Fachhochschule Nordwestschweiz
Hochschule für Technik

AI-based assistant for personalized relocation

Bachelor Thesis

Windisch, August 2025

Students	Stefan Simic Damjan Stojanovic
Expert	Sibylle Oeschger
Supervisors	Kevin Kim
	Nitish Patkar
Client	Byung Yun Cho
Project number	IIT04

Fachhochschule Nordwestschweiz, Hochschule für Technik

Abstract

The company *Swissplatz* offers a relocation service and is based in Geneva, Switzerland. Its core mission is to help clients find suitable housing solutions in Switzerland. Many of these clients come from abroad, which means they are often unfamiliar with the local language and Swiss rental laws. Throughout the entire process, the client does not have direct contact with landlords. All communication is handled by a mediator, ensuring that important information is not lost or misunderstood.

Currently, *Swissplatz* manages client preferences using Excel sheets and handles the search and communication manually. This is especially time-consuming, as the Swiss housing market is spread across several different platforms. Each platform must be checked individually in order to maintain an up-to-date overview.

The goal of this project is to simplify and accelerate this process by providing digital support. A central tool should help collect housing offers from various platforms and assist mediators in managing client interactions more efficiently. The system is intended to reduce repetitive tasks with automation and the use of Artificial Intelligence and improve the overall quality and speed of the service.

Acknowledgements

We would like to thank our supervisors, Prof. Dr. Kevin Kim and Dr. Nitish Patkar, for their support and guidance throughout the project. We are also thankful to our client, Byung Yun Cho, for the opportunity to work on this project and for the constructive collaboration.

Furthermore, we would like to thank everyone who took the time to review our document and provide valuable feedback. Special thanks go to all participants in the user testing sessions, whose input has been important to improving the quality and usability of our solution.

Contents

List of Figures

List of Tables

1 Introduction

1.1 Background and motivation

Swissplatz is a mediation company that connects individuals looking to rent an apartment with platforms that offer rental listings. Throughout the entire process, the client has no direct contact with the landlord. All communication goes through the mediator. This is especially important as many clients come from abroad and are unfamiliar with the local language and rental laws.

The workflow consists of three main phases: **Searching**, **Administration**, and **Viewing**.

In the Searching phase, the client shares their preferences with the mediator, who then searches for suitable apartments and informs the client. This process is iterative until a suitable property is found. If necessary, the mediator also contacts landlords to clarify missing information in listings.

The Administration phase involves the exchange of information between clients and landlords via the mediator. Since landlords often require different documents and use their own forms, automation is challenging, leading to extensive back-and-forth communication.

In the Viewing phase, the mediator arranges the apartment viewing appointment. Due to the high level of manual work required, these processes are time-consuming. Finding ways to optimize and partially automate certain steps could significantly improve efficiency.

1.2 Objectives and vision of the project

The goal of this project is to create an integrated platform that streamlines communication and automates key aspects of the rental mediation process. Currently, the client preferences are managed in an excel file, and mediators manually search for listings and manage communication. The new solution aims to centralize these processes on a single platform, improving efficiency and accessibility.

The platform will serve as a hub where clients can enter their housing preferences, while mediators manage rental offers in one place and upload them manually if needed. All interactions - such as feedback on listings, document requests, and appointment coordination - will take place within this platform, ensuring smooth and transparent communication between clients and mediators.

A key aspect of the project is automating the property search process. The system will gather listings from various sources, reducing the need for manual searching and allowing mediators to focus on high-value tasks. By integrating these functionalities, the platform will enhance scalability, improve user experience, and provide a structured yet flexible solution for rental mediation.

This project, therefore aims to answer following research questions:

- What are the key challenges and best practices in retrieving and processing data from various platforms to enable automation in the rental mediation process?
- How can the user experience be optimized to simplify data input and interaction between clients and mediators and how is it perceived by users?
- What are the possibilities of artificial intelligence that can be utilized to automate processes in the rental mediation workflow?

1.3 Structure of the documentation

This documentation outlines the systematic approach we followed for the design and integration of the developed solution.

The structure of the document reflects our workflow: we begin with a State of the Art analysis to address our research questions, examine current industry practices, and identify where our solution fits within this landscape. Based on these insights, we developed abstract solution concepts, independent of specific technologies. From these concepts and the resulting requirements, we proceeded with the concrete implementation.

Finally, the solution was evaluated, discussed, and potential directions for future work were identified.

Thus, the structure of this document mirrors the sequence of our methodology.

1.4 Solution approach

To develop an effective solution tailored to our customer's needs, we followed an iterative and feedback-driven approach. Early in the project, we conducted meetings with our supervisors and our customer to gather initial requirements and understand their expectations.

Based on these inputs, we created prototypes and proof-of-concept implementations, which we regularly presented to stakeholders to gather feedback. This iterative process allowed us to refine the solution continuously, adapt to changing goals, and ensure alignment with customer desires.

Throughout the project, we defined and adjusted the project scope collaboratively. This helped manage feature changes in a controlled way while keeping the main objectives in focus.

To validate the usability of our solution, we conducted user tests with individuals outside the development team. Their feedback gave us valuable insights into real-world usage and guided improvements to the user interface and overall user experience.

While conducting our research, we focused primarily on answering the defined research questions. This approach allowed us to align our work with established industry standards while integrating our own contributions.

For our research, documentation and development process, we relied on the following tools:

- **Google Scholar**: Utilized for targeted searches of scientific articles, publications, and other academic sources to obtain relevant theoretical foundations.
- **General Google Search**: Applied to gather broader information, explore practical applications, and identify industry practices.
- **AI-Based Chatbots**: Leveraged modern AI tools such as ChatGPT to structure and refine information, explore alternative perspectives, assist in problem-solving, and support the documentation process (including spelling, grammar, and overall document structure).

2 Background

2.1 Initial Situation

Swissplatz is a relocation company based in Geneva, Switzerland. Its core mission is to assist individuals-especially those moving from abroad-in finding suitable housing in Switzerland. The process is currently highly manual and is structured into three main phases:

- **Searching:** The client communicates their preferences (e.g., location, budget, number of rooms) to a mediator. The mediator manually searches across various real estate platforms for suitable listings and shares them with the client. This step often requires iteration until the client expresses interest in a specific property.
- **Administration:** Once a client shows interest, the mediator begins the administrative process of contacting landlords, gathering necessary documents, and managing forms. Since each landlord has their own requirements, this part is highly individualized and difficult to automate.
- **Viewing:** When an agreement is reached, the mediator arranges the apartment viewing. This includes coordinating schedules between client and landlord and confirming appointments.

Each of these phases is currently handled manually by the mediators. Client information and housing preferences are stored in Excel sheets, and all communication is done via email or phone. This makes the workflow time-consuming, error-prone, and difficult to scale.

The overall vision is to automate all phases of the process. However, the primary focus of this project is the automation of the **Searching Phase**. In doing so, we are developing the platform that serves as the foundation for further automation.

2.2 Initial Technical Setup

Swissplatz currently operates a website through which potential customers can register for the service and establish initial contact.

Once a customer has expressed interest, Swissplatz collects their preferences during a personal meeting. These details are then recorded manually in an Excel spreadsheet, which serves as the central working document.

From there, Swissplatz employees search for apartments on various external platforms and log the results in the same spreadsheet. This initiates an iterative process in which apartments are presented to the customer, feedback is gathered, and further searches are conducted until a suitable apartment is found.

All information related to a customer - including preferences, search history, and apartment details - is stored exclusively in this Excel file. As a result, Swissplatz currently lacks a centralized software solution, and critical business processes are spread across multiple disconnected tools, making scalability and collaboration more difficult.

Furthermore, communication with both landlords and customers takes place via different, unrelated channels such as email and phone calls, leading to fragmented information flow and potential inefficiencies.

2.3 Stakeholders

The primary stakeholders involved in this project are:

- **Clients:** Individuals looking to relocate to Switzerland. Their main concern is finding housing quickly and easily, despite language or legal barriers.
- **Mediators:** Employees at Swissplatz who manage the entire relocation process. They will be the primary users of the dashboard and internal tools.
- **Landlords:** Landlords could profit from unified communication, making the overall process better.

3 State of the Art

This chapter provides an overview of current technologies, research, and solutions relevant to our project. It sets the foundation for our design and implementation by analyzing existing approaches and identifying gaps or limitations. The chapter is divided into related work, competitor analysis, application of artificial intelligence in relocation services, scraping techniques, middleware/web technologies, and legal and ethical considerations.

3.1 Related Work / Literature Review

Several academic studies and industry reports address the use of AI and chatbots in the housing and relocation domain. These works provide insight into how digital tools can assist with data collection, search optimization, and user interaction. While many industries, such as finance and healthcare, have already started integrating artificial intelligence into their workflows, the relocation industry has been slower at adopting it.

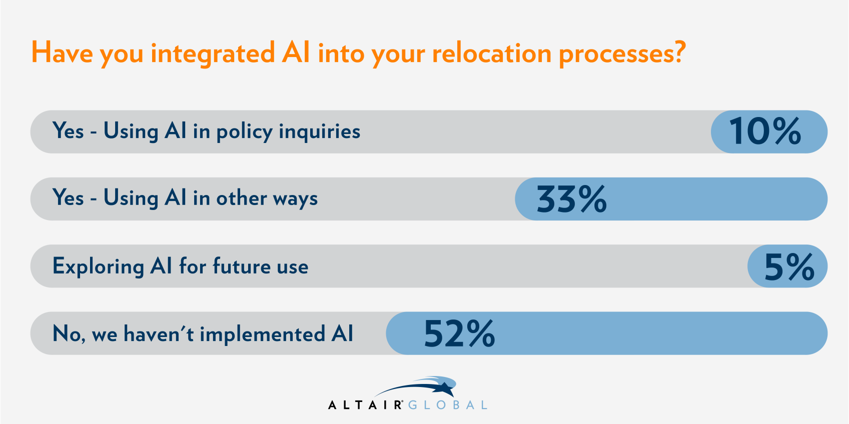


Figure 3.1: Use of AI in relocation industry

This diagram from Altair Global [1] shows that even more than half of the respondents haven't integrated AI into their relocation processes, and 5% are still exploring options. It showcases that there is still a lot of open potential in this industry.

While the relocation sector has been slower to adopt AI, research in related housing areas shows how these technologies can help solve complex information and process challenges. For example, Subedi [2] developed the Landlord-Tenant Rights Bot, an AI chatbot that gives tenants and small landlords state-specific legal information in the United States. It uses Retrieval-Augmented Generation (RAG) with a database of legal summaries to provide accurate and relevant answers about evictions, security deposits, and fair housing rules. Early testing showed a 90% satisfaction rate for how clear the answers were, showing that conversational AI can make legal information easier to understand and help improve housing stability. Even though this work focuses on landlord-tenant law rather than relocation directly, the same AI methods could be used in relocation to handle legal questions, manage documents, and give clients personalized guidance.

In a related but different domain, Isinkaye et al. [3] presented a mobile-based hostel location chatbot system with integrated recommendation capabilities. The chatbot helps students find hostels that match their preferences and uses a content-based filtering approach to generate a ranked list of suggestions. Their evaluation with 152 users showed high ratings in areas such as recommendation accuracy, platform compatibility, and user friendliness. Although targeted at student accommodation rather than relocation, this work demonstrates how recommendation systems and conversational interfaces can streamline the search process and improve user satisfaction. All this are concepts that could also benefit AI-driven relocation platforms as well.

Most of the systems in these studies are either commercial or only available as research demos, so there is still space for open and freely available AI tools in the relocation field. This research showcases that while there is some related work, there is still a clear gap for widely accessible AI solutions that directly address relocation needs.

3.2 Existing Solutions / Competitor Analysis

3.2.1 Existing Housing Search Platforms in Switzerland

Several platforms in Switzerland already provide online housing search services, including popular sites such as **ImmoScout24**, **Comparis**, **Homegate**, and **Flatfox**. These platforms are well-established and offer a wide range of search filters, map-based browsing, and up-to-date listings. They are designed for individual users who want to explore available apartments or houses on their own. While their search engines are powerful and provide large inventories, their functionality is limited to the discovery phase: they do not guide users through the application process, offer mediation between tenants and landlords, or provide direct assistance with legal and administrative questions. In practice, this means that users are left to handle document preparation, landlord communication, and contract questions entirely on their own.

A key limitation for many newcomers to Switzerland is that these platforms offer their interfaces and communication channels primarily in German, French, or Italian, with limited multilingual support. They also lack integrated features to assist non-local users in understanding the rental process, legal requirements, or cultural expectations. For someone unfamiliar with the Swiss housing market, this can make the process slow and confusing.

In contrast, some relocation services and human relocation agents offer more comprehensive support. These services do not just present listings, they actively assist with the entire rental process. This can include automated property matching based on client preferences, help with completing and submitting application documents, guidance on local rental laws, and translation or interpretation services for non-native speakers. Relocation agents may also coordinate viewings, negotiate with landlords, and clarify contractual details, ensuring that clients understand every step of the process. While such services can be highly valuable, they are typically offered at a premium cost and may not be accessible to all potential tenants.

The comparison between pure search platforms and relocation services highlights a clear gap in the Swiss market: there is no widely available, AI-supported platform that combines the convenience of powerful search tools with the personalized guidance and mediation found in relocation assistance. Bridging this gap could make the housing search significantly easier, especially for international clients or those unfamiliar with the Swiss rental system.

Feature	**Search Platforms (e.g., ImmoScout24, Comparis, Homegate, Flatfox)**	**Relocation Services / Agents**
Main Purpose	Provide online property listings and search filters	Assist clients through the entire rental process, from search to contract signing
User Guidance	No personal guidance, self-service only	Personalized support, tailored to client's needs and situation
Language Support	Mostly German, French, Italian; limited English support	Multilingual support
Mediation with Landlords	None	Acts as intermediary between tenant and landlord
Document Handling	No document preparation or submission help	Assists with preparing and submitting application documents
Legal and Process Support	No legal guidance or process explanations	Provides explanations of Swiss rental laws and local practices
Cost	Free for users	Premium service, usually paid by client or employer
Target Audience	Local residents familiar with the Swiss housing market	Newcomers, expats, and clients unfamiliar with Swiss rental system

Table 3.1: Comparison of Swiss housing search platforms and relocation services

This table provides a clear summary of the main differences between common Swiss housing search platforms and relocation services.

3.3 AI in Relocation Services

The use of artificial intelligence in relocation is still developing, but it already shows potential to improve many parts of the process. From guiding users during the initial inquiry to organizing data and supporting decision-making, AI can enhance efficiency and user experience. The following diagram illustrates one possible workflow where AI is integrated into different stages of relocation services.

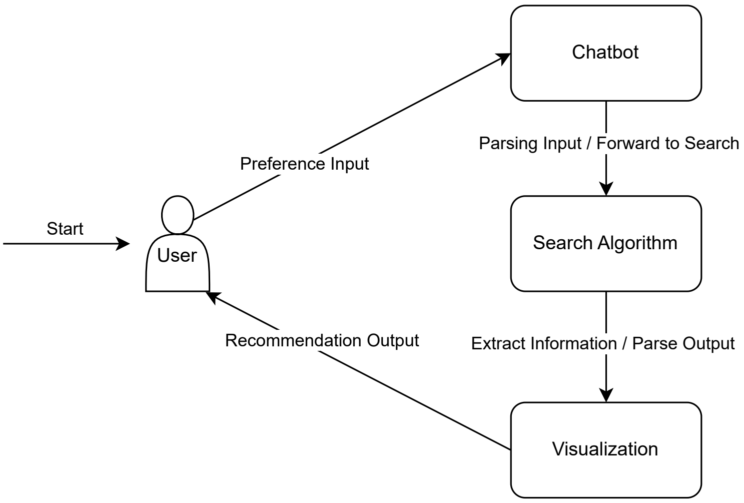


Figure 3.2: Workflow AI in Relocation Services

- **Chatbots:** Assist users in describing preferences and understanding processes.
- **Search Algorithm:** Suggest suitable listings based on prior selections or similar users.
- **Information extraction / visualization:** Parse listing descriptions and extract structured data (e.g., number of rooms, rent).

3.3.1 Chatbots

AI-driven Chatbots have become a regular tool for todays industry. Not only is it a daily work tool, for some people it has become a part of life which can provide help in any case. Their powerful ability to imitate and simulate a real human being is what makes it so appealing.

That is why it is important to take a close look and analyse what the industrial standard is.

First, we should take a moment to understand what a chatbot is and how it works behind the scenes. According to Oracle [4], a Chatbot is a computer program that simulates and processes human conversation (either written or spoken), allowing humans to interact with digital devices as if they were communicating with a real person. Driven by AI, automated rules, natural-language processing (NLP), and machine learning (ML), chatbots process data to deliver responses to requests of all kinds. Popular Examples are ChatGPT from OpenAI and Microsoft Copilot.

A key advantage of chatbots is their ability to work around the clock and respond instantly, which makes them particularly useful in industries where quick answers are important. In relocation services, they can guide users through the housing search by asking structured questions about preferences, such as location, budget, and property type. Instead of filling out static forms, users interact in a conversational way, which often feels more natural and less time-consuming. Modern chatbots can also be connected directly to databases or APIs, allowing them to provide real-time information about available listings.

For example, when a user specifies their criteria, the chatbot can immediately filter through housing data and present matching results. In more advanced systems, they can even learn from previous interactions to improve recommendations over time.

3.3.2 Examples in the industry

An example of an AI-supported tool in the Swiss relocation market is the *Prime Relocation AI Chatbot*[1]. This chatbot does not directly search for properties or browse housing listings. Instead, it is designed to support users with general information about the relocation process, provide access to city guides, forward specific questions to the company's expert team and help arrange appointments with relocation specialists. The actual property search is handled by human relocation experts, who have access to various resources, including some that are not publicly available, and can perform tailored searches based on the client's individual requirements.

From a functional perspective, the chatbot acts mainly as a communication and triage interface rather than an integrated search or mediation system. It demonstrates how AI can be used to provide quick, structured responses and guide users towards relevant human support channels. However, it is not connected to real-time property databases and does not automate steps such as landlord communication or legal guidance. In this sense, the AI component supports the service indirectly, with the core relocation work still carried out by human agents.

Another example is the relocation technology offered by *ARC Relocation*[2]. ARC provides a client portal called *RELO AI*, which is designed to centralize the relocation process in one place. Through this portal, clients can track every step of their move, access important documents, communicate with relocation coordinators, and receive updates in real time. The system also includes tools for budgeting, expense tracking, and task management, helping clients stay organized throughout the process.

While ARC's platform uses AI elements to improve recommendations and manage workflows, the main strength lies in its integration of all relocation services into a single interface. Clients can view status updates, receive personalized alerts, and interact with service providers directly through the platform. This reduces the need for multiple communication channels and gives both clients and relocation managers a clear overview of progress.

Similar to Prime Relocation, ARC's system does not focus on automated property search through public listings. Instead, the housing search is still handled by human relocation consultants, who use the platform to share options with clients. The AI and digital tools work mainly to improve efficiency and coordination, while the core property-matching process remains human-driven.

The examples of Prime Relocation and ARC Relocation show that AI is already being used in the relocation industry to improve communication and provide better client support. However, these systems do not fully automate the relocation process. Key tasks such as property searching and document handling are still mainly carried out by human experts, with AI acting as a supportive tool rather than a complete replacement.

[1] `https://www.primerelocation.ch/ai`

[2] `https://arcrelocation.com/relocation-technology/`

3.4 Data Gathering

In modern web systems, acquiring up-to-date external data is often essential. In our case, apartment listings are distributed across various platforms, each with their own structure, accessibility, and limitations. This section presents an overview of different data acquisition methods that are commonly used in such systems.

Each technique has trade-offs in terms of legality, stability, accuracy, and performance. Depending on the provider's technical offering and access policies, one or more methods may be necessary.

3.4.1 API-Based Access

An **Application Programming Interface (API)** offers a structured and standardized way to retrieve data from a provider. When publicly documented or contractually available, this is often the most reliable and legally compliant method for data gathering.

Advantages:

- Officially supported and typically stable over time.
- Returns well-structured data formats (e.g., JSON, XML).
- Developer-friendly, often with extensive documentation.

Challenges:

- May require registration, an API key, or a paid license.
- Access is limited to the data explicitly exposed.
- Many providers do not offer a public API at all.

In the scope of this project, we contacted various housing search platforms for possible access to their API. For example, *Flatfox*[3] and *ImmoScout24*[4] both provide APIs. However, these are unusable for our purposes: the ImmoScout24 API only covers Germany, and the Flatfox API is intended solely for internal use.

3.4.2 Direct Database Access

In rare cases, public services provide direct read access to their databases (or to a database replica) - for instance, via SQL queries or downloadable data dumps. This method enables full data retrieval without relying on frontend scraping or API limitations.

Advantages:

- Complete access to all available records, often including historical data.
- No rate limits imposed by a frontend or API.

Challenges:

- Typically requires an official partnership or internal system access.
- Data often needs extensive transformation to be usable.
- Requires deeper technical knowledge to interpret schema and relations.
- Risk of outdated schema or missing documentation.

Currently, there is no known direct database access for Swiss apartment or housing listings.

[3] https://flatfox.ch/en/docs/api/

[4] https://api.immobilienscout24.de/

3.4.3 Static Web Scraping

Based on this definition, we can distinguish between scraping *HTML pages* and *JavaScript-based pages*, since the approach differs significantly depending on how the website is built.

> Scraping is the process used to locate and retrieve DA (Development Applications) data from council websites: addresses are searched for on the DA trackers and relevant information is saved. The council websites that are scraped are served in either Hypertext Markup Language (HTML) or embedded in a JavaScript [5].

The first approach - **static scraping** - involves parsing HTML content from publicly visible webpages. Libraries such as `BeautifulSoup` or `Scrapy` can extract structured data directly from the HTML code, similar to how a browser's "Inspect Element" feature reveals a page's structure.

Advantages:

- No API access required.
- Can be adapted to almost any platform.
- Fast execution.
- All information visible to a regular website visitor can be retrieved.

Challenges:

- Website structure changes frequently → code requires maintenance.
- Legal gray areas (Terms of Service, `robots.txt` restrictions).
- Data often needs cleaning and normalization.
- Only publicly visible data is accessible.
- Can be blocked by anti-bot detection systems.

3.4.4 Dynamic / GUI-Based Scraping (Selenium)

Some platforms rely heavily on JavaScript to render content after the initial page load. In these cases, static HTML scraping fails because the desired data is not present in the raw HTML source. Instead, browser automation frameworks such as `Selenium` simulate actual user interaction and execute the JavaScript, allowing the scraper to capture dynamically generated content.

Advantages:

- Handles JavaScript-heavy, dynamically loaded pages.
- Can simulate complex user actions (clicks, scrolling, login, etc.).
- More resilient against some basic anti-bot measures.

Challenges:

- Significantly slower due to real-time rendering.
- Requires more resources (headless browser or full GUI).
- Can still be detected and blocked by advanced anti-bot systems.

3.4.5 Comparison of Techniques

Method	**Pros**	**Cons**	**Suitability for Housing Data**
API Access	Reliable, legal, structured data	Access restricted, not always available	Ideal (if available)
Database Access	Full data, no scraping needed	Requires direct agreement or dump	Very good (but rare for public platforms)
Static Scraping	Universal, no access needed	Fragile, terms of service sensitive, requires maintenance	Good for stable HTML sites
Dynamic / GUI Scraping	Works on dynamic pages	Fragile, Slow, setup-heavy	Only if JS prevents scraping or it gets detected as a bot

Table 3.2: Comparison of data gathering methods for housing platforms

In practice, scraping remains one of the few viable options for accessing housing data from platforms that do not provide an API or partnership. While it introduces risks and complexity, careful engineering and ethical considerations (e.g., rate limiting, identifying as a bot, not scraping protected content) can mitigate these issues.

As part of this analysis, requests were sent via email to selected providers, such as Flatfox and Immobilier, to ask about possible API access and the legal or technical feasibility of automated data collection. The replies from these providers gave additional context to the publicly available information and helped clarify certain limitations. The full email correspondence is included in Appendix.

3.5 Middleware

Modern web systems are composed of several layers that need to interact in a reliable and efficient way. The **middleware** is a crucial architectural layer that sits between the frontend (user interface) and the backend services (e.g., database, scraping logic, or external APIs) [6].

It handles business logic, validates and transforms data, routes requests, manages authentication, and ensures communication between all modules. In essence, it is the *central control unit* or *heart* of the application - responsible for keeping all parts synchronized and responsive [7].

Key responsibilities of the middleware:

- Accepting and responding to HTTP requests (e.g., from frontend or chatbot)
- Querying the database and transforming results
- Coordinating with the data scraping service
- Authenticating users and authorizing admin access
- Structuring the application's business logic (e.g., preference handling, result ranking)

Requirements for a good middleware:

- **Fast**: Low-latency communication between frontend and backend
- **Scalable**: Able to handle multiple concurrent sessions or users
- **Available**: Minimal downtime and quick recovery
- **Maintainable**: Clear structure and modern programming practices
- **Extensible**: Easy to integrate new components (e.g., new data sources or chatbot modules)

In this project, we decided to use Python for the middleware development, primarily due to the preference and technical background of the customer. Python offers excellent libraries and frameworks for API development, and it integrates well with machine learning models and database systems.

3.5.1 Popular Middleware Frameworks in Python

- **Django REST Framework (DRF)**[5]**:** Based on Django, this full-featured toolkit supports rapid development of RESTful APIs. It provides serialization, authentication, view handling, and an admin panel out of the box.
- **FastAPI**[6]**:** A newer, high-performance web framework built on top of Starlette and Pydantic. FastAPI is asynchronous, type-safe, and very fast. It is ideal for modern microservice architectures and supports automatic OpenAPI documentation.
- **Flask + Flask-RESTful**[7]**:** A lightweight option with flexible architecture, but requires more manual setup. Less suitable for complex business logic or scaling needs.
- **Sanic**[8]**:** Lower-level asynchronous web frameworks with a strong focus on performance, but steeper learning curves and fewer integrations.

[5] https://www.django-rest-framework.org/

[6] https://fastapi.tiangolo.com/

[7] https://flask.palletsprojects.com/en/stable/

[8] https://sanic.dev/en/

3.6 Frontend / Website Technologies

The frontend is the interface through which users interact with the application. In the context of a relocation assistant or apartment search platform, this interface must support real-time interactivity, dynamic filtering, chatbot integration, and data visualization.

Modern frontend development is increasingly based on component-driven frameworks. These frameworks help structure complex interfaces into reusable parts, improve maintainability, and support routing, state management, and user interaction.

This section presents a comparison of widely adopted frontend frameworks suitable for building responsive and interactive web applications.

3.6.1 Overview of Frontend Frameworks

The following table compares the most relevant technologies in terms of their suitability for interactive search-based platforms.

Aspect	**Angular**[9]	**React**[10]	**Vue / NuxtJS**[11]
Type	Full-featured framework	Library with ecosystem	Lightweight framework
Language	TypeScript (strict, structured)	JavaScript / TypeScript (flexible)	JavaScript (optionally TypeScript)
Learning Curve	Steeper; opinionated and structured	Medium; flexible but fragmented	Low to medium; intuitive syntax
Routing / State Management	Built-in modules	External libraries (React Router, Redux)	Vue Router, Vuex (optional in NuxtJS)
Performance	High; optimized for large apps	High; fast rendering via virtual DOM	Very fast; optimized reactivity system
Tooling	Strong CLI and dev tools	Wide variety of tools and starter kits	Lightweight CLI, integrated tooling
Ecosystem / Community	Mature, enterprise-level support	Very large and active community	Growing, especially in Asia and Europe
Suitability for Dashboard Interfaces	Excellent structure for complex admin panels	Flexible; many open-source UI libs	Good; simple for smaller dashboards
Suitability for Chatbot Integration	Strong binding and state management	Flexible; well-suited for dynamic components	Easily integrates via components

Table 3.3: Comparison of popular frontend frameworks for interactive web applications

[9] https://angular.dev/

[10] https://react.dev/

[11] https://nuxt.com/

3.6.2 Considerations for Framework Selection

When selecting a frontend technology for a relocation or apartment search platform, several technical and project-related factors must be considered:

- **Complexity of UI**: Does the project require a structured admin dashboard or just simple pages?
- **Developer experience**: Is TypeScript familiarity important? Are team members experienced in a specific framework?
- **Integration needs**: How will the frontend interact with middleware, databases, or chatbots?
- **Community and longevity**: Is long-term support, security, or active development important for the project?

3.7 Database

Modern digital platforms depend heavily on well-structured databases that support fast, reliable access to data. In our project, the database is a critical component that stores apartment-listings, user data, search preferences, and chatbot interactions. It must support structured models with relations between different entities, allow efficient querying, and scale with growing data volume.

In this section, we present different types of databases relevant to our domain.

3.7.1 Relational Databases

Relational databases store data in structured tables that are connected through defined relationships. Each row (record) represents an entity, and columns define its attributes. They are based on SQL (Structured Query Language) and follow a strict schema.

Advantages:

- Excellent for structured data with clear relationships (e.g., searchresult → apartment, chatbot → searchquery).
- Mature ecosystem (e.g., PostgreSQL, MySQL) with strong support for indexing and ACID transactions.
- Easy to enforce data integrity through constraints and foreign keys.

Disadvantages:

- Schema changes can be complex.
- Not ideal for semi-structured or unstructured data.

Famous relational databases are MySQL and PostgreSQL.

Aspect	**MySQL**[12]	**PostgreSQL**[13]
Advantages		
Popularity	Very user-friendly, widely supported by shared hosting providers	Also open source, widely adopted for professional applications
Performance	High performance for simple read operations	Optimized for complex queries and larger data sets
Community Support	Large user base, many tutorials available	Strong documentation, developer-focused
Compatibility	Strong integration with PHP platforms (e.g., WordPress)	Well-suited for modern stacks (Python, JS, etc.)
Disadvantages		
SQL Compliance	Limited SQL standards support (e.g., missing window functions)	Highly SQL-compliant (supports CTEs, window functions, etc.)
Complex Data Handling	Less suitable for heavily relational data	Strong handling of relations and foreign keys
Data Validation	Looser type enforcement, can lead to silent errors	Strict type system and constraint handling
JSON Support	Basic JSON functionality only	Advanced JSON support with indexing and JSONPath

Table 3.4: Comparison of MySQL and PostgreSQL

3.7.2 NoSQL Databases

NoSQL (Not Only SQL) databases were developed to handle large-scale, often unstructured data. They do not require a fixed schema and can store documents (e.g., JSON), key-value pairs, wide-columns, or graphs.

Advantages:

- Flexible structure: ideal for rapidly changing or nested data (e.g., chatbot session logs).
- High scalability and performance for large datasets.

Disadvantages:

- No standardized query language (SQL-like querying varies per DB).
- Harder to enforce data relationships and constraints.

Popular examples include MongoDB (document-based) and Redis (key-value).

