**Автономная некоммерческая организация высшего образования**
**«Университет Иннополис»**

**ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА**
**(БАКАЛАВРСКАЯ РАБОТА)**
**по направлению подготовки**
**09.03.01 - «Информатика и вычислительная техника»**

**GRADUATION THESIS**
**(BACHELOR'S GRADUATION THESIS)**
**Field of Study**
**09.03.01 – «Computer Science»**

**Направленность (профиль) образовательной программы**
**«Информатика и вычислительная техника»**
**Area of Specialization / Academic Program Title:**
**«Computer Science»**

| | |
|---|---|
| **Тема / Topic** | **Ориентированный на пользователя дизайн и разработка веб-платформы децентрализованной аренды для учреждений / User-Centered Design and Development of a Web-Based Peer-to-Peer Rental Platform for Institutions** |

| | | |
|---|---|---|
| Работу выполнил / Thesis is executed by | **Абделхади Асем Шавки Рамадан Мохамед / Asem Shawky Ramadan Mohamed Abdelhady,**<br>**Тотанжи Джаффар / Jaffar Totanji** | подпись / signature |
| Руководитель выпускной квалификационной работы / Supervisor of Graduation Thesis | **Маццара Мануэль / Manuel Mazzara** | подпись / signature |

Иннополис, Innopolis, 2024

# Contents

# List of Tables

# List of Figures

## Abstract

This work documents the development of a minimum viable product (MVP) for a community-based peer-to-peer rental platform. A survey of existing rental platforms revealed that none specifically catered to community-based restrictions on rentals. Additionally, these platforms suffered from performance limitations, or lack of adequate user engagement. In response, we developed an MVP specifically tailored to address these gaps, with a focus on community-centric features. The development process is thoroughly documented, encompassing the best practices used to optimize scalability and performance. While the MVP has been successfully built with excellent performance attributes, it has not yet been launched to the public, leaving the commercial viability of the business model untested.

# Chapter 1

# Introduction

In this work, we create a peer-to-peer rental platform that addresses the limitations of existing peer-to-peer rental solutions.

**What is the problem? Why is it important?**

Our concept is rooted in the observation that rentals are becoming increasingly significant, particularly within smaller communities. However, these activities often lack structured organization, monitoring, and centralization.

**What is known about the problem? What is not known about the problem?**

Existing peer-to-peer rental solutions suffer from multiple problems including poor performance and a lack of user engagement. Furthermore, the existing solutions do not provide the support and isolation that these communities need. Results of this analysis can be found in Appendix A.

**Our solution**

We will be creating an institution-based peer-to-peer (P2P) rental marketplace

that confines rentals to user-initiated communities and demonstrates high performance metrics.

**Overview of results achieved**

Our platform achieved high performance scores, maintained 100% availability, and ranked among the top 1% of websites worldwide when undergoing web benchmarks. Moreover, we provide a comprehensive account of the design and development phases, describing the technologies and methodologies employed to achieve said performance and a scalable architecture. We also address the performance issues of existing platforms by employing optimal technologies selected based on the latest research findings and industry trends.

**Research questions**

In the next chapter, we will find answers to the following research questions:

**RQ1: Which back-end framework should we use for the development of our application?**

**RQ2: Which front-end framework should we use for the development of our application?**

**RQ3: Which development methodology should we follow during the development of our application?**

# Chapter 2

# Literature Review

In this literature review, we examine the technologies fundamental to constructing a web application for a P2P rental marketplace. Front-end frameworks such as React, Angular, and Vue have redefined user experiences, adapting to the dynamic needs of modern users, while back-end technologies like Ruby on Rails, Node.js, Django, and Java Spring have strengthened the ability of platforms to process complex transactions and manage data efficiently. Moreover, the implementation of development methodologies, most notably Agile, SCRUM, and Lean, has had a profound impact on the way digital products are brought to life, ensuring they are responsive to user needs and market demands.

This chapter navigates through the body of knowledge surrounding the technical and methodological aspects of constructing a web application for a P2P rental marketplace. Further in this chapter, we dissect the array of back-end frameworks in Section 2.1, followed by front-end frameworks in Section 2.2. The review then explores different development methodologies in Section 2.3. Lastly, we summarize our choices in Section 2.4.

## 2.1 Back-end Technologies

This section seeks to answer the question:

**RQ1: Which back-end framework should we use for the development of our application?**

Our choice is based on a thorough examination of peer-reviewed scholarly articles that evaluate leading frameworks, complemented by insights from the industry. This approach guarantees that our selection is consistent with the prevailing industry norms and technological progress.

### 2.1.1 Search Process

The search for relevant literature was performed across several prominent academic databases: Google Scholar, IEEE Xplore, and ScienceDirect.

Table 2.1 presents the structured search terms, operators, and filters employed in our investigation.

TABLE 2.1
Back-end Search Query Details

| Concept Grouping | Search Terms | Search Operator | Filters |
|---|---|---|---|
| Back-end Web development | Web, Back end | OR | Past 5 years |
| Frameworks | Frameworks, Technologies | OR | Past 5 years |
| Analysis Type | Comparative, Comparison, Analysis | OR | Past 5 years |

The query formulated for this search was:

"Web" OR "back end" "frameworks" OR "technologies" "comparative" OR "comparison" OR "analysis"

The initial search yielded six relevant papers, summarized in Table 2.2 which details the distribution of papers across the databases.

TABLE 2.2
Back-end Search Results

| Database | Number of papers |
|---|---|
| Google Scholar | 2 |
| ScienceDirect | 1 |
| IEEE Xplore | 3 |

The findings were then reviewed and evaluated in-depth. Despite being unable to access one paper from IEEE, the remaining five papers underwent thorough analysis.

### 2.1.2 Search Results

The investigation began with a comprehensive search aimed at identifying key back-end frameworks that could be suitable for the development of our application. We explored a variety of frameworks, including Django, Spring, Ruby on Rails, Node.js (Express), and Laravel. This search was enriched by a structured comparative analysis, referencing foundational methodological approaches from scholarly sources and integrating current industry insights[1].

The methodology employed encompassed an extensive set of evaluative criteria of 14 primary benchmarks along with an additional 10 specific criteriadesigned to provide a detailed comparison of the frameworks. This rigorous analysis leveraged data from various sources, including technical forums, code repositories, professional networking platforms, and direct user testimonials.

[1]https://hackr.io/blog/web-development-frameworks

### 2.1.3 Result Analysis

Within the analytical framework, several frameworks were highlighted for their distinctive capabilities and alignment with our project requirements:

- **Spring:** Recognized for its robust performance and adaptability to contemporary business trends, Spring benefits from the highest developer availability with 530 211 LinkedIn profiles listing it as a skill. It boasts a strong project health with approximately 144 334 tags on StackOverflow and extensive resources on its official site [2]. In terms of job market presence, Spring leads significantly with 4 619 job positions listed, reflecting its dominant stance in the industry [1].

- **Django:** Django is notable within the back-end landscape for its comprehensive feature set that supports robust web application development. The framework shows healthy community engagement with about 212 681 developers listing it on LinkedIn and 186 288 tags on StackOverflow. It also has a solid presence in the job market with 353 job opportunities listed, and it is supported by extensive documentation and learning resources available on its official site [3] [1].

- **Node.js:** A non-blocking, event-driven I/O platform built on Google Chromes V8 engine. Node.js is designed around an asynchronous I/O and event-driven architecture, optimizing for high concurrency. Unlike traditional multi-threaded server architectures, Node.js operates on a single-threaded model with an event loop, effectively handling over 40 000 concurrent user connections on a server with only 8GB of memory[2].

[2] https://spring.io
[3] https://docs.djangoproject.com

- **Laravel:** Known for its extensive package ecosystem and ease of use, Laravel has 192 680 developers listing it on LinkedIn and features about 104 327 tags on StackOverflow. It offers 152 job positions, indicating a steady but smaller presence in the job market compared to Django and Spring. The Laravel community is vibrant, with a wealth of resources available on Laracasts [4] and official documentation that makes it particularly accessible to newcomers [1].

From the analysis of the initial papers, it was determined that Spring is a suitable choice for our requirements. Consequently, we sought a relevant use case for Spring within the same search query to back-up this selection.

### 2.1.4 Case Study Review 1

A study evaluating the implementation of Spring Boot and MySQL within an energy management system demonstrated their integration and performance in a practical setting [3]. This review assesses their application in the specific context of a manufacturing enterprise.

**System Architecture and Framework Utilization:** The system architecture utilized a three-tier approach: data collection, business logic, and user interface, featuring:

- **Spring Boot:** Positioned at the heart of the business logic layer, facilitating easier deployment and scalability. Its toolset supports rapid development and complex functionalities.

[4] https://laracasts.com

- **MySQL:** Employed as the data storage solution, enhancing performance and scalability through strategic data distribution.

**Database Design and Optimization:** The database was optimized using horizontal partitioning, with a well-organized structure to support efficient data management.

In the previously examined use case, MySQL was utilized as the database system. Prior to finalizing our decision to proceed with MySQL, we conducted an analysis of a study that compared its performance with another database system within a similar use case. This comparison was essential to determine the most appropriate database technology for our needs.

### 2.1.5 Case Study Review 2

A comparative study of MongoDB and performance of MySQL in cloud environments provided insights into the efficiency of these database systems under varying operational conditions [4]. The focus was on latency, database size, and resource efficiency.

