
# KODIRAJ.BA – ONLINE JUDGE WEB APPLICATION

A dissertation submitted to

the Faculty of Engineering and Natural Sciences of

International University of Sarajevo in partial

fulfillment of the requirements for the

degree of Bachelor of Science

in SOFTWARE ENGINEERING

by

Harun Tucaković and Muhammed Mušanović

June, 2022

Dissertation written by

HARUN TUCAKOVIĆ and MUHAMMED MUŠANOVIĆ

Graduation Committee

| | |
|---|---|
| Assoc. Prof. Dr. Kanita Karađuzović-Hadžiabdić | IUS, Bosnia and Herzegovina, Supervisor |
| Assoc. Prof. Dr. Khaldoun Al Khalidi | IUS, Bosnia and Herzegovina |

We hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. We also declare that, as required by these rules and conduct, we have fully cited and referenced all material and results that are not original to this work.

HARUN TUCAKOVIĆ,
MUHAMMED MUŠANOVIĆ

# INTERNATIONAL UNIVERSITY OF SARAJEVO

DECLARATION OF COPYRIGHT AND AFFIRMATION

OF FAIR USE OF UNPUBLISHED WORK



KODIRAJ.BA – ONLINE JUDGE WEB APPLICATION



Affirmed by Harun Tucaković, Muhammed Mušanović

Signatures Date

# ABSTRACT

The increase in popularity of online education over the last two decades caused the birth of many online judge educational platforms in the IT industry. With time, online judge educational platforms became one of the best alternatives to the traditional education system when it comes to IT education. Nevertheless, the topic of architecture and implementation details of such systems is still not explored to its full potential. Moreover, online judge educational platforms are generally available only in English and a few other most popular languages in the world. This project develops a new online judge educational platform as a web application called Kodiraj.ba. Kodiraj.ba web application is developed in the Bosnian language and is aimed at the market in Bosnia and Herzegovina. Additionally, this project provides a detailed explanation of the architecture and implementation of the application. Backend for Kodiraj.ba web application was developed in Node.js and Express.js, while frontend was developed using ReactJS and Redux state manager. Execution of user-submitted code was implemented as a set of microservices developed using Python and Bash shell. Kodiraj.ba web application is a functional online judge web application with a content management system that allows administrator users to manage content on the platform.

# TABLE OF CONTENTS

# LIST OF FIGURES

# LIST OF TABLES

# CHAPTER 1 **INTRODUCTION**

The massive expansion of the IT industry and increase in demand for software engineers, developers, and IT specialists over the last two decades has resulted in many innovations and improvements in the education system for these roles. One of the most popular tools for practicing and improving problem-solving skills outside of the bounds of the traditional education systems are online judge web applications. Online judge systems in general are web applications that provide a reliable and safe cloud-based evaluation of programming solutions submitted by users of the web application [1]. User-submitted code is compiled and executed against the set of test cases that are designed to check the correctness of the user's solution while measuring the performance of the solution in terms of execution time and memory usage.

## 1.1 Problem statement

Many online judge systems and web applications have been developed during the last two decades, with many of them providing very diverse sets of functionalities and different ways for users to improve their problem-solving skills. However, even with the immense popularity of online judge web applications, the topic of the architecture and implementation details of such systems was poorly explored. Existing research and papers on this topic are scares and they will be covered in the following chapter. Furthermore, despite the popularity of online judge systems, most of them are available just in the English language or a handful of the most popular languages in the world. Hence, there is a language barrier to entry and usability for people from smaller communities and countries. Bosnia and Herzegovina can be an example of one of the communities that have language barriers to the usability of most existing online judge

systems. Many students and young people from Bosnia and Herzegovina who are not proficient in the English language will have a hard time utilizing existing online judge systems to their full potential. The solution that this project presents for both above-mentioned problems is the development of the online judge web application in the Bosnian language and providing a detailed explanation of the architecture and technologies used to develop the system.

### 1.2 Objectives

Conversation and sharing of different ideas are a great way to improve common knowledge and understanding of any topic, and the same is the case with technical topics in the IT industry. By increasing the number of academic papers, research, and people participating in the conversation about the architecture and implementation details of online judge systems, the entire industry moves towards improvements and a better understanding of the topic. While also providing more resources for newcomers to learn and explore the technologies and architectures that are used in these systems. While researching the topic, the development of such a system for the market of Bosnia and Herzegovina can help many young people practice and improve their skills. Consequently, the main objectives of this project can be summarized in the following:

1. Propose and explain in detail our idea for the architecture and implementation of the online judge web application.
2. Develop Kodiraj.ba online judge web application for the market in Bosnia and Herzegovina.

### 1.3 Thesis structure

This paper consists of seven chapters: Introduction, Related work, Security concerns, Technologies, Requirements, System design, and Conclusion. The first chapter, Introduction, gives an overview of the project and the paper, explaining the problem statement, objectives, and the structure of the paper. The related work chapter explores existing research and systems on this topic, while also explaining how this project fits into the field. The Technologies chapter gives an overview of different technologies and tools that are used to develop web applications. The fourth chapter, Security concerns, explains the security considerations that come with developing an online judge web application and what is done in this project to improve the security of the system. The Requirements chapter lays out system features, user requirements, functional requirements, and non-functional requirements of the system. The sixth chapter, System design, dives into details of the system architecture and implementation of the system, illustrating the system design with a database design diagram, use case modeling, sequence diagrams, and user interface explanations. Finally, the chapter Conclusion gives the conclusion and the summary of the paper and the web application implementation.

# CHAPTER 2 **RELATED WORK**

This section discusses existing online judge web applications, as well as other web applications that implement code execution engine technologies. Furthermore, existing research and papers on the topic of online judge systems are reviewed.

The popularity of this medium for education and problem-solving skill development in the IT industry has given rise to many online judge systems and web applications. Some online judge web applications are primarily focused on competitive programming [2], [3], and some provide preparation for technical job interviews [4], [5], while others provide online courses [6], [7]. Several researchers published papers on the topic of architecture and system design of Online Judge web applications. Wirawan et al. [8] presented the high-level design of an online judge web application without any architecture or implementation details. While this paper gives a great high-level idea of the system design, it does not provide any details regarding how the user's solution is executed, how is it scored, or how are test cases evaluated. Similarly, Jianhua et al. [9] developed and presented their system design, but only high-level system design is provided without any insight into how the code gets executed and how are test cases evaluated. Himanshu et al. [10] write about security issues for online judge web applications, which according to them can be categorized into issues related to authentication of users, authorization of requested resources, malicious code execution, and network attacks. Kasahara et al. [11] showed that applying gamification to online judge educational platforms can motivate students to write better code.