[12] https://www.mysql.com/

[13] https://www.postgresql.org/

Aspect	MongoDB[14]	Redis[15]
Advantages		
Data Model	Document-based (JSON-like), ideal for flexible and nested data	Key-value store, extremely fast access and caching
Scalability	Horizontally scalable with sharding support	In-memory architecture, built for high-throughput workloads
Use Cases	Good for semi-structured data, logs, and rapidly changing schemas	Ideal for caching, session storage, queues, pub/sub
Querying	Supports rich queries, indexing, aggregation pipelines	Extremely low-latency key-based lookups
Tooling	Mature ecosystem, GUI tools like Compass, Atlas cloud hosting	Lightweight, widely used with strong community support
Disadvantages		
Consistency	Uses eventual consistency by default (can be tuned)	Data is ephemeral unless persistence is explicitly configured
Memory Usage	Stores full documents (can be large)	Fully in-memory - limited by RAM capacity
Relationships	No joins; manual handling of relations needed	No built-in support for complex structures or queries
Data Durability	Needs replication or journaling for strong durability	Requires configuration to persist data between reboots

Table 3.5: Comparison of MongoDB and Redis (NoSQL Databases)

3.7.3 Object-Oriented Databases

Object-oriented databases store data as objects, closely reflecting object-oriented programming paradigms. They are rarely used in web-scale applications today but can be suitable when application logic maps tightly to complex data structures.

Advantages:

- Natural mapping to code in OOP languages.
- Reuse of objects directly in code (no ORMs needed).

Disadvantages:

- Less common, limited tooling, smaller community.
- Poor interoperability with standard web systems.

[14] https://www.mongodb.com/

[15] https://redis.io/

3.7.4 Comparison of Database Types in the Context of Apartment Search

Choosing the appropriate type of database is essential for the long-term scalability, performance, and maintainability of the application. The apartment search domain typically requires structured, interconnected data (e.g., apartments, users, preferences, interactions). In this section, we compare relational databases, NoSQL document stores, and in-memory key-value stores regarding their suitability for such a system [8], [9].

Aspect	**Relational DB (e.g., PostgreSQL)**	**Document Store (e.g., MongoDB)**	**Key-Value Store (e.g., Redis)**
Data Structure	Tables with rows and columns; strict schema	Flexible JSON-like documents; schema-free	Simple key-value pairs (strings, lists, sets)
Relations Between Entities	Strong support via foreign keys and joins	Manual linking; no join support	No native relationship modeling
Schema Flexibility	Rigid but controlled; ideal for predictable data	Highly flexible; good for evolving data models	No structure enforcement
Query Capabilities	Powerful SQL queries, aggregation, joins	Rich queries and aggregations, but limited joins	Extremely limited (key-based access only)
Use Case Fit: Listings	Suited for structured fields (price, size, location)	Can store variable fields in documents	Inefficient for anything but quick lookups
Use Case Fit: User Preferences	Easy to model user → preferences → results relations	Flexible but harder to normalize or constrain	Poor fit unless extremely simple
Performance (Read)	Optimized with indexes and caching	Good, especially when denormalized	Very fast (RAM-based)
Performance (Write)	Transactional integrity, slower but safer	Fast inserts; eventual consistency	Extremely fast
Scaling	Vertical and horizontal (via replication, sharding)	Horizontal via sharding	Scales well but memory-limited
Tooling / Ecosystem	Mature (ORMs, admin tools, migrations)	Good GUI and cloud services (e.g., Atlas)	Lightweight tools; mostly used for caching
Suitability for Core Data (Listings, Users)	**Very suitable**	Suitable with trade-offs	Not very suitable for structured storage

Table 3.6: Comparison of database types for apartment search applications

4 Methods

4.1 Project Methodology

To ensure the successful implementation of the project, we rely on methodologies that cover the following key aspects:

Project Planning: Combination of Scrum and Kanban

For high-level project planning, Scrum is used to define clear phases, milestones, and dependen- cies, ensuring a structured workflow throughout the entire project timeline. This provides a solid framework while maintaining flexibility. Kanban complements this approach by enabling agile task management, allowing the team to adapt to changes as needed. Kanban boards are used to visualize progress continuously, ensuring transparency and improving responsiveness to new requirements or challenges.

Requirements Analysis (Requirements Engineering)

To define user needs, expectations, and technical requirements, an iterative process is applied. Regular meetings with clients and supervisors are held to gather and document requirements, serving as a foundation for development and helping to establish clear goals. The focus is on answering key questions that drive the project forward. Additionally, literature research and competitor analysis are conducted to derive industry-specific requirements for the product.

User-Centered Design

A user-centered approach is followed to ensure that the platform is intuitive, user-friendly, and functional. Iterative prototypes serve as the foundation for continuous usability testing, where feedback from potential users is gathered and integrated into the development process. This ensures that the final product meets user expectations and enhances overall usability.

Scientific Research and Literature Review

To address specific research questions and requirements, a thorough literature review is conducted. This includes analyzing scientific articles, technical reports, and industry best practices, as well as comparing competitor products. Additionally, discussions with industry experts provide valuable insights, helping to translate theoretical knowledge into practical solutions.

Prototypes

To validate early ideas and gain rapid feedback, low- and high-fidelity prototypes were created throughout the development process. These prototypes served as a communication tool between the development team, the client, and potential users. Early wireframes and interface mockups were used to visualize the core user interactions, layout structures, and essential features of the platform.

By presenting these prototypes in regular feedback meetings, we were able to collect targeted input, align expectations with stakeholders, and identify potential usability issues before implementation. This iterative prototyping approach allowed us to make informed design decisions and reduce the risk of late-stage rework.

4.2 Proof of Concepts

To evaluate technical feasibility and assess critical assumptions, we implemented multiple proof-of-concept (PoC) solutions during the early and middle stages of development. These small-scale experiments focused on validating essential components, such as the scraping logic, chatbot integration, and backend communication.

Each PoC provided a focused insight into a specific challenge and enabled the team to test libraries, APIs, and frameworks in a controlled environment. By doing so, we were able to identify potential technical limitations early, explore alternative implementations, and strengthen our overall architecture based on proven methods.

4.3 System Usability Scale

In order to systematically assess the usability of our solution, we conducted evaluations using the System Usability Scale (SUS). SUS is a standardized questionnaire that allows us to measure perceived usability based on real user feedback. It consists of 10 statements that users rate after interacting with the system.

This method provides a quick and reliable overview of user satisfaction, highlighting both strengths and weaknesses in the user interface. The insights gathered through SUS helped us prioritize improvements, validate our design choices, and ensure that the platform meets a high standard of user experience.

4.4 Usability Testing

Usability Testing is a user-centered evaluation method used to assess how easily and effectively people can use a product or system. It involves observing representative users as they attempt to complete typical tasks, with the goal of identifying usability issues, measuring user performance, and gathering feedback to improve the design. This method focuses on practical, real-world interactions rather than theoretical assumptions, making it a valuable approach for ensuring that a solution meets the needs and expectations of its target audience.

5 Conceptual Design

This chapter presents the theoretical foundation and design concepts that guide the implementation of our platform. It focuses on the high-level structure of the system and its components while abstracting from the technical implementation details. The goal is to establish a clear architectural vision that aligns with the requirements defined in the requirements chapter and serves as a blueprint for the development phase.

The chapter is structured into key subsystems: the overall system architecture, the database design, the chatbot logic, middleware, data gathering module, and the frontend. Each of these components contributes to fulfilling the functional and non-functional requirements.

5.1 Requirements Specification

Before implementing a digital assistant for relocation, it is essential to clearly define what the system must achieve. This chapter presents the functional and non-functional requirements for each of the core components. The requirements are derived from the existing workflow at Swissplatz and the goals defined earlier in the project.

They serve as the foundation for system design and ensure that all technical decisions align with the actual needs of stakeholders. The following sections are structured by component: data gathering, dashboard and chatbot.

5.1.1 Data Gathering

Gathering relevant apartment data from various external platforms (e.g., ImmoScout24, Comparis) is a fundamental part of this project. The quality and reliability of this data directly influence the user experience and the success of the recommendation system. Therefore, precise functional and non-functional requirements are defined to guide the development and ensure robustness and consistency.

Functional Requirements

Nr.	Requirement	Description
DFA-1	Real data from different sources is found	The system must retrieve apartment listings from multiple online platforms reliably.
DFA-2	Data accuracy	The scraped information (price, rooms, location) must reflect what is shown on the source site, despite possible anti-scraping mechanisms.
DFA-3	Data storage	Retrieved listings must be stored in a database and retrievable on demand.
DFA-4	Search filters	The system must return only listings that match user-defined filters (e.g., location, price range, number of rooms).
DFA-5	Relevant data only	Irrelevant entries (e.g., parking spots, garages) must be filtered out.
DFA-6	Source reference	Each listing must include a reference to the original source (e.g., URL).
DFA-7	Field extraction	The scraper must correctly extract fields such as address, city, surface size, number of rooms, and rental price.

Table 5.1: Functional requirements for data gathering

Non-Functional Requirements

Nr.	Requirement	Description
DNF-1	Fault tolerance	The system must retry failed requests (e.g., due to timeouts or bad status codes).
DNF-2	Logging	All scraping actions and exceptions must be logged for debugging and traceability.
DNF-3	Extensibility	New sources must be integrable without breaking existing scraper logic.

Table 5.2: Non-functional requirements for data gathering

5.1.2 Dashboard

The Relocation Dashboard is the central interface for mediators to manage listings and user interactions. It should provide a clean, responsive, and intuitive user experience while enabling efficient workflows. Functionalities include displaying listings, tagging, and managing user sessions.

Functional Requirements

Nr.	Requirement	Description
DBF-1	Listing display	The dashboard must list apartments with essential fields like price, location, and size.
DBF-2	Status tagging	Listings can be tagged (e.g., “liked”, “pinned”, “disliked”) for tracking client feedback.
DBF-3	Apartment management	The system must support searching for apartments.

Table 5.3: Functional requirements for the dashboard

Non-Functional Requirements

Nr.	Requirement	Description
DBNF-1	Load performance	The dashboard must load fully in under 2 seconds on a standard desktop connection.
DBNF-2	UX consistency	The design must follow a consistent layout and visual language across all components.

Table 5.4: Non-functional requirements for the dashboard

5.1.3 Chatbot / AI Assistant

The chatbot will serve as the primary interface for client interaction. It collects preferences, answers frequently asked questions, and ensures smooth onboarding for users. The chatbot must maintain a professional tone, avoid off-topic discussions, and ensure that data is transferred correctly to backend services.

Functional Requirements

Nr.	Requirement	Description
CBF-1	Preference collection	The chatbot must collect structured preferences from users (location, price range, etc.).
CBF-2	Dialogue handling	The chatbot must be able to handle follow-up questions, misunderstandings, and clarifications.
CBF-3	Backend communication	Data collected by the chatbot must be forwarded in real time to backend services.
CBF-4	Conversation control	The chatbot must avoid and redirect off-topic or irrelevant discussions.
CBF-5	Chat memory	Chat history must be stored and available when the user returns.
CBF-6	Context awareness	The chatbot should keep track of conversation context and refer to previous user inputs.

Table 5.5: Functional requirements for the chatbot

Non-Functional Requirements

Nr.	Requirement	Description
CBNF-1	Response time	The chatbot must respond within 3 second after each user input.
CBNF-2	Concurrency	The chatbot must support at least 5 parallel conversations.
CBNF-3	Logging	All conversations must be logged securely for improvement and traceability.
CBNF-4	Professional tone	The chatbot must use polite and professional language at all times.

Table 5.6: Non-functional requirements for the chatbot

5.2 System Architecture

Our system consists of multiple modular components that interact to support and streamline the relocation process. The main subsystems are:

- **Frontend / Admin Interface:** The primary interface through which mediators manage users, apartment listings, and chatbot communication.
- **Data Gathering Engine:** A scraping module responsible for collecting apartment rental offers from multiple external platforms.
- **Middleware:** Acts as a communication layer between the frontend, backend services, database, and AI modules. It handles business logic and ensures secure, structured data flow.
- **Database:** Stores all persistent data, including user profiles, listings, preferences, search filters, and chatbot conversations.
- **Chatbot:** An AI-powered assistant that interacts with users to gather preferences, assist in the apartment search, and answer relevant questions.

All components communicate through the middleware, with the exception of the chatbot. The logic and operation of the chatbot is encapsulated and isolated, meaning it operates independently from the middleware. This loosely coupled architecture promotes modularity, flexibility, and scalability.

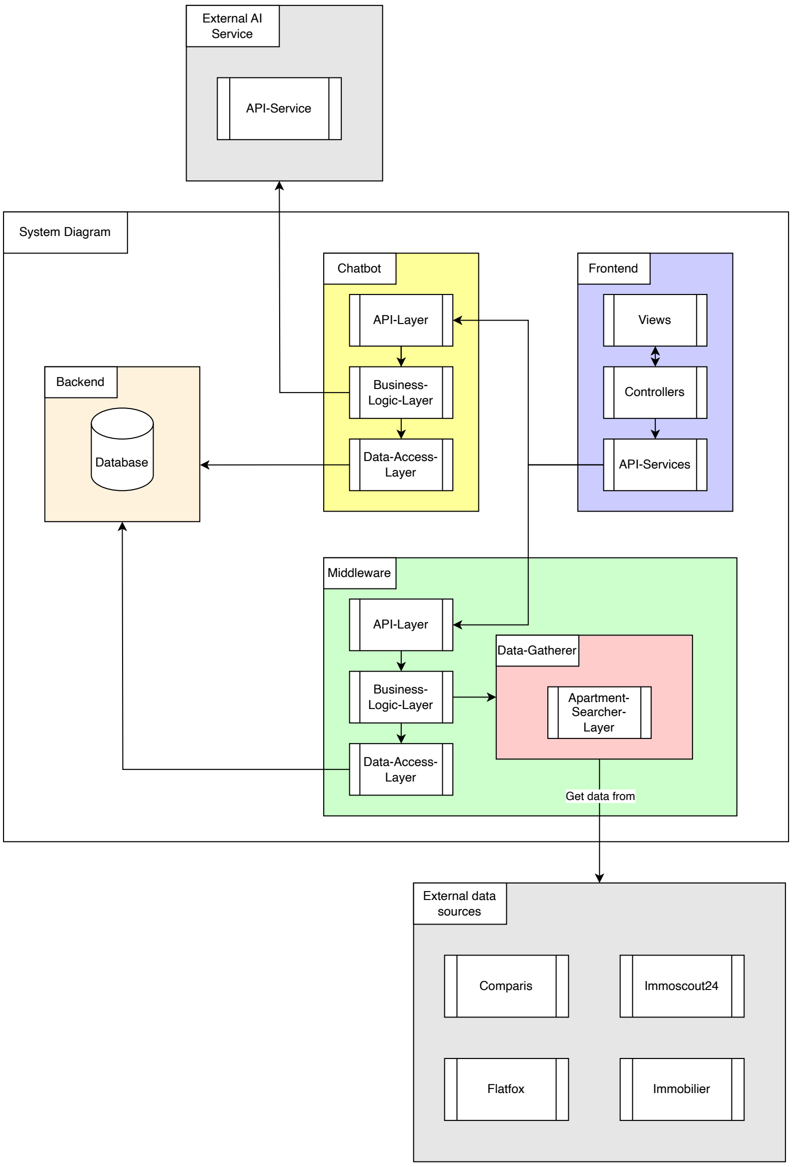


Figure 5.1: Conceptual System Architecture

The frontend loads all available data - including apartment listings and active or past conversations - and processes user inputs in real time. These inputs and associated data are sent through the middleware, which manages their integration and persistence in the backend database.

Most system data (excluding chatbot content) flows through the middleware. New information related to apartment preferences triggers the **Data Gathering Engine**, which immediately initiates a targeted search across configured external sources.

The **Chatbot** is fully responsible for managing user interactions, storing conversation histories, and communicating with AI services. Alongside the Data Gathering Engine, it is one of the two subsystems that directly communicate with external platforms (e.g., AI APIs and real estate websites).

5.3 Database

The database is designed to support the core data flows of the application: storing apartment listings, capturing user preferences, tracking chatbot conversations, and enabling administrative workflows.

The system relies on a relational database due to its structured and interconnected nature. Key entities are linked via foreign key relationships to maintain data integrity and traceability.

The main entities include:

- `Apartments`: Listings collected from external platforms via scraping or APIs.
- `Preferences`: A set of structured filters (e.g., location, price range, room count), created per chat session based on user input.
- `Chat`: Logs the conversation between the user and the AI assistant.
- `Feedback`: User responses related to individual apartments (e.g., “like”, “pin”, “dislike”).

The following diagram illustrates a simplified interaction workflow between the user, chatbot, and the database.

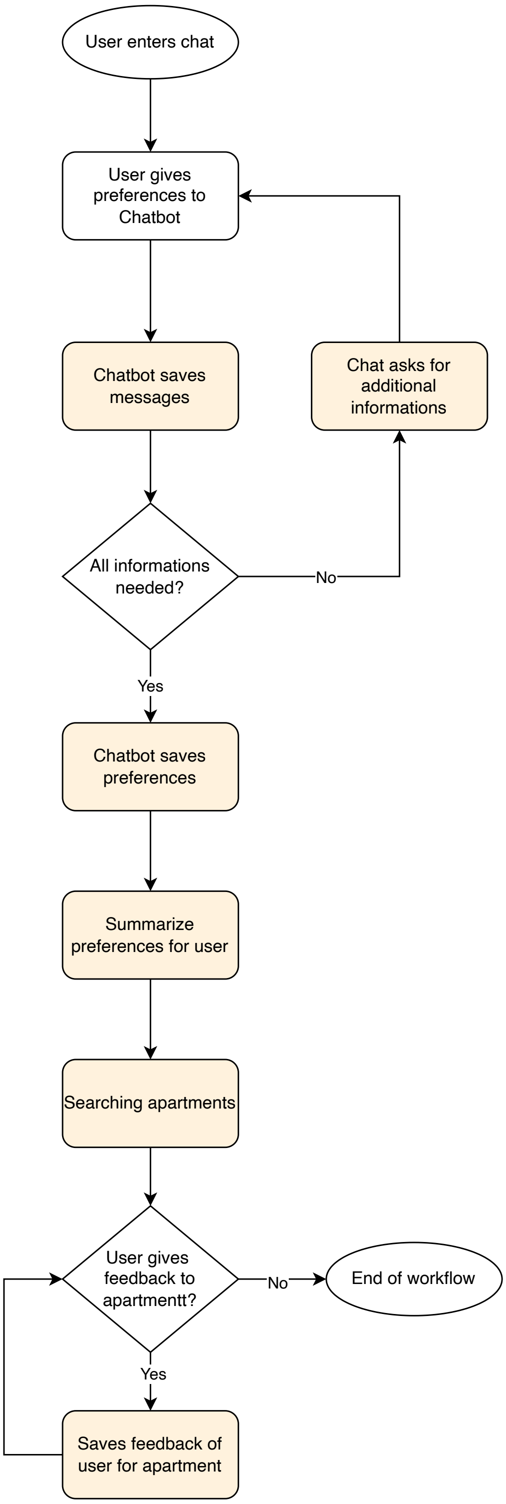

User enters chat
User gives preferences to Chatbot
Chatbot saves messages
Chat asks for additional informations
All informations needed?
No
Yes
Chatbot saves preferences
Summarize preferences for user
Searching apartments
User gives feedback to apartmentt?
No
End of workflow
Yes
Saves feedback of user for apartment

Figure 5.2: Simple user-chatbot interaction workflow

The colored elements indicate when database operations (store/update) occur. During a session, the chatbot iteratively collects user preferences and stores all exchanged messages - including both user inputs and bot responses. Once sufficient information has been gathered, the chatbot stores the final `Preferences` and triggers the apartment search.

Search results are retrieved from the data gatherer module and presented to the user via the UI. User feedback (such as ”pinning” or ”liking” a listing) is then saved to the `Feedback` entity.

This interaction pattern results in a database schema composed of four main relational tables:

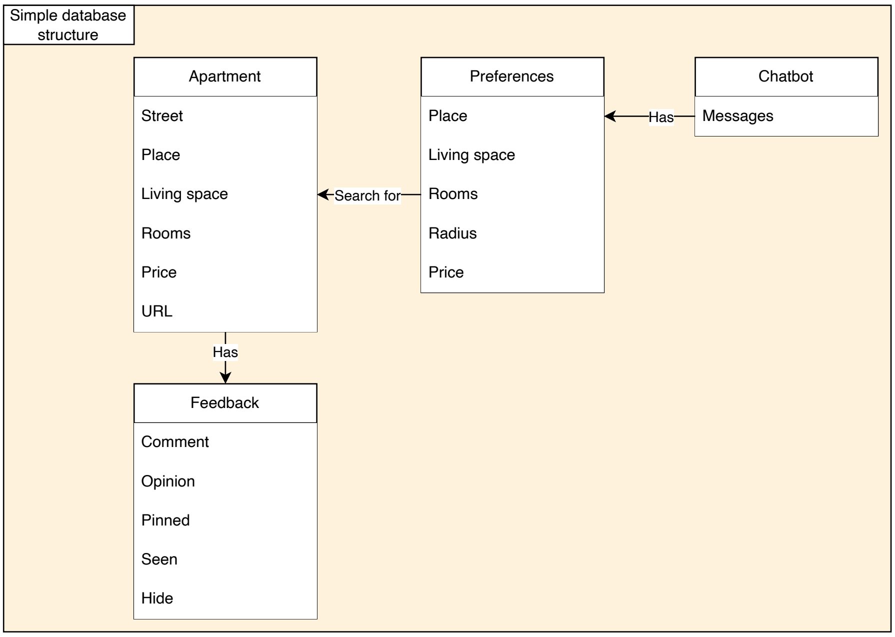


Figure 5.3: Simple database structure

Each `Chat` instance is associated with multiple `Messages`, and each chat can result in a single `Preference` object (which can be updated, if the user gives other preferences to the chatbot). The `Preferences` are then used to search for suitable `Apartments`. For each apartment shown to the user, a corresponding `Feedback` entry may be stored.

This structure ensures traceability across the full user journey, from first interaction to final apartment evaluation.

The database is essential for all requirements and needs to persist all the data, to fullfill them. With this concept apartments, preferences, chat-messages and feedback are all saved.

5.4 Chatbot

The chatbot acts as the first point of contact for users. Its primary role is to collect user preferences (e.g., location, rent limit) and to answer basic questions regarding the relocation process[16]. Once sufficient information has been gathered, the chatbot summarizes the input and initiates the apartment search[17]. All user interactions are stored and forwarded to the backend for further processing[18].

Key design goals:

- Natural language interaction, primarily in English, with support for multilingual dialogues where required.
- Structured and goal-oriented conversations in which the chatbot actively guides the user through the process.
- Input validation to ensure quality and relevance (e.g., numeric rent values, realistic price-location combinations).

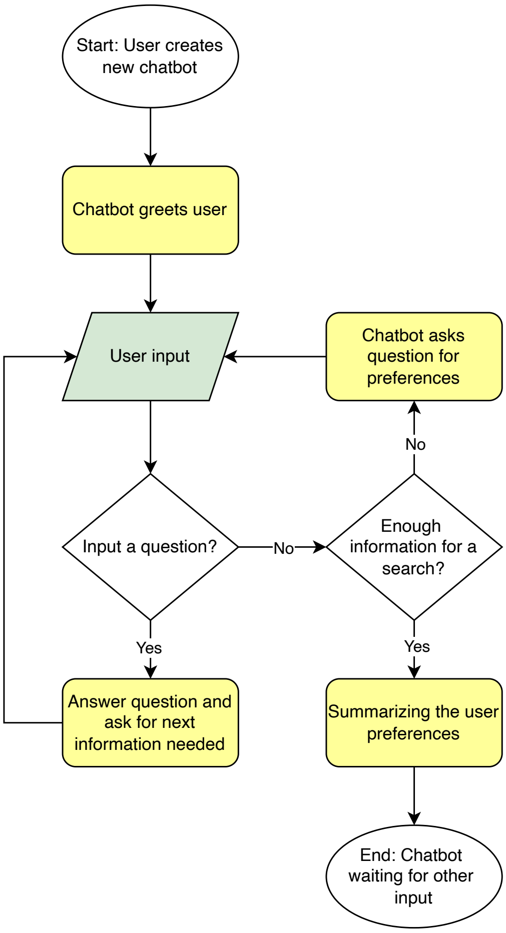

Start: User creates new chatbot
Chatbot greets user
User input
Chatbot asks question for preferences
No
Input a question?
No
Enough information for a search?
Yes
Yes
Answer question and ask for next information needed
Summarizing the user preferences
End: Chatbot waiting for other input

Figure 5.4: Chatbot interaction workflow diagram

The central element of the workflow is the user input (marked in green). The chatbot (shown in yellow) begins by greeting the user and prompting for input. Once a message is received, the chatbot evaluates its content to determine whether it is a **question** or an **answer**.

- If the input is a *question*, the chatbot responds accordingly and waits for further user input.
- If the input is an *answer*, the chatbot checks whether it now has sufficient information to perform a meaningful apartment search.

If required data is still missing, the chatbot asks targeted follow-up questions (e.g., regarding price range or number of rooms). Once all necessary information is collected, it summarizes the preferences and ends the interaction phase.

Even after this point, the chatbot remains responsive: the user may continue the conversation, and new inputs are processed in real time. If new or updated information is provided, the chatbot dynamically updates the user's preferences and, if needed, restarts the search process.

[16] Requirements CBF-1, CBF-2 & CBF-4
[17] Requirement CBF-3
[18] Requirements CBF-5 & CBF-6

5.4.1 Technology Overview

The development of a relocation assistant involves multiple technology layers, each serving a specific role in the system. At the conceptual level, it is important to understand the general options without committing to a specific implementation yet.

For the chatbot component, there are several possible approaches. Some solutions use pre-built chatbot platforms (e.g., Microsoft Bot Framework, Google Dialogflow) that provide ready-to-use conversation flows and can be connected to external services via APIs. Others rely on large language models (LLMs) from providers such as OpenAI or Anthropic, where the conversation logic is defined through prompts and API calls. In both cases, the chatbot acts as an interface that receives user input, processes it using natural language understanding (NLU), and returns a structured response.

By understanding these technology categories early in the design process, the implementation phase can focus on selecting the tools and frameworks that best fit the functional and non-functional requirements of the relocation assistant.

5.4.2 Conceptual Chatbot Architecture

This workflow results in a system architecture as illustrated below:

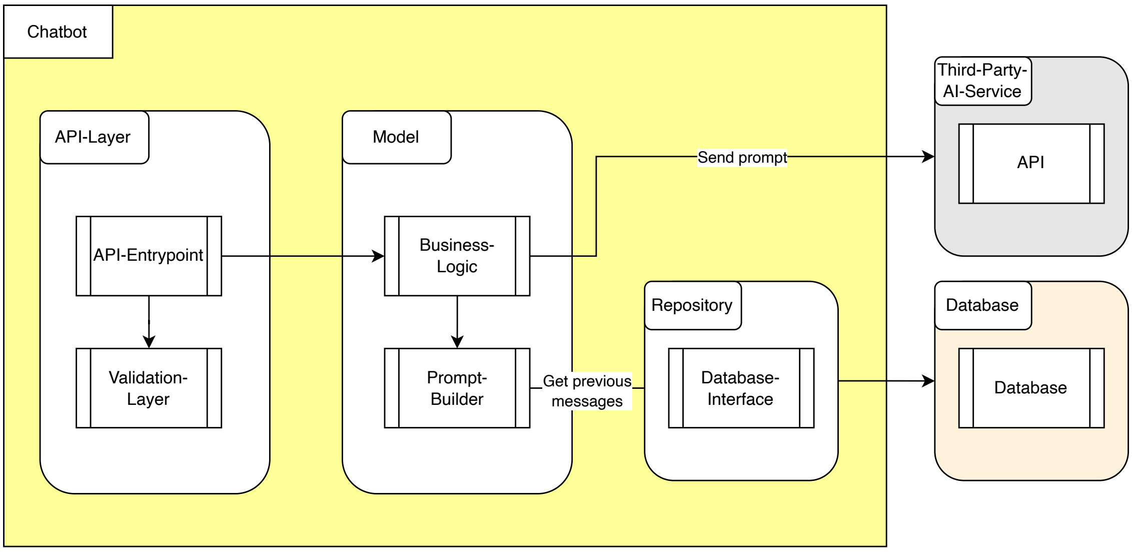


Figure 5.5: Conceptual Chatbot Architecture Diagram

The API layer receives user input through a dedicated interface. It validates incoming requests and forwards them to the internal model logic. A dedicated **Prompt Builder** constructs a prompt for the third-party AI service. This prompt includes the current message, relevant instructions, and previous conversation context.

To retrieve past messages, the Prompt Builder accesses the chat history stored in the database. Once the prompt is assembled, it is sent to the external AI service (e.g., via API), which processes the input and generates a response.

This response is then saved back into the database as part of the ongoing conversation and returned to the user through the same interface.

5.5 Middleware

The middleware acts as the central communication layer of the system, connecting and coordinating the different subsystems. It receives incoming requests and routes them to the appropriate endpoints where the business logic is executed. Typical operations include storing and retrieving data, validating inputs, and initiating apartment searches.

The Data Gathering module is implemented as an encapsulated component within the middleware itself. Outsourcing it as an independent subsystem would increase architectural complexity without offering substantial benefits. Therefore, it is directly integrated into the middleware to reduce overhead and ensure seamless internal communication.

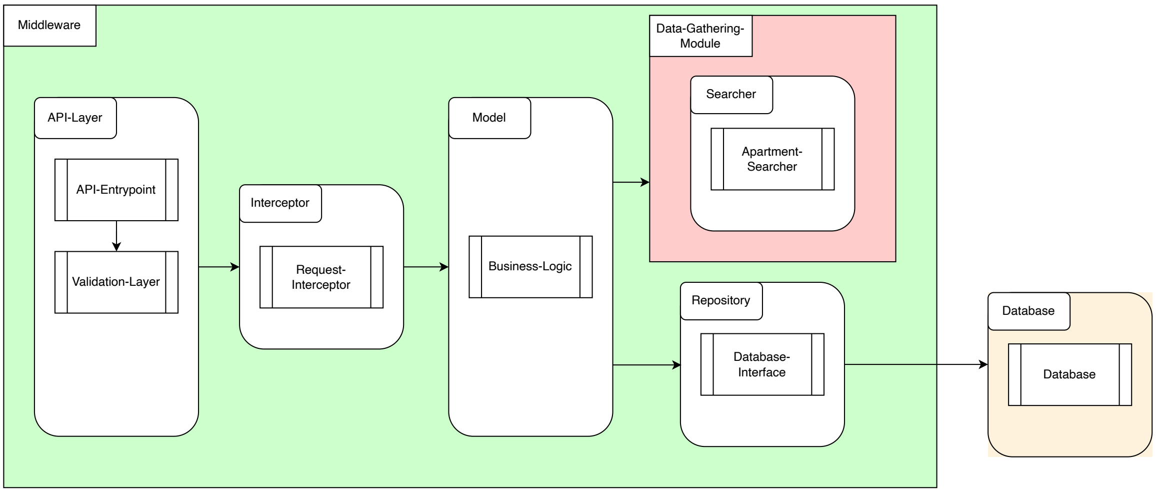


Figure 5.6: Conceptual Middleware Architecture Diagram

The starting point of the workflow is the API layer. Here, incoming HTTP requests are directed to specific endpoints that determine which business logic to execute. The first step is input validation, ensuring correctness and completeness of the data.