**Findings and Implications:**

- **Performance Metrics:** Analysis focused on data handling and resource efficiency, with MySQL showing advantages in certain scenarios.
- **Predictive Modeling:** Proposed models aid in database selection, optimizing performance based on specific cloud configurations and data volumes.

**Conclusion of Case Study Review:** The findings from this study influenced the decision to utilize MySQL, affirming its suitability for structured data management in cloud settings and providing a foundation for future resource planning.

Based on the outcomes of the comparative analysis and the insights derived from the use cases, we opted for Spring Boot as the framework for the back end and MySQL as the database system. Given the extensive range of technologies incorporated within Spring Boot, further investigation was necessary. Consequently, we consulted additional research, specifically a study on database bridging within Spring Boot [5], to ensure a comprehensive understanding and integration strategy. The insights from the paper guided us to go with the JDBC connection because of the following reasons:

- **Moderate Performance:** The combination of Spring Framework 4.0.1 with JDBC shows moderate performance [5]. This level of performance is neither negative nor positive, suggesting a reliable and stable choice for back-end applications where consistent performance is key.
- **Alignment with Business Trends and Developer Availability:** As identified in [1], the Spring Framework aligns well with contemporary business trends and developer availability. This alignment is crucial for our application, as it ensures that our back-end is built on a framework that is both current in its approach and supported by a robust community of developers .
- **Stability Across Various Scenarios:** The Spring and JDBC combination maintains a steady performance across different scenarios, making it suitable for our application that may encounter diverse workloads and requirements [5].

- **Flexibility and Ease of Use:** JDBC, as a standard Java API for database connectivity, provides a straightforward approach to database interaction. When integrated with Spring, this combination allows for ease of configuration and flexibility, beneficial for developers seeking an efficient solution for database operations.

- **Suitability for a Range of Applications:** The balanced performance of Spring with JDBC makes this combination versatile, suitable for various back-end systems, especially where complex ORM tools may not be required.

After a comprehensive review of various technologies and frameworks, we decided to implement Spring Boot as our back-end framework, complemented by JDBC for database connectivity and MySQL as the database system. This decision was influenced by the moderate and reliable performance of the Spring and JDBC combination, which maintains a steady performance across different scenarios [5]. Spring Boot, with its strong project health as mentioned in 2.1.3, aligns well with current business trends and developer availability [1]. The integration of JDBC offers ease of use and operational flexibility, critical for managing the diverse workload of our application efficiently.

## 2.2 Front-end Technologies

This section aims to answer the question:

**RQ2: Which front-end framework should we use for the development of our application?**

This decision was informed by a comprehensive review of peer-reviewed

literature comparing the prominent frameworks, as well as industry insights. This ensured that the selected framework aligns with current industry standards and technological advancements.

### 2.2.1 Search Process

The literature search was conducted across three major databases: Google Scholar, the ACM Digital Library (ADL), and IEEE Xplore.

Table 2.3 describes the terms, operators, and filters used for the search.

TABLE 2.3
Front-end Search Query Details

| Concept Grouping | Search Terms | Search Operator | Filters |
|---|---|---|---|
| Web development | Web, Front end | OR | Past 6 years |
| Frameworks | Frameworks, Libraries | OR | Past 6 years |
| Analysis Type | Comparative, Comparison, Analysis | OR | Past 6 years |

The following query was used to search the Google Scholar database:

Web OR "front end" "frameworks" OR "libraries" "comparative" OR "comparison" OR "analysis"

The same query was adapted to search IEEE Xplore and ADL adhering to their search process and inputs.

Our initial search yielded five papers. Table 2.4 describes the distribution.

TABLE 2.4
Front-end Search Results

| Database | Number of papers |
|---|---|
| Google Scholar | 2 |
| ADL | 1 |
| IEEE Xplore | 2 |

We were able to access all except one of these papers. After a careful review and analysis of the remaining four, three were selected as significantly influential to our decision-making process, based on relevance and depth of information.

### 2.2.2 Search Results

The investigation highlighted three main frameworks commonly adopted in the industry: React.js, Vue.js, and Angular. This was further supported by the industry insights in the latest comprehensive annual developer survey - State of JavaScript[5]

After identifying these frameworks, we extended our search with more keywords (React, Angular, Vue) in order to find detailed information about each of these technologies. Our second search yielded another four sources, totalling seven peer-reviewed sources informing this decision.

### 2.2.3 Result Analysis

Our analysis first involved taking a closer look at each technology. We conducted further research, and found the following:

- React.js: Initially met with skepticism due to its novelty, React (launched by Facebook engineer Jordan Walke) gained traction through open-sourcing in 2013 and early adoption by Netflix. The focus of React on core User Interface (UI) functionality resonated with developers. Key milestones include the performance-enhancing React Fiber (2017) and state management improvements with Hooks (2019). The emphasis React places on efficient

[5]https://2022.stateofjs.com/en-US/libraries/front-end-frameworks/

rendering, re-usability of components, and a vibrant ecosystem makes it a solid option for developing modern web applications [6].

- Vue.js: A JavaScript framework released in early 2014 by Evan You, has seen explosive growth, ranking 8th most starred on GitHub and downloaded 40 000 times daily from npm. Its focus on developer experience is evident in its features: a powerful templating syntax, data reactivity for automatic Document Object Model (DOM) updates, and streamlined data manipulation [7].

- Angular: A development platform built by Google that utilizes TypeScript for scalable web application creation. Released in 2016, it superseded its predecessor AngularJS. A key advantage of Angular is its cross-platform compatibility achieved through integrations like Angular Universal for server-side rendering and Ionic Framework for mobile app development. The framework offers extensive tooling to streamline development, including the Angular Command-Line Interface (CLI) for project management and Angular DevTools for debugging. Additionally, Angular comes with a rich set of pre-built libraries for common tasks such as Application Programming Interface (API) communication, form handling, and routing. Widespread adoption by Google and a large developer community further solidify the position of Angular as a prominent web development framework [8].

Based on the insights from the comparative studies, as well as the detailed analysis of each framework, we first excluded Angular from our choices. This decision was influenced by the fact that we are developing a small to medium-scale Single Page Application (SPA). Literature and industry reports suggest that

Angular, while robust for large-scale applications due to its comprehensive tooling and features, can be too cumbersome for smaller projects [9], [10].

Among the remaining options, we chose React for several reasons. The design of React prioritizes quick rendering and flexible component management, which is ideal for fast-paced development of SPAs. The framework has consistently received strong support from a large community of developers, which guarantees an abundance of collective knowledge and troubleshooting resources, as well as a rich ecosystem of plugins and tools. [11].

We decided to pair Next.JS with React in order to leverage the additional benefits that Next.JS offers, including the support for dynamic routing, API management, and Server-Side Rendering (SSR). SSR can significantly improve the initial loading time and Search Engine Optimization (SEO) of our application, while dynamic routing provides a more flexible way to manage content and navigation flows. These features of Next.JS complement the capabilities of React and align with our goal of building a responsive, user-friendly SPA [12].

This comprehensive approach, grounded in our research and aligned with industry trends, positions us to develop a robust front-end architecture that meets our specific needs and expectations.

## 2.3 Development Methodologies

This section focuses on answering the question:

**RQ3: Which development methodology should we follow during the development of our application?**

The decision is grounded in a thorough review of peer-reviewed literature on modern technologies. By doing so, we ensure that the chosen methodology is

consistent with current industry standards.

### 2.3.1 Search Process and Results

The literature search was carried out in two databases: Google Scholar and IEEE Xplore.

Table 2.5 describes the terms, operators, and filters used for the search.

TABLE 2.5
Development Methodologies Search Query Details

| Concept Grouping | Search Terms | Search Operator | Filters |
|---|---|---|---|
| Software development methodologies | Development methodologies, Agile methodologies, Traditional methodologies | OR | Past 15 years |
| Analysis Type | Comparative, Comparison, Analysis | OR | Past 15 years |

Four relevant papers were extracted from the search. Table 2.6 describes the distribution.

TABLE 2.6
Development Methodologies Search Results

| Database | Number of papers |
|---|---|
| Google Scholar | 2 |
| IEEE Xplore | 2 |

### 2.3.2 Result Analysis

The investigation highlighted two main categories of methodologies: Agile and Traditional.

- Agile Methodologies: Agile methodologies have garnered widespread preference in the software development industry, primarily due to their adaptability to rapidly evolving requirements. Unlike traditional methodologies, which often exhibit rigidity and slower response times, Agile approaches provide a more lightweight and flexible framework. This agility enables developers to accelerate the development process and respond to change efficiently. The emphasis of Agile on early and continuous delivery of valuable software, along with its focus on customer needs, sets it apart from its more plan-driven counterparts. These attributes make Agile especially suitable in the current dynamic software landscape, where change is the only constant [13], [14].

  - Feature-Driven Development (FDD): Known for its adaptability to late changes and its capability to deliver high-quality outputs in each development phase. However, the approach of FDD offers no extensive guidance on requirement analysis and risk management, which can be critical in complex projects [13].
  - Test-Driven Development (TDD): Emphasizes enhancing code quality through iterative testing, reducing development complexity by focusing on individual problems. TDD poses challenges in documentation and requires developers to possess skills in test case writing, which is not typically within their primary skill set [13].
  - Dynamic System Development Method (DSDM): Advocates for rapid application development, drawing best practices from various methodologies. Despite its benefits in quick development cycles, DSDM involves a complex structure with numerous roles, potentially com-

plicating project administration [13].