The code execution engine is one of the key features of the Kodiraj.ba web application. This feature has been previously developed and implemented in other online judge web applications, online compilers, IDEs, and web APIs that provide a

code execution engine as a SaaS product. The code execution engine is a topic that has been previously researched in scientific papers, even though the research and papers on this topic are scarce, as mentioned before. Papers that are found on this topic provided their system design details for the implementation of code execution engine apart from the topic of online judge systems. Došilović and Mekterović [12], and Hafiz and Jin [13] developed and explained their idea for a remote code execution engine with a focus on scalability, flexibility, and distributed systems. Stammerjohann [14] proposed different methods of sandboxing code execution with the intent to improve code execution isolation and system security. Sandboxing is the best way to isolate program execution from the base system. This isolation in most cases is used for security reasons, to prevent the program from affecting the base system. Some sandboxing approaches can be resource-heavy and time-consuming. Derval et al. [15] implement the code execution engine for the automatic grading system using the Docker virtualization and sandboxing. Docker containers have also been used to build a code execution engine for online judge applications by Yibo et al. [16]. While Docker virtualization in this context can be straightforward and easier to implement, alternatives like LXC containers provide lower-level sandboxing, have much less overhead, and can have a massive impact on performance.

# CHAPTER 3 **TECHNOLOGIES**

This chapter covers technologies and tools used for the implementation of the application. Overall, technologies used can be divided into three sections: backend technologies, frontend technologies, and technologies used for the deployment of the application.

## 3.1 Backend technologies

The backend server for Kodiraj.ba online judge web application is developed using Node.js [17] and Express.js [18]. Node.js is an open-source JavaScript runtime environment that enables the execution of JavaScript code outside of the web browser. It is used to write JavaScript back-end web servers. Node.js has single-threaded asynchronous event-driven architecture and it runs on the V8 JavaScript engine. On top of plain Node.js, this application uses Express.js as a web application framework which provides a set of features for easier and more convenient web development.

The database used for data storage is PostgreSQL [19], which is also known as Postgres. It's a free and open-source relational database management system. Node.js communicates with the PostgreSQL database using Sequelize. Sequelize is an object-relational mapper (ORM) for Postgres, MSSQL, SQLite, MariaDB, and MySQL databases, and it is promise-based featuring good transaction support, eager and lazy loading, and relations [20].

Code execution engine consists of Python and Bash scripts. For code execution Bash scripts use Unix command-line tools like "time", "timeout", and "LXC". The command-line program called "time" is used to measure program execution time and memory usage. The command-line program called "timeout" is used to set a time limit

for the program execution and terminate the program after the given time expires. LXC [21] is a Linux container runtime. It's an operating-system-level virtualization tool that provides a method of running multiple isolated Linux systems using a single Linux kernel. In this project, LXC is used to virtualize and isolate user-submitted code execution. Bash scripts are written to interact with LXC containers and other Unix command-line tools mentioned above. The Python script is designed to run as a separate service from the main backend web server. It is used to start and stop Bash execution scripts, handle the results of the execution, and communicate to the main backend web server over the RabbitMQ message broker.

RabbitMQ [22] is an open-source message broker. Kodiraj.ba web application uses RabbitMQ for communication between the main backend web server and Python services. The web server emits messages to the RabbitMQ queue to start code executions, while Python services consume messages from the queue and start code executions one by one as messages are consumed.

### 3.2 Frontend technologies

Before frontend development starts, low-fidelity and high-fidelity wireframes are created using Figma [23]. Figma is web-based graphics editing and prototyping software that is used for wireframing, prototyping, and UI design.

Frontend for Kodiraj.ba application is developed using ReactJS [24]. ReactJS is a free and open-source frontend JavaScript library that is mainly used for the development of single-page applications. More precisely, it is a declarative and component-based library whose only focus is managing the application state and rendering it to the document object model (DOM). Consequently, ReactJS is commonly

used along with some other libraries to facilitate the development process and improve user experience.

However, managing the state in React application can become cumbersome and unmanageable, which is why some additional state management libraries are used, such as Redux [25]. Redux is an open-source library used for state management and state centralization. This makes code more readable and state sharing between components easier.

For application styling, Sass [26] preprocessor scripting language is used. It represents an extension of standard CSS and is compiled into CSS. Additionally, CSS modules are utilized which allow writing CSS (or Sass) in local scope rather than global scope.

### 3.3 Deployment technologies

Kodiraj.ba web application is hosted on the Contabo [27] virtual private server (VPS). VPS is running Ubuntu Linux 20.04. Backend and frontend servers are run and managed by PM2 [28] and Nginx [29]. PM2 is a process manager for Node.js applications that simplifies the process of starting, restarting, and stopping Node.js servers. Code is hosted on GitHub, and the deployment process is automated using GitHub Actions [30].GitHub Actions are a tool for CI/CD, that allows for automatic building, testing, and deployment of the code directly from GitHub.

# CHAPTER 4 **SECURITY CONCERNS**

Modern web applications are prone to having security risks and vulnerabilities. In 2021, OWASP (Open Web Application Security Project) identified that 94% of the total 500,000 tested web applications had some form of broken access control [31]. In this section, we discuss security risks and concerns related to Kodiraj.ba web application. Additionally, we present the steps that are taken to mitigate those risks.

## 4.1 General Security Concerns

The first step in creating a secure web application is designing and implementing quality authentication and authorization methods. Kodiraj.ba web application implements authentication using registration and login with email and password. During the registration process, all user passwords are encrypted using bcrypt password-hashing function [32] before they are stored in the database. On all login attempts, passwords are securely compared on the backend server. Furthermore, authorization is implemented using JSON Web Tokens (JWT). Each JWT token holds user information required to ensure that user has access only to resources that they are authorized to access.

In the literature survey done by Kumar et al. [33] they discovered that the two most common security vulnerabilities in web applications today are SQL injection and cross-site scripting (XSS). Both SQL injection and XSS attacks are based on the premise of executing malicious code in the web application. SQL injection attack exploits web application input fields to send malicious parameters to the server which in return illegally modifies the database of the application. An XSS attack is done by inserting malicious code into the web application frontend, which is usually done to steal cookies

and credentials from the browser. Kodiraj.ba web application uses parameter binding (parameterized queries) to mitigate the risk of SQL injection attacks. Multiple research papers on the topic of SQL injection attacks propose parameter binding as one of the best methods of protecting web applications against SQL injection attacks [33]–[35]. To prevent the XSS attack attempts, Kodiraj.ba web application implements input validation on each input field. Input validation ensures that potentially malicious user input is detected and not executed.

### 4.2 Code execution engine

Kodiraj.ba web application uses a custom code execution engine to run and test users' code. Execution of users' code on the Kodiraj.ba server raises some obvious security concerns but also implies several inconspicuous security issues that require careful handling. User code can cause damage to the system in the following ways:

- Infinite loops or recursion without a base condition – user code that contains infinite loops or recursion without a base condition, if executed on the server, can starve other processes out of resources. This can cause serious performance loss and even system crashes due to exhausted resources.
- Memory or CPU heavy tasks – if user code is trying to run CPU or memory-intensive tasks it can starve other processes on the server out of resources. Consequently, we can expect performance loss or in some cases even a system crash.
- Modifying server filesystem – submitted code can attempt to create, delete, or modify files on the server filesystem. This, in turn, can cause major damage to the system in the case important files get deleted/modified, or new malicious

files are created. Additionally, user code can starve the system out of storage space by infinitely creating new files.