Before reaching the model layer, each request passes through an *interceptor*. The interceptor enriches the request by analyzing additional metadata such as the chatbot ID and user-specific information (e.g., email and role). This approach ensures that individual endpoint calls remain clean and minimal, while maintaining context-awareness and supporting integration with external systems like Wix, from where we get information like email and role.

In the model layer, the corresponding business logic is executed. This may involve querying the database, saving new records, or invoking the apartment search logic via the integrated Data Gathering module.

5.6 Data Gatherer

This module is responsible for retrieving relevant apartment listings from external platforms [19]. Although the task may sound straightforward, the actual implementation involves significant complexity. Each platform has its own structure, access method, and level of bot protection, requiring a highly adaptable and modular design.

The following diagram illustrates the high-level architecture of the Data Gatherer module:

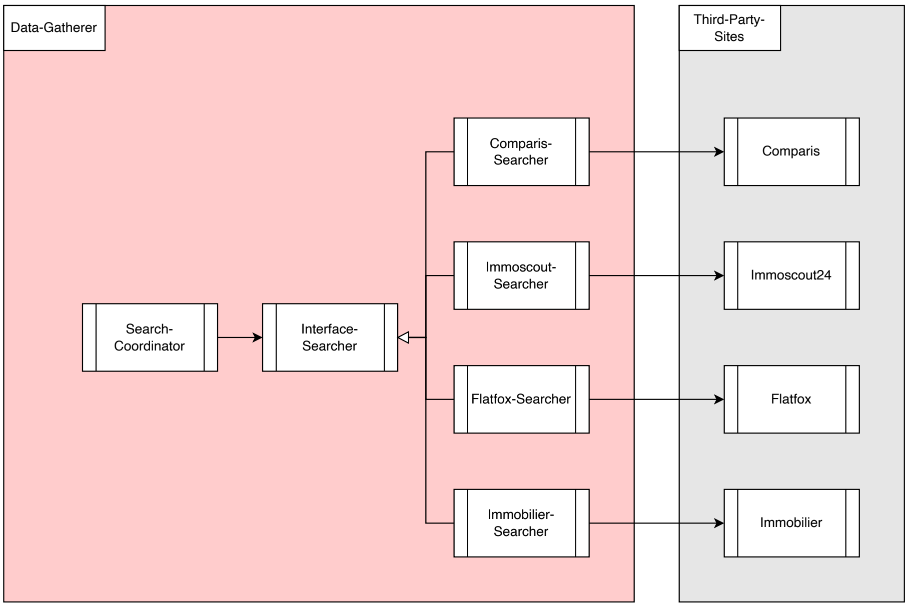


Figure 5.7: Conceptual Data-Gatherer Architecture Diagram

At the core of the system is an **orchestrator** that coordinates multiple searchers. Each searcher implements a common interface, ensuring consistent behavior across all platforms. This design abstracts the internal implementation details of individual searchers and allows seamless integration of new ones.

The use of a unified interface brings several benefits:

- The orchestrator can trigger and manage all searchers in the same way.
- Concrete searchers can be implemented independently using different technologies or scraping strategies.
- Changes in one searcher do not affect others, enhancing maintainability and scalability.

This flexibility is essential due to the diverse nature of the data sources:

- Some websites require advanced browser automation (e.g., Selenium) due to bot detection and JavaScript-heavy rendering.
- Others allow simple HTML parsing, which can be handled efficiently by lightweight libraries such as BeautifulSoup or Requests.

By allowing each searcher to use the most suitable approach for its target platform, the system ensures both robustness and efficiency. This modular design makes it easy to extend the system with additional platforms or improve individual scraping strategies without disrupting the overall architecture.

The apartment results are then returned to the middleware, where they are saved to the database.

[19] Requirements DFA-1 to DFA-7

5.7 Frontend

The frontend serves as the main access point for users interacting with the system. Through the interface, a user (in this case, an administrator) can select a chatbot and initiate a conversation. Based on the gathered input, the chatbot assists in searching for suitable apartments[20]. The found apartments are then listed in the frontend with additional tools to tag the apartments with information [21].

At the current stage of development, the interface displays a list of available chatbot instances. This allows administrators to select and manage specific conversations.

In the long-term vision of the system, the end-user will directly access the chatbot on the main page, without the need for manual administrator intervention. However, due to legal and practical uncertainties, this functionality has been postponed. For this project, the chatbot will be operated by an administrator on behalf of the user. That is the reason, why a chatbot list is needed. Otherwise a user would have only one chatbot available.

After selecting a chatbot, the administrator can access the chat interface, view past messages, and interact further. Apartment search requests are issued from within this view and processed accordingly.

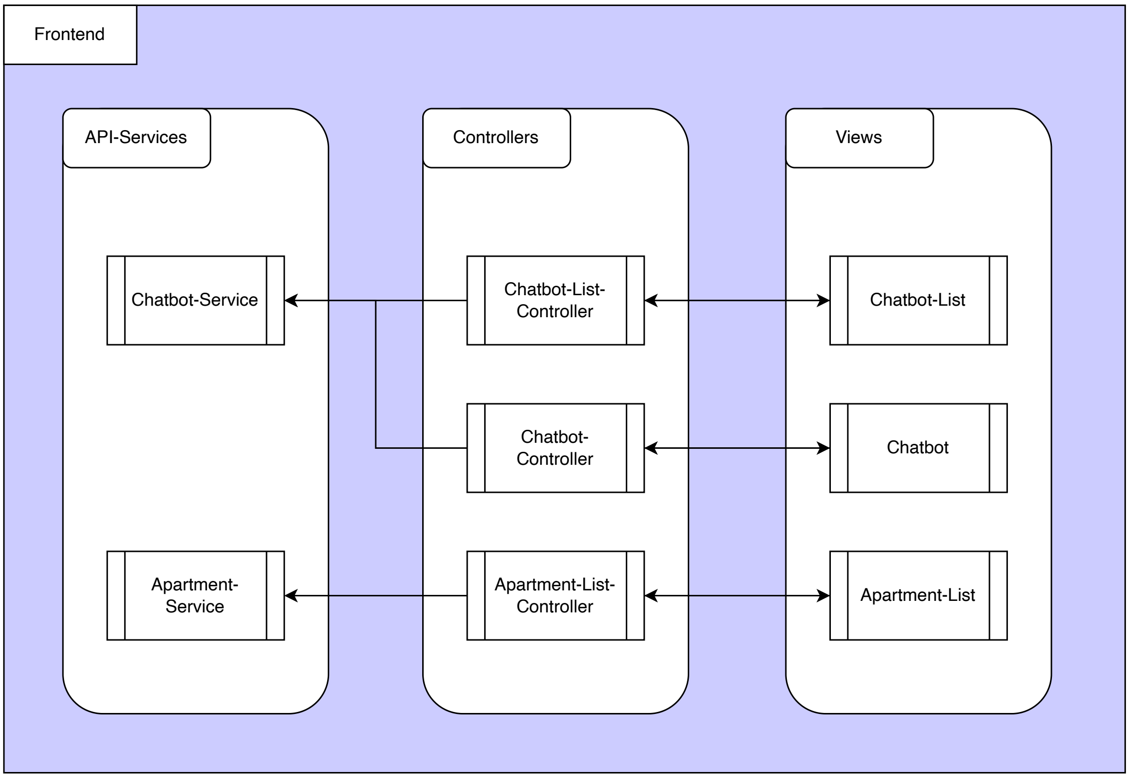


Figure 5.8: Conceptual Frontend Architecture Diagram

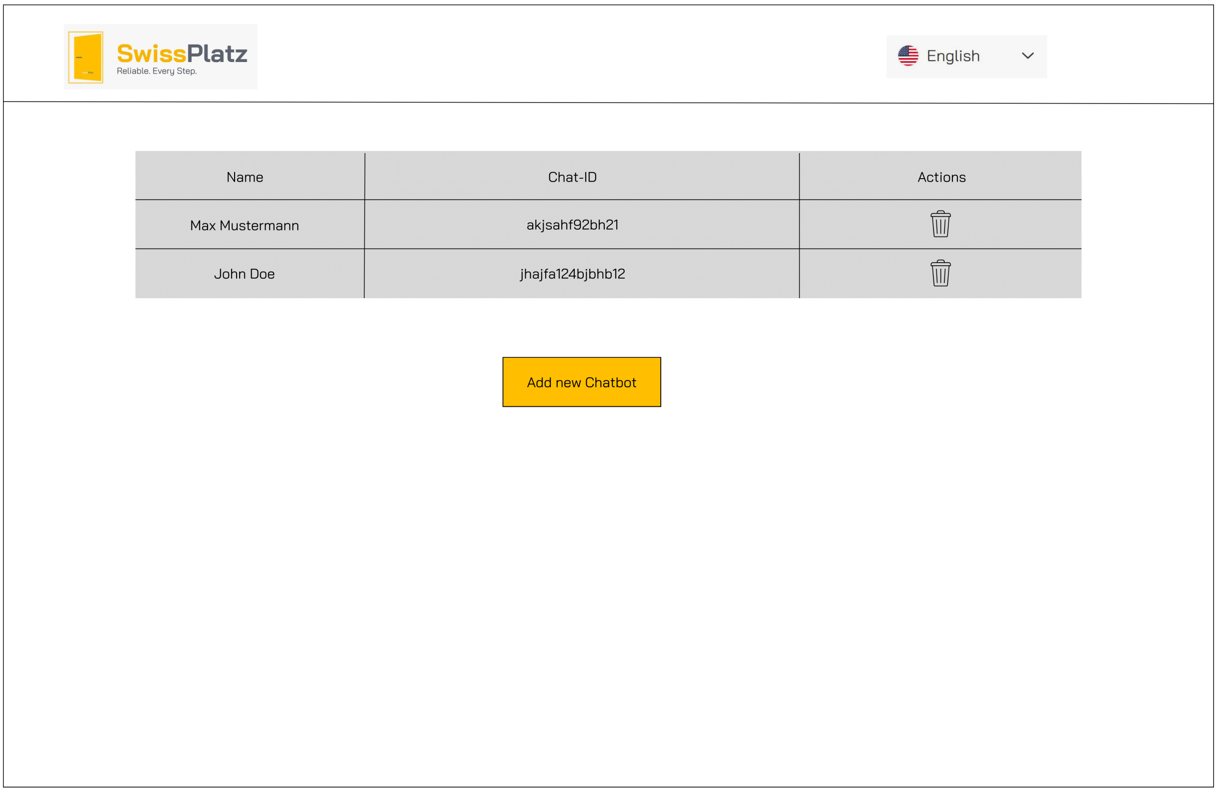
The frontend is structured around two main services that handle communication with the underlying subsystems. Each service is associated with a corresponding controller, which manages the business logic and coordinates the display logic.

The visual elements on the screen are defined in distinct views. Each view represents a part of the site, such as the chatbot overview, the chat interface, or the apartment search results.

[20] Requirement DBF-2

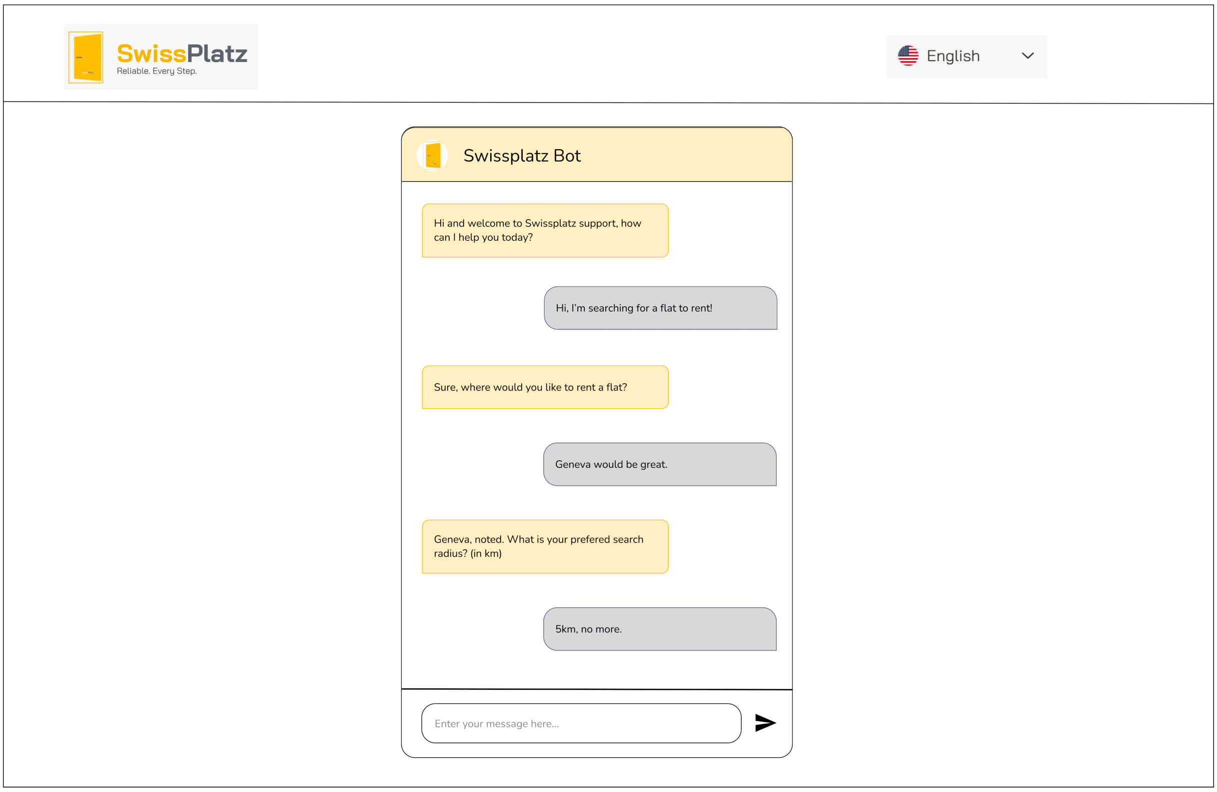
[21] Requirements DBF-1 and DBF-3

5.7.1 Prototyping and Design Iterations



Name	Chat-ID	Actions
Max Mustermann	akjsahf92bh21	🗑
John Doe	jhajfa124bjbhb12	🗑

Figure 5.9: Relocation Chatbot Overview



Hi and welcome to Swissplatz support, how can I help you today?

Hi, I'm searching for a flat to rent!

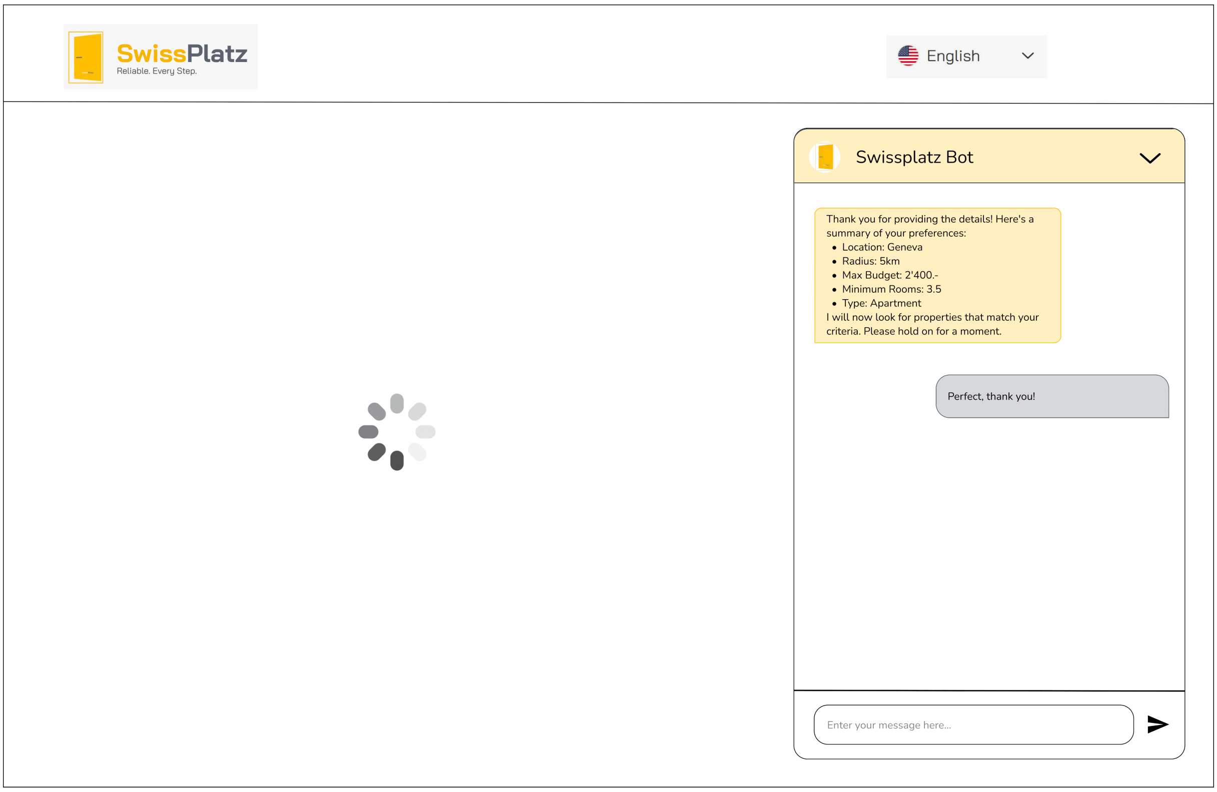
Sure, where would you like to rent a flat?

Geneva would be great.

Geneva, noted. What is your prefered search radius? (in km)

5km, no more.

Figure 5.10: Relocation Dashboard



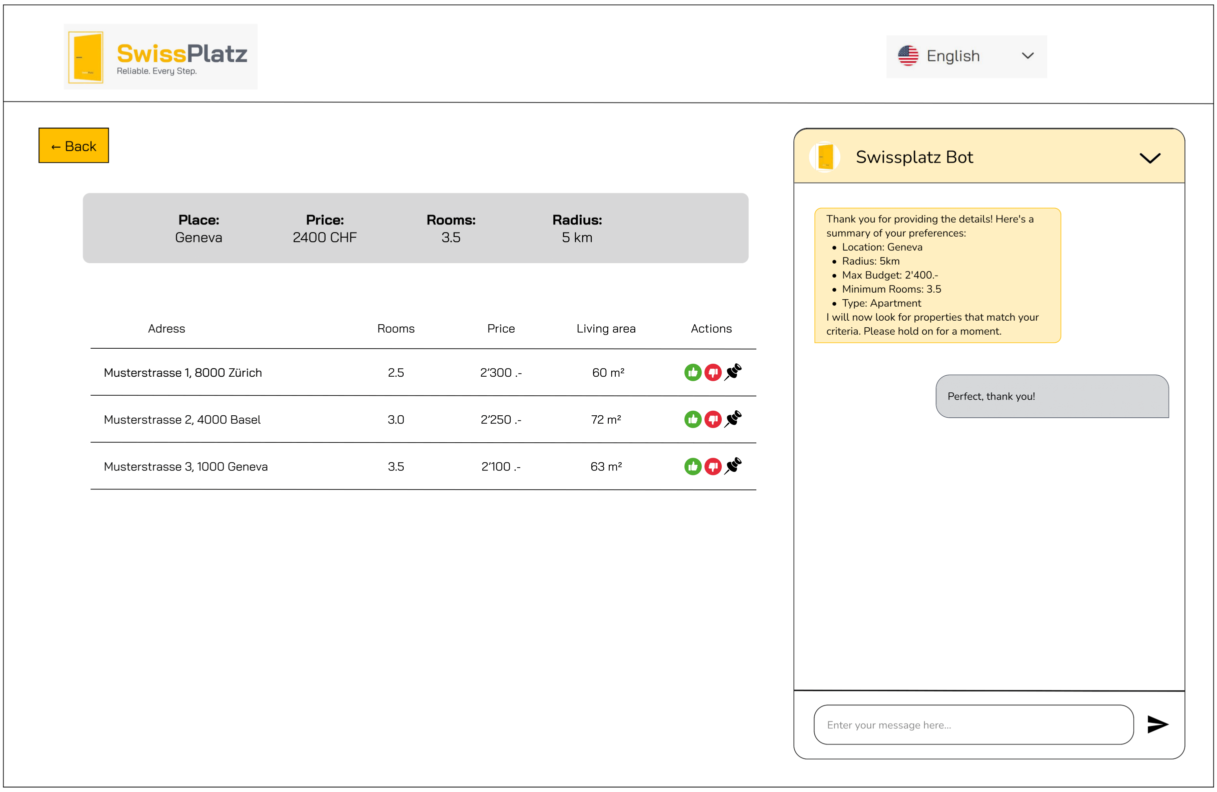
Thank you for providing the details! Here's a summary of your preferences:

- Location: Geneva
- Radius: 5km
- Max Budget: 2'400.-
- Minimum Rooms: 3.5
- Type: Apartment

I will now look for properties that match your criteria. Please hold on for a moment.

Perfect, thank you!

Figure 5.11: Loading Screen



Place:	Price:	Rooms:	Radius:
Geneva	2400 CHF	3.5	5 km

Adress	Rooms	Price	Living area	Actions
Musterstrasse 1, 8000 Zürich	2.5	2'300 .-	60 m²	👍 👎 📌
Musterstrasse 2, 4000 Basel	3.0	2'250 .-	72 m²	👍 👎 📌
Musterstrasse 3, 1000 Geneva	3.5	2'100 .-	63 m²	👍 👎 📌

Thank you for providing the details! Here's a summary of your preferences:

- Location: Geneva
- Radius: 5km
- Max Budget: 2'400.-
- Minimum Rooms: 3.5
- Type: Apartment

I will now look for properties that match your criteria. Please hold on for a moment.

Perfect, thank you!

Figure 5.12: Results in Relocation Dashboard

To ensure that the frontend meets usability expectations and supports the desired workflow, several prototypes were created during the early stages of development. These prototypes served as the foundation for discussions with our customer and allowed us to validate layout ideas and visual structure before implementation. This prototype showcases our final prototype on which we based the implementation of our application.

- Relocation Chatbot Overview: The first screen's main goal is to provide an overview over all created chatbots and being able to manage them (e.g adding, deleting).
- Relocation Dashboard: The second screen shows the Relocation Dashboard. No other information is being shown, to force a conversation with our AI-based chatbot. The goal is to be guided through the process regarding the housing search.
- Loading Screen: When the conversation is done, and the search criteria is found, an algorithm is searching various platforms for suiting advertisements.
- Result in Relocation Dashboard: Suitable housing options are being shown in the table, where the entries can be managed with liking, disliking and pinning. Clicking on the address will open the advertisement for further information and pictures.

Originally, the system was intended to include a client-facing interface, where end-users could interact with the chatbot directly. However, due to legal and strategic considerations, the focus shifted toward an internal tool designed exclusively for Swissplatz mediators. As a result, early prototypes differ significantly from the final version. These earlier concepts are included in the appendix to document the evolution of the system design.

6 Implementation

6.1 Analyzis of the Current State

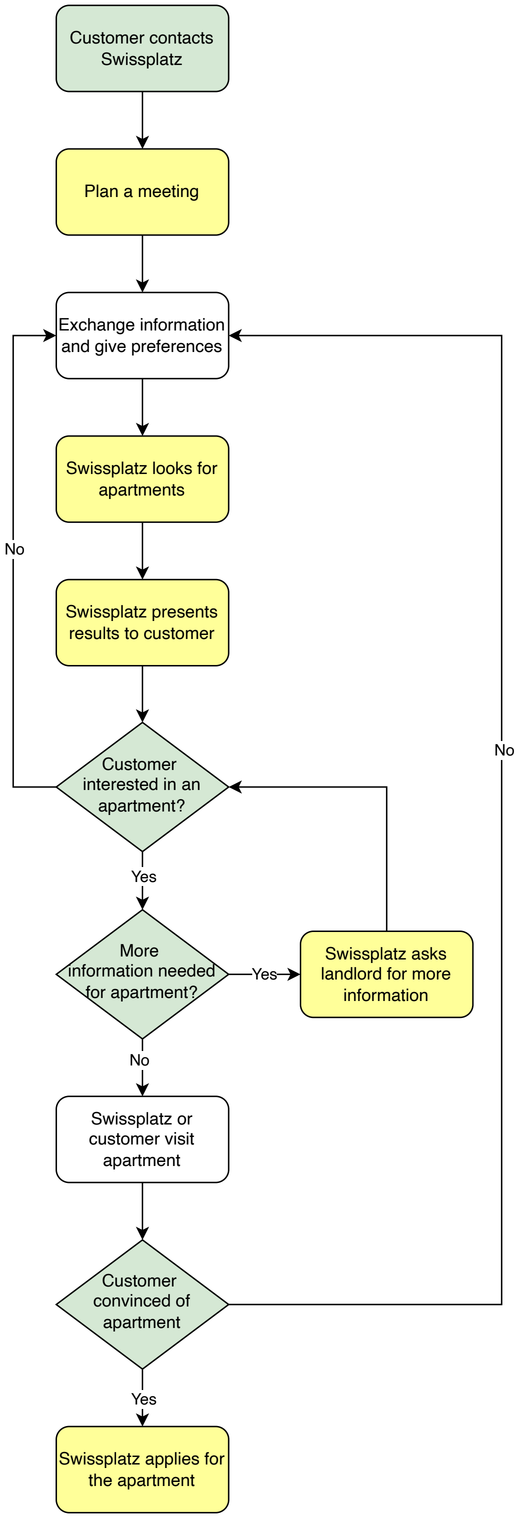

Customer contacts Swissplatz
Plan a meeting
Exchange information and give preferences
Swissplatz looks for apartments
Swissplatz presents results to customer
Customer interested in an apartment?
No
Yes
More information needed for apartment?
Yes
Swissplatz asks landlord for more information
No
Swissplatz or customer visit apartment
Customer convinced of apartment
No
Yes
Swissplatz applies for the apartment

Figure 6.1: Simple Swissplatz Workflow Diagram

Currently, Swissplatz (highlighted in yellow) operates primarily through its website, which provides general information about the company and its relocation services. When a customer (highlighted in green) becomes interested, they can create an account and contact Swissplatz directly. From that point on, the website is rarely used - all further communication takes place via email or phone calls.

To collect the necessary information, Swissplatz schedules a personal meeting with the customer. During this meeting, preferences and requirements are gathered and manually entered into an Excel spreadsheet.

As suitable apartment listings are found, they are also added to the same spreadsheet. Once enough options are available, they are shared with the customer for review.

The customer then decides how to proceed. If one or more listings are of interest, Swissplatz contacts the landlords to obtain further details or to arrange apartment viewings.

This initiates a feedback loop, where the customer provides updated input or expresses new preferences, and Swissplatz continues the search and coordination accordingly - until a decision is reached.

If the customer applies for an apartment and is accepted, the relocation process is completed. If not, the loop restarts and the process continues until a successful match is found.

6.2 Data Gathering Engine

The data gathering engine is responsible for retrieving apartment listings from various external platforms. In this section, we first examine the types of data sources used, then explain how data retrieval is performed, and finally discuss implementation details.

6.2.1 Source Platforms

To understand the functionality of the engine, it is essential to analyse the platforms from which data is collected.

As previously discussed, there are different approaches to retrieving data. Ideally, a platform provides a public API - the most robust and reliable way to access structured data. However, this is not always the case. When no API is available, scraping becomes necessary.

Our implementation focuses on two major platforms in Switzerland: **ImmoScout24** and **Comparis**. The following subsections describe how data is accessed from each platform.

ImmoScout24

ImmoScout24 does not offer a public API or direct access to their internal database. However, their `robots.txt` file explicitly allows scraping, which provides a legal and ethical basis for data collection.

Upon visiting the homepage, users are presented with a search bar that accepts various filters such as location, radius, price, and number of rooms.

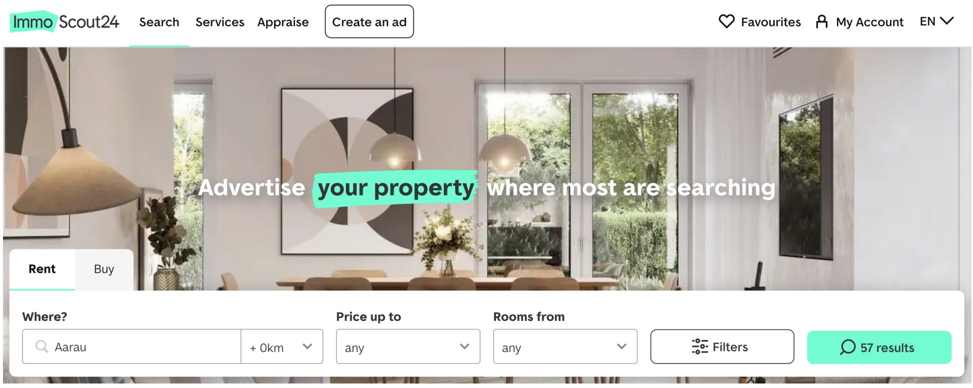
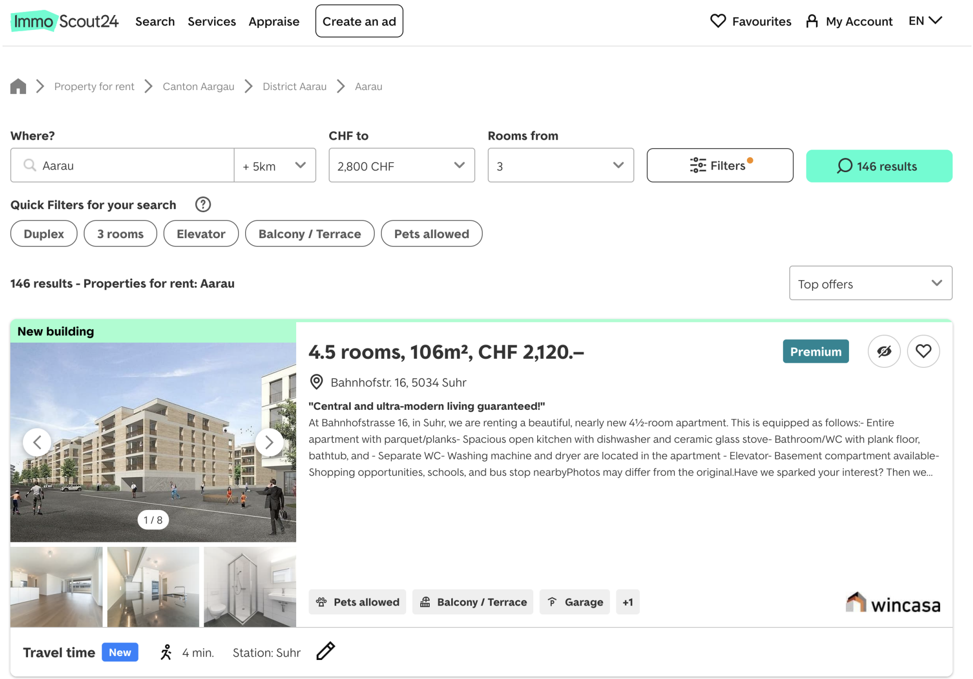


Figure 6.2: ImmoScout24 Frontpage with Search Bar

After submitting a query, the site redirects to the apartment listing page. A typical URL might look like the following:

```
https://www.immoscout24.ch/en/real-estate/rent
/city-aarau?r=5000&nrf=3&pt=2800
```

This shows that search parameters (e.g., radius, minimum number of rooms, price threshold) are embedded directly in the URL. As a result, listings can be accessed directly by constructing appropriate URLs - there is no need to simulate user interaction with the search bar.