  - SCRUM: Stands out for its simplicity and focus on managing software projects. It fosters transparency and self-organization within teams. Nonetheless, the general approach of SCRUM specifies no particular practices or technical methods, which could lead to ambiguities in implementation [13].
  - Extreme Programming (XP): Targets incremental development with frequent system releases and focuses on maintaining simplicity in code. However, the reliance of XP on informal documentation and its less effective application in distributed team settings present notable limitations [13].

- Traditional Methodologies: Traditional methodologies such as Waterfall, Iterative, and Spiral models play a significant role in software development.

  - Waterfall Model: Appreciated for its straightforwardness and clear structure, making it ideal for projects with well-defined requirements. However, its rigidity and lack of flexibility in handling changes pose significant drawbacks [15].
  - Iterative Waterfall Model: Improves upon the traditional Waterfall model by allowing feedback and corrections at each stage, but still shares some of the limitations of the original model [15].
  - Spiral Model: Known for its risk management capabilities, making it particularly suited for large, mission-critical projects despite its complexity and cost [16].

Comparing the Agile methodologies and the traditional ones, modern methodologies such as Agile emphasize adaptability and rapid response to change. Agile methodologies, like SCRUM and XP, promote iterative development and frequent releases, fostering better collaboration and quicker adaptation to new requirements. These methodologies have become increasingly popular due to their ability to produce high-quality software in shorter timeframes, meeting the dynamic needs of current software industry [16].

Each Agile methodology, therefore, brings its unique set of advantages and constraints, reflecting the diverse needs and contexts of software development projects in the modern era.

### 2.3.3 Conclusion of the Analysis

After careful consideration, we chose SCRUM as our software development methodology because it is a concrete implementation of an agile framework that delivers the highest value in the shortest time. SCRUM is team-oriented, with distinct roles and short, time-boxed iterations called sprints, where the system is incrementally developed and produces different artifacts to coordinate work. The simplicity of SCRUM and its focus on software management issues, rather than technical development practices, make it widely applicable to any domain [13].

SCRUM is preferred for its simplicity and effectiveness in managing software development projects, emphasizing iterative progress through sprints, fostering transparency, and accountability within teams [17], [18]. The structure of SCRUM, including roles like SCRUM Master and Product Owner, helps streamline project management and enhance team collaboration [19], [20]. However,

challenges include potential ambiguity in practices and the need for team members to be well-versed in SCRUM principles [21]. While SCRUM benefits distributed software development environments, effective communication tools and practices are required to mitigate coordination challenges. Overall, the adaptability and focus on team empowerment within SCRUM make it a robust choice for modern software development projects [22].

According to [13] and [23], the SCRUM lifecycle involves three main phases:

1. **Pre-sprint Phase**: This initial planning phase outlines the general objectives, defines the project team, and determines the required tools and resources. It generates a Product Backlog that documents customer requirements as user stories and features, prioritized and estimated by the product owner.

2. **Sprint Phase**: This development phase consists of a series of iterative sprint cycles, each lasting 2-4 weeks. Each sprint starts with a planning meeting to agree on the work for the next sprint. The team implements and tests features, holding daily SCRUM meetings to enhance communication and focus. The sprint ends with a review meeting to inspect and assess the output and a retrospective meeting to discuss improvements.

3. **Project Closure Phase**: This final phase occurs when the requirements are met, and the goals are achieved. The latest product version is ready for release and distribution, along with complete documentation and user manuals.

## 2.4 Outcome

### 2.4.1 Back-end Decisions

**Answer to RQ1:** From Section 2.1, it was determined that Spring Boot, coupled with a JDBC connection and MySQL, would be utilized for the development of the back-end of the marketplace.

### 2.4.2 Front-end Decisions

**Answer to RQ2:** Based on insights from Section 2.2 Next.js is a strong choice for the front-end development of our peer-to-peer rental marketplace.

### 2.4.3 Development Methodology Decisions

**Answer to RQ3:** From the results of 2.3, SCRUM is our choice as a software development methodology.

# Chapter 3

# System Design and Implementation

## 3.1 Development Methodology

In an effort to efficiently navigate the complexities of product development and in alignment with our Scrum methodology, our team resolved to create an MVP encompassing essential functionalities. Through iterative cycles within the MVP development process, we aimed to refine the product to better align with expected user needs and preferences. This approach was rooted in agility and responsiveness, key factors in nowadays dynamic market.

To achieve this, we set a timeline of 4 months to finalize our MVP. This time-frame was structured around bi-weekly sprints, a method that enabled us to maintain a steady pace and clear focus. Each sprint ended with a retrospective meeting, allowing us to assess our progress and address any challenges. To further streamline our process and enhance collaboration, we also had sprint planning meetings. These meetings were vital in ensuring synchronicity between the front-

end and back-end teams. Emphasizing our commitment to a collaborative and empowered team structure, each member assumed the role of a product owner for their respective service - be it front-end or back-end. This structure not only fostered ownership and accountability but also ensured that each aspect of our product received the dedicated focus it deserved.

To manage our product development process effectively, we decided to use ClickUp[1], a versatile project management tool. This platform enabled us to organize tasks, track progress, and maintain clear communication across the team.

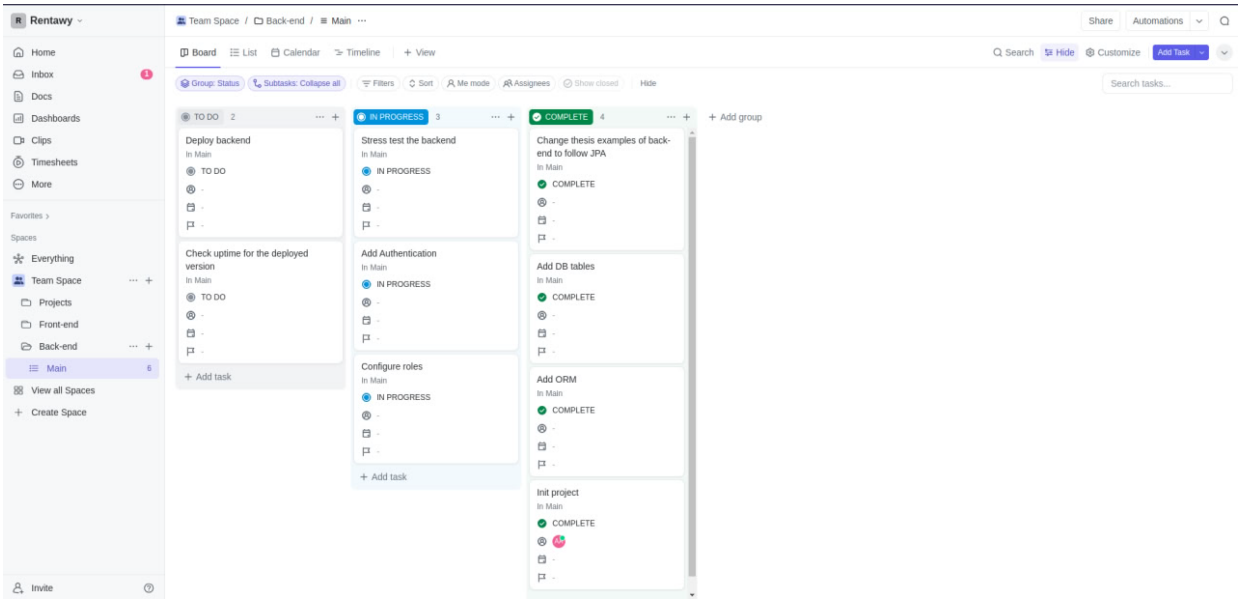


Fig. 3.1. Clickup back-end board

## 3.2 App Mock-ups

In the development of our rental application, we created mock-ups that illustrate the key functionalities it offers.

[1]https://clickup.com/


Fig. 3.2. Mock-up of Listings page, 'All' tab

Fig. 3.2 shows the functionality where users can browse through a list of all available listings. This feature allows users to view all the key information about a listing, then quickly rent an item by tapping the 'Rent' button, streamlining the initiation of rental transactions.