- Interfering with system processes – user code may try to interfere with running system processes, whether by modifying them or killing them, resulting in an application or system crash.
- Fork bombs – submitted code can contain a fork bomb that guarantees a system crash if it is executed. Nakagawa and Oikawa [36] define fork bobs as "a Denial-of-Service attack (DoS attack) that exhausts the resource of the target system by creating many processes rapidly".
- Using the network connection – the user submitted code can attempt to connect to remote devices over the network. This can be done to download malicious files, send information to the attacker, or just saturate the server network as a DDoS attempt. Furthermore, user-submitted code can try to spoof network information if allowed to access the network.

The above-mentioned security risks can be mitigated with a careful system design. Kodiraj.ba web application follows a list of steps to ensure all risks are mitigated:

- Using LXC containers to isolate user code execution – LXC is a Linux userspace interface that provides an API for Linux users to create and manage application and system containers [37]. All user submitted code is compiled and executed in separate LXC containers that separate all processes inside of the container from the rest of the system. Therefore, any malicious code that is executed inside of the container affects only that container and disappears completely when the container is killed. This protects the rest of the system

from any malicious code that would attempt to modify the server filesystem, execute fork bombs, or interfere with system processes.

- Disabling network interfaces – each container is prohibited from accessing network interfaces and therefore from establishing any kind of outgoing network connection, which protects our system from unnecessary network saturation, remote access, and file transfers.
- Setting resource limits – LXC containers are configured to have strict resource limits, which prevents a single user code execution to exhaust system resources.
- Runtime execution limit – each user code execution process is created with a runtime execution limit. This time limit prevents the process from running infinitely in case of infinite loops and recursion without a base condition. While also preventing slow user code from wasting system resources.

In summary, there are many security risks and concerns involved with contemporary web application development. An online judge system with a code execution engine brings additional challenges to the security aspect of web application development. User code execution opens many approaches for the attackers to damage the system. Kodiraj.ba web application is developed in consideration of the latest research and scientific papers on the topic of web application security and implements all necessary measures to reduce the risk of security threats.

# CHAPTER 5 **SOFTWARE REQUIREMENTS**

This chapter explains in detail system features, user requirements, functional requirements, and non-functional requirements. These requirements usually serve as a contract between the client and developers by making it clear what is expected, and what needs to be implemented. This paper presents the requirements below as an overview of the scope of the project and its features and functionalities. Figure 5.1 provides a convenient way to get a quick overview of the application.

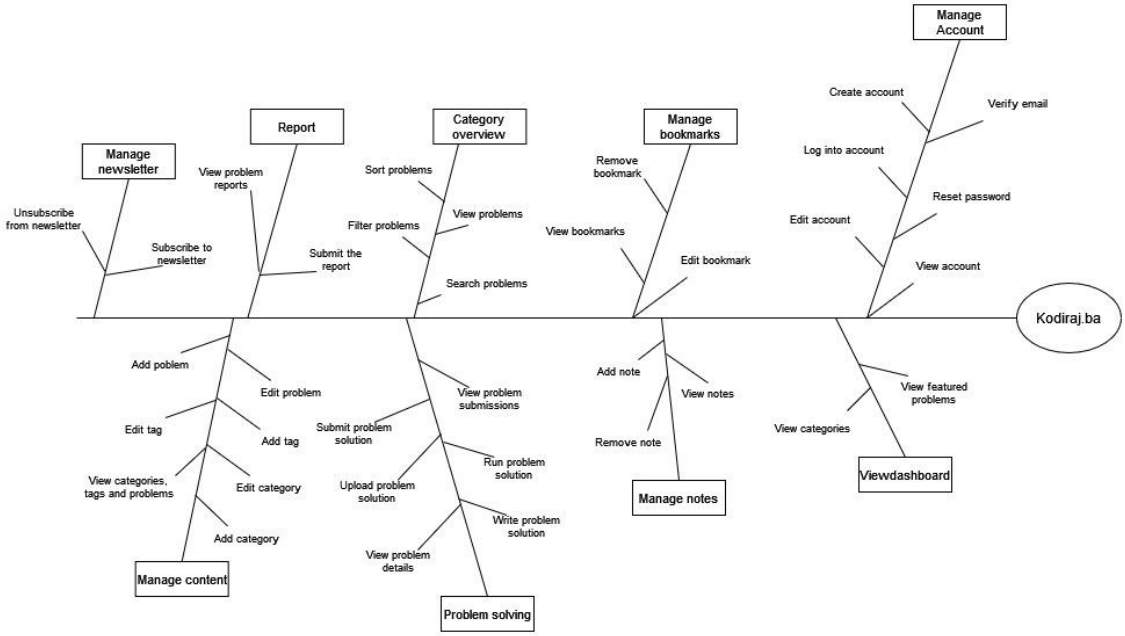


Figure 5.1: Feature tree

## 5.1 System features

**FTR1** Manage account: Users can create, manage, view, and use their accounts.

Priority: High

UR1.1 Create account

FR1.1.1 Users shall be able to enter their username, email, password, and confirm password to create an account.

FR1.1.2 Users shall be able to create an account using their Google account.

FR1.1.3 Users shall be redirected to the page with information about email verification, after account registration.

FR1.1.4 Users shall not be allowed to create an account with a username or email that is used by another account.

FR1.1.5 Users shall not be required to verify their email address if the account is created with a Google account.

UR1.2 Verify email

FR1.2.1 Users shall receive a verification email after they have created an account.

FR1.2.2 Users shall be able to verify their email by clicking the link in the email message.

FR1.2.3 Users shall be redirected to the login page after they verify the email. The notification message should be displayed on the login page informing the user that email verification was successful.

UR1.3 Log into account

FR1.3.1 Users shall be able to log into their accounts using their email and password.

FR1.3.2 Users shall be able to log in using their Google accounts.

FR1.3.3 Users shall not be able to log in if their email is not verified.

UR1.4 Reset password

FR1.4.1 Users shall be able to click the “Forgot password” link on the login page to start the process of password reset. It should redirect them to the page with password reset instructions.

FR1.4.2 Users shall be able to submit their email to receive a password reset URL.

FR1.4.3 Users shall be able to reset their password by entering a new password and confirming the password on the page that is only accessible through the password reset URL from the email message.

UR1.5 View account

FR1.5.1 Users shall be able to see their username, avatar, first name, last name, email, location, GitHub, and LinkedIn URLs and the "Profile settings" button, which will open the profile settings page.

FR1.5.2 Users shall be able to see their account statistics: total number of solved problems, number of solved problems in each difficulty level, acceptance percentage, the total number of points collected, and number of points per points category.

FR1.5.3 Points categories shall be problem solved, streak, category completion, tag completion, and membership time.

UR1.6 Edit account

FR1.6.1 Users shall be able to view their avatar, username, email, and settings panel in profile settings.