Where? Aarau + 5km

CHF to 2,800 CHF

Rooms from 3

146 results - Properties for rent: Aarau

4.5 rooms, 106m², CHF 2,120.–

Bahnhofstr. 16, 5034 Suhr

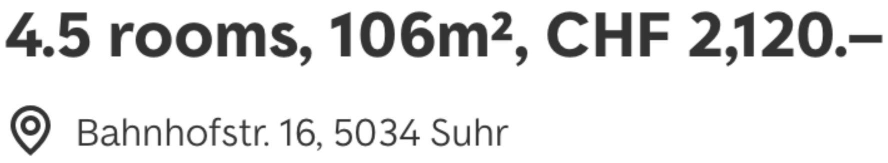
"Central and ultra-modern living guaranteed!"

At Bahnhofstrasse 16, in Suhr, we are renting a beautiful, nearly new 4½-room apartment. This is equipped as follows:- Entire apartment with parquet/planks- Spacious open kitchen with dishwasher and ceramic glass stove- Bathroom/WC with plank floor, bathtub, and - Separate WC- Washing machine and dryer are located in the apartment - Elevator- Basement compartment available- Shopping opportunities, schools, and bus stop nearbyPhotos may differ from the original.Have we sparked your interest? Then we...

4 min. Station: Suhr

Figure 6.3: ImmoScout24 Apartment Listing Page

Each apartment is displayed as a separate HTML element, containing basic metadata such as title, location, rent, and number of rooms. The scraper loops over each of these elements to extract relevant information.



4.5 rooms, 106m², CHF 2,120.–

Bahnhofstr. 16, 5034 Suhr

Figure 6.4: Zoomed View of Apartment Details on ImmoScout24

This zoomed-in screenshot of a listing element shows the essential data fields - including number of rooms, living space, price, street, and location - which are extracted and displayed on our application's front page.

At the current stage of development, only this summary-level data is retrieved, as it is sufficient for the apartment overview and initial filtering. However, the system architecture is designed to be extensible: more detailed information can be collected in the future by following the links to individual apartment detail pages, where additional metadata is available (e.g., availability date, amenities, description text).

Based on this analysis, it becomes clear that scraping the apartment listing page alone is sufficient for ImmoScout24. This means that there is no need to render or simulate user interaction via a headless browser with Selenium, and the implementation can rely on lightweight static HTML scraping.

To achieve this, we evaluated two different libraries: **BeautifulSoup** and **Selectolax**. Both libraries are designed for parsing HTML documents and are widely used in the context of web scraping.

Initially, we implemented an ImmoScout24 scraper using BeautifulSoup. However, after a short time, ImmoScout24 began identifying our scraper as a bot, resulting in incomplete or incorrect data being returned.

To mitigate this, we switched to **Selectolax**, which provides a faster HTML parser and more advanced selector tools that allow us to more closely emulate human-like scraping behavior. This significantly improved the reliability of our data collection and reduced the risk of detection.

Therefore, the current implementation of the ImmoScout24 searcher is based on Selectolax and will be described in more detail in the following section.

Implementation of the Searcher

This section focuses on the technical logic behind the searcher's implementation. It does not cover structural design aspects such as interface inheritance or the orchestration logic of how the class is invoked. Instead, the emphasis is on the operational steps required to retrieve apartment data from the ImmoScout24 website.

Each platform-specific searcher adheres to a common interface, ensuring consistent interaction with the rest of the system. All required search parameters - such as location, radius, number of rooms, and price limit - are passed into the method as arguments.

```python
def getApartments(self, chatbot_id: str, place: str, radius: float,
                  rooms: int, price: float) -> List[SearchResult]:
```

To reduce the likelihood of being detected as a bot, the scraper emulates a typical browser environment. In this case, we simulate a Chrome browser on a Windows machine using the `cloudscraper` library:

```python
scraper = cloudscraper.create_scraper(
    browser={'browser': 'chrome', 'platform': 'windows', 'mobile': False}
)
```

After establishing a browser fingerprint, the search URL is dynamically constructed using the input parameters. The scraper then requests the apartment listing page and parses the HTML content. Each listing element is analysed, and structured apartment data is extracted and stored in application-specific data models.

This logic allows for fast, lightweight, and reliable data retrieval from ImmoScout24 - as long as the platform layout and detection mechanisms remain stable.

Comparis

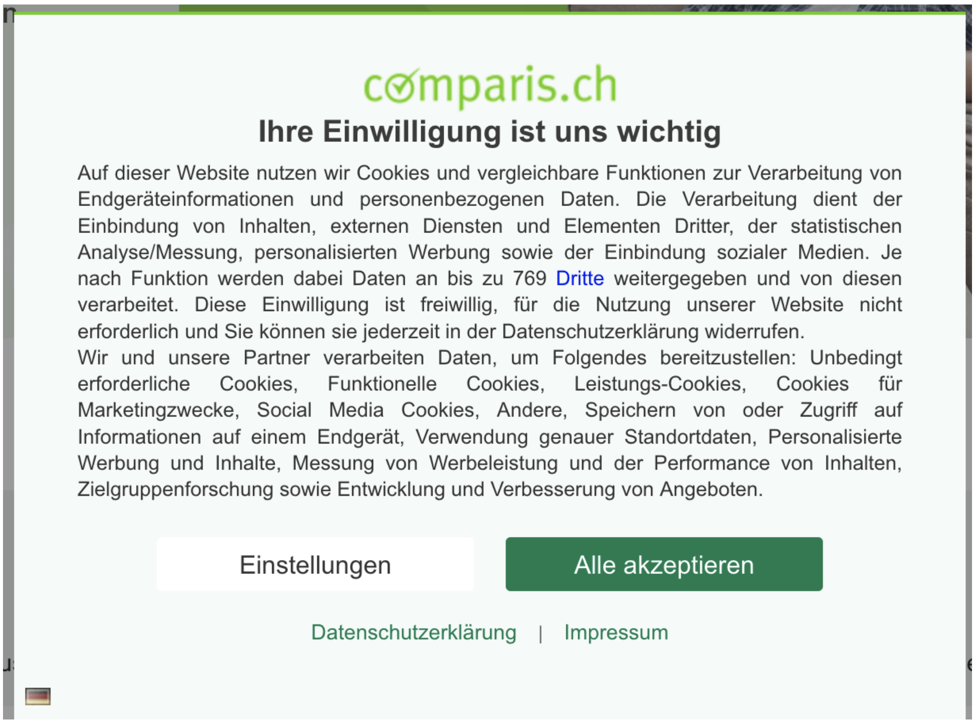
The situation with Comparis is similar to ImmoScout24: there is no public API, and no direct database access is provided. However, scraping is permitted according to their `robots.txt` file.

In practice, Comparis presents significantly more challenges. It has stricter bot detection mechanisms and a more dynamic and complex architecture. Unlike ImmoScout24, where the URL clearly contains search parameters, Comparis uses encoded request parameters in the URL. An example of such a URL is:

`https://www.comparis.ch/immobilien/result/list?requestobject=%7B%22D.`

These long, encoded URLs make it impractical to directly construct the search URL by hand - especially since the internal structure of the query object is undocumented and subject to change.

Therefore, we must interact with the website dynamically, mimicking a real user session from the start. This process begins on the Comparis homepage, where users are prompted to accept cookie settings:



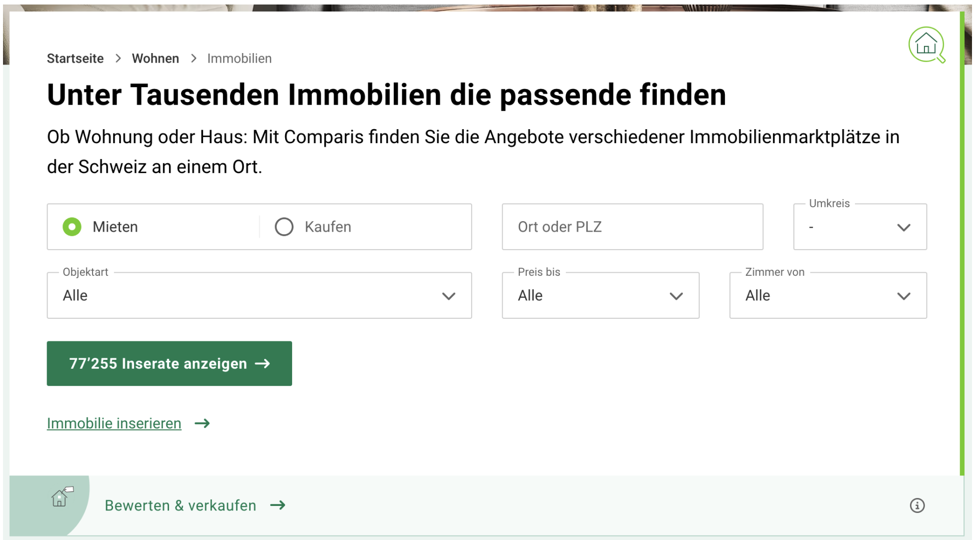
comparis.ch

Ihre Einwilligung ist uns wichtig

Auf dieser Website nutzen wir Cookies und vergleichbare Funktionen zur Verarbeitung von Endgeräteinformationen und personenbezogenen Daten. Die Verarbeitung dient der Einbindung von Inhalten, externen Diensten und Elementen Dritter, der statistischen Analyse/Messung, personalisierten Werbung sowie der Einbindung sozialer Medien. Je nach Funktion werden dabei Daten an bis zu 769 Dritte weitergegeben und von diesen verarbeitet. Diese Einwilligung ist freiwillig, für die Nutzung unserer Website nicht erforderlich und Sie können sie jederzeit in der Datenschutzerklärung widerrufen.
Wir und unsere Partner verarbeiten Daten, um Folgendes bereitzustellen: Unbedingt erforderliche Cookies, Funktionelle Cookies, Leistungs-Cookies, Cookies für Marketingzwecke, Social Media Cookies, Andere, Speichern von oder Zugriff auf Informationen auf einem Endgerät, Verwendung genauer Standortdaten, Personalisierte Werbung und Inhalte, Messung von Werbeleistung und der Performance von Inhalten, Zielgruppenforschung sowie Entwicklung und Verbesserung von Angeboten.

Figure 6.5: Comparis Cookie Banner

Only after accepting the cookie policy is it possible to interact with the search interface:



Startseite > Wohnen > Immobilien

Unter Tausenden Immobilien die passende finden

Ob Wohnung oder Haus: Mit Comparis finden Sie die Angebote verschiedener Immobilienmarktplätze in der Schweiz an einem Ort.

● Mieten ○ Kaufen

Ort oder PLZ

Umkreis: -

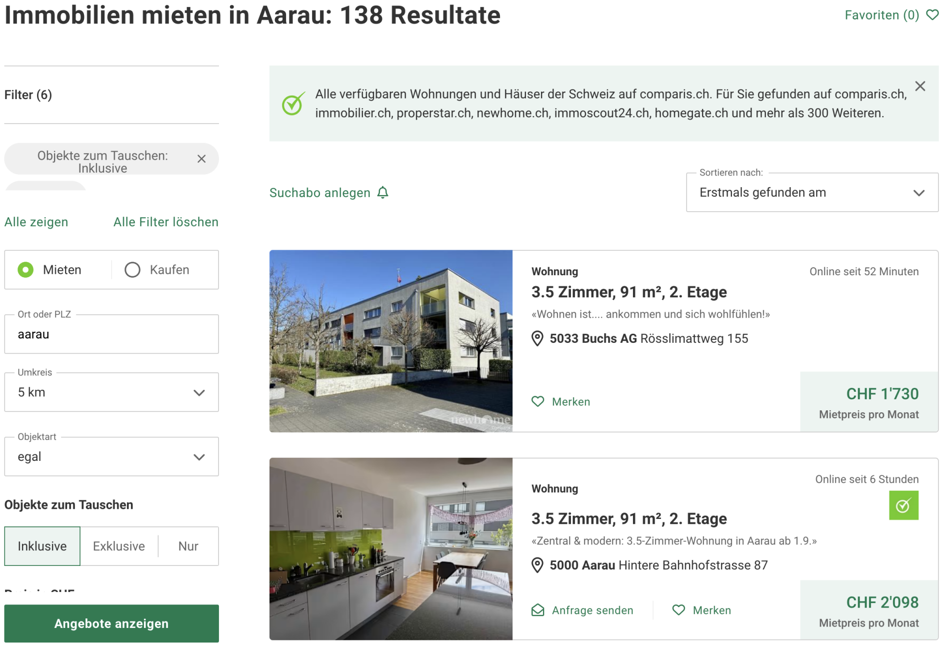
Objektart: Alle

Preis bis: Alle

Zimmer von: Alle

Figure 6.6: Comparis Frontpage with Search Bar

Once the user provides the desired search criteria and submits the query, the platform navigates to the apartment listing page:



Immobilien mieten in Aarau: 138 Resultate

Favoriten (0)

Alle verfügbaren Wohnungen und Häuser der Schweiz auf comparis.ch. Für Sie gefunden auf comparis.ch, immobilier.ch, properstar.ch, newhome.ch, immoscout24.ch, homegate.ch und mehr als 300 Weiteren.

Filter (6)

Objekte zum Tauschen: Inklusive

Alle zeigen Alle Filter löschen

● Mieten ○ Kaufen

Ort oder PLZ: aarau

Umkreis: 5 km

Objektart: egal

Objekte zum Tauschen: Inklusive | Exklusive | Nur

Angebote anzeigen

Suchabo anlegen

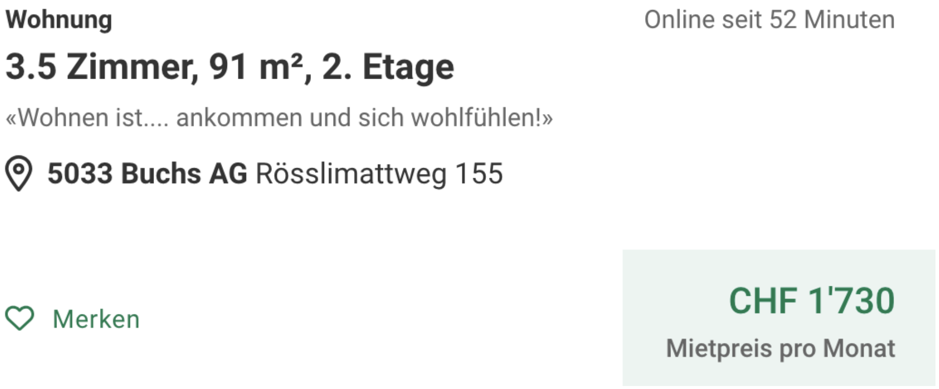
Sortieren nach: Erstmals gefunden am

Wohnung
Online seit 52 Minuten
3.5 Zimmer, 91 m², 2. Etage
«Wohnen ist.... ankommen und sich wohlfühlen!»
5033 Buchs AG Rösslimattweg 155
Merken
CHF 1'730
Mietpreis pro Monat

Wohnung
Online seit 6 Stunden
3.5 Zimmer, 91 m², 2. Etage
«Zentral & modern: 3.5-Zimmer-Wohnung in Aarau ab 1.9.»
5000 Aarau Hintere Bahnhofstrasse 87
Anfrage senden Merken
CHF 2'098
Mietpreis pro Monat

Figure 6.7: Comparis Apartment List

Here, the structure is again similar to ImmoScout24: each apartment is represented as an HTML element containing essential attributes such as price, address, and number of rooms - all visible on the summary card. There is no need to visit the detailed listing page unless additional metadata is required.



Wohnung
Online seit 52 Minuten
3.5 Zimmer, 91 m², 2. Etage
«Wohnen ist.... ankommen und sich wohlfühlen!»
5033 Buchs AG Rösslimattweg 155
Merken
CHF 1'730
Mietpreis pro Monat

Figure 6.8: Comparis Detail View (Zoomed)

Implementation of the Searcher

Due to the dynamic nature of the Comparis website and its strong anti-scraping protections, a static HTML parser such as BeautifulSoup or SelectoLax is insufficient. Instead, we employ **Selenium** - a browser automation library that simulates full user interactions within a real browser window.

The process begins by launching a Selenium-controlled browser (e.g., Chrome), navigating to the Comparis homepage, and programmatically accepting the cookie banner. Next, the script fills in the search parameters via the web interface and submits the query.

Once the apartment list is loaded, each listing element is parsed, and the relevant data fields are extracted.

Because the browser must fully render the page and respond to DOM events between each step, the script includes pauses (`sleep`) to ensure stability. As a result, scraping Comparis using Selenium takes significantly longer - a typical session can last up to **3 minutes**. In contrast, static HTML scraping (e.g., using SelectoLax) completes in under **10 seconds** for comparable platforms.

The added robustness of Selenium comes at the cost of speed and resource usage. However, for platforms with high bot protection or dynamic content loading, it remains one of the most reliable scraping options available.

6.2.2 Search Interface and Data Model

The searcher modules implement a common interface to ensure consistency and modularity across all platform-specific implementations. This abstraction allows new searchers to be added easily, without requiring changes to the logic that uses them.

The interface is defined as follows:

```python
class ApartmentSearcherInterface(ABC):

    @abstractmethod
    def getApartments(self,
                      chatbot_id: str,
                      place: str,
                      radius: float,
                      rooms: int,
                      price: float) -> List[SearchResult]:
        pass
```

This interface defines a single method: `getApartments`. It receives the necessary search parameters, including the `chatbot_id` (used to assign search results to a specific user session), as well as location-specific constraints such as `place`, `radius`, `rooms`, and `price`.

The method returns a list of `SearchResult` objects - a unified data model for apartment listings, defined as follows:

```python
class SearchResult(models.Model):
    id = models.AutoField(primary_key=True)
    chatbot_id = models.CharField(max_length=100, null=True, blank=True)
    address = models.CharField(max_length=255)
    living_space = models.IntegerField()
    rooms = models.FloatField()
    price = models.IntegerField()
    url = models.URLField()
    available = models.BooleanField()
    liked = models.IntegerField(default=0)  # 1 = liked, 2 = disliked, other = no feedback
    hide = models.BooleanField(default=False)
    pined = models.BooleanField(default=False)
```

This model encapsulates all relevant data for a single apartment, including metadata such as `address`, `living_space`, `price`, and `rooms`, as well as additional attributes related to user interaction and apartment state.

While the feedback (e.g., liked, hide, pined) could be separated into a dedicated feedback model, this integration was chosen deliberately. Since each apartment listing can only be associated with one feedback state, a one-to-one relationship would introduce unnecessary complexity in the frontend. Combining this information into the SearchResult simplifies API responses and reduces the number of network requests required to render apartment lists.

Two additional fields, available and pined, support further interaction logic:

- available: Indicates whether the apartment is still listed on the source platform. This flag is automatically updated during subsequent searches.
- pined: Used to highlight specific listings in the UI (e.g., recommended or important offers).

This design strikes a balance between data normalization and frontend performance, favoring simplicity and responsiveness in the user-facing components.

6.2.3 Search Orchestrator

The orchestration of apartment searchers takes place within the middleware. Each search request is handled by the method searchForApartments, which receives a searchQuery object. This object contains all relevant parameters for the apartment search (e.g., location, radius, number of rooms, price). The chatbot ID is added to the arguments before dispatching the searchers.

To reduce total wait time and improve responsiveness, the system leverages a ThreadPoolExecutor, allowing all registered searchers (e.g., for ImmoScout24 and Comparis) to run in parallel. Each searcher is called with the same set of arguments through the shared interface described previously.

The results returned by each thread are collected and stored directly in the database. Importantly, no data is returned immediately to the frontend from this function.

Instead, the frontend polls the middleware at regular intervals to check for newly available apartment results. This asynchronous architecture ensures loose coupling and avoids blocking operations.

```python
def searchForApartments(self, searchquery):
    chatbot_id = searchquery.chatbot_id
    args = (chatbot_id, searchquery.location, searchquery.radius,
            searchquery.rooms, searchquery.price)
    searchers = [self.immoscout_searcher, self.comparis_searcher]

    all_results = []

    with ThreadPoolExecutor() as executor:
        futures = [executor.submit(searcher.getApartments, *args)
                  for searcher in searchers]

        for future in futures:
            try:
                result = future.result()
                all_results.extend(result)
            except Exception as e:
                print(f"Fehler bei einem Searcher: {e}")

    self.searchresult_repository.saveSearchResults(all_results)
```

6.3 Middleware

The middleware serves as the unified controller for data flow and system communication. It exposes APIs that enable clients to perform CRUD (Create, Read, Update, Delete) operations while abstracting away the underlying implementation details.

In our project, the middleware is implemented using `Django REST Framework`. This choice provides several advantages, including:

- Built-in serialization for handling complex data structures.
- Flexible URL routing.
- Scalability and performance optimization features.
- A rich ecosystem of third-party libraries and extensions.

Overall, Django REST offers a solid foundation for building a fast, scalable, and flexible middleware layer.

The middleware fulfills four main tasks (see numbered elements in the following figure):

1. **Routing and Endpoints** - Accepting incoming API requests and directing them to the correct handlers.
2. **Validation** - Ensuring that incoming requests are well-formed and contain valid data before processing.
3. **Database Communication** - Reading from and writing to the database as needed.
4. **Business Logic Handling** - Executing the core application logic, including interactions with external services.

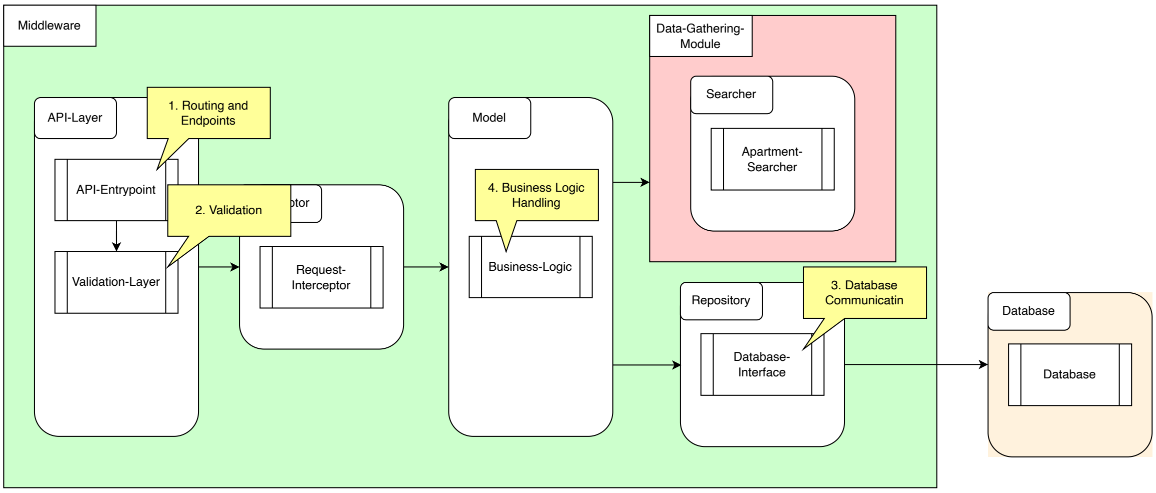


Figure 6.9: Numbered Middleware Diagram

6.3.1 Routing and Endpoints

With the urls.py file in Django REST, endpoints can be defined and linked to specific classes for handling requests:

```
urlpatterns = [
  path('apartments/', ApartmentLogicView.as_view()),
  path('apartments/all', ApartmentLogicView.as_view()),
  path('apartments/searchquery', ApartmentLogicView.as_view()),
]
```

All requests are directed to the ApartmentLogicView, which inherits its HTTP method definitions (e.g., get, put) from Django's APIView:

```
def get(self, request):
def put(self, request):
```

For the three endpoints above, an if-else statement is currently used to determine which action to perform. Since there are only two GET methods and one PUT method, creating additional view classes would add unnecessary complexity.

6.3.2 Validation

Validation of incoming parameters happens inside the get and put methods. If required fields (e.g., price) are missing, the middleware assigns default values (e.g., -1) so that downstream components can identify unset parameters. Additional validations (such as range checks for numeric values or format checks for URLs) ensure robust handling of user input.

6.3.3 Database Communication

To reduce coupling between the database and the middleware, all database operations are abstracted behind repository interfaces. This design allows the underlying database implementation to be swapped without affecting the business logic.

For example, at the start of the project, a Redis in-memory database was used for rapid prototyping. Thanks to the interface-based design, switching to PostgreSQL for production required minimal code changes.

```python
class SearchResultRepositoryInterface(ABC):

  @abstractmethod
  def saveSearchResult(self, result: SearchResult) -> SearchResult:
    pass

  @abstractmethod
  def updateSearchResult(self, updated_result: SearchResult) -> SearchResult:
    pass

  @abstractmethod
  def saveSearchResults(self, results: List[SearchResult]) -> List[SearchResult]:
    pass
```

Each entity has its own repository interface and a concrete implementation class, ensuring modular and maintainable database access.

6.3.4 Business Logic Handling

Business logic in the middleware is distributed across multiple service classes. Key responsibilities include:

- Creating and populating `SearchQuery` objects for the apartment searchers.
- Scheduling search jobs and executing them in predefined time slots.
- Intercepting API calls to enrich them with metadata, such as chatbot ID, user role, or email address.
- Managing middleware configuration, including:
  - Defining allowed CORS origins.
  - Setting authentication and authorization rules.
  - Registering and enabling/disabling specific searcher modules.

This separation ensures that the middleware remains both adaptable to future changes and easy to maintain. Such a workflow can be represented as the following simplified sequence diagram:

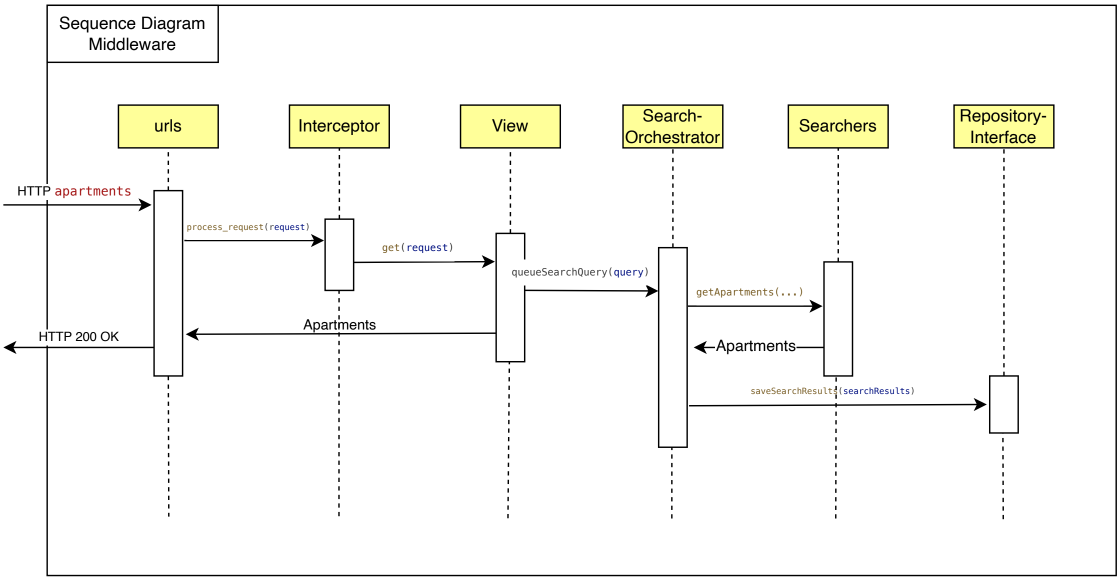


Figure 6.10: Middleware Sequence Diagram

In this example, the client calls an endpoint to submit a `SearchQuery`, which specifies the apartment search parameters. In our current implementation, the search is triggered immediately upon receiving the query.

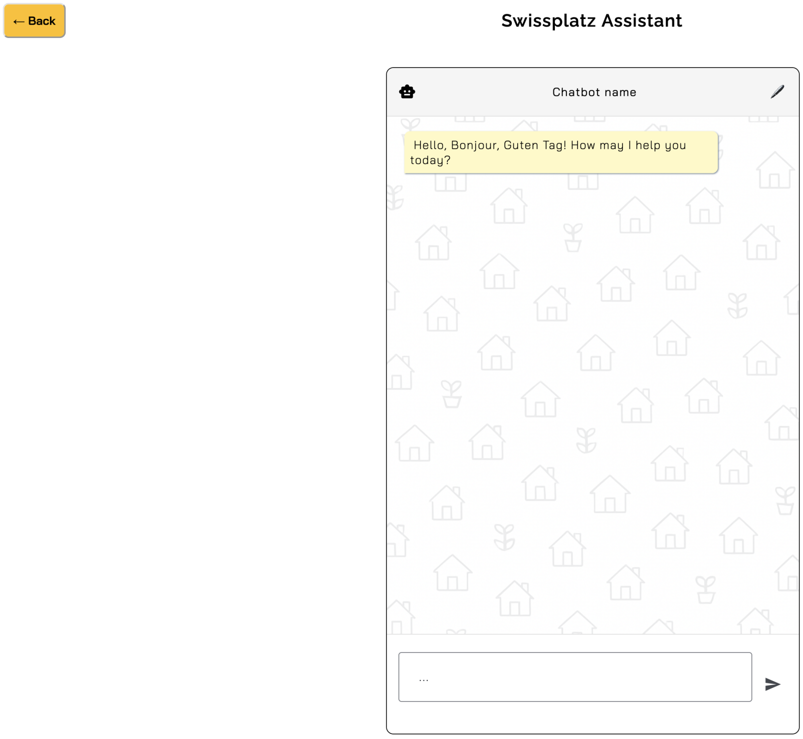
The `View` component returns an HTTP `200 OK` response to acknowledge receipt of the request - without including the apartment data itself. The actual search process is handled asynchronously: the middleware invokes the relevant searchers, which gather the apartment listings in parallel.

Meanwhile, the frontend periodically polls the middleware at fixed intervals to check whether new results have been stored in the database. Once the data becomes available, it is retrieved and displayed to the user.

6.4 Relocation Dashboard

The *Relocation Dashboard* serves as the central interface between Swissplatz staff and the underlying system logic. It replaces many of the manual processes Swissplatz previously carried out, combining apartment search, customer feedback management, and the ability to obtain additional information through the chatbot. In future projects, the dashboard can be extended to include further functionality.

Upon opening the dashboard, the user is first presented with the *Relocation Chatbot Overview*. If there is no existing chat, the overview automatically creates a chat and we start in the *Relocation Dashboard*.