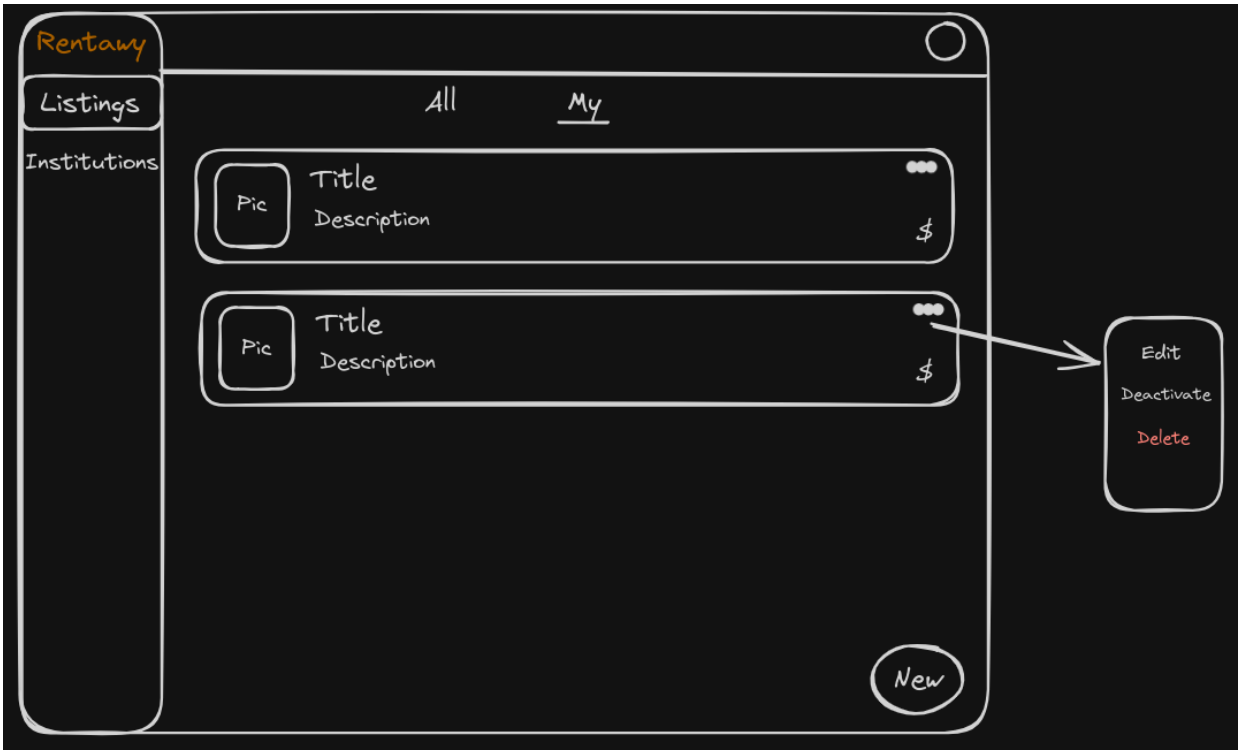


Fig. 3.3. Mock-up of Listings page, 'My' tab

Fig. 3.3 shows the personalized listing management feature. It provides users with the capability to oversee their individual rental listings, offering the flexibility to add new items or edit existing ones. This allows users to maintain and control their rental offerings within the application.


Fig. 3.4. Mock-up of Institutions page, 'All' tab

Fig. 3.4 illustrates the Institutions section of the app, which allows users to view the essential information about an institution. The 'Join' button simplifies the integration into these communities.

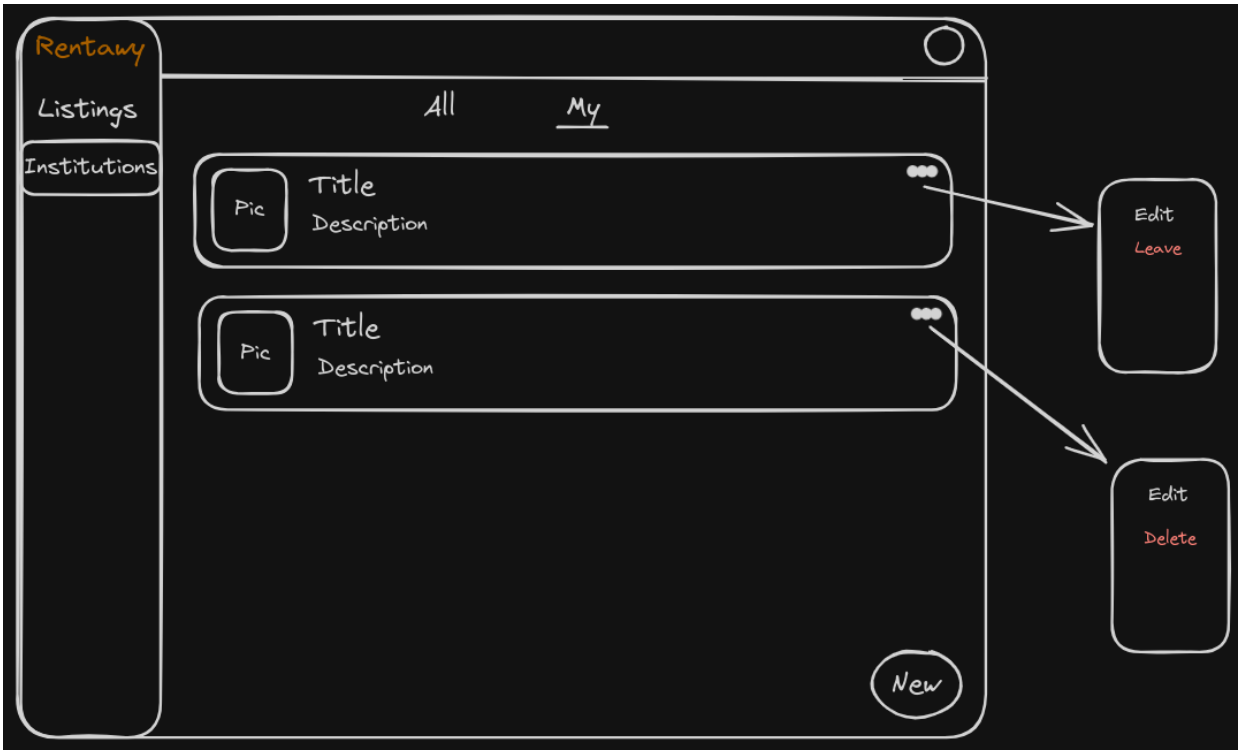


Fig. 3.5. Mock-up of Institutions page, 'My' tab

Fig. 3.5 displays the options for users to manage their institutions or their involvement with one. It provides the tools to edit institutional details or to opt-out, giving users the autonomy to manage their participation according to their preferences.

In conclusion, the mock-ups serve as a blueprint for the fundamental functionalities of our rental application. Each figure is a testament to our commitment to providing a user-friendly interface that simplifies the rental process while empowering users to manage their activities with ease and confidence.

## 3.3 Back-end

### 3.3.1 Project Structure

The design of the back-end commences with an illustration of the project structure, focusing specifically on user-related components. This structure, depicted in Listing 3.1, adheres to a layered architecture, centralizing functionalities around the user entity.

**Listing 3.1:** Project Directory Structure

```
src
|-- main
|   |-- java
|   |   `-- com
|   |       `-- rentawy
|   |           |-- Application.java
|   |           |-- config
|   |           |   `-- SecurityConfig.java
|   |           |-- controller
|   |           |   `-- InstituteController.java
|   |           |-- service
|   |           |   `-- InstituteService.java
|   |           |-- DAO
|   |           |   `-- InstituteDAO.java
|   |           |-- model
|   |           |   `-- Institute.java
|   |           |-- dto
|   |           |   `-- InstituteDTO.java
|   |           `-- exception
|   |               `-- InstituteNotFoundException.java
|   `-- resources
|       |-- application.properties
|       |-- static
|       `-- templates
`-- test
    `-- java
        `-- com
            `-- rentawy
                |-- InstituteControllerTest.java
                `-- InstituteServiceTest.java
```

### Application Entry Point

The Application.java file serves as the entry point, initiating the Spring Boot application and running the embedded Tomcat server.

### Layered Architecture

The structure is divided into three distinct layers which are Control layer, Service layer and Repository layer.

**Controller Layer** The InstituteController.java demonstrates the handling of CRUD operations for user management through RESTful endpoints.

**Listing 3.2:** InstituteController.java

```
package com.rentawybackend.controller;
...

@RestController
@RequestMapping("/institutes")
public class InstituteController {

  private InstituteServiceI instituteService;

    @Autowired
    public InstituteController(InstituteServiceI instituteService) {
        this.instituteService = instituteService;
    }
    ...
}
```

**Service Layer** The InstituteService.java illustrates the encapsulation of business logic and abstraction of data access.

**Listing 3.3:** InstituteService.java

```
package com.rentawybackend.service;
...

@Service
public class InstituteService implements InstituteServiceI {

    private InstituteDAOI instituteDAO;

    @Autowired
    public InstituteService(InstituteDAOI instituteDAO) {
        this.instituteDAO = instituteDAO;
    }
...
}
```

**Repository Layer** The repository layer, exemplified by InstituteDAO.java, is pivotal in managing database interactions, particularly in establishing connections, executing SQL queries, and mapping results to objects.

**Listing 3.4:** InstituteDAO.java

```
package com.rentawybackend.DAO;
...

@Repository
public class InstituteDAO implements InstituteDAOI {
    private EntityManager entityManager;
```

```
    @Autowired
    public InstituteDAO(EntityManager entityManager) {
        this.entityManager = entityManager;
    }
...
}
```

The flexibility of Spring Boot with JDBC allows for seamless integration of various databases, as depicted in Fig. 3.6.

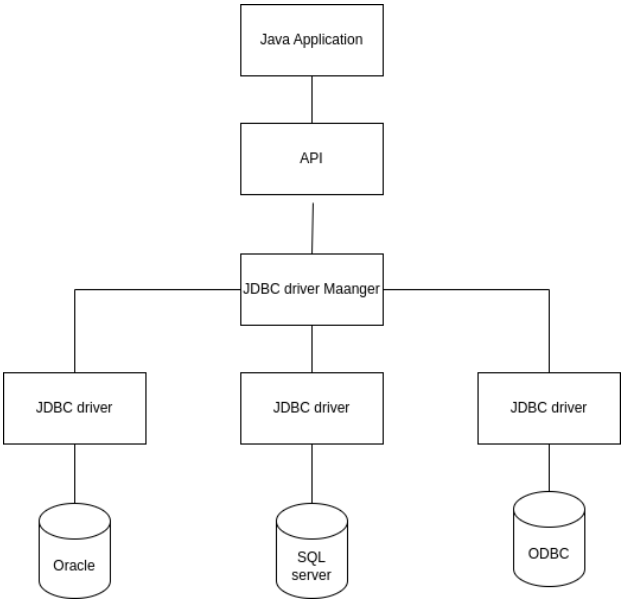


Fig. 3.6. Database integration in Spring Boot with JDBC

**Auxiliary Components of the Project Structure** The project structure also encompasses several additional folders, each serving a distinct role in the functionality of the application:

- **Model:** The Model directory comprises Plain Old Java Objects (POJOs), representing the domain model of the application. These classes generally reflect the data structures and are frequently mapped to database tables,

particularly in the context of Object-Relational Mapping (ORM), where they are typically annotated as entities.