FR1.6.2 Users shall be able to upload a new avatar image. The new avatar image must be less than 2MB in size.

FR1.6.3 Users shall be able to view and edit their first name, last name, location, date of birth, summary, GitHub and LinkedIn URLs, username, and password in the settings panel.

FR1.6.4 Users that created their account via an external authentication method (e.g., Google) cannot change the password.

FR1.6.5 Users shall be able to toggle their newsletter subscription in the setting panel.

**FTR2** View dashboard: The dashboard represents a homepage for logged-in users. Users can easily navigate the application from the dashboard and view their progress and summary of the account statistics.

Priority: Medium

UR2.1 View categories

FR2.1.1 Users shall be able to view categories they started to solve.

FR2.1.2 Users shall be able to see category name, category points, progress bar, percentage of problems solved, and “Continue” button for each category they started solving.

FR2.1.3 Users shall be able to view a list of all available categories.

FR2.1.4 Users shall be able to see category names, category points, and the number of total problems.

FR2.1.5 Users shall be able to click on the category to open a page with a list of all problems in that category.

UR2.2 View featured problems

FR2.2.1 Users shall be able to see the problem of the day. For the problem of the day, there should be displayed: “Problem of the day” text, problem title, summary, tags, and category.

FR2.2.2 Users shall be able to view a list of ten suggested problems with “Suggested problems” text, problem names, categories, tags, and difficulties displayed.

FR2.2.3 Suggested problems should not include problems that the user has already solved.

FR2.2.4 Users shall be able to click on the problem in the suggested problems list to navigate to the problem page.

FR2.2.5 Users shall be able to see and click the “Random problem” button, which will redirect them to the page of a random problem.

**FTR3** Manage bookmarks: Users can bookmark, upvote, and downvote the problems, as well as view the lists of the bookmarked.

Priority: Low

UR3.1 View bookmarks

FR3.1.1 Users shall be able to view a list of all bookmarked problems, with problem name, problem tags, problem category, problem difficulty level, problem summary, and the timestamp of the bookmark displayed for each bookmarked problem.

UR3.2 Save bookmark

FR3.2.1 Users shall be able to see and click the “Bookmark” button on the problem page, to save that problem to their bookmarks list.

FR3.2.2 Users shall be able to see and click the upvote button on the problems page to upvote the problem, or to remove the upvote from the already upvoted problem.

FR3.2.3 Users shall be able to see and click the downvote button on the problems page to downvote the problem, or to remove the downvote from the already downvoted problem.

UR3.3 Remove bookmark

FR3.3.1 Users shall be able to see and click the “Remove Bookmark” button on the problem page of the bookmarked problem, to remove that problem from their bookmarks list.

FR3.3.2 Users shall be able to remove a problem from the bookmarks list directly on the page that displays a list of bookmarks.

FR3.3.3 User upvotes shall be removed if the user downvotes the upvoted problem and vice versa.

**FTR4** Manage notes: Users can create and manage notes. Notes can be general notes and problem notes which are connected to the problem user started to solve.

Priority: Medium

UR4.1 View notes

FR4.1.1 Users shall be able to view a list of all notes that they left for problems, with note title, note text, the timestamp when the note was last edited, and the problem title displayed for each note.

FR4.1.2 Users shall be able to view a list of all general notes, with note title, note text, and timestamp when the note was last edited displayed for each note.

FR4.1.3 All notes shall be sorted by the date they were last edited in descending order.

FR4.1.4 Users shall be able to view a note for the problem on the problem page of that problem, with the note title and text displayed.

FR4.1.5 All notes shall be displayed in the color user has picked.

UR4.2 Add note

FR4.2.1 Users shall be able to add a general note on the notes page by providing the note title, and note text and picking the note color.

FR4.2.2 The user shall be able to add a note for the problem on the problem page by providing the note title, and note text and picking the note color.

FR4.2.3 Users shall be able to add only one note per problem.

FR4.2.4 Users shall be able to edit the note title, note text, and note color of any note on the notes page.

FR4.2.5 Users shall be able to edit the note title, note text, and note color of the problem note on the problem page of that problem.

UR4.3 Remove note

FR4.3.1 Users shall be able to delete any notes from the notes page.

FR4.3.2 Users shall be able to delete the problem note from the problem page of that problem.

**FTR5** Category overview: Users can overview the category. In the category overview users can view, search, filter, and sort all the problems in that category.

Priority: High

UR5.1 View problems

FR5.1.1 Users shall be able to view a list of all problems that belong to the category.

FR5.1.2 Users shall be able to see the problem name, summary, tags, difficulty, and problem points for each problem in the list.

UR5.2 Search problems

FR5.2.1 Users shall be able to search for a problem using the search bar.

FR5.2.2 Problem search shall be based on the problem name.

UR5.3 Filter problems

FR5.3.1 Users shall be able to filter a list of problems based on difficulty level, problem tags, and has user solved/not solved problems before.

UR5.4 Sort problems

FR5.4.1 Users shall be able to sort the list of problems by name alphabetically, by popularity, by points, and by difficulty.

FR5.4.2 Users shall be able to sort the list of problems in both ascending and descending order.

FR5.4.3 Sorting problems by popularity shall be based on the number of times the problem has been solved, with the most popular problem being the problem that has been solved the most times.

**FTR6** Problem solving: Users can view problem details, write a solution for the problem in the online code editor, upload their problem solution, run, and submit the solution, navigate to other problems, and view their previous submissions.

Priority: High

UR6.1 View problem details

FR6.1.1 Users shall be able to view the problem description panel, with the following information displayed: problem name, problem difficulty level, number of upvotes and downvotes, problem points, bookmark problem button, number of total submissions and accepted submissions for the problem, problem description text, sample input, and sample output.

FR6.1.2 Users shall be able to see a button to reveal the first and second hints.

FR6.1.3 Users shall be able to see a button to view the video solution for the problem.

FR6.1.4 Users shall be able to resize the problem description panel.

FR6.1.5 Users shall be able to minimize the problem description panel.

UR6.2 Write problem solution

FR6.2.1 Users shall be able to see and use an integrated code editor to write code.

FR6.2.2 The integrated code editor shall have line numbers displayed.

FR6.2.3 Users shall be able to choose a programming language in which they want to write a solution for the problem from the dropdown list.

UR6.3 Upload problem solution

FR6.3.1 Users shall be able to upload code and load it into the online code editor, by clicking the “Upload button” and selecting a file from the filesystem of their device.

FR6.3.2 File formats that can be uploaded shall be only the file formats of the source code written in one of the available programming languages that the user can select from the dropdown list.

FR6.3.3 File upload size for user code shall be limited to a maximum of 1MB in size.

UR6.4 Run problem solution

FR6.4.1 Users shall be able to click the “Run code” button which will run the code from the integrated code editor using the sample input.

FR6.4.2 Users shall be able to see the console output of their code execution in the console panel.

FR6.4.3 The console panel shall be empty by default.