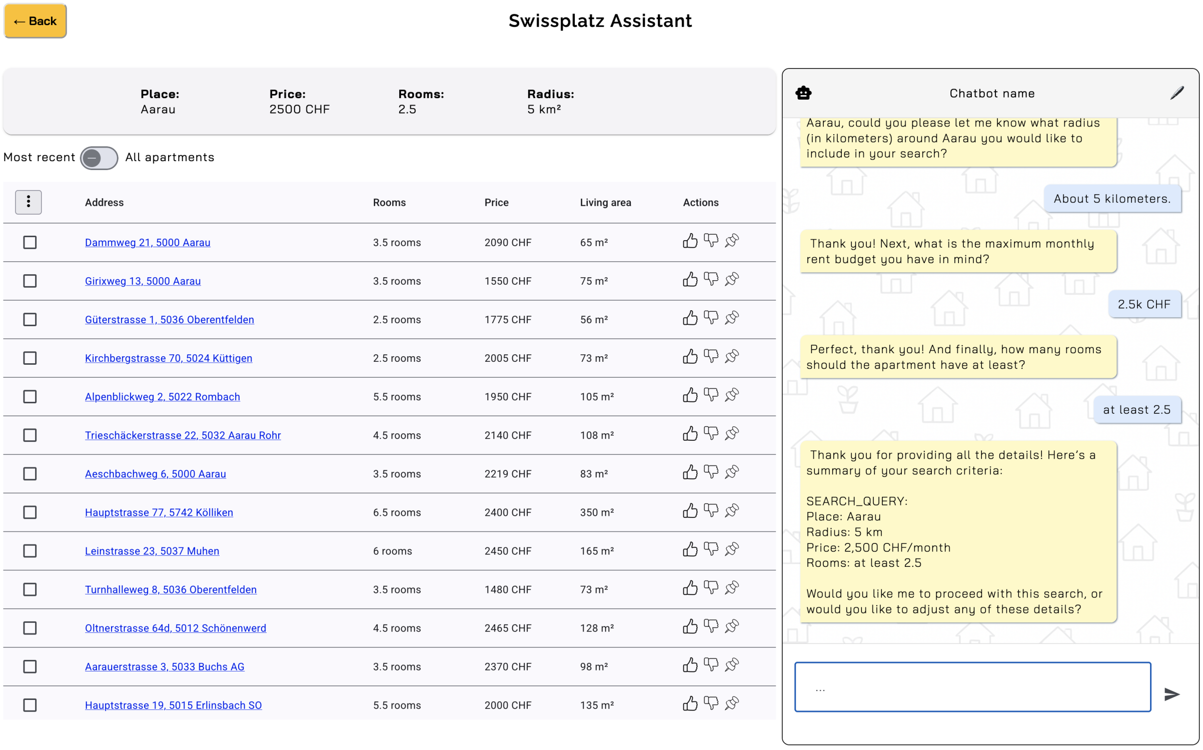
Hello, Bonjour, Guten Tag! How may I help you today?

Figure 6.11: Relocation Dashboard with Empty Chat

The chatbot greets the user and subtly demonstrates multilingual capabilities. From here, staff can interact with the bot, ask questions, and enter customer preferences.

The chatbot guides the user by requesting the information required to initiate an apartment search. It can also assist with additional relocation-related queries, such as identifying the sunniest locations in Switzerland or estimating realistic rent-prices.

Once all required preferences have been collected, the chatbot summarizes them and triggers the apartment search. When results are found, the layout automatically changes to a 2/3-1/3 split, with the chatbot occupying one-third of the view and the apartment list taking the remaining space.



Place: Aarau **Price:** 2500 CHF **Rooms:** 2.5 **Radius:** 5 km²

Most recent All apartments

⋮	Address	Rooms	Price	Living area	Actions
□	Dammweg 21, 5000 Aarau	3.5 rooms	2090 CHF	65 m²	👍 👎 📌
□	Girixweg 13, 5000 Aarau	3.5 rooms	1550 CHF	75 m²	👍 👎 📌
□	Güterstrasse 1, 5036 Oberentfelden	2.5 rooms	1775 CHF	56 m²	👍 👎 📌
□	Kirchbergstrasse 70, 5024 Küttigen	2.5 rooms	2005 CHF	73 m²	👍 👎 📌
□	Alpenblickweg 2, 5022 Rombach	5.5 rooms	1950 CHF	105 m²	👍 👎 📌
□	Trieschäckerstrasse 22, 5032 Aarau Rohr	4.5 rooms	2140 CHF	108 m²	👍 👎 📌
□	Aeschbachweg 6, 5000 Aarau	3.5 rooms	2219 CHF	83 m²	👍 👎 📌
□	Hauptstrasse 77, 5742 Kölliken	6.5 rooms	2400 CHF	350 m²	👍 👎 📌
□	Leinstrasse 23, 5037 Muhen	6 rooms	2450 CHF	165 m²	👍 👎 📌
□	Turnhalleweg 8, 5036 Oberentfelden	3.5 rooms	1480 CHF	73 m²	👍 👎 📌
□	Oltnerstrasse 64d, 5012 Schönenwerd	4.5 rooms	2465 CHF	128 m²	👍 👎 📌
□	Aarauerstrasse 3, 5033 Buchs AG	3.5 rooms	2370 CHF	98 m²	👍 👎 📌
□	Hauptstrasse 19, 5015 Erlinsbach SO	5.5 rooms	2000 CHF	135 m²	👍 👎 📌

Aarau, could you please let me know what radius (in kilometers) around Aarau you would like to include in your search?

About 5 kilometers.

Thank you! Next, what is the maximum monthly rent budget you have in mind?

2.5k CHF

Perfect, thank you! And finally, how many rooms should the apartment have at least?

at least 2.5

Thank you for providing all the details! Here's a summary of your search criteria:

SEARCH_QUERY:
Place: Aarau
Radius: 5 km
Price: 2,500 CHF/month
Rooms: at least 2.5

Would you like me to proceed with this search, or would you like to adjust any of these details?

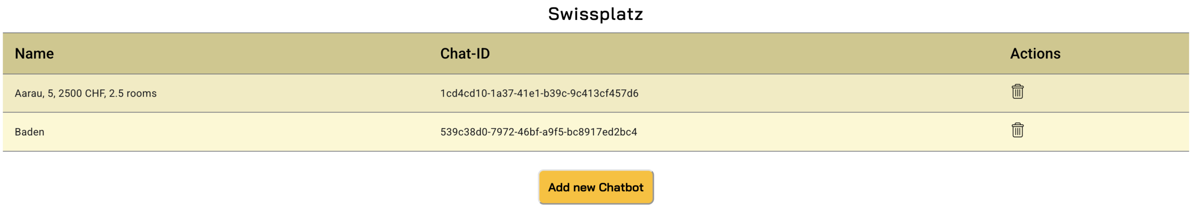
...

Figure 6.12: Relocation Dashboard with Search Results

The apartment list offers features such as pinning, liking, disliking, and selecting listings for export to Excel, which Swissplatz still uses for client reporting.

When new apartments are found in a subsequent search, older listings that are not pinned are automatically hidden, as they no longer match the updated preferences. However, a toggle option allows all listings (old and new) to be displayed.

From the dashboard, users can also access an overview of all active chatbots and create new ones. This ensures that Swissplatz can maintain a dedicated chatbot for each customer.



Name	Chat-ID	Actions
Aarau, 5, 2500 CHF, 2.5 rooms	1cd4cd10-1a37-41e1-b39c-9c413cf457d6	🗑
Baden	539c38d0-7972-46bf-a9f5-bc8917ed2bc4	🗑

Figure 6.13: Relocation Chatbot Overview - Chatbot List

In the long-term vision, the goal was for every Swissplatz customer to directly use their own chatbot instance to search for apartments independently. Due to legal uncertainties, this was not part of the current project scope, so the chatbot interactions are orchestrated by Swissplatz staff via the *Relocation Dashboard*.

6.5 Chatbot / AI assistant

This chapter describes the transition from the conceptual design to the fully implemented system. It focuses on the technical decisions, implementation steps, and integration of each module, ensuring that the relocation assistant functions reliably and meets the specified requirements.

6.5.1 Technology Decision for Chatbot

When selecting a chatbot technology, several providers were evaluated: OpenAI GPT-4o, Anthropic Claude, and Microsoft Azure OpenAI. The decision was based on the following criteria:

- Cost per 1,000 tokens (input/output separately).
- Quality of responses and multi-language support (German/English).
- Latency and performance.
- API flexibility (function calls, system prompt, context length).
- Data privacy and hosting options.

This table shows a comparison of the main options.

Criteria	**OpenAI GPT-4o**	**Anthropic Claude 4**	**Azure OpenAI (GPT)**
Cost / 1K Tokens	$0.0025 (in), $0.01 (out)	$0.003 (in), $0.015 (out)	Same as OpenAI, plus Azure fees
Max Context Length	128k tokens	200k tokens	128k tokens
Latency	Low (fast inference)	Medium	Low
Language Support	Excellent (DE/EN)	Excellent (DE/EN)	Excellent (DE/EN)
Privacy	US-based, SOC2 compliant	US-based	Azure compliance, EU hosting possible

Table 6.1: Comparison of chatbot provider options

There were more options of creating chatbots, which seemed even more appealing due to it’s easyness. Examples are **Botpress**, where we initially created a chatbot with our needed prompt for testing purposes. The integration would have been really simple, since botpress would be hosting the chatbot, and we could have just integrated into our website. But financally it wouldn’t have made sense, since the prices were 89 US-Dollars per month to get 5’000 incoming messages.

Considering all these factors and after consulting it with our customer and expert, **OpenAI GPT-4o** was chosen due to its balance between cost, performance, and API features.

6.5.2 Backend Implementation

The backend of the chatbot was implemented in **Python** using **Flask** for the HTTP API.

The core logic is handled in `ChatbotGpt.py`, which:

- Loads the API key and connects to OpenAI's GPT model.
- Defines a strict *system prompt* with a fixed question protocol for housing requests.
- Formats the chat history into OpenAI's API format.
- Sends user messages to the GPT model and returns responses.

The `ChatRoutes.py` file exposes REST endpoints for:

- Creating, updating, and deleting chatbot sessions.
- Sending and retrieving messages.
- Listing all chatbots.

The conversation history is stored in the database repository with the interface `ChatRepositoryInterface`, ensuring consistent method signatures.

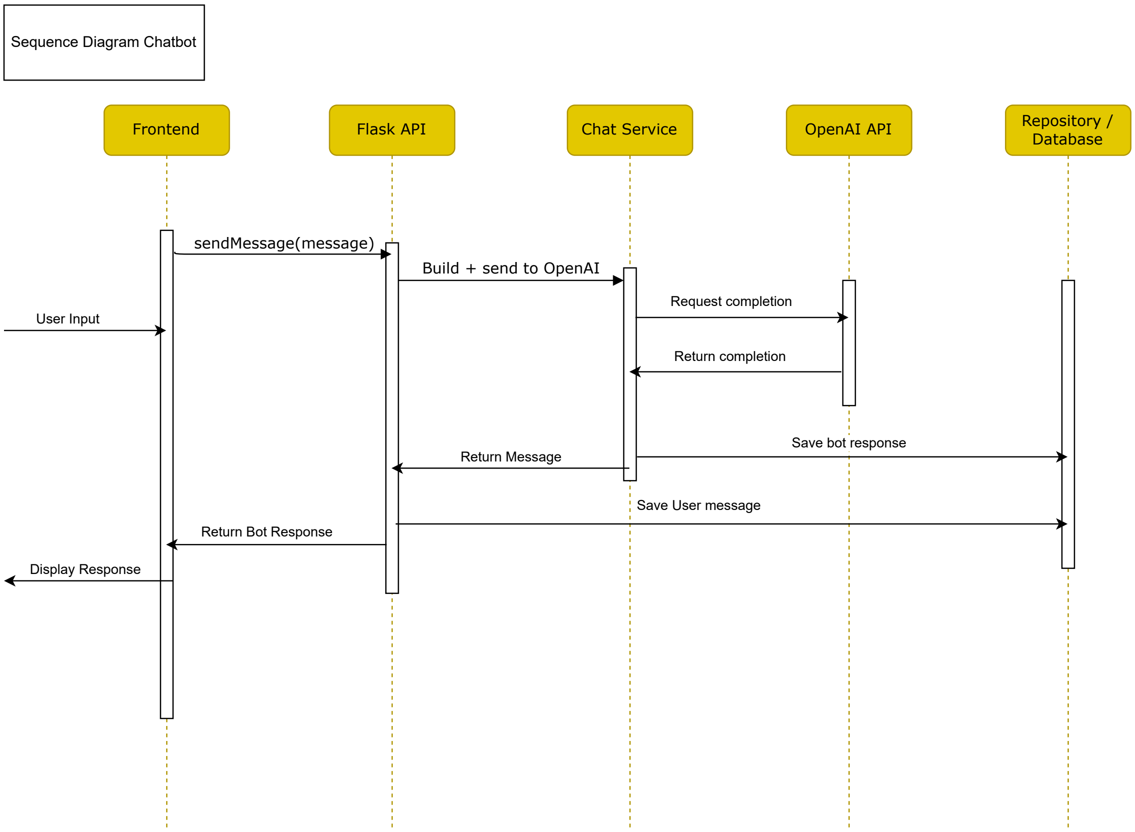


Figure 6.14: Sequence diagram of chatbot message handling

This figure illustrates the request flow from the frontend to the AI model.

6.5.3 Frontend Integration

The frontend was implemented in **Angular**, with a dedicated `ChatbotService` to handle all HTTP requests to the backend. The decision for using Angular was made, based on our prior experience with the framework. This allowed us to work efficiently and focus more on implementing the project-specific logic rather than learning new tooling. The built-in modules and strong CLI work well for our application, which made our decision for the framework rather easy.

The main UI component (`ChatbotComponent`) is responsible for:

- Displaying the conversation history.
- Sending new messages to the backend and showing a loading indicator while waiting for a response.
- Allowing the chatbot's name to be edited.
- Automatically parsing the AI's summary into a structured search query and broadcasting it to the housing search form.

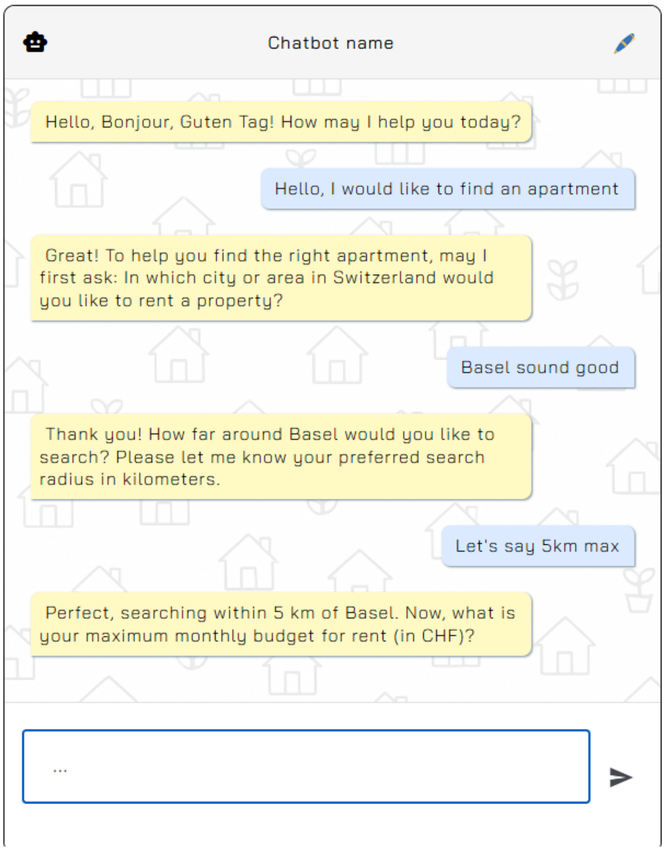
The HTML template (`chatbot.component.html`) uses Angular Material for styling and responsive layouts.

6.5.4 Data Flow

When a user sends a message:

1. The frontend calls `POST /chatbots/{id}` with the message.
2. The backend saves the user's message in the database.
3. The backend compiles the full conversation and sends it to OpenAI's GPT-4o API.
4. The AI's response is saved in the database and returned to the frontend.
5. The frontend updates the chat view and, if a search query is detected, triggers the search form autofill.

This setup makes the interaction for the user smooth and simple. They just type their answers, and in the background the system takes care of saving the messages, asking the AI for a reply, and filling in the housing search form when all information is complete. It connects the conversation directly to the search process, which helps avoid unnecessary manual work and makes the overall workflow smoother.



Hello, Bonjour, Guten Tag! How may I help you today?

Hello, I would like to find an apartment

Great! To help you find the right apartment, may I first ask: In which city or area in Switzerland would you like to rent a property?

Basel sound good

Thank you! How far around Basel would you like to search? Please let me know your preferred search radius in kilometers.

Let's say 5km max

Perfect, searching within 5 km of Basel. Now, what is your maximum monthly budget for rent (in CHF)?

Figure 6.15: Chatbot interaction

6.6 Integration with Existing Systems

Swissplatz currently hosts its public website on Wix. The initial goal of the project was to integrate our application into the existing Wix site via an iFrame. Wix provides several tools to support such integrations.

The communication between Wix and our application is handled through a message, which is send from Wix to us. We then enhance our HTTP headers with this information, which enable the exchange of user-specific information - for example, identifying the logged-in user and their role. This allows the system to distinguish between administrators and regular customers.

To support this mechanism, custom header handling was implemented in the frontend, middleware, and database layers. Although the project scope later shifted from a customer-facing solution to an administrator-focused *Relocation Dashboard*, this integration code remains in the codebase for potential future use.

Both the middleware and frontend include interceptors dedicated to this task. They extract user information from incoming requests and, if necessary, enrich it with missing metadata. This is essential for ensuring that API endpoints receive complete and consistent context data.

Embedding the application into Wix via iFrame is straightforward: Wix offers a built-in iFrame component where a URL or IP address can be specified. Custom JavaScript can also be injected into the Wix page to interact with Wix APIs and retrieve user data. For example, the following script enriches HTTP calls with user information:

```
import { currentUser } from 'wix-users';

async function getEmailOfUser(currentUser) {
        return await currentUser.getEmail();
}

async function sendCurrentUserToRelocationSite(currentUser) {
        if (currentUser.loggedIn) {
                const id = currentUser.id;
                const role = currentUser.role;
                const email = await getEmailOfUser(currentUser);

                $w("#ip6-relocation-iframe").postMessage({
                        type: "CUSTOMER_INFO",
                        payload: { id, role, email }
                });
          }
}

$w.onReady(function () {
        sendCurrentUserToRelocationSite(currentUser);
});
```

In this example, the currently logged-in user is imported via the Wix wix-users module.
The $w.onReady() function triggers once the page has loaded, calling our custom function to send user data to the declared iFrame ID. On the frontend side of our relocation application, this data can be received and processed accordingly. At present, this code is not active in production, as the relocation application is not yet deployed within Wix.

6.7 Database Model

To persist application data, the system relies on a dedicated database layer. In the code, each database table or entity is represented through an interface, while the concrete implementation remains hidden within the respective subsystem. This abstraction layer enables the database technology to be swapped without requiring changes in higher-level business logic. It also improves testability and allows for parallel development, as the application logic can be implemented and tested before the final database technology is chosen.

6.7.1 Choice of Database Technology

At the start of development, `Redis` was selected as the database technology. This choice was made because Redis offers a flexible, schema-less data model, allowing for rapid prototyping without the need to enforce strict typing. Its in-memory nature also provides fast read/write operations, which was beneficial during early iterations of the project.

Later, the database layer was reimplemented using `PostgreSQL` to take advantage of its strong typing, relational integrity, advanced query capabilities, and ACID-compliant transaction model. PostgreSQL is also more suitable for long-term scalability and analytical queries.

6.7.2 Challenges with PostgreSQL Integration

While the PostgreSQL implementation worked in isolation, issues arose in the integrated, multi-service setup. Both the middleware and the chatbot service require direct database access, but they maintain separate entity ownership and different startup sequences. This led to conflicts, especially when the middleware attempted to resolve primary keys generated by the chatbot service during its own startup phase. Due to PostgreSQL's strict schema enforcement, these conflicts caused service startup failures.

Redis, with its more relaxed typing and flexible schema handling, did not suffer from these issues. As a result, Redis remains the primary database in the current deployment. However, the PostgreSQL implementation is still part of the repository and can be re-enabled at any time thanks to the interface-based abstraction layer.

6.7.3 Future Migration Path

The current architecture is designed so that migrating to PostgreSQL or another database technology will only require implementing the corresponding repository interfaces. This minimizes risk and development effort when switching technologies in the future.

6.7.4 Evolution of the Data Model

During the conceptual design phase, a simplified database schema was proposed (see Figure 6.16). It consisted of only four core tables or classes, representing the minimal set of entities required to operate the system.

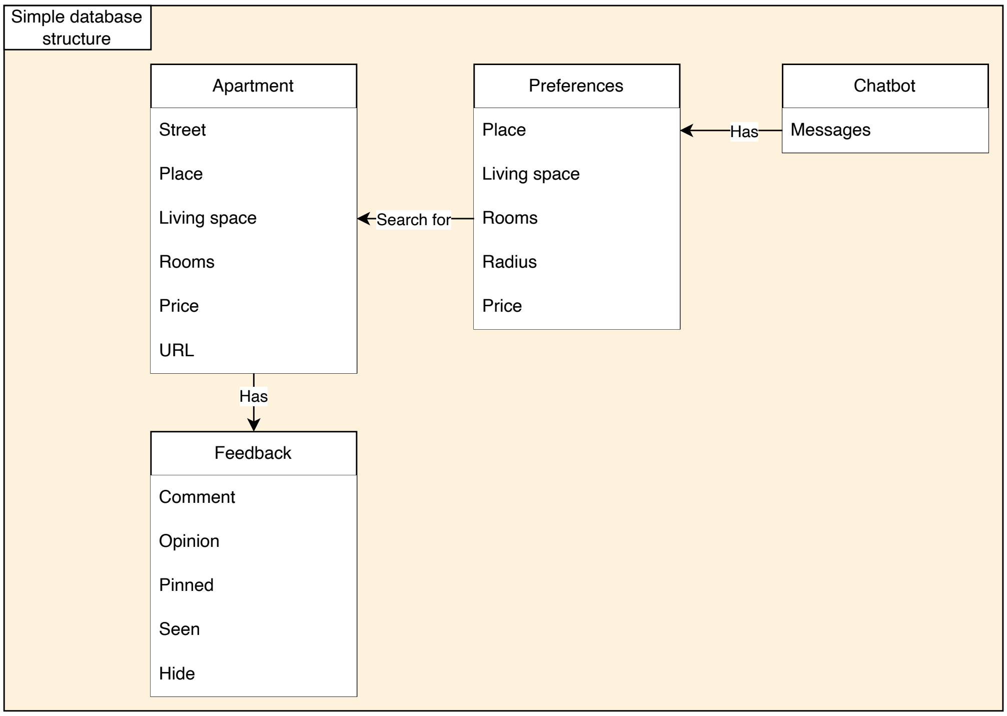


Figure 6.16: Conceptual and Simplified Database Structure

As development progressed, additional requirements and features led to a more complex schema. This evolved structure includes additional datatypes, primary and foreign key relationships, and two entirely new tables: `Queue` and `Timeslot`. These tables are essential for scheduling periodic apartment searches and updating the database at defined intervals.

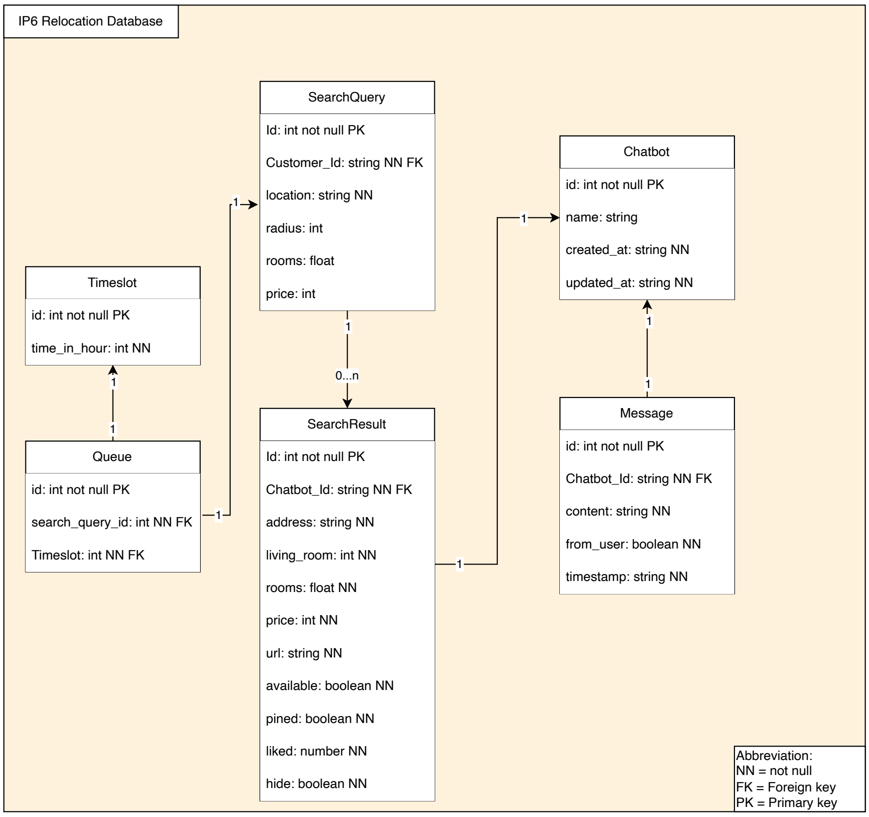


Figure 6.17: Complex Database Structure

Another significant change is the merging of the feedback entity into the `SearchResult` table. As explained earlier, this integration reduces the number of database queries required by the frontend and simplifies the overall data retrieval process.

The result is a data model that is more expressive, operationally efficient, and better aligned with the system's real-world usage patterns.

6.8 Docker-based Containerization

To ensure scalability, maintainability, and deployment flexibility, each subsystem is containerized. This means that the **Frontend**, **Middleware**, **Chatbot**, **Database**, and **Reverse Proxy** run as independent services, each packaged with its own dependencies and runtime environment.

Scalability and Resource Management

Containerization allows fine-grained control over resource allocation. For example, the Middleware service - responsible for executing apartment searches - can be allocated more CPU cores or memory, or horizontally scaled to multiple replicas when the workload exceeds a given threshold. Such scaling can be configured dynamically in production environments using orchestration tools (e.g., Docker Swarm or Kubernetes).

Networking and Communication

All containers run in a shared Docker network, which enables service-to-service communication via internal DNS names (e.g., `middleware` can reach `database` simply by using its service name in the connection string). This ensures isolation from the host system and consistent communication across environments.

Persistent Data

While most containers are stateless, the database container
(e.g., Redis or PostgreSQL) requires persistent storage. Docker volumes are used to retain data across container restarts and deployments, ensuring no data loss.

Development and Deployment

For local development, containers can be started with hot-reloading enabled (e.g., Angular dev server, Python auto-reload). In production, optimized builds are used to reduce image size and improve performance. The only required dependency for deployment is Docker itself; all other dependencies (Python, Angular, Node.js, etc.) are downloaded and configured within the container images.

Orchestration with docker-compose

The `docker-compose.yaml` file defines and starts all services with a single command. Each subsystem has its own `Dockerfile`, tailored to its needs. For example, the Chatbot service's `Dockerfile`:

```
FROM python:3.10-slim

WORKDIR /app

COPY requirements.txt ./
RUN pip install --no-cache-dir --upgrade pip
RUN pip install -I --no-cache-dir -r requirements.txt

COPY . .

CMD ["python", "run.py"]
```

Here, the base image specifies the correct Python runtime, dependencies are installed from `requirements.txt`, and finally the service is started with the appropriate entrypoint.

Security and Isolation

By running each subsystem in its own container, the architecture benefits from process isolation. Potential failures or vulnerabilities in one service do not directly affect the others, improving both reliability and security.

Overall, containerization makes the relocation platform easier to develop, deploy, and scale, while maintaining a clean separation of concerns between subsystems.

7 Evaluation

This chapter evaluates the developed system in terms of its usability, performance, and overall effectiveness compared to existing solutions. The goal is to present clear, fact-based results that show whether the requirements defined at the beginning of the project have been met, and to identify any areas that still need improvement. While subjective impressions and broader interpretations are discussed later in the *Discussion* chapter, the focus here is on measurable outcomes and concrete observations gathered during testing.

7.1 Usability testing

7.1.1 Why are Usability Tests needed?

Usability testing is a key part of building any software that is meant to be used by real people. Even if a system is technically correct and complete, it will not be successful if users don't understand how to use it or struggle to complete important tasks. A usability test helps to uncover exactly those problems early.

Especially in our case, where the system is used as a daily tool by Swissplatz mediators, it's important that the workflow feels smooth, efficient, and intuitive. A confusing interface or unclear logic could slow them down and reduce trust in the tool. The usability test gives us real-world feedback and helps us make the tool better, not just from a technical perspective, but from a user experience point of view. It also helps us to avoid spending time on features that are not useful, and instead focus on what really matters to the users.

To check how easy and effective our system is to use, we conducted usability tests with real users ourselves. The feedback from these tests helped us improve the interface and make sure the system is clear and efficient to use.

To evaluate the usability of our system, we conducted a structured usability test with several test participants. The goal was to find out how well users could interact with the tool, how intuitive the interface was, and whether the chatbot and dashboard supported a smooth workflow.

7.1.2 Execution of the usability test

The test was based on a realistic usage scenario that reflects the daily tasks of Swissplatz mediators.

Participants

A total of eight participants were randomly selected for the usability test. All of them had at least basic computer skills, and some were already familiar with using AI-based chatbots.

According to usability research by Nielsen Norman Group [10], testing with just five users is often enough to uncover the majority of usability problems. However, since we had more volunteers available, we decided to include all eight participants in order to gain broader feedback and identify additional edge cases.

Equipment and Pre-Test Instructions

The usability test was conducted using a laptop or desktop computer with a stable internet connection and a modern web browser such as Chrome, Firefox, or Edge. The participants accessed the test version of the application through a dedicated URL. Before starting, they were instructed to open a browser of their choice, navigate to the test link where the application was running, and take a few moments to familiarize themselves with the layout and structure of the tool. Every participant received short and brief instructions on what usability testing is about, and what the final goal is.

Tasks

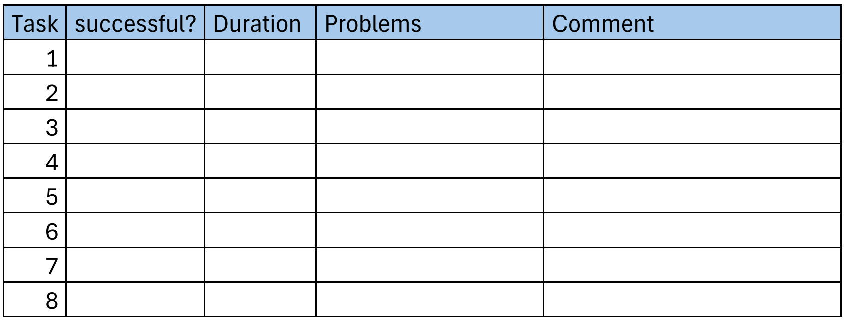
Each participant was asked to complete the following step-by-step tasks using the application:

1. Create a new chatbot
2. Name the Chatbot ”Usability Test X”
3. Provide search preferences to the chatbot:
    - Location: Baden, Switzerland
    - Minimum number of rooms: 2.5
    - Maximum budget: CHF 2,500
    - Search radius: 10 km
4. Pin the first two apartments from the result
5. Mark the first pinned apartment as ”liked”
6. Mark the second pinned apartment as ”disliked”
7. Open the third apartment from the results and check for additional information.
8. Use the filter function to show all available apartments.