- **Configuration and Security:** The Configuration (Config) folder predominantly houses classes responsible for various application configurations. Pertinent to security, this directory includes one or more classes dedicated to establishing security protocols within the application. These configurations might entail setting up authentication and authorization mechanisms, security filters, and defining access to protected resources.

- **Data Transfer Object (DTO):** Contained within the DTO folder are classes specifically designed for the transfer of data between different processes, especially the server and the client. These Data Transfer Objects play a crucial role in the controller layer, where they are utilized to receive incoming data and send responses, effectively decoupling the external interface of the API from the internal domain model.

- **Exception:** This folder is designated for classes that define custom exceptions. These exceptions are closely tied to the business logic of the application and are instrumental in handling specialized error scenarios, such as when a resource is not found, or a specific operation is not feasible due to established business rules.

The process flow for an API request, particularly for adding a user, is illustrated in the diagram below. Flowchart in Fig. 3.7 delineates how the request traverses through the different architectural layers of the application:

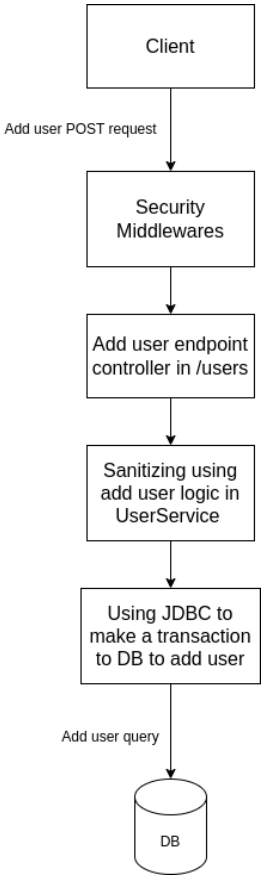


Fig. 3.7. Process flow of an API request for adding a User

## 3.4 Front-end

Our front-end implementation leveraged the versatility and efficiency of Next.js [2] and React [3], complemented by the robust design principles of Material UI [4]. These choice of these three main building blocks ensured a seamless development process as they are highly compatible with one another.

### 3.4.1 Project Structure

The project adheres to the standard Next.js structure, embracing the latest "app" directory pattern, with the optional "src" folder containing the source code of the project and separating it from the external configuration files. The source code was structured according to the principles of Feature-Oriented Software Development (FOSD), which is a paradigm that emphasizes the decomposition of a software system in terms of the features it provides. FOSD aims to construct well-structured software that can be tailored to the needs of the user and application scenario, offering significant benefits in terms of modularity, reusability, and variation. This approach ensures a modular and organized codebase, facilitating ease of maintenance and scalability. Additionally, this paradigm aids in maintaining a clean mapping between the representations of features across all phases of the software life cycle, ensuring that the software system can easily evolve and adapt to new requirements or feature changes [24]. Next, we list the project structure and explain its different functions.

**Listing 3.5:** Frontend Project Directory Structure

[2] https://nextjs.org/docs
[3] https://react.dev/
[4] https://mui.com/

```
rentawy-front
|-- src
|   |-- app
|   |   |-- institutions
|   |   |   `-- page.tsx
|   |   |-- offers
|   |   |   `-- page.tsx
|   |   |-- layout.tsx
|   |   |-- page.tsx
|   |   `-- globals.css
|   |-- components
|   |   |-- app-mainbar
|   |   |   `-- app-mainbar.tsx
|   |   |-- app-sidebar
|   |   |   |-- app-sidebar.tsx
|   |   |   `-- sidebar-menu.tsx
|   |   |-- offer-card
|   |   |   `-- offer-card.tsx
|   |   `-- profile-button
|   |       `-- profile-button.tsx
|   |-- features
|   `-- lib
|       |-- services
|       |-- theme
|       |   `-- theme.ts
|       `-- types
|           |-- index.ts
|           |-- institution.ts
|           `-- offer.ts
|       `-- utils
```

- **app**: Managed by Next.js, this directory encompasses the core functionality of our application. Each folder within "app" represents a distinct page, housing a page.tsx file containing the corresponding code of the page. Additionally, an optional layout.tsx file is included, which defines the layout of the page. Next.js allows each page to inherit the layout of its parent. This hierarchical structure enhances code reusability and simplifies the management of layouts throughout the application.

- **components**: This directory houses "dumb components,". [5] These components are stateless and primarily responsible for displaying data. Designed to be reusable across various sections of the application, they contribute to code modularity and maintainability.

- **features**: "A feature is a unit of functionality of a software system that satisfies a requirement, represents a design decision, and provides a potential configuration option" [24]. The "features" directory houses the independent features of our application. These features encapsulate specific functionality, promoting code reusability and facilitating modular development.

- **lib**: This directory hosts non-component code, including utility functions and backend service integrations. By segregating such code from the components, we ensure a clear separation of concerns and maintain a clean codebase structure. The content inside this folder is usually used across different parts of the app, outsourcing this content here makes it easy to manage it in one place.

- **services**: Backend request functions are grouped into logical files within the

[5] https://www.viking-js.com/post/understanding-smart-and-dumb-components-in-react

"services" directory, based on the corresponding backend models. This organization enhances code clarity and simplifies the management of backend interactions.

- **types**: TypeScript type definitions for backend models reside in the "types" directory. By providing type safety throughout the application, we mitigate potential runtime errors and improve code robustness.

- **utils**: Utility functions used for various purposes are centralized within the "utils" directory. By abstracting common functionalities into separate utility files, we keep the component files clean and readable, and embrace reusability.

- **theme**: The "theme" directory contains configuration files for Material UI, enabling consistent styling and theming across the application.

With the exception of the "app" folder, each folder and sub-folder in our structure is a Typescript Module [6]. More on this in the next section.

**Listing 3.6:** Example of import before aliases & modules

```
import { OfferCard } from "../../components/offer-card/offer-card"
```

**Listing 3.7:** Example of import after aliases & modules

```
import { OfferCard } from '@/components'
```

### 3.4.2 Imports & Import Aliases

The structure described above comes with many benefits. However, it adds complexity to import statements, where we often need to navigate through multiple

[6] https://www.typescriptlang.org/docs/handbook/modules/introduction.html

levels of folders to find the files we need. To address this, we use a combination of import aliases and Typescript Modules. These aliases simplify the import process and hide the inner folder structure from the import statement. This means that changes in file structure or names won't affect imports in other files. Listing 3.8 shows the import aliases. Listings 3.6 and 3.7 show the benefits of this approach.

**Listing 3.8:** Import aliases defined within the front-end project

```
"paths": {
    "@/*": ["*"],
    "@components/*": ["components/*"],
    "@lib/*": ["lib/*"]
}
```