UR6.5 Submit problem solution

FR6.5.1 Users shall be able to click the “Submit” button which will submit the code from the integrated code editor as a problem solution and run it with the test cases.

FR6.5.2 Users shall be able to see the results of their submission in the new “Results” tab in the console panel.

FR6.5.3 Results of the problem submission shall contain a list of all test cases for that problem, with each test case displaying the test case name, an indication of whether the submitted code passed the test case, runtime, and memory usage of the code execution with that test case.

UR6.6 View problem submissions

FR6.6.1 Users shall be able to open and view a list of their previous submissions for the current problem.

FR6.6.2 Each submission in the list shall display the submission date, programming language, and results of the problem submission as described in FR6.5.3.

FR6.6.3 Users shall be able to open and view a list of all previous submissions for all problems. This will be displayed on a separate page.

FR6.6.4 Each submission in the list of all submissions shall display the problem name, problem category, submission date, and results of the submission as described in FR6.5.3.

FR6.6.5 Both submission lists shall be sorted by submission date in descending order.

**FTR7** Report: Users can report problems, while admin users can view submitted reports.

Priority: Low

UR7.1 View problem reports

FR7.1.1 Admin shall be able to see a list of all problem reports, with the problem name, the username of the user that reported the problem, and the title of the report displayed for each report in the list.

FR7.1.2 Admin shall be able to view problem report details, with the problem name, username, and email of the user that submitted the report, report title, and full report text displayed.

FR7.1.3 Reports in the list of all problem reports shall be ordered by the submission timestamp, in descending order.

UR7.2 Submit the report

FR7.2.1 Users shall be able to submit the problem report from the problem page by providing the report title and report text.

**FTR8** Manage content: Admin users shall be able to view, add, and edit.

Priority: Medium

UR8.1 View categories, tags, and problems

FR8.1.1 Admin shall be able to see a list of all categories in the system, with category name and a total number of problems in the category displayed for each category in the list.

FR8.1.2 Admin shall be able to see a list of all tags for each category in the system, with the tag name.

FR8.1.3 Admin shall be able to click on a category or a tag to open a list of all problems in that category or tag, on a category overview page.

FR8.1.4 Admin shall be able to see problem name, problem summary, problem tags, problem difficulty level, and problem points for each problem in the list of all problems.

FR8.1.5 Admin shall be able to click on a problem in the list of problems and open a problem-solving page.

FR8.1.6 Admin shall not be able to write code in the online code editor, upload code, run or submit the code.

UR8.2 Add category

FR8.2.1 Admin shall be able to add a new category to the system by providing a category name, category points, and category color.

FR8.2.2 All categories must have unique names.

FR8.2.3 Admin shall be notified in case the new category name is not unique. This notification should be displayed as soon as the admin enters a non-unique category name, even before it is submitted.

FR8.2.4 All empty categories are marked as “hidden” by default.

UR8.3 Edit category

FR8.3.1 Admin shall be able to modify the name of every category.

FR8.3.2 Admin shall be able to modify category points by choosing a number of points from the dropdown menu.

FR8.3.3 Admin shall be able to mark any category as “hidden”.

FR8.3.4 Hidden categories shall be visible only to the admin and shall not be displayed to regular users.

FR8.3.5 Problems in hidden categories shall not be displayed to regular users.

FR8.3.6 Admin shall not be able to mark an empty category as “visible”.

UR8.4 Add tag

FR8.4.1 Admin shall be able to add a new tag to any existing category by providing a tag name.

FR8.4.2 All tags in a single category must have unique names.

FR8.4.3 Admin shall be notified in case the new tag name is not unique. This notification shall behave in the same way as the notification described in F8.2.3.

FR8.4.4 All empty tags are marked as “hidden” by default.

UR8.5 Edit tag

FR8.5.1 Admin shall be able to modify the name of any tag in any category.

FR8.5.2 Admin shall be able to mark any tag as “hidden”.

FR8.5.3 Hidden tags shall be visible only to the admin and shall not be displayed to regular users.

FR8.5.4 Admin shall not be able to mark an empty tag as “visible”.

UR8.6 Add problem

FR8.6.1 Admin shall be able to add a new problem to any existing category, by uploading an archive file that contains new problem information and data.

FR8.6.2 The archive file that is uploaded to add a new problem must be in ZIP file format.

FR8.6.3 The archive file that is uploaded to add a new problem must contain the following files: description.md file, problem.json file, test_cases.json file, and any additional files that might be required.

FR8.6.4 Description.md file shall contain problem description, sample inputs, and sample outputs written in Markdown markup language.

FR8.6.5 The file called problem.json shall contain the problem title and the exact name of the description.md file, problem difficulty, problem summary, video solution URL, exact name of test_cases.json file, number of test cases, list of tag IDs, points category ID, points activity ID, first hint, second hint, and a list of programming languages parameters in JSON format.

FR8.6.6 The list of programming language parameters in the problem.json file shall contain language ID, language time limit, language memory limit, compiler arguments, and runtime arguments for each language in the list.

FR8.6.7 The file named test_cases.json shall contain a list of all test cases in JSON format.

FR8.6.8 Each test case in the test_cases.json file shall contain a test case name, test case description, and list of all tests in that test case.

FR8.6.9 Each test in the test case shall contain test input, expected output, the input type, and output type values that define whether test input and the expected output are actual values or represent a file path of the file that holds actual values.

FR8.6.10 Admin shall be able to download the sample ZIP file, which will contain sample files required for new problem upload with necessary instructions that explain how to upload new problems.

UR8.7 Edit problem

FR8.7.1 Admin shall be able to mark any existing problem as “hidden”.

FR8.7.2 Hidden problems shall be visible only to the admin and shall not be visible to regular users.

FR8.7.3 Admin shall be able to edit problem name, summary, difficulty level, video URL, points, category, tags, first hint, and second hint in the online problem editor.

FR8.7.4 Admin shall be able to upload a new description.md file for the existing problem to replace the existing description.md file.

FR8.7.5 Admin shall be able to download the existing problem description.md file.

UR8.8 Remove problem

FR8.8.1 Admin shall be able to delete any existing problem.

**FTR9** Manage newsletter subscription: Users without an account can subscribe to a newsletter subscription.

Priority: Low

UR9.1 Subscribe to newsletter

FR9.1.1 Users shall be able to subscribe to Kodiraj.ba newsletter by entering their email address in the input field and clicking the button to submit their email address.

FR9.1.2 The notification message should be displayed to the user when the email is successfully added to the newsletter list.

UR9.2 Unsubscribe from newsletter

FR9.2.1 Users shall be able to unsubscribe from the newsletter.

FR9.2.2 Users shell be redirected to the web page with the message informing the user that they are unsubscribed after clicking the unsubscribe button.

## 5.2 Non-functional requirements

Non-functional requirements are constraints that are put on the development of the system to provide additional value to the software. In general, they define how a system is supposed to be, in contrast to functional requirements which define what is a system supposed to do. Non-functional requirements provide insurance that software will work as intended. Kodiraj.ba web application non-functional requirements are specified in Table 5.1.