During the test, we observed how the participants interacted with the system. We paid attention to any visible confusion or mistakes. Testers were encouraged to think out loud and comment on anything that seemed unclear, or frustrating.

Evaluation

To be able to measure usability in a structured way, we tracked each participant's progress using a predefined evaluation table:



Task	successful?	Duration	Problems	Comment
1				
2				
3				
4				
5				
6				
7				
8				

Figure 7.1: Measurement table

This table contained one row per task and allowed us to record:

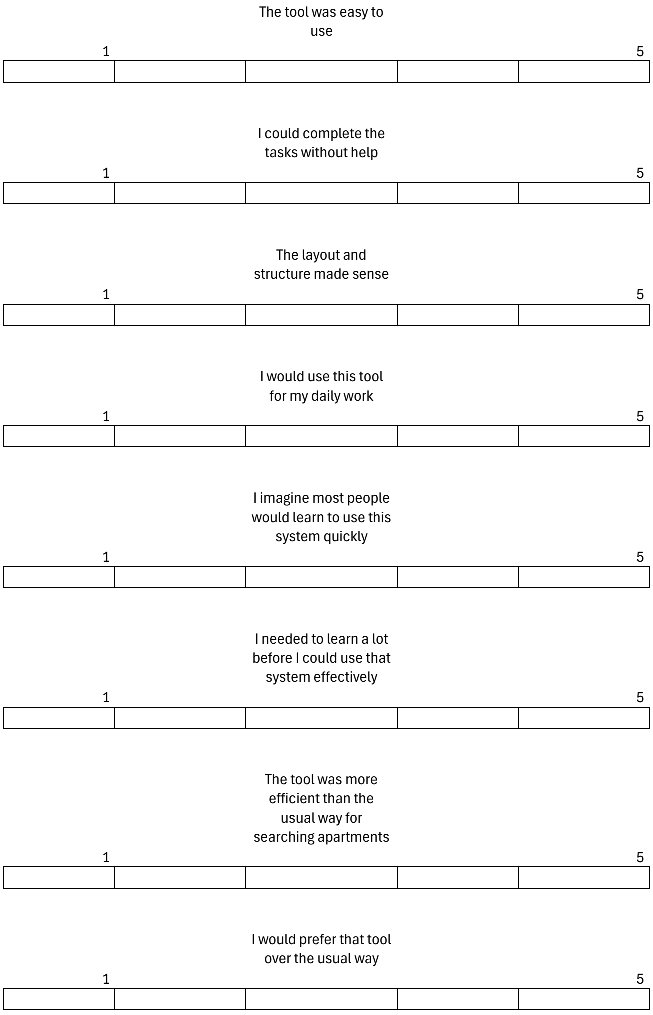
- whether the task was completed successfully
- how long the participant needed
- whether any problems occurred
- and additional comments or observations from the test supervisor

Each task was evaluated individually to see where users had difficulties, which steps were completed without hesitation, and how long the process took in total. This helped us identify which parts of the interface worked well and which required improvement. The comment field was especially useful to capture spontaneous feedback and insights that would not appear in numeric ratings.

The collected data was then summarized and compared across all participants to reveal common patterns and usability issues.

Participants evaluation

At the end of the test, participants were asked to provide feedback and rate the application in several categories.



The tool was easy to use

1 □ □ □ □ □ 5

I could complete the tasks without help

1 □ □ □ □ □ 5

The layout and structure made sense

1 □ □ □ □ □ 5

I would use this tool for my daily work

1 □ □ □ □ □ 5

I imagine most people would learn to use this system quickly

1 □ □ □ □ □ 5

I needed to learn a lot before I could use that system effectively

1 □ □ □ □ □ 5

The tool was more efficient than the usual way for searching apartments

1 □ □ □ □ □ 5

I would prefer that tool over the usual way

1 □ □ □ □ □ 5

Figure 7.2: Participants evaluation table

With this additional information, we earned more insights and gave the participants a chance to add more feedback to the application and experience as well.

7.1.3 Test Results

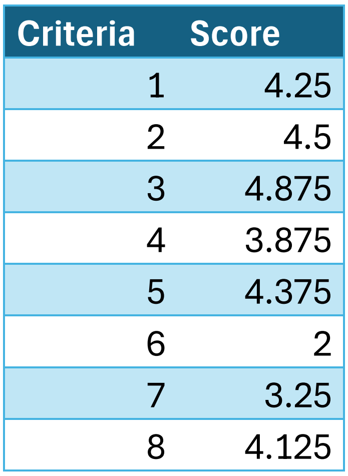
The usability tests provided valuable insights into how well the system supports users during the apartment search process. All eight participants were able to complete the given tasks successfully, although a few minor issues and usability challenges were observed along the way.

Most users found the system easy to use, and the core workflow was generally understood without the need for help.

Based on the evaluation table and the recorded feedback, we identified the following key findings:

- **Language inconsistency:** Switching between English and German in the user interface caused confusion. A consistent language setting is recommended.
- **Chat input field size:** The text input field was considered too small. Some users had difficulty seeing what they were typing, especially for longer messages.
- **Repeated questions:** The chatbot sometimes repeated previously asked questions, which disrupted the conversation flow.
- **Pinning functionality:** The pinning functionality was unclear in regards of how it differed from the ”like” option.
- **Cursor reset on tab change:** When switching tabs and returning to the application, the cursor was no longer active in the input field, which broke the typing flow.
- **Multi-location input:** The chatbot did not support entering more than one location at once, which some users expected.
- **Search success and completion:** Despite the mentioned issues, all participants managed to finish the tasks and find suitable apartments. The general structure was considered clear.

The collected feedback from all participants was evaluated and averaged across eight key criteria. The resulting scores (on a scale from 1 (strongly disagree) to 5 (strongly agree)) give a clear picture of how the application performed from a user experience perspective.



Criteria	Score
1	4.25
2	4.5
3	4.875
4	3.875
5	4.375
6	2
7	3.25
8	4.125

Figure 7.3: Results of criteria

Overall, the average scores were clearly above 4.0 in most categories, which shows that the application is already quite usable and intuitive for its target group. Still, targeted refinements to the chatbot logic and feedback design could further enhance the experience.

7.2 Requirement Fulfillment

The evaluation checked if the system meets the requirements set at the start of the project. Most of the main goals were reached: mediators can create and manage chatbot sessions, collect client preferences through a simple conversation, and get housing listings from multiple platforms using the scraping modules. The shared dashboard works for real-time collaboration and feedback between chatbot and mediators.

Non-functional needs were also covered. Usability tests showed that the interface is mostly easy to use, with all participants able to finish their tasks. Some smaller issues, such as mixed languages in the interface or the chatbot asking the same question twice, were found but did not stop the system from working as intended.

In summary, the system meets its key requirements, with only a few details left to improve in future updates.

7.3 Comparison to the State of the Art

Compared to existing Swiss housing platforms such as ImmoScout24, Comparis, or Flatfox, our solution offers significant advantages:

- Real-time collaboration between client and mediator in a single interface.
- Automatic reuse of client preferences for search and application preparation.
- Integrated conversational guidance that supports multiple languages.

While current market platforms act primarily as search engines, our solution bridges the gap to full relocation support by combining powerful search capabilities with process guidance and communication tools. This aligns with the gap identified in the *State of the Art* chapter and shows the added value of integrating AI-assisted interaction.

8 Discussion

8.1 Addressing research questions

What are the key challenges and best practices in retrieving and processing data from various platforms to enable automation in the rental mediation process? Retrieving and processing data from different housing platforms proved to be one of the most challenging parts of the project. Many platforms in Switzerland do not offer public APIs, which means that data must be collected either through static HTML scraping or by using tools such as Selenium for dynamically rendered content. This comes with technical hurdles like anti-bot measures, variable HTML structures, and inconsistent data formats. In our implementation, we used a modular scraping architecture with separate "searcher" modules for each platform. These modules follow a common interface and can be run in parallel by an orchestrator, which speeds up the process and keeps the design extensible. A central middleware layer handles validation, logging, and database interactions, ensuring that scraped data is normalized into a consistent format with fields for address, size, price, and status. Legal and ethical considerations, such as respecting robots.txt and avoiding excessive request rates, were also part of the design from the beginning. This combination of technical flexibility, clear structure, and compliance resulted in a robust and maintainable data pipeline.

How can the user experience be optimized to simplify data input and interaction between clients and mediators and how is it perceived by users? The user experience was designed to make data input and communication between clients and mediators as smooth as possible. Instead of long static forms, the platform uses a conversational chatbot to guide users through the preference-gathering process. This allows questions to be asked one at a time and in a natural order, while the system automatically summarizes the answers and converts them into a structured search query. The preferences are stored centrally and reused throughout the process, so users do not have to enter the same information multiple times. Clients and mediators interact through a shared dashboard, where search results appear in real time and can be marked as liked, pinned, or hidden. Feedback actions are immediately visible to both sides, making collaboration easier. The system also supports multilingual communication and provides clear status updates, which is especially valuable for clients unfamiliar with the Swiss housing market. Usability tests showed that users appreciated the reduced manual effort, the clear structure, and the feeling of transparency in the process. Some minor issues, such as inconsistent language or unclear button functions, were identified and fixed, demonstrating the benefit of iterative testing.

What are the possibilities of artificial intelligence that can be utilized to automate processes in the rental mediation workflow? AI plays a supporting but important role in the current version of the platform. At present, it is mainly used for conversational intake of client preferences and for answering general questions about the process. The chatbot can clarify missing details and guide users towards the next steps without replacing the human mediator. While this already improves efficiency, there is room for expansion. Literature such as Subedi's work on the Landlord-Tenant Rights Bot shows how retrieval-augmented generation (RAG) can deliver accurate, context-aware legal information. Similarly, research on hostel recommendation systems demonstrates the potential for ranking results based on content similarity to user preferences. In the future, the platform could integrate these techniques to provide personalized ranking of search results and automated preparation of application documents. However, it will be important to maintain a human-in-the-loop approach, where AI suggestions are transparent and always subject to review by the mediator. This balance between automation and human oversight can increase speed and scalability without reducing trust in the process.

Summary

To wrap up our work, let's briefly revisit the starting point of Swissplatz. Out of the three main phases of their workflow - Searching, Administration, and Viewing - our developed solution focuses on, and help with automating the Searching phase.

We built a solid platform foundation capable of supporting processes for all three phases, with the current emphasis on search. Swissplatz employees can now manage the entire searching process in one place: documenting customer preferences, running targeted searches for suitable housing options, and keeping everything organized without switching between multiple tools.

A key part of this solution is our chatbot, which assists employees during the search. It provides personalized answers, guides them through the process, and automatically searches for apartments based on the information provided - making the search faster, more structured, and easier to handle.

9 Conclusion and future work

Researching the state of the art for this project was not easy. Because the use of Artificial Intelligence in relocation services is still very new, there is little documentation or research available. The industry is also not yet very advanced in this area, so we had to rely on smaller related studies and examples from similar fields. Still, this research gave us a good starting point and helped us define clear and realistic goals for the project.

During development, legal uncertainties and communication with third-party platforms slowed us down. Some questions about data handling and possible access for end users made it necessary to rethink parts of the project. In the end, we found a good solution by changing the product so that it is now used by Swissplatz employees instead of external clients. Although it changed the course of our project, it turned out to be the right choice and worked to our advantage.

Data scraping was another big challenge. It was a new topic for our team, and many platforms made it difficult for us with technical restrictions. Several times we had to change our approach when websites updated their structure or blocked automated requests. Our early planning helped a lot here, because it allowed us to react quickly and keep the project on track.

We also had to make sure that the different parts of the system - frontend, middleware, chatbot, and database - worked well together. Clear structure and separation between components were important to make the system reliable and easy to improve later.

Overall, we can say that we are very happy and proud with the result. The solution works well for the searching phase and creates a strong base for future improvements. We learned that flexible design, early risk management, and the ability to adapt are very important when working with new technologies. We also saw the value of close collaboration with stakeholders, continuous testing, and building a solution that can grow with changing requirements.

9.1 Open challenges

One of the ongoing challenges is the constantly changing nature of the platforms we scrape data from. Whenever a platform updates its structure or security measures, our scraping methods may need to be adapted, which can slow down the process.

Another challenge is the long-term goal of unifying all functions into one single platform. While our current solution already centralizes many tasks, developing additional tools for Swissplatz will be necessary to fully replace their existing workflows.

Adding more data sources also remains difficult. Many platforms protect their data with strong security measures, making it hard to access the information in a structured way. This often requires creative technical solutions and careful planning.

Finally, some legal questions remain unresolved. Data protection regulations and unclear platform policies mean that legal risks could still arise in the future, so continuous review and adaptation will be necessary.

9.2 Further development potential

Looking ahead, there are several ways the system could be expanded to provide even greater value for Swissplatz. One area is the development of additional tools to support employees in their daily work, such as integrated comment functions, direct chat with clients, and a more advanced feedback system.

Another improvement would be to increase the capabilities of the AI. This could include gathering more details from housing advertisements through the chatbot, offering better recommendations, and analyzing user preferences more precisely to improve search results.

In the long term, the platform could be redesigned so that clients themselves become the direct end users, rather than Swissplatz acting as an intermediary. Finally, the system could be extended to cover the other two phases of the relocation process - administration and viewing - in order to create a fully automated and end-to-end solution.

By following these directions, the platform could evolve from a specialized internal tool into a fully integrated relocation assistant that automates the entire process and significantly improves the customer experience.

Sources

[1] Altair Global, *Exploring the opportunity of ai in corporate relocation*, Accessed: 2025-08-10, Apr. 2024. [Online]. Available: `https://www.altairglobal.com/news/exploring-the-opportunity-of-ai-on-corporate-relocation/`.

[2] N. Subedi, *Empowering housing equity: An ai-driven chatbot for landlord-tenant rights and community stability*, SSRN Electronic Journal, Preprint, accessed: 2025-08-10, 2025. DOI: `10.2139/ssrn.5227597`. [Online]. Available: `https://papers.ssrn.com/sol3/papers.cfm?abstract_id=5227597`.

[3] F. O. Isinkaye, I. G. AbiodunBabs, and M. T. Paul, „Development of a mobile-based hostel location and recommendation chatbot system“, *Int. J. Inf. Technol. Comput. Sci*, vol. 14, pp. 23–33, 2022.

[4] Oracle. „What is a chatbot?“ Accessed: 2025-08-09, Accessed: Aug. 9, 2025. [Online]. Available: `https://www.oracle.com/chatbots/what-is-a-chatbot/`.

[5] R. Morshedi, B. Chu, E. Huang, and L. Ivers, „Web scraping: Applications in infrastructure planning“, *New South Wales*, vol. 3, 2019.

[6] N. Medvidovic, „On the role of middleware in architecture-based software development“, in *Proceedings of the 14th international conference on Software engineering and knowledge engineering*, 2002, pp. 299–306.

[7] Amazon Web Services. „Was ist middleware?“ Accessed: 2025-08-14. [Online]. Available: `https://aws.amazon.com/de/what-is/middleware/`.

[8] Matillion. „The types of databases (with examples)“. Accessed: 2025-08-11. [Online]. Available: `https://www.matillion.com/blog/the-types-of-databases-with-examples`.

[9] Rivery. „Database types guide“. Accessed: 2025-08-11. [Online]. Available: `https://rivery.io/data-learning-center/database-types-guide/`.

[10] J. Nielsen. „Why you only need to test with 5 users“. Nielsen Norman Group, Accessed: Aug. 9, 2025. [Online]. Available: `https://www.nngroup.com/articles/why-you-only-need-to-test-with-5-users/`.

Declaration of honesty

We hereby declare that we have written the present bachelor thesis independently and solely using the sources indicated. All passages taken literally or in substance from the sources listed are marked in the work as quotations or paraphrases. This bachelor thesis has not yet been published. It has neither been made accessible to other parties nor submitted to any other examination authority.

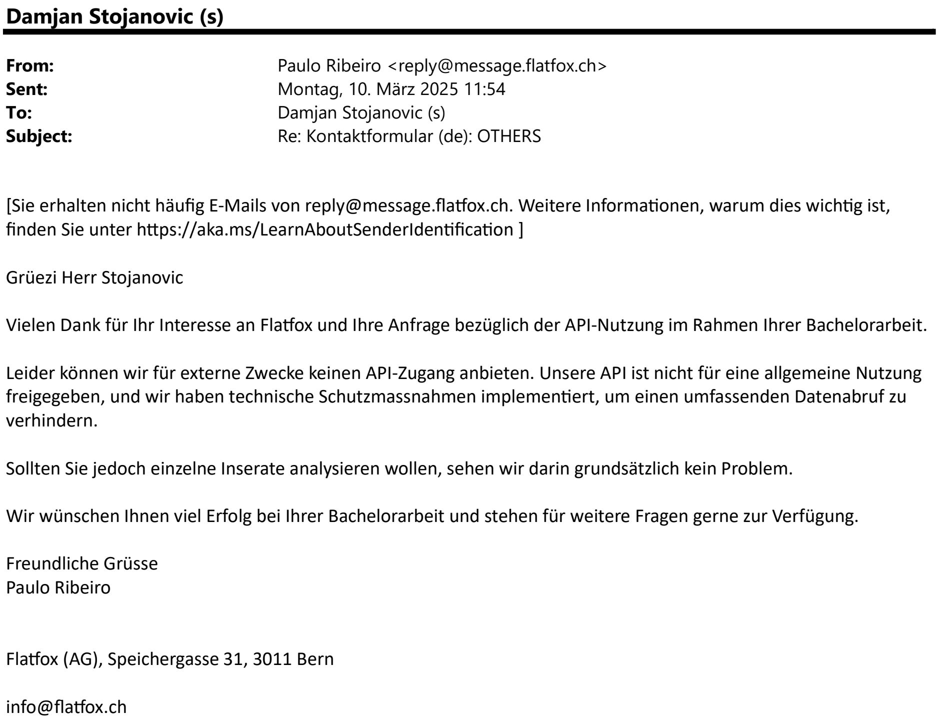
Windisch, 13. August 2025

Name: Stefan Simic
Signature: Stefan Simic

Name: Damjan Stojanovic
Signature: Stojanovic D.

A Appendix

A.0.1 Platform Requests E-Mail



Damjan Stojanovic (s)

From:	Paulo Ribeiro <reply@message.flatfox.ch>
Sent:	Montag, 10. März 2025 11:54
To:	Damjan Stojanovic (s)
Subject:	Re: Kontaktformular (de): OTHERS

[Sie erhalten nicht häufig E-Mails von reply@message.flatfox.ch. Weitere Informationen, warum dies wichtig ist, finden Sie unter https://aka.ms/LearnAboutSenderIdentification]

Grüezi Herr Stojanovic

Vielen Dank für Ihr Interesse an Flatfox und Ihre Anfrage bezüglich der API-Nutzung im Rahmen Ihrer Bachelorarbeit.

Leider können wir für externe Zwecke keinen API-Zugang anbieten. Unsere API ist nicht für eine allgemeine Nutzung freigegeben, und wir haben technische Schutzmassnahmen implementiert, um einen umfassenden Datenabruf zu verhindern.

Sollten Sie jedoch einzelne Inserate analysieren wollen, sehen wir darin grundsätzlich kein Problem.

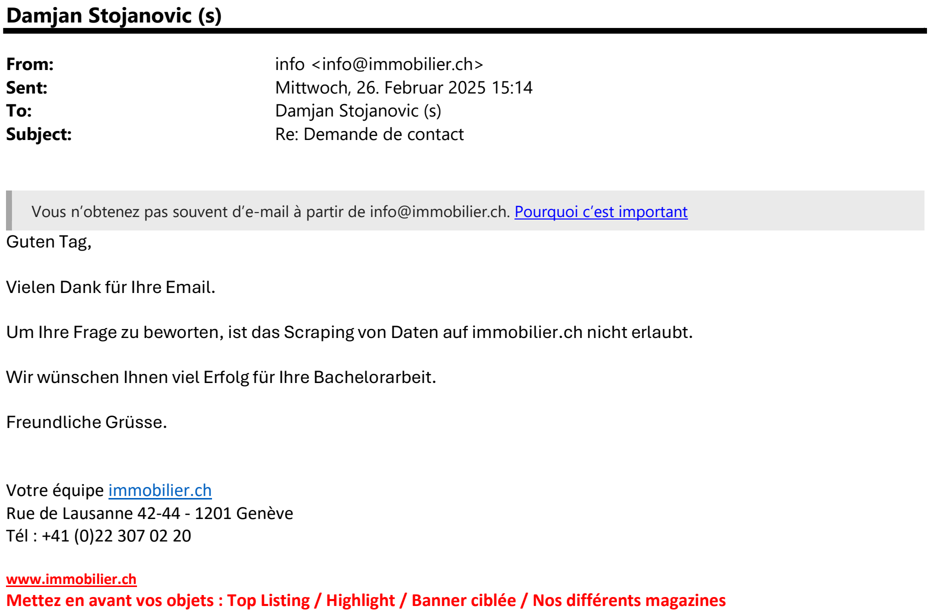
Wir wünschen Ihnen viel Erfolg bei Ihrer Bachelorarbeit und stehen für weitere Fragen gerne zur Verfügung.

Freundliche Grüsse
Paulo Ribeiro

Flatfox (AG), Speichergasse 31, 3011 Bern

info@flatfox.ch

Figure A.1: API Access request to Flatfox



Damjan Stojanovic (s)

From:	info <info@immobilier.ch>
Sent:	Mittwoch, 26. Februar 2025 15:14
To:	Damjan Stojanovic (s)
Subject:	Re: Demande de contact

Vous n'obtenez pas souvent d'e-mail à partir de info@immobilier.ch. Pourquoi c'est important

Guten Tag,

Vielen Dank für Ihre Email.

Um Ihre Frage zu beworten, ist das Scraping von Daten auf immobilier.ch nicht erlaubt.

Wir wünschen Ihnen viel Erfolg für Ihre Bachelorarbeit.

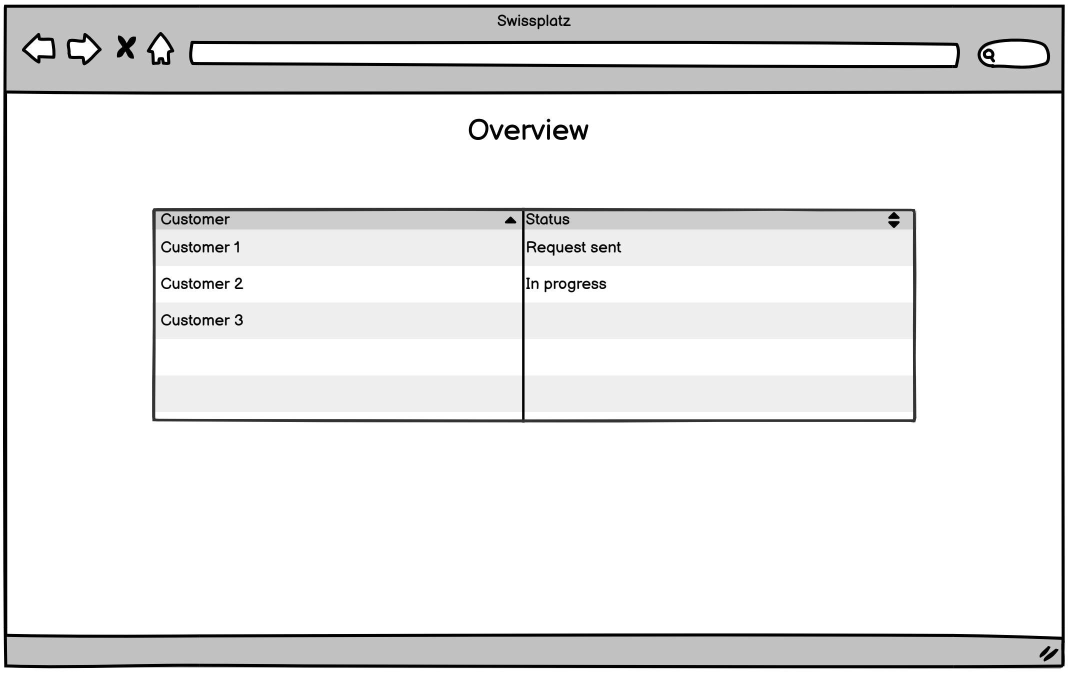
Freundliche Grüsse.

Votre équipe immobilier.ch
Rue de Lausanne 42-44 - 1201 Genève
Tél : +41 (0)22 307 02 20

www.immobilier.ch
Mettez en avant vos objets : Top Listing / Highlight / Banner ciblée / Nos différents magazines

Figure A.2: Data Scraping request to immobilier

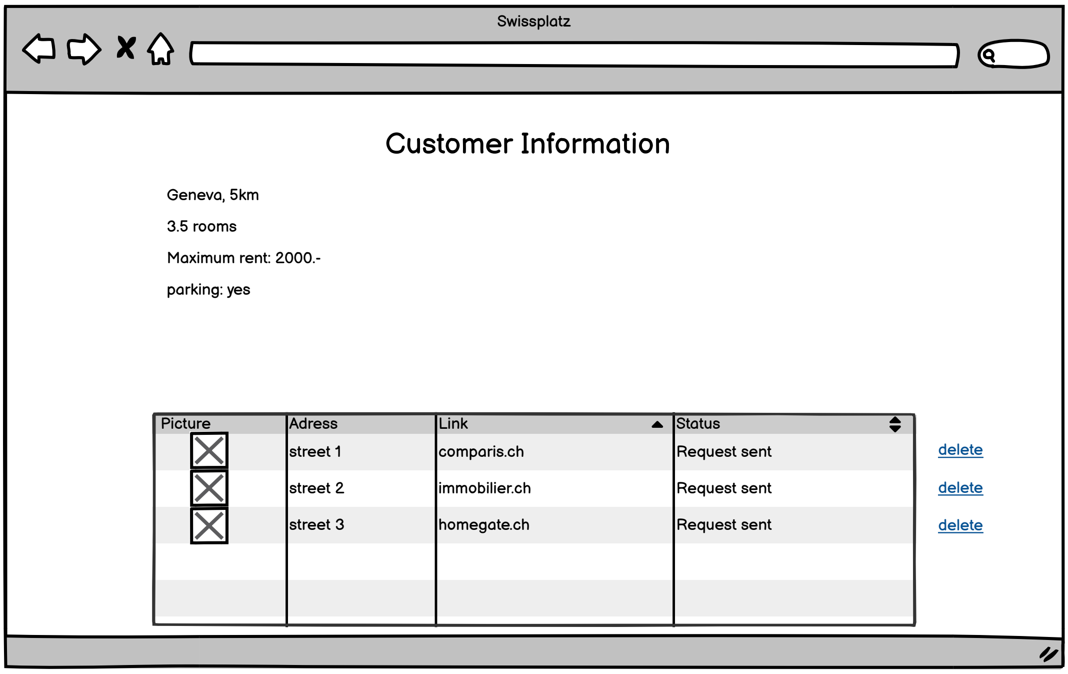
A.0.2 First Version of Lo-FI prototype



Overview

Customer	Status
Customer 1	Request sent
Customer 2	In progress
Customer 3	

Figure A.3: First Lo-Fi Prototype Admin overview



Customer Information

Geneva, 5km

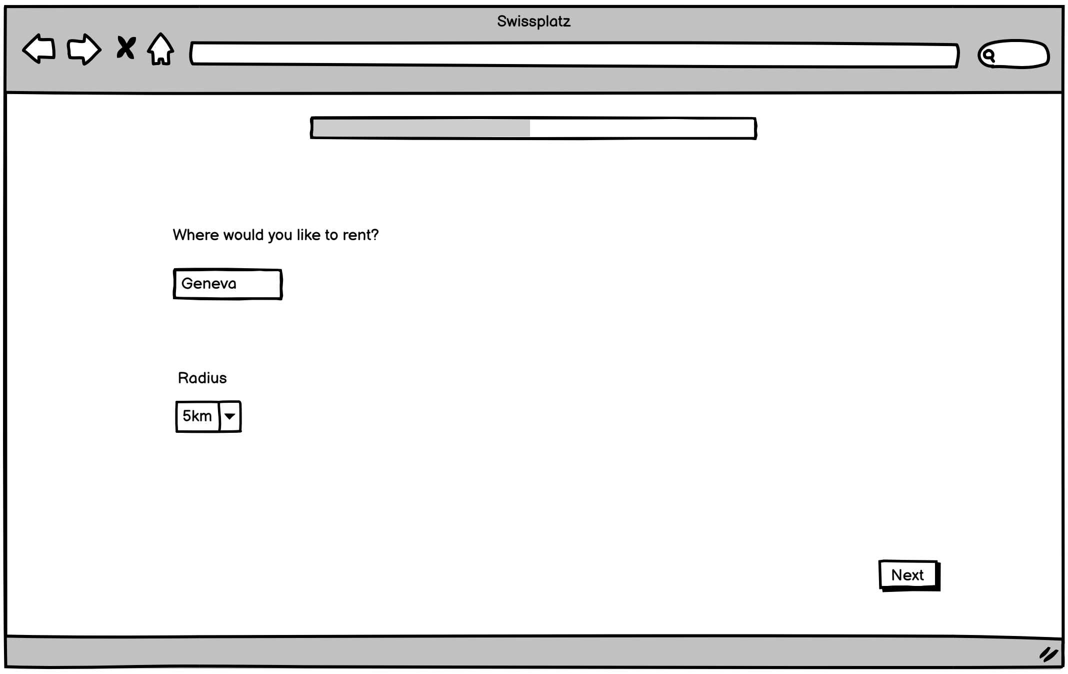
3.5 rooms

Maximum rent: 2000.-

parking: yes

Picture	Adress	Link	Status	
☒	street 1	comparis.ch	Request sent	delete
☒	street 2	immobilier.ch	Request sent	delete
☒	street 3	homegate.ch	Request sent	delete

Figure A.4: First Lo-Fi Prototype Customer Information



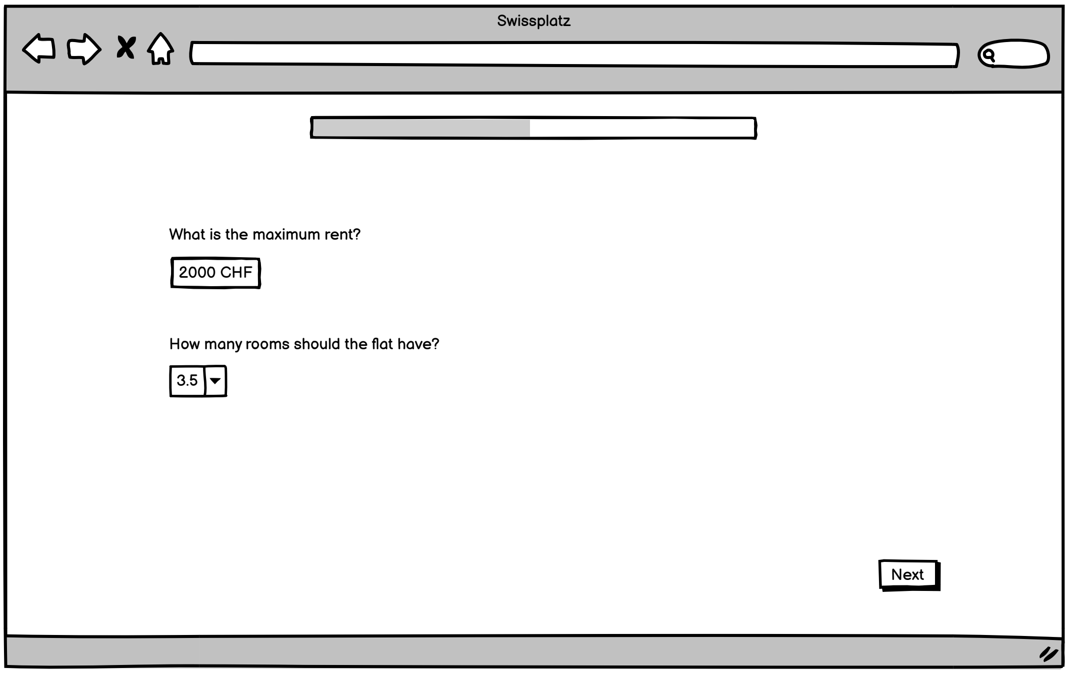
Where would you like to rent?