### 3.4.3 Front-end Project Interactions

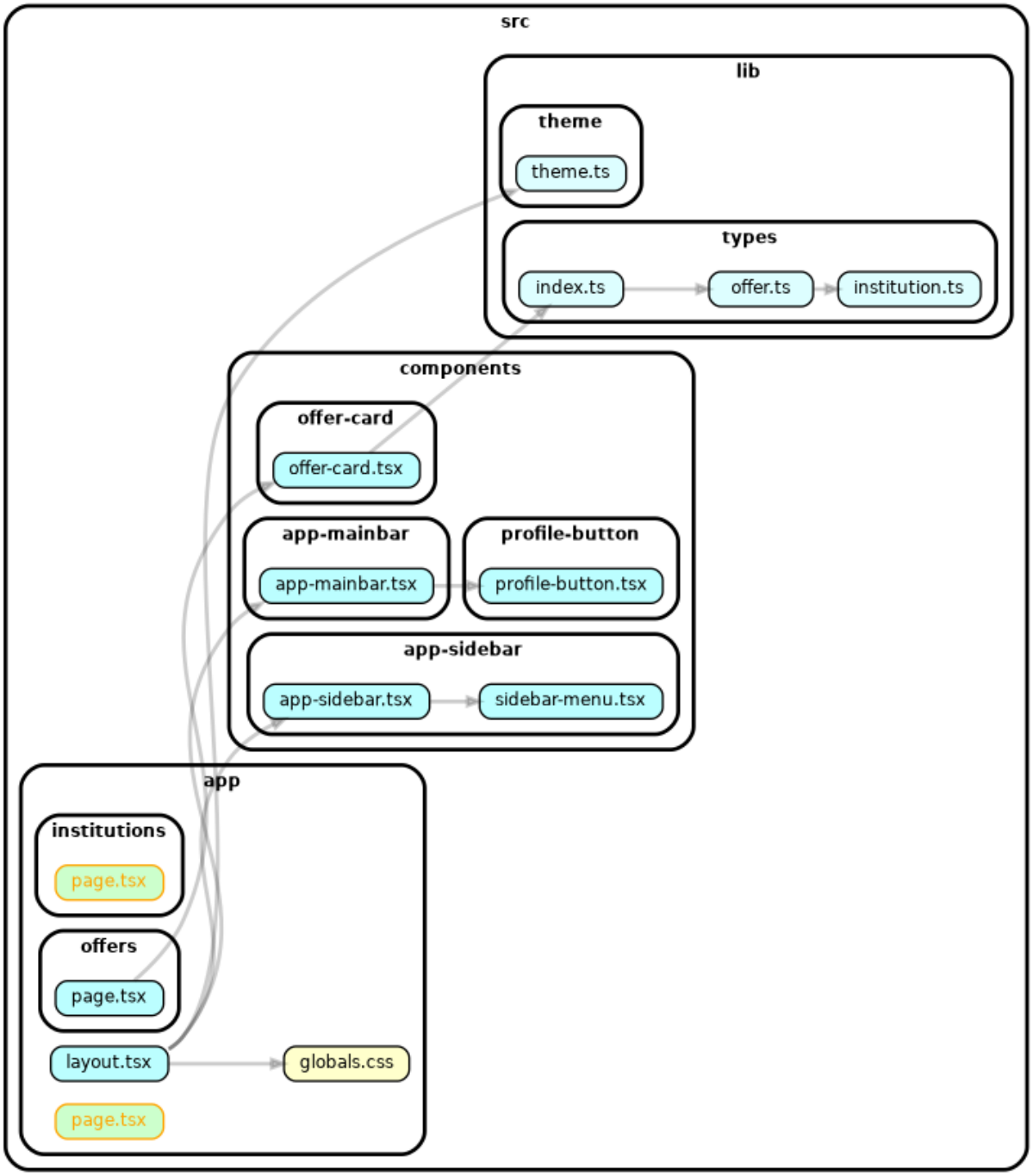


Fig. 3.8. Interactions within the front-end project

Fig. 3.8 depicts the component hierarchy within the front-end project. The blank rectangles represent folders, and the colored ones represent files. Each

arrow is an import statement. The architecture can be visualized as layers where each layer imports only from a lower layer, the outermost one being the app layer. We can see that no arrows go the opposite direction i.e. no lower layers import from ones that are higher up. This rule eliminates the risk of encountering circular dependencies during development.

# Chapter 4

# Evaluation

In dissecting the results of our work, we utilized specialized metrics to assess the performance of the individual components of our application across various testing scenarios. More on metrics and scenarios in Sections 4.1.1 , 4.2.1, and 4.2.2.

Key front-end metrics included a First Contentful Paint (FCP) of 0.2 seconds and a Speed Index (SI) of 1.1 seconds. The Largest Contentful Paint (LCP) occurred at 0.8 seconds, with a Total Blocking Time (TBT) of only 10 milliseconds. The Cumulative Layout Shift (CLS) scored a perfect zero, reflecting a stable user interface. These metrics contributed to an overall Performance Score of 99 out of 100.

The back-end performance was analyzed under various load-testing scenarios, revealing robust stability and scalability. The server was tested for its response to fixed, ramp-up, spike, and peak loads using metrics such as Total Requests, Throughput, Response Times, and Error Rates. Across all tests, the server consistently managed high loads with low response times and zero error rates.

## 4.1 Front-end Results

This section assesses key performance metrics of a web application, offering a snapshot of its responsiveness, speed, and stability from the perspective of the user. These performance aspects, widely recognized in web development, are critical for ensuring a quality user experience. They cover the initial loading of content, the speed of content filling on the screen, the time taken to become fully interactive, and the visual stability of the page as it loads.

### 4.1.1 Definition of Web Metrics

- First Contentful Paint (FCP): A user-centric metric that marks the time at which the first element is painted on the screen. For the end-user, this metric is indicative of the perceived load speed; a faster FCP generally suggests that the website is responsive, contributing to a positive initial user experience.

- Speed Index (SI): A performance metric that measures how quickly the contents of a webpage are visibly populated, with lower scores indicating faster perceived load times. For users, a better SI score means less time waiting for the page to appear fully populated, which can improve user satisfaction and engagement.

- Largest Contentful Paint (LCP): Refers to the time taken for the largest content element on the page to load and become visible to the user. This is crucial from the perspective of the user as it directly affects the time they wait to view the main content of the website, impacting their perceived load time and overall experience.

- Total Blocking Time (TBT): Quantifies the total time that a webpage is unresponsive to user input, such as clicks or key presses. A lower TBT is better for the user experience as it indicates a more responsive site, where interactions can be performed without frustrating delays.
- Cumulative Layout Shift (CLS): The sum total of all individual layout shift scores for every unexpected layout shift that occurs during the lifespan of the page. For users, a low CLS means the page is stable as it loads, without shifts that might cause misclicks or reading disruptions.

### 4.1.2 Performance Results

TABLE 4.1
FRONT-END PERFORMANCE METRICS

| | |
|---|---|
| First Contentful Paint (FCP) | 0.2 s |
| Speed Index (SI) | 1.1 s |
| Largest Contentful Paint (LCP) | 0.8 s |
| Total Blocking Time (TBT) | 10 ms |
| Cumulative Layout Shift (CLS) | 0 |

TABLE 4.2
LIGHTHOUSE SCORES (OUT OF 100)

| | |
|---|---|
| Performance Score | 99 |
| Accessibility Score | 100 |
| Best Practices Score | 100 |
| SEO Score | 100 |

We used Google Lighthouse [1] to evaluate the performance of the web app[2]. The obtained results in Tables 4.1 and 4.2 demonstrate a commitment to providing an optimal user experience. The First Contentful Paint (FCP) was recorded at 0.2 seconds, substantially quicker than the average benchmark, indicating that users are presented with visible content shortly after navigation begins. Similarly, the Speed Index (SI) was observed at 1.1 seconds, suggesting that content fills the viewport rapidly and the site appears responsive.

Largest Contentful Paint (LCP) occurred at 0.8 seconds, which represents a small timeframe for delivering the main content to the user. The Total Blocking Time (TBT) stood at a minimal 10 milliseconds, highlighting that the application remains interactive and does not hinder user actions.

A Cumulative Layout Shift (CLS) score of 0 indicates no unexpected layout shifts occur, thus providing a stable and reliable user interface.

These results collectively contribute to an overall Performance score of 99 out of 100, falling into the highest category set by the performance metrics, thereby indicating that the web application offers an excellent user experience from a performance standpoint. Fig. 4.1 is a visualization of the performance results using Lighthouse Scoring Calculator [3]. The performance score is calculated as follows:

[1] https://developer.chrome.com/docs/lighthouse/overview

[2] Rentawy performance report

[3] https://googlechrome.github.io/lighthouse/scorecalc/

$$\text{Performance Score} = 0.1 \cdot \text{FCP Score} + 0.1 \cdot \text{SI Score} + 0.25 \cdot \text{LCP Score} + 0.3 \cdot \text{TBT Score} + 0.25 \cdot \text{CLS Score}$$

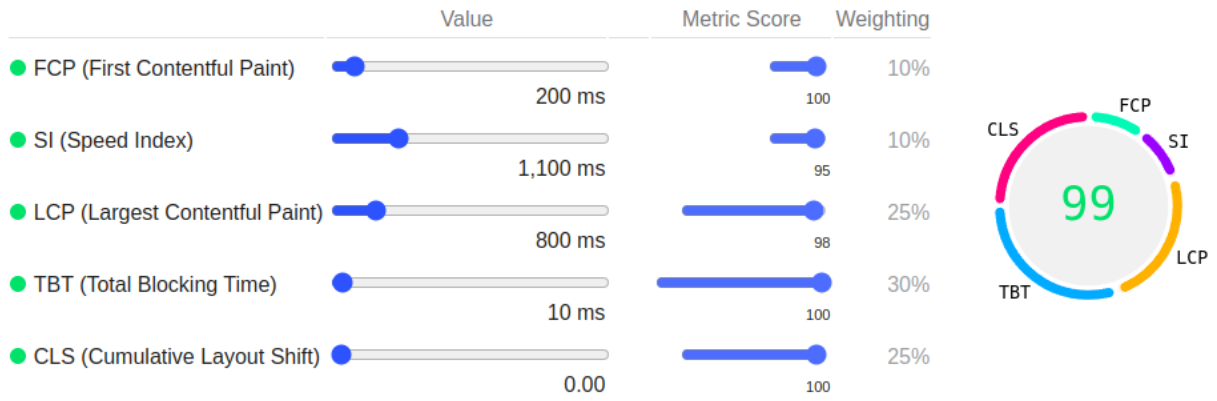


Fig. 4.1. Google lighthouse scoring calculator

In addition to performance, the application achieved full marks in Accessibility and Best Practices, indicating robustness in areas such as user accessibility and adherence to modern web standards. While the SEO of the application was also rated at 100, reflecting strong optimization for search engines.

Overall, the performance metrics of the web application rank among the top 1% of websites worldwide, which represents a significant factor in facilitating user retention and satisfaction, and a strong foundation for future development and scalability.

### 4.1.3 Performance Comparison

To contextualize the performance of our front-end, we compared its key metrics with those of one of the existing rental solutions - RentItems[4]. As well as the two of the well-known P2P platforms - Amazon[5] and Avito[6].