Table 5.1: Non-functional requirements

| **Requirement** | **Definition** | **More details** |
|---|---|---|
| NFR1: Hardware Interface – I | Minimum hardware | OS: Ubuntu Linux 20.04<br>CPU: Any 8 core CPU or higher |

| | | |
|---|---|---|
| | requirements for the server that is hosting the application. | Disk space: 400GB<br>RAM: 16GB DDR4 |
| NFR2: Hardware Interface – II | Minimum hardware requirements for the client machine. | OS: Windows 8, Linux, Mac<br>Processor: Pentium 4<br>Disk space: -<br>RAM: 4GB DDR3 |
| NFR3: Software Interface | Software requirements for the system. | Database: PostgreSQL<br>Message broker: RabbitMQ<br>Virtualization: LXD virtualization manager |
| NFR4: Communication Interface | The user needs an internet connection to access the system via a web browser. | Internet connection. |
| NFR5: Availability | Availability of the applications for users. | AVL-1: System shall be always available.<br><br>AVL-2: Excluded from the calculation of availability is down-time that is reserved for maintenance which happens between 1:00 A.M. through 2 A.M. Central European Time once a week. |
| NFR6: Performance | Speed and responsiveness of the system, and the amount of traffic it can sustain. | PER-1: Response time for any action in the application shall not be more than 0.5 seconds.<br><br>PER-2: Code execution shall not take more than 20 seconds to complete.<br>PER-3: Code submission action shall not take more than 40 seconds to complete.<br><br>PER-4: The application should handle a maximum of 100 queries in 1 second.<br><br>PER-5: The application shall handle at most five code submissions at the same time within 40 seconds. |
| NFR7: Usability | How easy is it to use the application? | USY-1: Users are expected to be able to navigate and use the application after 20 minutes of usage. |
| NFR8: Security | Security expectations of the system. | SEC-1: Only admin users can manage content (categories, tags, and problems) in the application. |

| | | |
|---|---|---|
| | | SEC-2: User-submitted code shall not be able to affect the main hosting server and should be completely isolated.<br><br>SEC-3: Virtualization containers are not persisted after code execution is completed.<br><br>SEC-4: Only authenticated users shall be able to execute and submit code. |
| NFR9: Design and implementation constrains | Design and implementation constraints. | CON-1: The application should be supported for Firefox, Chrome, Brave, Safari, and Opera.<br><br>CON-2: Relational database shall be used. |
| NFR10: Internationalization and localization | System availability in different languages and accessibility. | IL-1: The application should be available in the Bosnian language. |

# CHAPTER 6 **SYSTEM DESIGN**

The system design chapter explores the architecture and the implementation of the system in detail. Backend and frontend architectures are discussed separately and the reasoning behind choosing the architectures in question is provided. Technical details about system implementation are discussed by decomposing the system into modules and exploring each module and its implementation. Communication between different system components is explained. Database structure and diagram are presented in this chapter. Subsequently, use cases and actors in the system are listed and visualized using the use case diagram. To emphasize the crucial system flows sequence diagrams are shown and discussed. Finally, parts of the user interface are introduced to complete the picture of the system and the web application.

## 6.1 Backend architecture

The architecture of the backend part of the Kodiraj.ba web application can be classified as a hybrid between monolithic and microservices architectures. The main backend server that includes REST API for the frontend application is designed as a monolithic application. The code execution engine is designed as a cluster of microservices. Monolithic architectures are generally easier to understand and maintain and have a simpler deployment process [38]. Monolithic architecture implies that all functionality is encapsulated into a single application. This means that modules of the application cannot be executed independently because everything is tightly coupled. Microservices refer to the architecture of the system where each module of the system is developed as an independent service or application. Microservices can even be developed using different technologies and being hosted on different machines. They

usually communicate over message brokers. This architecture provides extreme scalability and handling of higher traffic [39].

In the case of Kodiraj.ba web application code execution microservices communicate with the main backend server via RabbitMQ message broker and REST API. RabbitMQ is used to queue execution requests for code execution microservices, while REST API is used to return the results of the execution from the microservice to the main backend server. This microservice architecture of the code execution engine allows for a theoretically unlimited number of code executions at the same time if enough instances of the service are running.

### 6.2 Frontend architecture

Kodiraj.ba application is developed as a single-page application that uses client-side rendering and Redux as state manager. A single-page application (SPA) is a web application in which the content of the page is dynamically changed after data is fetched from the server, without refreshing the whole page. This is closely related to the client-side rendering which is the process of rendering pages directly in the browser, on the client-side, rather than generating these pages on the server-side and sending them to the client, which is known as server-side rendering. One of the main benefits of this approach is that all resources needed for page displaying are loaded only once, at the first application render, and thus it decreases the loading time for the subsequent request. This also consumes less user bandwidth and provides a better user experience. However, this approach also has some drawbacks, such as the use of more browser resources and more complex state managing on the client-side. To avoid the issue of complex state managing, the Redux library is used.

Redux stores the global application state in an object called *store*. Parts of the state are then accessed by UI components, depending on component functionality. To change a specific part of the state, from component *action creator* function needs to be dispatched. This *action creator* function then sends the *action* object to *reducers* functions. Action objects typically consist of two properties, *type,* and *payload. Reducers* update application state based on received action type and payload. Subsequently, components that use the state from the store are re-evaluated and re-rendered. The whole process is visualized in Figure 6.1 [40].

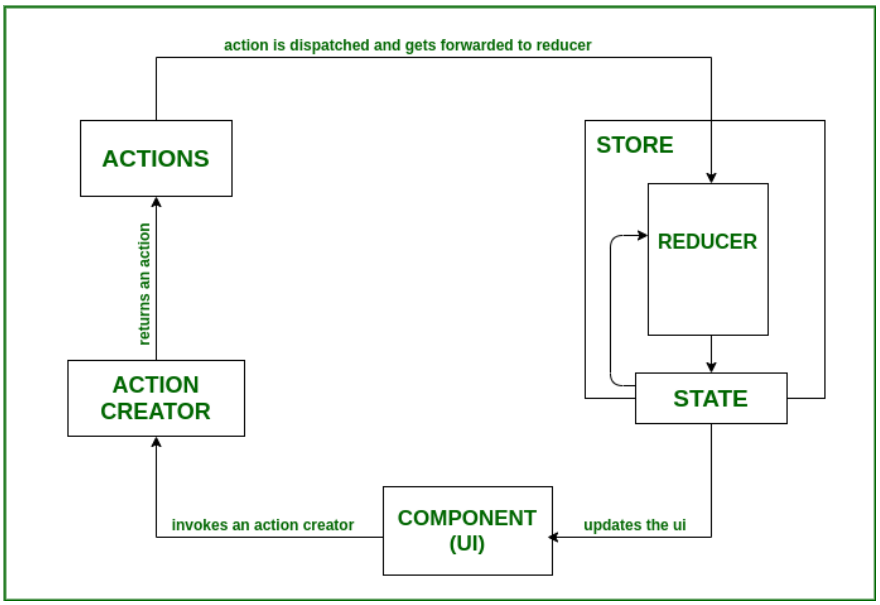


Figure 6.1: Redux state flow

### 6.3 System Decomposition

Kodiraj.ba is an online judge web application that allows its users to solve programming problems and evaluate their solutions concerning correctness, execution time, and memory performance. The content of the Kodiraj.ba web application consists of programming problems that are grouped into categories and tags. Each problem can belong to only one category. Each category has its tags. One problem can have multiple tags. The user is expected to browse categories and problems, find a problem to solve

and solve the problem. Aside from the main functionalities, the user is provided with functionalities like problem bookmarks and problem notes, which can be grouped into user account management functionalities. Content in the Kodiraj.ba web application is managed by administrator users through the content management system included in the web application.