Geneva

Radius

5km

Figure A.5: First Lo-Fi Prototype Form Page 1



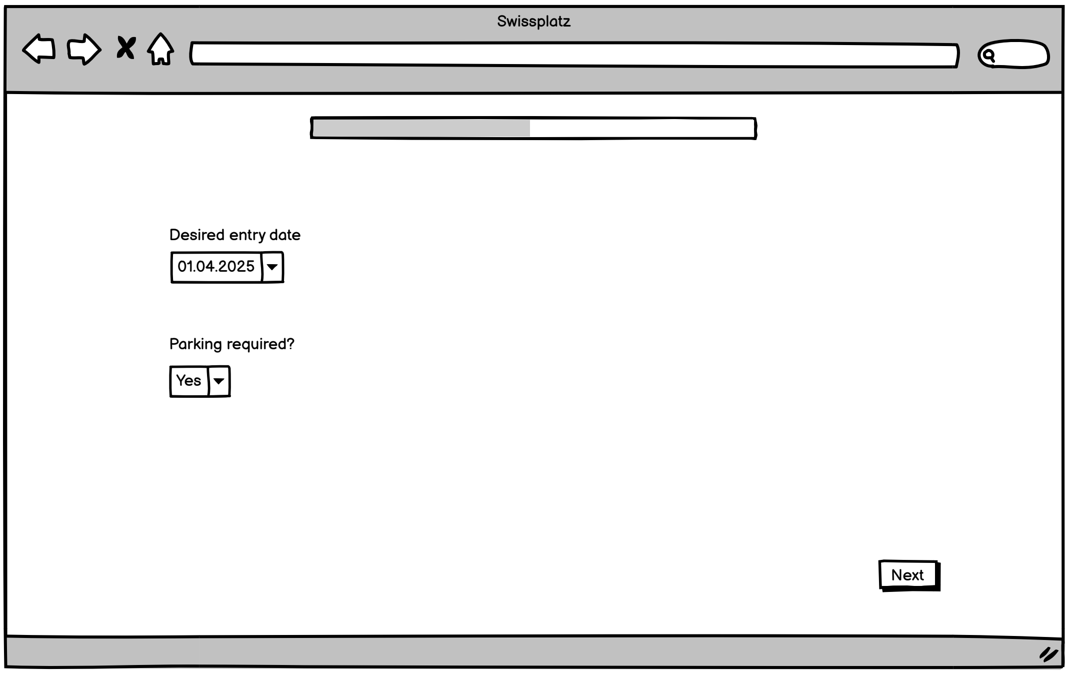
What is the maximum rent?

2000 CHF

How many rooms should the flat have?

3.5

Figure A.6: First Lo-Fi Prototype Form Page 2



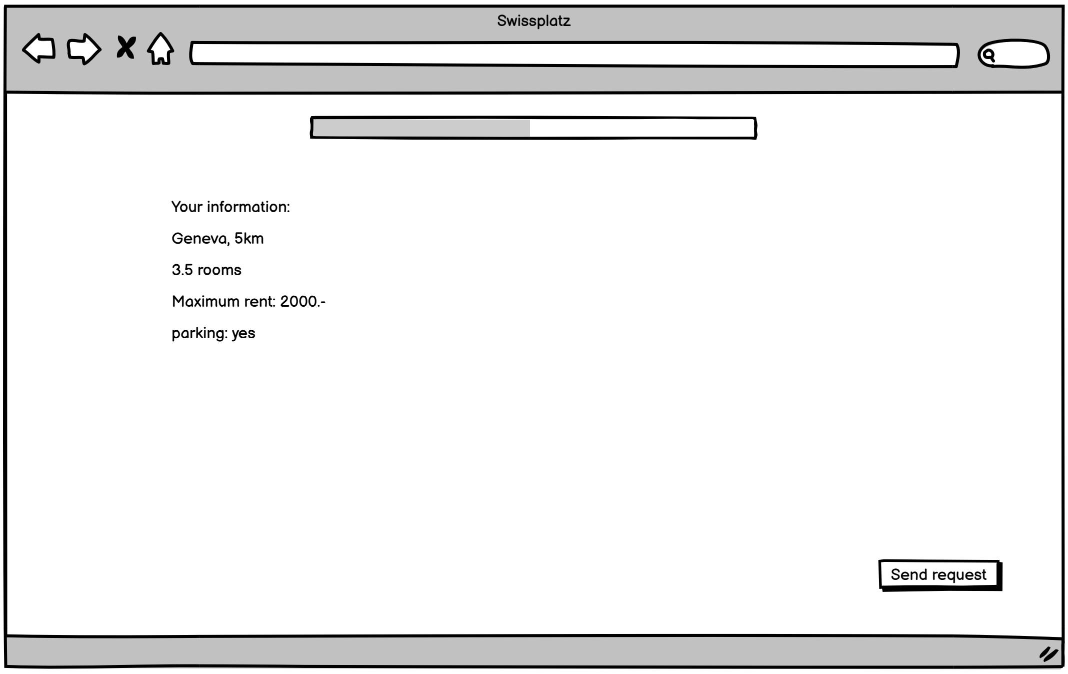
Desired entry date

01.04.2025

Parking required?

Yes

Figure A.7: First Lo-Fi Prototype Form Page 3



Your information:

Geneva, 5km

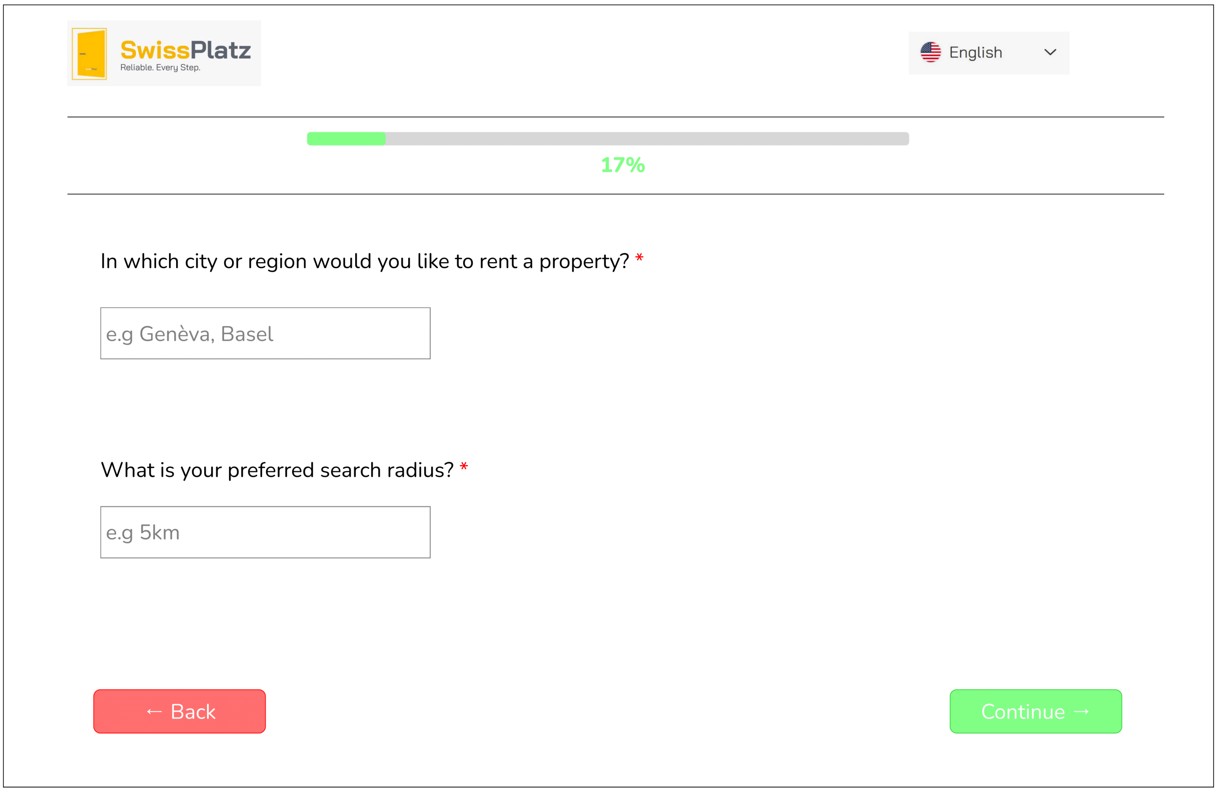
3.5 rooms

Maximum rent: 2000.-

parking: yes

Figure A.8: First Lo-Fi Prototype Summary

A.0.3 First Version of Hi-FI prototype



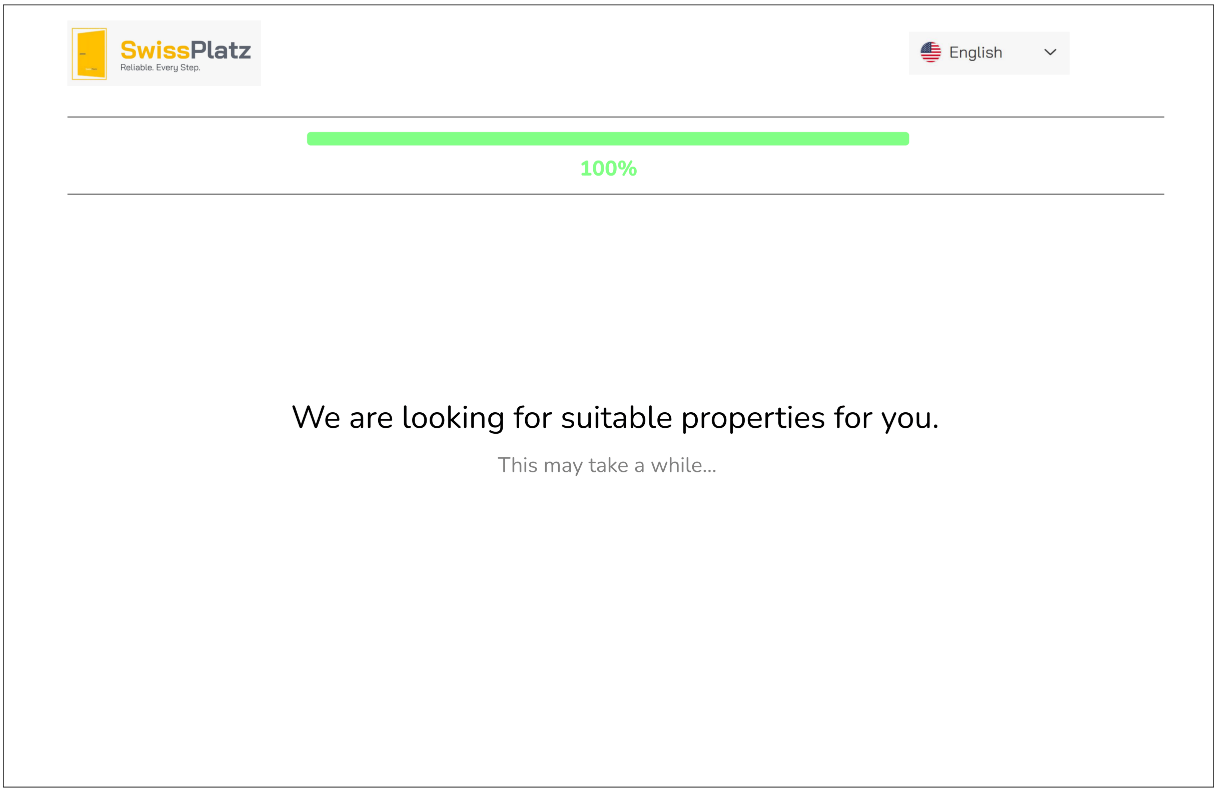
In which city or region would you like to rent a property? *

e.g Genèva, Basel

What is your preferred search radius? *

e.g 5km

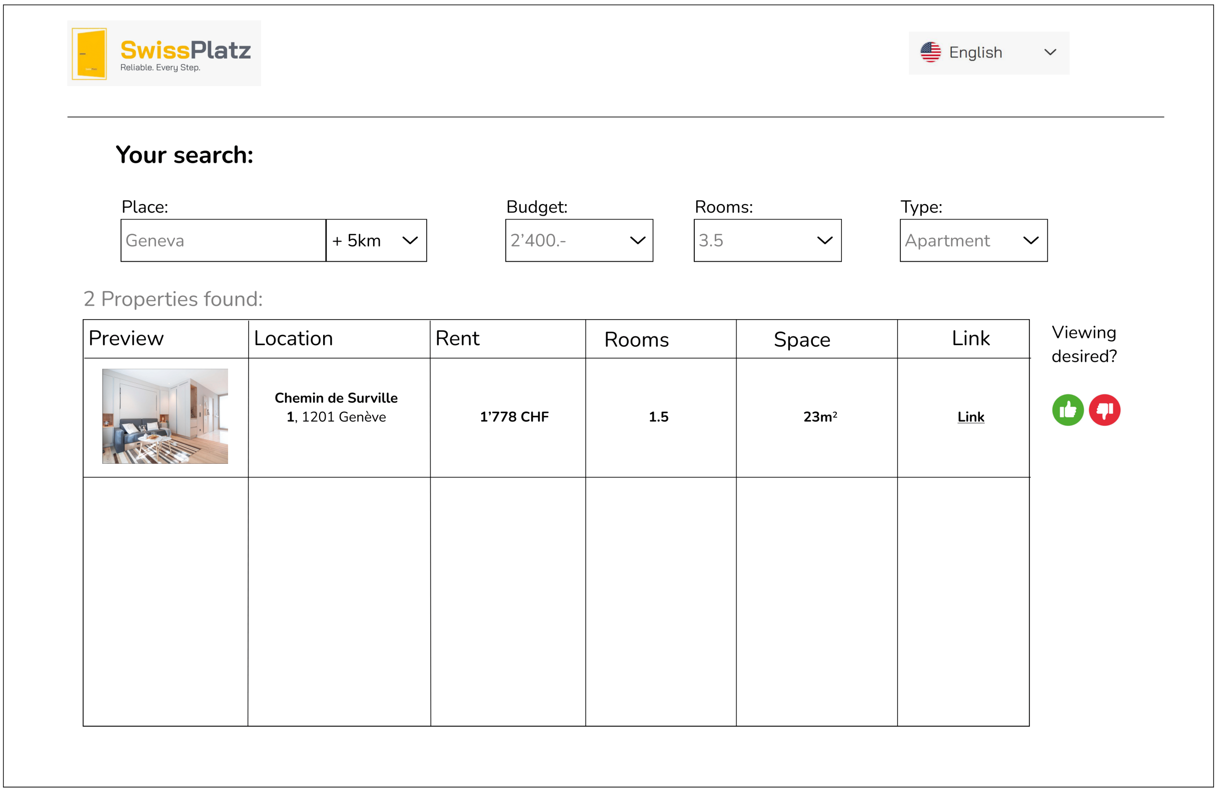
Figure A.9: First Prototype Form Screen



We are looking for suitable properties for you.

This may take a while...

Figure A.10: First Prototype Loading Screen



SwissPlatz
Reliable. Every Step.

English

Your search:

Place:		Budget:	Rooms:	Type:
Geneva	+ 5km	2'400.-	3.5	Apartment

2 Properties found:

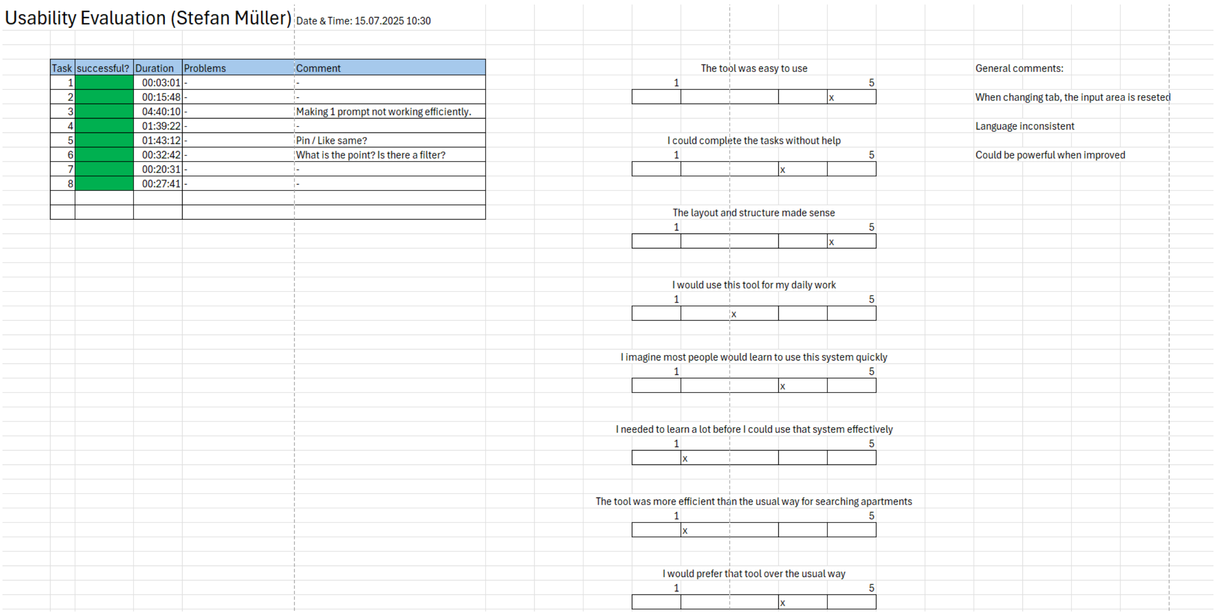
Preview	Location	Rent	Rooms	Space	Link
	Chemin de Surville **1**, 1201 Genève	**1'778 CHF**	**1.5**	**23m^2**	**Link**

Viewing desired?

👍 👎

Figure A.11: First Prototype Result Screen

A.0.4 Detailed Results of the Usability Tests



Usability Evaluation (Stefan Müller) Date & Time: 15.07.2025 10:30

Task	successful?	Duration	Problems	Comment
1	🟩	00:03:01	-	-
2	🟩	00:15:48	-	-
3	🟩	04:40:10	-	Making 1 prompt not working efficiently.
4	🟩	01:39:22	-	-
5	🟩	01:43:12	-	Pin / Like same?
6	🟩	00:32:42	-	What is the point? Is there a filter?
7	🟩	00:20:31	-	-
8	🟩	00:27:41	-	-

The tool was easy to use

1				5
				x

I could complete the tasks without help

1				5
			x	

The layout and structure made sense

1				5
				x

I would use this tool for my daily work

1				5
		x		

I imagine most people would learn to use this system quickly

1				5
			x	

I needed to learn a lot before I could use that system effectively

1				5
	x			

The tool was more efficient than the usual way for searching apartments

1				5
	x			

I would prefer that tool over the usual way

1				5
			x	

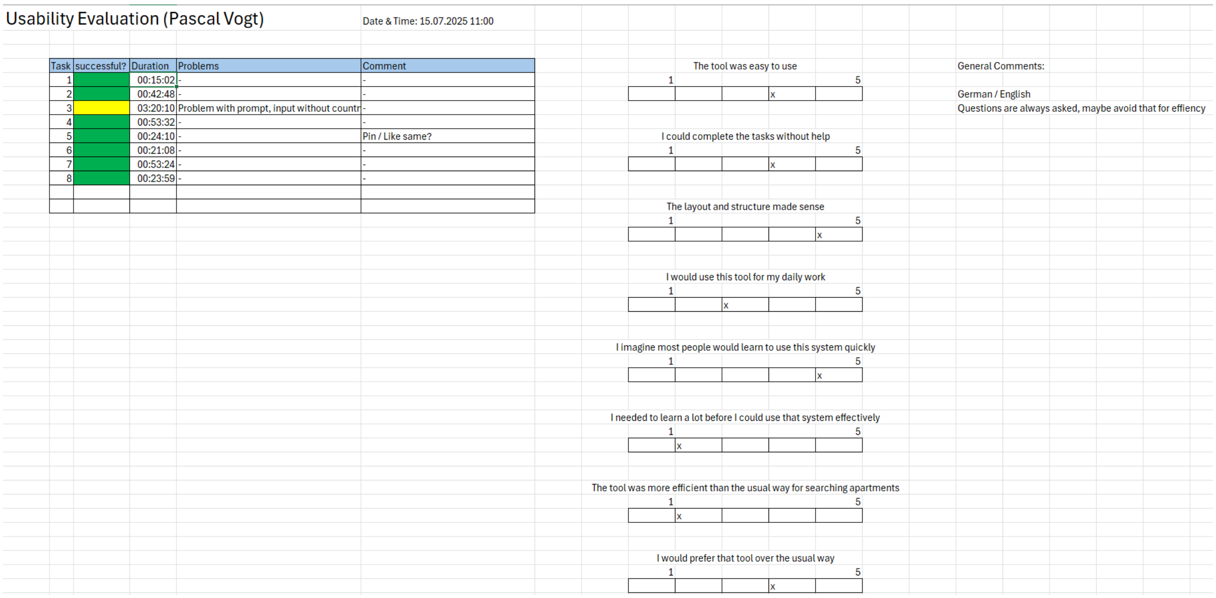
General comments:

When changing tab, the input area is reseted

Language inconsistent

Could be powerful when improved

Figure A.12: Usability Testing Stefan Müller



Usability Evaluation (Pascal Vogt) Date & Time: 15.07.2025 11:00

Task	successful?	Duration	Problems	Comment
1	🟩	00:15:02	-	-
2	🟩	00:42:48	-	-
3	🟨	03:20:10	Problem with prompt, input without countr	-
4	🟩	00:53:32	-	-
5	🟩	00:24:10	-	Pin / Like same?
6	🟩	00:21:08	-	-
7	🟩	00:53:24	-	-
8	🟩	00:23:59	-	-

The tool was easy to use

1				5
			x	

I could complete the tasks without help

1				5
			x	

The layout and structure made sense

1				5
				x

I would use this tool for my daily work

1				5
		x		

I imagine most people would learn to use this system quickly

1				5
				x

I needed to learn a lot before I could use that system effectively

1				5
	x			

The tool was more efficient than the usual way for searching apartments

1				5
	x			

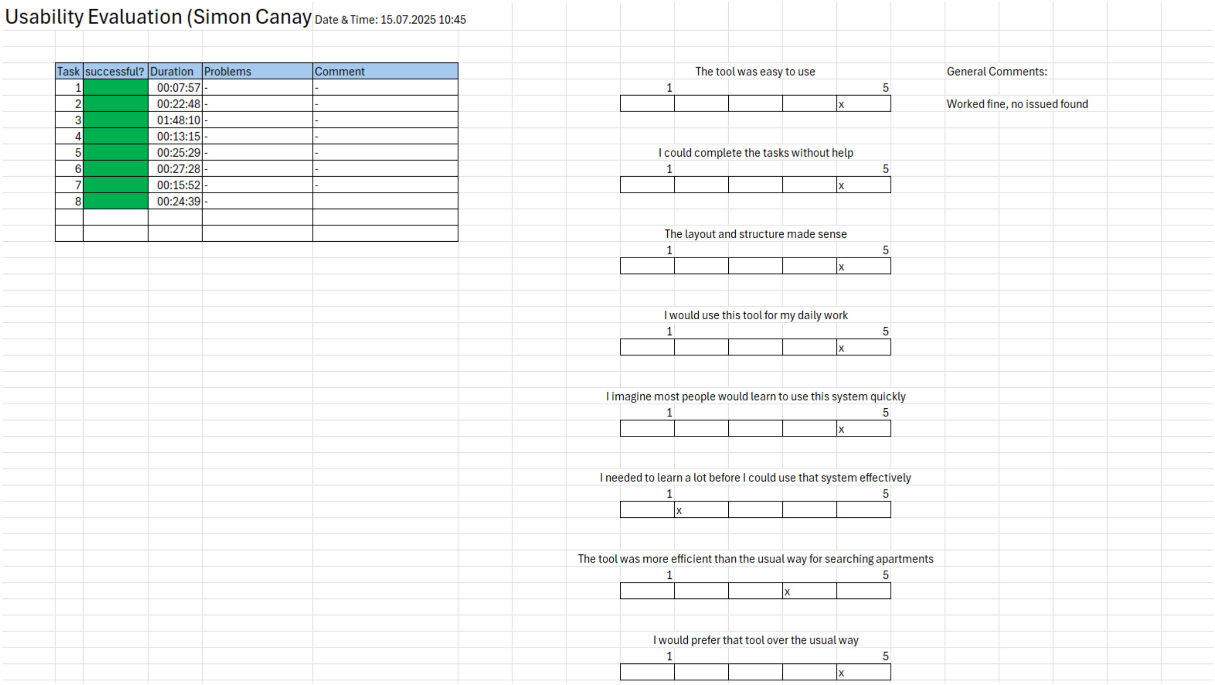
I would prefer that tool over the usual way

1				5
			x	

General Comments:

German / English
Questions are always asked, maybe avoid that for effiency

Figure A.13: Usability Testing Pascal Vogt



Usability Evaluation (Simon Canay Date & Time: 15.07.2025 10:45

Task	successful?	Duration	Problems	Comment
1	🟩	00:07:57	-	-
2	🟩	00:22:48	-	-
3	🟩	01:48:10	-	-
4	🟩	00:13:15	-	-
5	🟩	00:25:29	-	-
6	🟩	00:27:28	-	-
7	🟩	00:15:52	-	-
8	🟩	00:24:39	-	

The tool was easy to use

1				5
			x	

I could complete the tasks without help

1				5
			x	

The layout and structure made sense

1				5
			x	

I would use this tool for my daily work

1				5
			x	

I imagine most people would learn to use this system quickly

1				5
			x	

I needed to learn a lot before I could use that system effectively

1				5
	x			

The tool was more efficient than the usual way for searching apartments

1				5
		x		

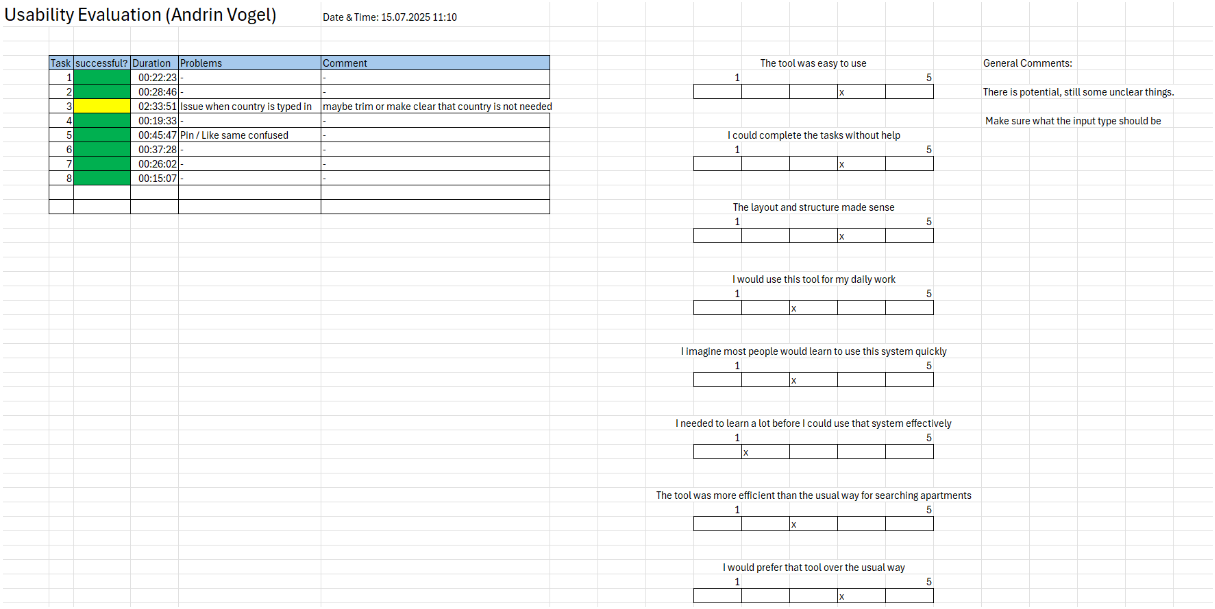
I would prefer that tool over the usual way

1				5
			x	

General Comments:

Worked fine, no issued found

Figure A.14: Usability Testing Simon Canay



Usability Evaluation (Andrin Vogel) Date & Time: 15.07.2025 11:10

Task	successful?	Duration	Problems	Comment
1	🟩	00:22:23	-	-
2	🟩	00:28:46	-	-
3	🟨	02:33:51	Issue when country is typed in	maybe trim or make clear that country is not needed
4	🟩	00:19:33	-	-
5	🟩	00:45:47	Pin / Like same confused	-
6	🟩	00:37:28	-	-
7	🟩	00:26:02	-	-
8	🟩	00:15:07	-	-

The tool was easy to use

1				5
			x	

I could complete the tasks without help

1				5
			x	

The layout and structure made sense

1				5
			x	

I would use this tool for my daily work

1				5
		x		

I imagine most people would learn to use this system quickly

1				5
		x		

I needed to learn a lot before I could use that system effectively

1				5
	x			

The tool was more efficient than the usual way for searching apartments

1				5
		x		

I would prefer that tool over the usual way

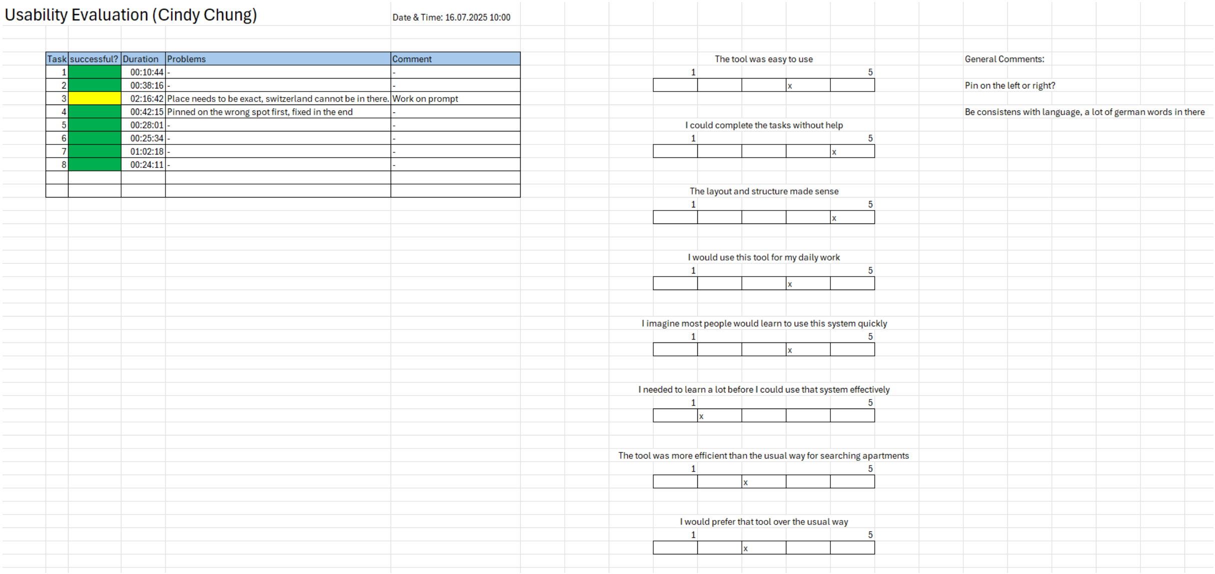
1				5
			x	

General Comments:

There is potential, still some unclear things.