TABLE 4.3
FRONT-END PERFORMANCE COMPARISON

| Metric | Rentawy | RentItems | Avito | Amazon |
|---|---|---|---|---|
| First Contentful Paint (FCP) | 0.2 s | 1.2 s | 1.2 s | 0.6 s |
| Speed Index (SI) | 1.1 s | 2.3 s | 4.2 s | 3.2 s |
| Largest Contentful Paint (LCP) | 0.8 s | 3.6 s | 1.7 s | 1.3 s |
| Total Blocking Time (TBT) | 10 ms | 690 ms | 1830 ms | 30 ms |
| Cumulative Layout Shift (CLS) | 0 | 0.333 | 0.114 | 0.002 |
| Performance Score | 99 | 31 | 48 | 89 |

Table 4.3 illustrates this comparison, indicating that the performance of our website significantly surpasses that of one of the existing solutions, is comparable to prominent platforms like Amazon, and even surpasses them in the case of Avito.

## 4.2 Back-end Results

The back-end performance of the web application is a crucial aspect that directly impacts the user experience [25]. This analysis explores how the server handles and responds under different load-testing scenarios, each simulating a unique user interaction pattern, and allowing us to evaluate the performance of

[4]RentItems performance report
[5]Amazon performance report
[6]Avito performance report

the server under various conditions. We first explain the scenarios, and then the metrics used below.

### 4.2.1 Performance Testing Scenarios

Below are the scenarios we tested. We ran each of them for 10 minutes in 6-Core Intel Core i7 processor and 16 memory machine.

#### Fixed Load Testing

In the Fixed Load test, a constant number of users simultaneously interact with the application. This test assesses the stability of the server under a steady load.

#### Ramp-Up Testing

Ramp-Up testing gradually increases the load on the server. This test evaluates how the server scales and accommodates increasing numbers of users over time.

#### Spike Testing

Spike testing involves suddenly increasing the load significantly for a short period, then reducing it. This test checks the stability of the server to handle sudden bursts of traffic.

#### Peak Testing

Peak testing puts the server under the highest anticipated load. It assesses the maximum capacity of the server and identifies potential performance bottlenecks.

### 4.2.2 Server Performance Metrics

For each testing scenario, we focused on the following key metrics:

- **Total Requests**: The total number of requests processed.
- **Throughput**: The number of requests processed per second.
- **Average Response Time**: The average time taken to respond to a request.
- **Peak Response Time**: The longest time taken to respond to any single request.
- **Error Rate**: The percentage of requests that resulted in errors.

#### Fixed Load Test Results

TABLE 4.4
BACK-END FIXED PERFORMANCE METRICS

| Metric | Value |
|---|---|
| Total Requests | 218 905 |
| Average Response Time | 13 ms |
| Peak Response Time | 2 089 ms |
| Error Rate | 0% |

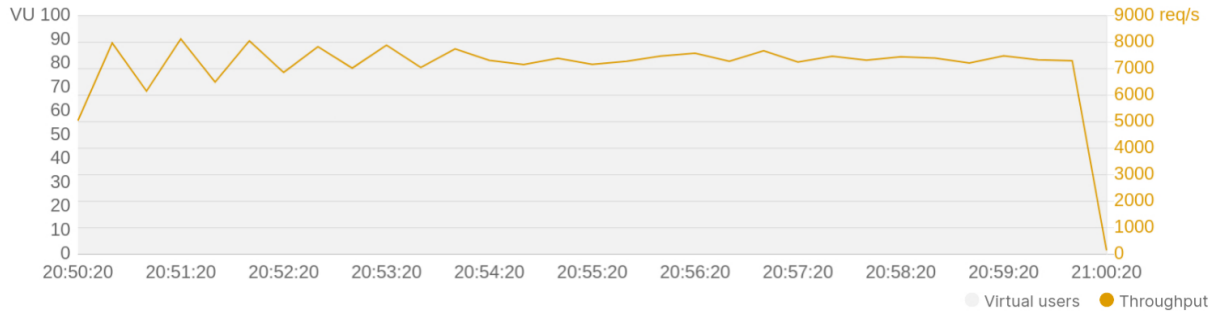


Fig. 4.2. Fixed testing throughput

Table 4.4 reveals the performance of the system under constant load, handling over 218 905 requests with an average response time of 13 ms and a peak of 2 089 ms, all with no errors. This indicates the system can comfortably support a sizable user base continuously without sacrificing speed or accuracy. Figure 4.2 focuses on the throughput being fixed at 8 000 req/s approximately for the 10 minutes period.

### Ramp-Up Test Results

TABLE 4.5
Back-end Ramp-up Performance Metrics

| Metric | Value |
|---|---|
| Total Requests | 176 128 |
| Average Response Time | 10 ms |
| Peak Response Time | 2 404 ms |
| Error Rate | 0% |

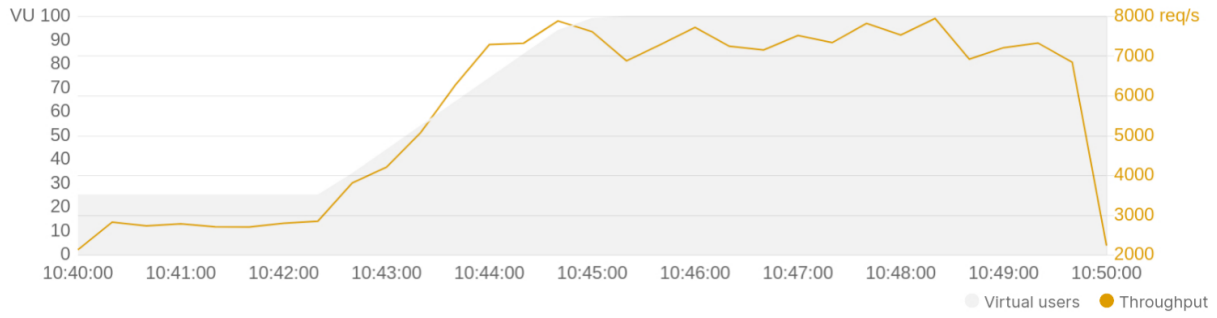


Fig. 4.3. Ramp-up testing throughput

Table 4.5 examines how the system copes with gradually increasing user load. Maintaining zero errors. This suggests the system can scale effectively to accommodate growing user numbers without significant performance degradation. Figure 4.3 demonstrates the ramp started from the second minute from 10 virtual users to span extra eight minutes and reach 100 virtual users.

### Spike Test Results

TABLE 4.6
Back-end Spike Performance Metrics

| Metric | Value |
|---|---|
| Total Requests | 80 337 |
| Average Response Time | 5 ms |
| Peak Response Time | 532 ms |
| Error Rate | 0% |

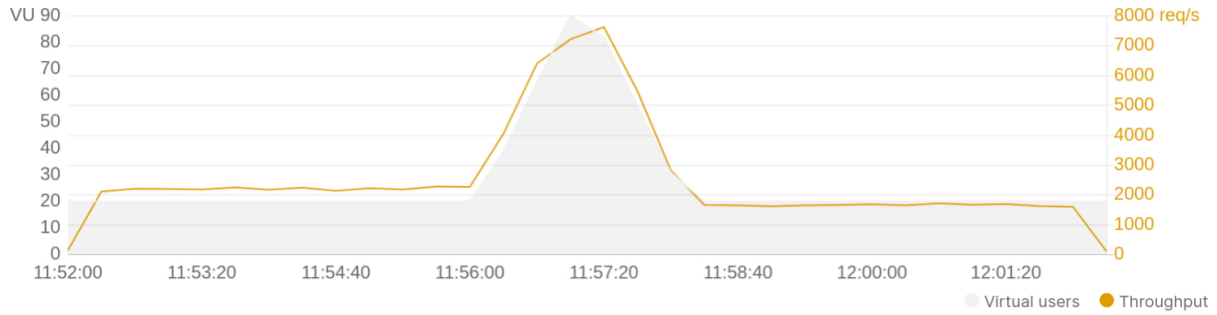


Fig. 4.4. Spike testing throughput

In table 4.6, the response of the system to sudden load spikes is assessed. Successfully handling about 80 337 requests at peak times, maintaining an average response time of five ms throughout the test and no errors. This means that the system is well-equipped to deal with sudden surges in usage, ensuring stability during unexpected high-traffic events. Figure 4.4 illustrates that spike was at the beginning of the fourth minute from 3 000 req/s to span extra four minutes and reach up to 7 500 req/s

### Peak Test Results

TABLE 4.7
Back-end Peak Performance Metrics

| Metric | Value |
|---|---|
| Total Requests | 191 015 |
| Average Response Time | 7 ms |
| Peak Response Time | 1030 ms |
| Error Rate | 0% |

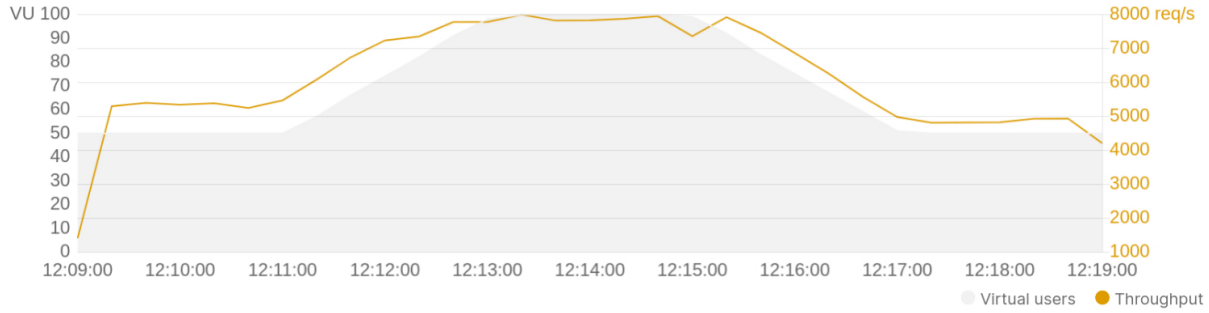


Fig. 4.5. Peak testing throughput

Table 4.7 focuses on performance under maximum load. The system was able to process 191 015 requests, all while maintaining an average response time of seven ms and zero errors. This shows it can handle peak usage times efficiently, ensuring a smooth user experience even under the heaviest loads. Figure 4.5 shows the system can get up to 8 000 req/s throughput.