To achieve a better understanding of the system it can be broken down into five separate modules: authentication and authorization, content management, code execution, account management, and the database module. Exploring these modules one at a time will allow for a more detailed and clearer understanding.

### 6.3.1 Authentication and authorization

Kodiraj.ba web application implements two different ways of authenticating users. Firstly, users can authenticate using their account email and password. User entered email and passwords are securely sent from the web browser to the backend web server using encrypted HTTPS protocol. On the server, email and password are compared to the email and encrypted password stored in the database. Additionally, users can authenticate using their Google accounts. In case of successful authentication, the user is provided with the JSON web token (JWT) which is further used for user authorization.

JWT provided to users after successful authentication contains encrypted user identifiers with user roles and permission levels which allow granular access control. With each request to the backend web server users provide their JWT which is decrypted by the web server and lets the server know who is making a request and what that user can or cannot access.

### 6.3.2 Content management

The content management module of the application consists of all functionalities related to creating and editing categories, tags, and problems. These functionalities are only available to administrator users. Categories and tags are created and edited using the UI in the web application. The administrator can mark both categories and tags as hidden. Marking categories or tags as hidden makes them invisible to regular users. Moreover, all problems in hidden categories or tags are also invisible to regular users.

Problems are created by uploading a problem .zip file through the UI in the web application. Uploaded compressed problem file contains *problem.json*, description.md, *test_cases.json* files, and additional resource files as needed. File *problem.json* contains general problem information like problem category, tags, difficulty, solution video URL, etc. File *test_cases.json* contains information about tests and test cases. It defines each test case, all tests for each test case, and specifies inputs and outputs for each test. Inputs and outputs for tests can be small and specified directly in the JSON file. In cases where inputs or outputs are massive, they are stored in text files as additional resources, and the file path is specified in the JSON file. File description.md is a markdown markup file that defines problem description that will be directly shown to users when they open the problem-solving page.

The uploaded compressed problem file is decompressed on the backend and all the files inside are processed. File description.md is kept as a file on the storage file system. The rest of the files are read, parsed, and processed into the database. At this point, a new problem is created and connected to all other objects in the database like the appropriate database, all the tags, etc.

Problems are edited through the web application UI. General problem details are edited through pop-up modals, while problem description is changed by uploading a new description.md file. The administrator can also download the existing problem description.md file for the convenience of editing. The administrator can mark any problem as hidden, which makes it completely invisible to regular users.

### 6.3.3 Account management

The account management part of the application contains all the functionalities that are connected to the user account and user navigation towards problem-solving. Main functionalities are accessible to user profile, editing of a user profile, password reset, bookmarking problems, viewing bookmarked problems, viewing problems that the user started solving, searching, sorting, and filtering problems, and browsing categories. Implementation of these functionalities, even though important for the overall application, is not technically interesting, hence, details will be omitted in favor of topics like code execution engine.

### 6.3.4 Code execution

The code execution module of the Kodiraj.ba web application consists of two main functionalities: user code execution and user code submission. User code execution refers to the user request to just execute the solution against sample input without running all test cases. This user request is designed for situations where the user is not ready to submit the code but wants to execute it and see the result of the execution. User code submission refers to the situation where the user is done with the solution and wants to submit the solution by running it against all test cases. In the first case, code is executed only once, while in the second case code is executed once per the test, where

the problem can have many tests. Due to the large similarity of these functionalities, only the code submission will be explained. Code submission flow from the user opening the problem page to the user receiving submission results is illustrated in Figure 6.2.

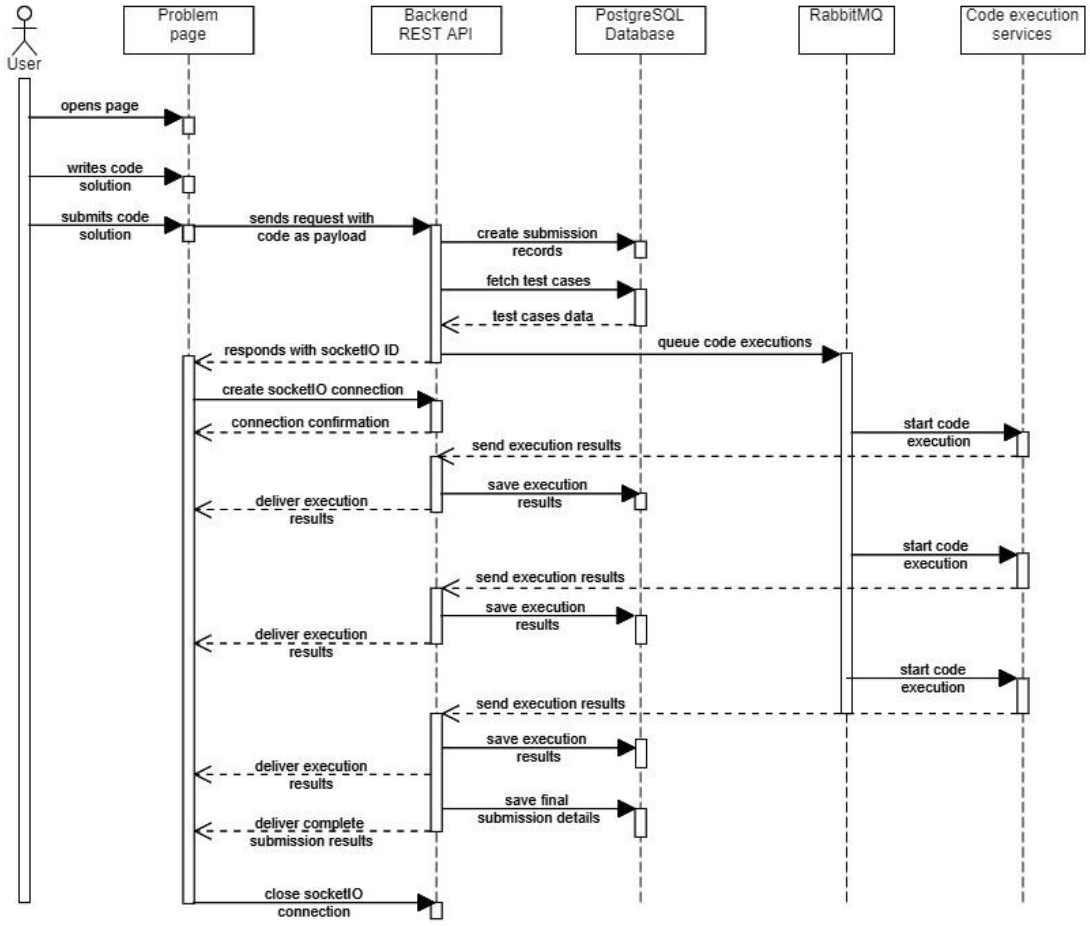


Figure 6.2: Sequence diagram of code submission

Code submission flow starts with the user opening the problem page. On the problem page, users can view the problem description and they can write the solution in the online code editor. When the solution is ready user submits the solution. Submit solution action sends users' code as a file payload to the backend web server via REST API. Backend receives the submitted code and saves it to the storage file system. Then, the backend creates submission records in the database and reads test case information