Make sure what the input type should be

Figure A.15: Usability Testing Andrin Vogel



Usability Evaluation (Cindy Chung) Date & Time: 16.07.2025 10:00

Task	successful?	Duration	Problems	Comment
1	🟩	00:10:44	-	-
2	🟩	00:38:16	-	-
3	🟨	02:16:42	Place needs to be exact, switzerland cannot be in there.	Work on prompt
4	🟩	00:42:15	Pinned on the wrong spot first, fixed in the end	-
5	🟩	00:28:01	-	-
6	🟩	00:25:34	-	-
7	🟩	01:02:18	-	-
8	🟩	00:24:11	-	-

The tool was easy to use

1				5
			x	

I could complete the tasks without help

1				5
				x

The layout and structure made sense

1				5
				x

I would use this tool for my daily work

1				5
			x	

I imagine most people would learn to use this system quickly

1				5
			x	

I needed to learn a lot before I could use that system effectively

1				5
	x			

The tool was more efficient than the usual way for searching apartments

1				5
		x		

I would prefer that tool over the usual way

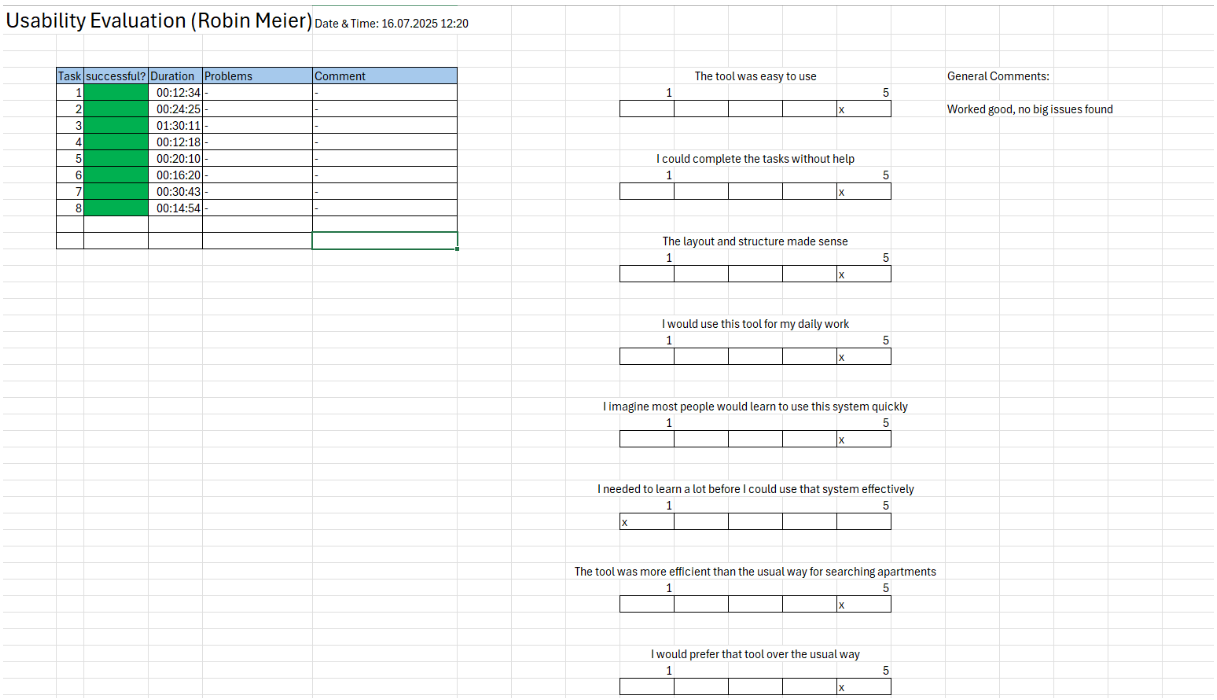
1				5
		x		

General Comments:

Pin on the left or right?

Be consistens with language, a lot of german words in there

Figure A.16: Usability Testing Cindy Chung



Usability Evaluation (Robin Meier) Date & Time: 16.07.2025 12:20

Task	successful?	Duration	Problems	Comment
1	🟩	00:12:34	-	-
2	🟩	00:24:25	-	-
3	🟩	01:30:11	-	-
4	🟩	00:12:18	-	-
5	🟩	00:20:10	-	-
6	🟩	00:16:20	-	-
7	🟩	00:30:43	-	-
8	🟩	00:14:54	-	-

The tool was easy to use

1				5
				x

I could complete the tasks without help

1				5
				x

The layout and structure made sense

1				5
				x

I would use this tool for my daily work

1				5
				x

I imagine most people would learn to use this system quickly

1				5
				x

I needed to learn a lot before I could use that system effectively

1				5
x				

The tool was more efficient than the usual way for searching apartments

1				5
				x

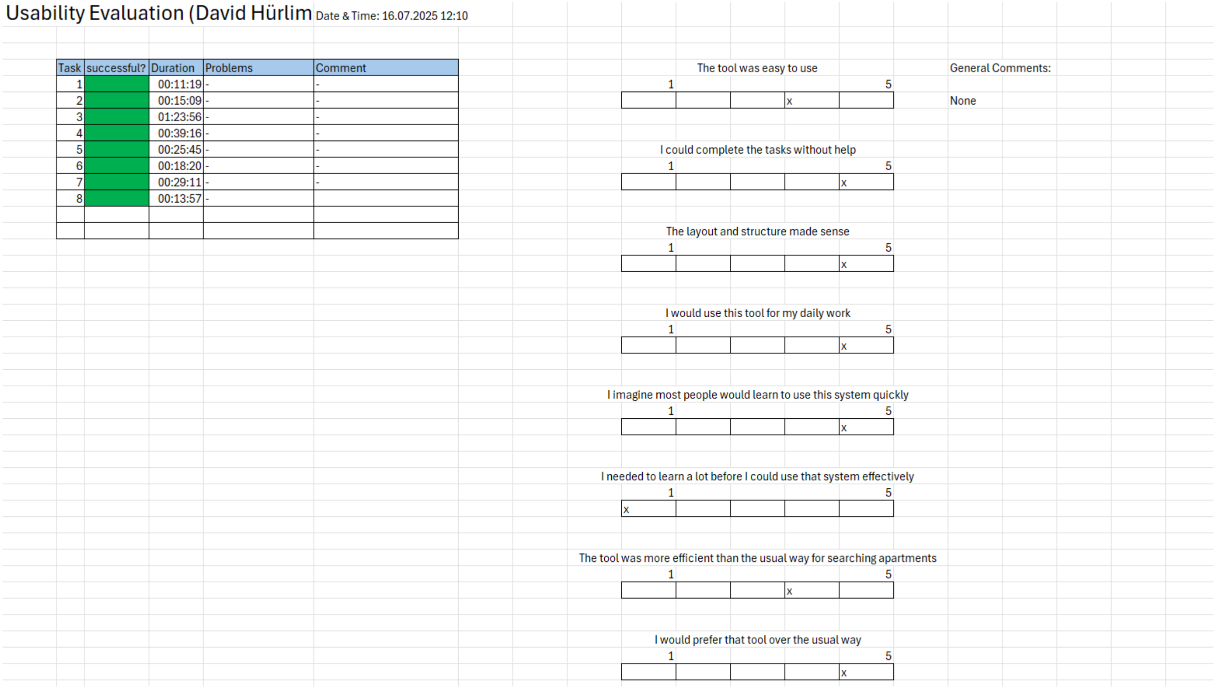
I would prefer that tool over the usual way

1				5
				x

General Comments:

Worked good, no big issues found

Figure A.17: Usability Testing Robin Meier



Usability Evaluation (David Hürlim Date & Time: 16.07.2025 12:10

Task	successful?	Duration	Problems	Comment
1	🟩	00:11:19	-	-
2	🟩	00:15:09	-	-
3	🟩	01:23:56	-	-
4	🟩	00:39:16	-	-
5	🟩	00:25:45	-	-
6	🟩	00:18:20	-	-
7	🟩	00:29:11	-	-
8	🟩	00:13:57	-	

The tool was easy to use

1				5
			x	

I could complete the tasks without help

1				5
				x

The layout and structure made sense

1				5
				x

I would use this tool for my daily work

1				5
				x

I imagine most people would learn to use this system quickly

1				5
				x

I needed to learn a lot before I could use that system effectively

1				5
x				

The tool was more efficient than the usual way for searching apartments

1				5
			x	

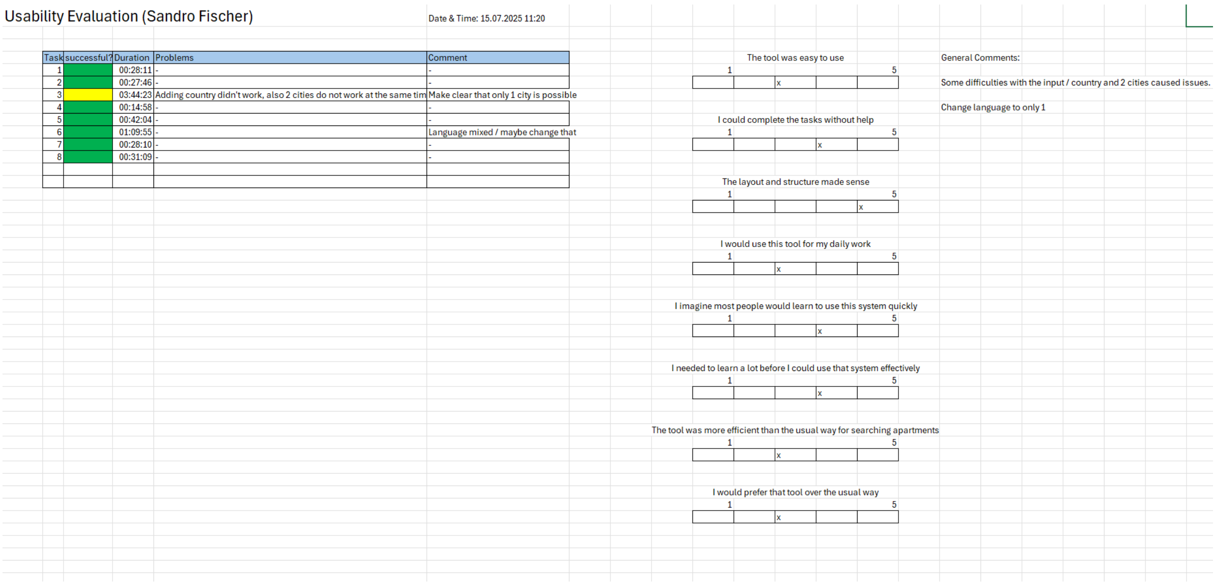
I would prefer that tool over the usual way

1				5
				x

General Comments:

None

Figure A.18: Usability Testing David Hürlimann



Usability Evaluation (Sandro Fischer) Date & Time: 15.07.2025 11:20

Task	successful?	Duration	Problems	Comment
1	🟩	00:28:11	-	-
2	🟩	00:27:46	-	-
3	🟨	03:44:23	Adding country didn't work, also 2 cities do not work at the same tim	Make clear that only 1 city is possible
4	🟩	00:14:58	-	-
5	🟩	00:42:04	-	-
6	🟩	01:09:55	-	Language mixed / maybe change that
7	🟩	00:28:10	-	-
8	🟩	00:31:09	-	-

The tool was easy to use

1				5
		x		

I could complete the tasks without help

1				5
			x	

The layout and structure made sense

1				5
				x

I would use this tool for my daily work

1				5
		x		

I imagine most people would learn to use this system quickly

1				5
			x	

I needed to learn a lot before I could use that system effectively

1				5
			x	

The tool was more efficient than the usual way for searching apartments

1				5
		x		

I would prefer that tool over the usual way

1				5
		x		

General Comments:

Some difficulties with the input / country and 2 cities caused issues.

Change language to only 1

Figure A.19: Usability Testing Sandro Fischer

A.0.5 Project agreement

Windisch, 19.03.25

Information on the project procedure & project agreement IP6 ' AI-based assistant for personalized relocation '

Supervisor: Kevin Kim
Nitish Patkar

Client: Byung Yun Cho

Project duration: 11.02.2025 until 14.08.2025

Task

1. Familiarization

1.1 Expectations for the project process

Dates
Fix appointments early, i.e., reviews with the customer and about every 2-3 weeks a meeting appointment with your supervisors. Clarify any absences right at the start of the project.

Meetings
Meetings are basically intended to discuss the current status of the project, clarify questions, discuss ideas and plan the next steps.
Send a list of agenda items and all other necessary documents to the supervisors. At the beginning of each project meeting, explain the current status of the project, the progress and problems as well as the planned steps.
You can use the meetings by arrangement and, if necessary, also for specific questions (e.B micro-teaching, brainstorming, presentation of results or mentoring). However, come to a meeting with as specific questions as possible.
Please record the discussed contents and decisions in a timely manner.

1.2 Specifications for the agreement

As a first task in your work you have to complete this agreement (cf. point 3). A first version should be produced by 2-4 weeks (BB 4-6 weeks) after kick-off. For projects that re-quire technical analysis, it may be useful to carry out a first implementation iteration before the sub-mission of the project agreement. Please complete the following items:

Initial situation
Formulate the project and the initial situation in your own words.

Project vision
Describe which goals and results are to be achieved with the project. The vision serves to derive quality criteria.

Project specific issues
In addition to the general questions, formulate 2-3 project-specific questions. These serve as a basis for scientifically structured research and the derivation of suitable solutions.

Examples of questions and solutions:

- Which approaches do you use to reach the defined target group?
  Solution approach: Development of concepts for user-centered approaches and implementation

of the user interface of the application, e.g., in the form of storyboards with a continuous user story or GUI prototypes.

- With which technical concept do you achieve the desired solution?
  Solution approach: Technology evaluation, development of technical solution concept (PoC), definition of subsystem decomposition, architectural style and technologies.
- Which interaction concepts, interface designs and visual languages are suitable for your approach?
  Solution approach: Development of interaction concepts and graphically carefully designed, clearly structured imagery for interface design, which meet the requirements of an innovative user experience.
- With which technical implementation do you meet the requirements for functionality, usability, reliability, efficiency and maintainability?
  Solution approach: Implementation of a executable application for a previously evaluated setup and defined usage scenario based on suitable technologies and frameworks
- Correctness, usability and reliability are central to the successful introduction of the software. How can you ensure and test them?
  Solution approach: In-depth testing of correctness, usability and reliability, documentation of Test results, demonstration of the fulfillment of the requirements by means of live test.

Methodology
Describe how the goals are achieved. Which methodologies do you use for this (e.g. Scrum, Agile, scientific approach, etc.).

Planning
Create an initial project schedule. Define work packages and their deliverables.

Risk Assessment
Identify and evaluate risks within the project and develop strategies for dealing with them.

2. Documentation

2.1 Written documentation (Thesis Rapport)

Document in writing and electronically your approach, the theoretical background, the application of methods and concepts, the implementations and test results. Also check the planned with the actual schedule, the achievement of goals and reflect on experiences.
Be sure to strictly separate personal comments from facts. **The main part of the documentation is completely fact-based**. This means that no sentences of the kind "Then we had the problem x and tried to solve it with y" are allowed to occur. But if such a problem x really exists and not only you did not get to the edge with it, then you should write: "Tests z have clearly shown that a problem x exists. Possible approaches to solve problem x are a, b and c. We chose variant c for reasons e and f." Only in an extra section can you formulate your personal impressions, experiences, problems and the like.
It is also important that a good documentation must still be read after many years and that it gives the reader a well-rounded picture, even if he was not directly involved in the work. Please also attach great importance to linguistic quality.
The target audience of this documentation are the supervisors, the experts, the client and future students who want to continue working in this area.
The documentation is created during the course of the project. For the second coaching meeting, a table of contents of the report should be prepared so that it can be discussed with the supervisors.
The parts for research and analysis are to be presented after the first third of the project.

On the web portal of the FHNW you create a project presentation (web summary). For bachelor theses in the spring semester, you will also create a poster for the exhibition. Both artifacts must be discussed with the supervisors prior to publication.

The following information must be mentioned on all publications:

- Logo FHNW
- Semester project IP5 or Bachelor thesis (IP6)
- Project name
- Spring- or Autumn Semester 202x, Degree Program Computer Science (Profiling iCompetence), University of Applied Sciences and Arts Northwestern Switzerland
- Submitted by: Name of Students
- Submitted to: Name of Supervisor
- Client: Company / Institution
- Date

Further information on writing reports can also be found on the Information Literacy Platform

2.2 Presentations

Presentations take place in consultation with the supervisors and the client. The expert will also be present when defending your bachelor's thesis.

On the one hand, presentations provide an overview of the entire project and the results achieved and deepen one or two important interesting questions. Also part of the presentation is a concise demonstration of how to use your software. With the audience, you can expect a technically experienced professional audience. Schedule 30' for the presentation and demonstration and reserve 30' for questions and discussion.

2.3 Publication of the project results

If the work or parts of the work are published, all names of the project participants (students, supervisors, clients) as well as the name of the institution (FHNW) must be mentioned. Before each publication, supervisors and clients must be asked for their consent in advance.

2.4 Protocols

Protocols are an important part of the documentation. Professionally managed protocols contain the following points:

- Date, Space, Time, Participants, Excused
- Agenda
- Project status (possibly with screenshots, sketches, etc.; Status according to planning)
- Content (fact-based, thematically structured and comprehensible in terms of content; Decisions are recorded)
- Open questions
- Next steps; Appointments & tasks (who, what & until when)

2.5 Document repository

Set up access to your document storage for the maintainers. If there are no compelling reasons against it, use the Gitlab infrastructure of the FHNW[1].
Also, use this document cabinet to store additional documentation, such as how to run your code. Make sure that an adequate commit history is visible to the caregivers.

2.6 Submission

The project submission includes (unless otherwise defined with the project manager) the following artifacts:

- Written documentation (Thesis Rapport)
- Project agreement (on the shelf as an appendix in the thesis)
- Codebase (documented & with readme to explain the setup), hosted on GitLab of the FHNW (https://gitlab.fhnw.ch/iit-projektschiene/[semester]/[project]) and as a ZIP archive

[1] https://gitlab.fhnw.ch/

- Link to the project appearance on the FHNW web portal
- other artifacts, if available (screencast recommended, ...)

3. Project-specific agreement

3.1 Initial position

Swissplatz is a mediation company that connects individuals looking to rent an apartment with platforms that offer rental listings. Throughout the entire process, the client has no direct contact with the landlord—all communication goes through the mediator. This is especially important as many clients come from abroad and are unfamiliar with the local language and rental laws.

The workflow consists of three main phases: Searching, Administration, and Viewing.

In the **Searching** phase, the client shares their preferences with the mediator, who then searches for suitable apartments and informs the client. This process is iterative until a suitable property is found. If necessary, the mediator also contacts landlords to clarify missing information in listings.

The **Administration** phase involves the exchange of information between clients and landlords via the mediator. Since landlords often require different documents and use their own forms, automation is challenging, leading to extensive back-and-forth communication.

In the **Viewing** phase, the mediator arranges the apartment viewing appointment.

Due to the high level of manual work required, these processes are time-consuming. Finding ways to optimize and partially automate certain steps could significantly improve efficiency.

3.2 Project vision

The goal of this project is to create an integrated platform that streamlines communication and automates key aspects of the rental mediation process. Currently, the client preferences are managed in an excel file, and mediators manually search for listings and manage communication. The new solution aims to centralize these processes on a single platform, improving efficiency and accessibility.

The platform will serve as a hub where clients can enter their housing preferences through an online form, while mediators manage rental offers in one place and upload them manually if needed. All interactions—such as feedback on listings, document requests, and appointment coordination—will take place within this platform, ensuring smooth and transparent communication between clients and mediators.

A key aspect of the project is automating the property search process. The system will gather listings from various sources, reducing the need for manual searching and allowing mediators to focus on high-value tasks. By integrating these functionalities, the platform will enhance scalability, improve user experience, and provide a structured yet flexible solution for rental mediation.

3.3 Questions

A. **What are the key challenges and best practices in retrieving and processing data from various platforms to enable automation in the rental mediation process?**
To address this question, a structured approach is taken. First, an analysis of existing data retrieval methods is conducted, examining industry standards such as APIs, web scraping, and data aggregation techniques. Various approaches used in real estate platforms are evaluated to understand their effectiveness and limitations. Additionally, existing automation solutions are reviewed to identify best practices for handling dynamic and structured data. Based on these findings, a concept is developed that enables efficient data collection while ensuring scalability and compliance with platform-specific restrictions.

B. **How can the user experience be optimized to simplify data input and interaction between clients and mediators and how is it perceived by users?**
To ensure a seamless and intuitive experience for both clients and mediators, a user-centered design approach is applied. First, the key pain points in the current workflow are identified through user research and process analysis. Best practices in usability testing, and prototyping are then reviewed to explore how an interactive platform can enhance communication and data management. Through iterative design and user evaluations, different input methods and interaction models are tested to develop an optimal solution that reduces complexity and improves efficiency for all users involved. Additional user tests will be done to ensure the validation of the platform.

C. **What are the possibilities of artificial intelligence that can be utilized to automate processes in the rental mediation workflow?**
To explore the role of AI in automating the rental mediation process, existing AI applications in real estate and process automation are analyzed. Key areas of focus include automated property matching based on client preferences, intelligent document handling, and chatbot-assisted communication. Various machine learning models are evaluated for their ability to optimize workflow efficiency while maintaining data security and accuracy. Based on these insights, a structured concept is developed that integrates AI-driven automation while ensuring transparency and user control over critical processes.

In addition to the project-specific questions, the following generic questions will be considered in the implementation of their work:

D. Identification of suitable scenarios and user interface prototyping: Which approaches do you use to reach the defined target group?

E. Technical concept: With which technical concept do you achieve the desired solution?

F. User Interface Design: Which interaction concepts, interface designs and visual languages are suitable for your approach?

G. Implementation: With which technical implementation do you meet the requirements for functionality, usability, reliability, efficiency and maintainability?

H. Testing: Correctness, usability and reliability are central to the successful introduction of the software. How can you ensure and test them?

3.4 Methodology

To ensure the successful implementation of the project, we rely on methodologies that cover the following key aspects:

Project Planning: Combination of Scrum and Kanban
For high-level project planning, Scrum is used to define clear phases, milestones, and dependencies, ensuring a structured workflow throughout the entire project timeline. This provides a solid framework while maintaining flexibility. Kanban complements this approach by enabling agile task management, allowing the team to adapt to changes as needed. Kanban boards are used to visualize progress continuously, ensuring transparency and improving responsiveness to new requirements or challenges.

Requirements Analysis (Requirements Engineering)
To define user needs, expectations, and technical requirements, an iterative process is applied. Regular meetings with clients and supervisors are held to gather and document requirements, serving as a foundation for development and helping to establish clear goals. The focus is on answering key questions that drive the project forward. Additionally, literature research and competitor analysis are conducted to derive industry-specific requirements for the product.

User-Centered Design
A user-centered approach is followed to ensure that the platform is intuitive, user-friendly, and functional. Iterative prototypes serve as the foundation for continuous usability testing, where feedback from potential users is gathered and integrated into the development process. This ensures that the final product meets user expectations and enhances overall usability.

Scientific Research and Literature Review
To address specific research questions and requirements, a thorough literature review is conducted. This includes analyzing scientific articles, technical reports, and industry best practices, as well as comparing competitor products. Additionally, discussions with industry experts provide valuable insights, helping to translate theoretical knowledge into practical solutions.

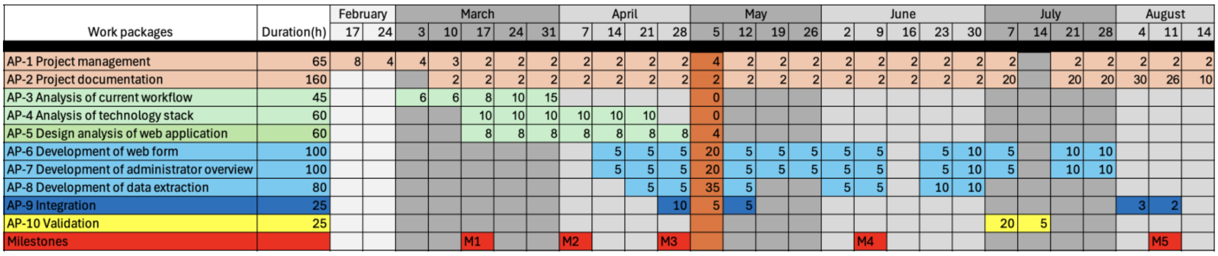
3.5 Planning

Nr.	Work package	Duration	Deliverable	Activities
AP-1	Project management	65h	Project agreement, Protocols, project planning,	Documentation, Project agreement, Planning, Meetings, Create epics and user stories
AP-2	Project documentation	160h	Documentation	Documentation
AP-3	Analysis of current workflow	45h	Protocols of results, internal report for documentation	Research, Comparison of competitors, Documentation, Evaluation
AP-4	Analysis of technology stack	60h	Protocols of results, Internal report for documentation, Proof of Concepts	Literature research, Comparison of competitors Documentation, Evaluation, Developing PoCs
AP-5	Design analysis of web application	60h	Prototypes, Personas, Internal report for documentation, User testing	Literature research, Comparison of competitors, User testing, Documentation, Evaluation, Prototyping, Creating personas
AP-6	Development of web form: replacing Excel workflow	100h	Web application, Documentation	Developing web form application, Testing, Evaluation, Documentation
AP-7	Development of administrator overview	100h	Web application, Documentation	Developing administrator overview, Testing, Evaluation, Documentation
AP-8	Development of data extraction and integration	80h	Web application, Documentation	Developing data extractor, Testing, Evaluation, Documentation
AP-9	Integration	25h	Web application, Documentation	Integrating new website into current solution
AP-10	Validation	25h	Documentation, Result of user testing	Documentation, Validation, Evaluation, User testing

Nr.	Milestone	Date
M-1	Project agreement signed	17.03.2025
M-2	Proof of Concept for data extraction	07.04.2025
M-3	Proof of Concept for integration of WIX with another website	28.04.2025
M-4	Development of web form finished	09.06.2025
M-5	Delivery of IP6	11.08.2025

Timetable

The following table shows when work is being done on which work packages. The milestones are marked in red. The schedule is not yet complete and will be continuously updated. The orange-marked fields (05.05) indicate the project week.



		February		March					April				May				June					July				August		
Work packages	Duration(h)	17	24	3	10	17	24	31	7	14	21	28	5	12	19	26	2	9	16	23	30	7	14	21	28	4	11	14
AP-1 Project management	65	8	4	4	3	2	2	2	2	2	2	2	4	2	2	2	2	2	2	2	2	2		2	2	2	2	2
AP-2 Project documentation	160				2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	20		20	20	30	26	10
AP-3 Analysis of current workflow	45			6	6	8	10	15					0															
AP-4 Analysis of technology stack	60					10	10	10	10	10	10		0															
AP-5 Design analysis of web application	60					8	8	8	8	8	8	8	4															
AP-6 Development of web form	100									5	5	5	20	5	5	5	5	5		5	10	5		10	10			
AP-7 Development of administrator overview	100									5	5	5	20	5	5	5	5	5		5	10	5		10	10			
AP-8 Development of data extraction	80										5	5	35	5			5	5		10	10							
AP-9 Integration	25											10	5	5												3	2	
AP-10 Validation	25																					20	5					
Milestones						M1			M2			M3						M4									M5	

3.6 Risk Assessment

Nr.	Risk	Description
R-01	Communication problems	Insufficient communication between the team members can lead to misunderstandings. Also, the customer and supervisors can be affected.
R-02	Exceeding Deadline for project	Exceeding the deadline due to insufficient time planning.
R-03	Absence of a team member	The absence of a team member can lead to limitations in progress and delays.
R-04	Changes of requirements	Additional requests from supervisors/customer may trigger changes in the schedule.
R-05	Issues with Scraping data / legal issues	If the data isn't available to us, the project may lead to more work.
R-06	Changes of data / scraping not working anymore	If the data we scrape is being changed, due to updates from the websites.
R-07	Connection to WIX-Website not possible	Our created website won't connect to the existing WIX-Website

Risk Evaluation

Probability of occurrence	very likely	🟧	🟥	🟥	🟥
	likely	🟧	🟧	🟥	🟥
	possible	🟩	🟧 R-06	🟧 R-05, R-02	🟥
	unlikely	🟩	🟩 R-04	🟧 R-01, R-07	🟧 R-03
	very unlikely	🟩	🟩	🟩	🟩
		very low	low	critical	very critical
		Extent of damage			

Risk Actions

Nr.	Risk	Actions
R-01	Communication problems	Regular meetings, clear protocols and use of communication platforms.
R-02	Exceeding Deadline for project	Creating a detailed project plan with milestones and regular reviews of resources and progress.
R-03	Absence of a team member	Regular exchange and uploading of work to shared repositories can soften this problem.
R-04	Changes of requirements	Open/honest communication and a clear definition of goals from the beginning prevent this problem.
R-05	Issues with scraping data / legal issues	Early testing and clarification with the provider can identify problems early.
R-06	Changes of data / scraping not working anymore	Using different sources to prevent missing data and building the scraping process so that it is easily adaptable.
R-07	Connection to WIX-Website not possible	Early testing in this case could identify problems early.

4. **Final provisions**
The undersigned acknowledge that they have read and understood the text and undertake to comply with the points listed and the general duty of care with their signature.

Windisch, 19.03.25

Supervisors

Kevin Kim ....

Nitish Patkar ....

Student

Damjan Stojanovic Stojanovic D.

Stefan Simic Stefan Simic