### 4.2.3 Comparison of the Testing Scenarios

#### Peak Response Times

There is a notable difference in peak response times across the tests. The ramp-up test has the longest peak response time at 2,404 ms, followed by the fixed load test at 2,089 ms, the peak test at 1,030 ms, and the spike test at 532 ms. This variability indicates that while average response times remain stable, the server's response to the most demanding individual request can fluctuate significantly. This fluctuation could be a concern for operations requiring consistent response times, as it may impact user experience during critical interactions.

#### Resource Limitations

The varying peak response times could be indicative of resource limitations, such as CPU or memory constraints, that become apparent under different testing conditions. This is particularly evident in the ramp-up and fixed load tests, where the highest peak response times are observed.

### 4.2.4 Availability

Ever since the Backend was deployed, it has 100% availability according to uptime robot [7]. This shows that the Backend is ready to accept requests all the time.

### 4.2.5 Performance comparison

To the best of our knowledge, there are no similar studies to to benchmark these results against, however, the results demonstrate the proficiency of the server in managing various user interaction scenarios. The system exhibits robustness and scalability, ensuring a seamless user experience by maintaining low response times and high reliability across different load conditions.

[7] https://stats.uptimerobot.com/w2RZTPZ8yQ

# Chapter 5

# Discussion & Conclusions

## 5.1 Discussion

The issues we identified with existing rental solutions include performance problems and a lack of support for confined rentals.

Results from the previous chapter demonstrate how we addressed the performance issues:

Firstly, our front-end achieved outstanding results, ranking in the top 1% of websites worldwide according to web benchmarks. This performance significantly surpassed that of existing platforms, as detailed in the report linked in Appendix A.

Although we measured the performance of our back-end, we were unable to benchmark it against other back-ends or those of existing solutions due to a lack of available data.

Regarding the lack of support for confined rentals, we were unable to test this hypothesis. Our platform was not launched to the public due to insufficient resources. As a result, this aspect of our work remains unexplored and could be

thoroughly tested in future works.

## 5.2 Conclusions

The goal of this work was to create a peer-to-peer rental platform that addresses the limitations of existing peer-to-peer rental solutions as this is an area of increasing importance. The limitations we sought to address were low performance scores, and a lack of support for community-confined rentals.

The contributions of this work include:

- The successful development of the platform, which yielded high performance scores.
- To the best of our knowledge, there is no documented process in the existing literature for building a web application with this specific set of technologies.
- The selection process, analysis, and results of choosing the most suitable technologies for building the project, guided by peer-reviewed literature, comparative analysis, and contemporary industry trends.

The implications of these contributions are:

- The methods and practices used in this project may serve as a guideline for others utilizing similar technologies.
- The Literature Review chapter could serve as a comprehensive overview of the modern development methodologies and technologies for building web applications.

- Results from the literature review may inform the future development process of platforms with similar features.

Our work did not test the hypothesis of community-driven rentals. Future work could launch the platform, collect user feedback, and test the viability of the idea.

# Bibliography cited


[1] M. Kalua, M. Kalanj, and B. Vukeli, "A comparison of back-end frameworks for web application development," *Zbornik veleuilita u rijeci*, vol. 7, no. 1, pp. 317–332, 2019.

[2] X. Huang, "Research and application of node. js core technology," in *2020 International Conference on Intelligent Computing and Human-Computer Interaction (ICHCI)*, IEEE, 2020, pp. 1–4.

[3] F. Zhang, G. Sun, B. Zheng, and L. Dong, "Design and implementation of energy management system based on spring boot framework," *Information*, vol. 12, no. 11, p. 457, 2021.

[4] M. M. Eyada, W. Saber, M. M. El Genidy, and F. Amer, "Performance evaluation of iot data management using mongodb versus mysql databases in different cloud environments," *IEEE access*, vol. 8, pp. 110 656–110 668, 2020.

[5] A. Ginanjar and M. Hendayun, "Spring framework reliability investigation against database bridging layer using java platform," *Procedia Computer Science*, vol. 161, pp. 1036–1045, 2019.

[6] A. Banks, E. Porcello, and a. O. M. C. Safari, *Learning React, 2nd Edition*. O'Reilly Media, Incorporated, 2019. [Online]. Available: https://books.google.ru/books?id=BBBOzQEACAAJ.

[7] C. Macrae, *Vue.js: Up and Running: Building Accessible and Performant Web Apps*. O'Reilly Media, 2018, ISBN: 9781491997246. [Online]. Available: https://books.google.ru/books?id=YNRKswEACAAJ.

[8] A. Bampakos and P. Deeleman, *Learning Angular: A no-nonsense guide to building web applications with Angular 15*. Packt Publishing, 2023, ISBN: 9781803237343. [Online]. Available: https://books.google.ru/books?id=tV-vEAAAQBAJ.

[9] M. Kalua, K. Troskot, and B. Vukeli, "Comparison of front-end frameworks for web applications development," *Zbornik Veleuilita u Rijeci*, vol. 6, no. 1, pp. 261–282, 2018.

[10] Y. Xing, J. Huang, and Y. Lai, "Research and analysis of the front-end frameworks and libraries in e-business development," in *Proceedings of the 2019 11th International Conference on Computer and Automation Engineering*, 2019, pp. 68–72.

[11] R. Vyas, "Comparative analysis on front-end frameworks for web applications," *International Journal for Research in Applied Science and Engineering Technology*, vol. 10, no. 7, pp. 298–307, 2022.

[12] V. Ballamudi, K. Lal, H. Desamsetti, and S. Dekkati, "Getting started modern web development with next. js: An indispensable react framework," *Digitalization & Sustainability Review*, vol. 1, no. 1, pp. 1–11, 2021.

[13] S. Al-Saqqa, S. Sawalha, and H. AbdelNabi, "Agile software development: Methodologies and trends.," *International Journal of Interactive Mobile Technologies*, vol. 14, no. 11, 2020.

[14] A. Moniruzzaman and D. S. A. Hossain, "Comparative study on agile software development methodologies," *arXiv preprint arXiv:1307.3356*, 2013.

[15] V. Chandra, "Comparison between various software development methodologies," *International Journal of Computer Applications*, vol. 131, no. 9, pp. 7–10, 2015.

[16] G. Kumar and P. K. Bhatia, "Comparative analysis of software engineering models from traditional to modern methodologies," in *2014 Fourth International Conference on Advanced Computing & Communication Technologies*, IEEE, 2014, pp. 189–196.

[17] V. Hema, S. Thota, S. N. Kumar, C. Padmaja, C. B. R. Krishna, and K. Mahender, "Scrum: An effective software development agile tool," in *IOP Conference Series: Materials Science and Engineering*, IOP Publishing, vol. 981, 2020, p. 022 060.

[18] E. P. Wonohardjo, R. F. Sunaryo, and Y. Sudiyono, "A systematic review of scrum in software development," *JOIV: International Journal on Informatics Visualization*, vol. 3, no. 2, pp. 108–112, 2019.

[19] P. A. G. Permana, "Scrum method implementation in a software development project management," *International Journal of Advanced Computer Science and Applications*, vol. 6, no. 9, pp. 198–204, 2015.

[20] V. T. Faniran, A. Badru, and N. Ajayi, "Adopting scrum as an agile approach in distributed software development: A review of literature," in *2017 1st International Conference on Next Generation Computing Applications (NextComp)*, IEEE, 2017, pp. 36–40.

[21] E. Hossain, M. A. Babar, and H.-y. Paik, "Using scrum in global software development: A systematic literature review," in *2009 Fourth IEEE International Conference on Global Software Engineering*, Ieee, 2009, pp. 175–184.

[22] V. Mahnic, "A capstone course on agile software development using scrum," *IEEE Transactions on education*, vol. 55, no. 1, pp. 99–106, 2011.

[23] K. S. Rubin, *Essential Scrum: A practical guide to the most popular Agile process*. Addison-Wesley, 2012.

[24] S. Apel and C. Кдstner, "An overview of feature-oriented software development.," *J. Object Technol.*, vol. 8, no. 5, pp. 49–84, 2009.

[25] A. Golmohammadi, M. Zhang, and A. Arcuri, "Testing restful apis: A survey," *ACM Transactions on Software Engineering and Methodology*, vol. 33, no. 1, pp. 1–41, 2023.

# Appendix A

# Existing Rental Platforms

TABLE A.1
Existing Rental Platforms

| Platform | Link | Limitations | Notes |
|---|---|---|---|
| Rent Items | https://rentitems.com/ | • No data<br>• Performance issues | Performance report |
| RentMy | https://www.rentmy.com/ | • Possibly fake data<br>• Very small user-base<br>• SEO issues | The platform was closed as of May 28th 2024 and is planning to re-launch. The information may no longer be relevant. |
| KidzKit | https://rentkidzkit.com/en/ | • Limited to kids' gear | None |