from the database. When test cases information is loaded backend iterates through all tests from all test cases and for each test it queues the execution of the solution with that test. Test execution is queued by sending a message to the RabbitMQ queue. Message sent to the RabbitMQ contains details about the location of the user-submitted code on the file system, test input, and any additional parameters relevant to the code execution. All executions are queued at once, and then the backend responds to the client with the information on how to connect and register to the SocketIO web socket. A web socket connection is used for the backend to be able to deliver execution results to the client in real-time.

Execution messages that were queued in the RabbitMQ queue by the backend server are consumed by code execution services. Code execution services are Python services that are running on the server with the task of consuming messages from RabbitMQ and performing code execution for each queued message. Python code execution services are developed to be standalone services, independent of the main backend server and independent of each other. Multiple Python code execution services can be running on the server at the same time. This microservice architecture allows for many code executions at the same time.

Python execution services execute user code with help of Bash shell scripts. Bash shell scripts are helper scripts that collect all files required files for the execution, launch and configure the LXC container, transfer all files from the storage file system into the LXC container file system, install dependencies into the container, remove the network interface from the LXC container, set timeout to kill the execution, and finally start execution of the code inside the LXC container. Furthermore, the Bash script is responsible for collecting the code output from the container file system, collecting

performance metrics, saving them into the file, and in the end deleting the LXC container. After the Bash script is done executing, the Python execution service collects output and performance metrics from the files on the storage file system and creates POST HTTP requests to the main backend server to deliver the execution results.

The backend collects execution results that come in via HTTP requests, saves the to the database, and forwards them to the client on the frontend via a web socket connection. When the last execution result comes into the REST API server, all results are grouped, final submission details are written to the database, and everything is sent to the frontend over a web socket.

### 6.3.5 Database

Figure 6.3 illustrates the database schema. This figure is added with purpose of the paper completeness, while the high-quality illustration of the diagram is available with the digital version of the paper.

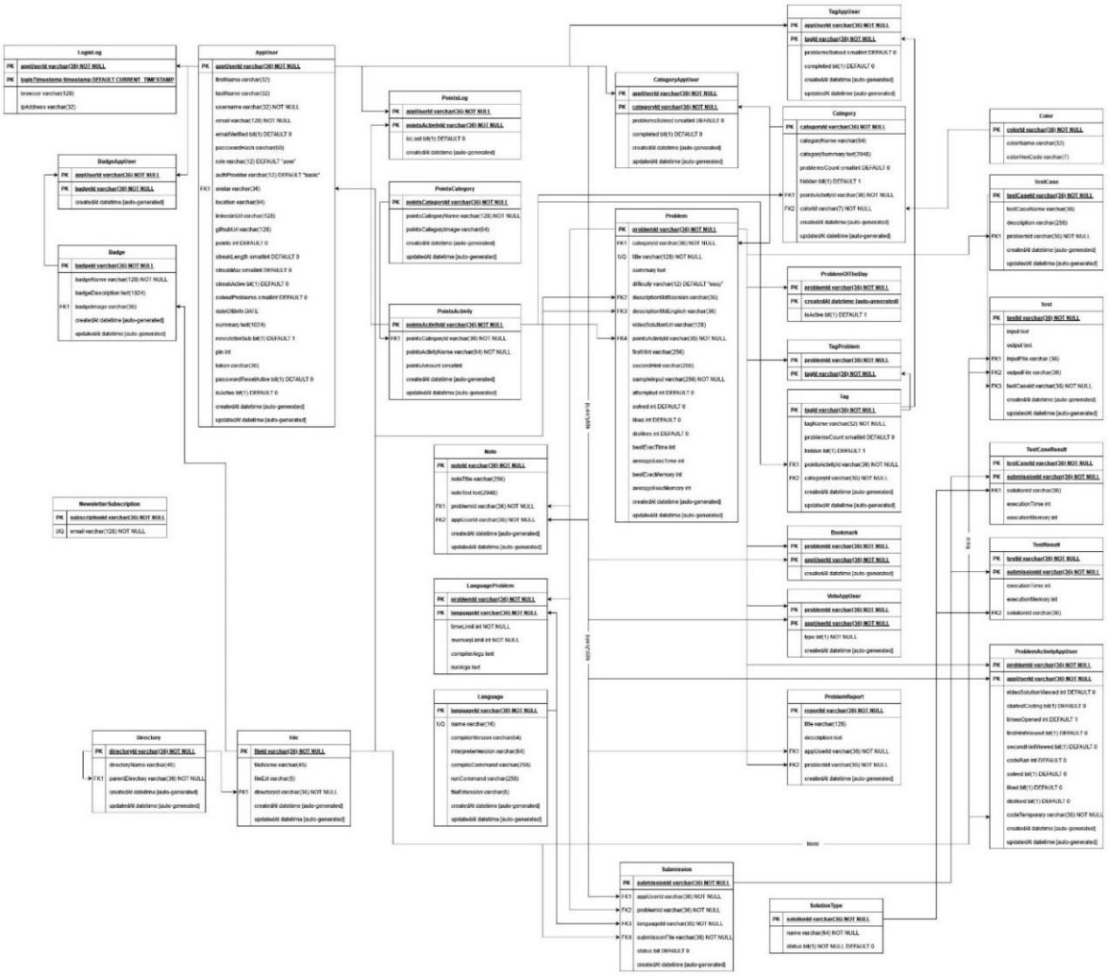

Figure 6.3: Database schema

### 6.4 Use case modeling

This section will present flows of use cases derived from user requirements. Use cases are grouped into three groups, based on the type of the user: Guest user use cases, Logged-in user cases, and Administrator use cases. A guest user is a user that is not logged in and that has restricted access to all main system functionalities. Logged-in user is a user that has created an account and is successfully authenticated. This type of user can use most system functionalities. Lastly, the administrator is a user type that has access to all system functionalities and is mainly responsible for content

management. In Figure 6.4 relationship between all use cases and user, types are visualized.

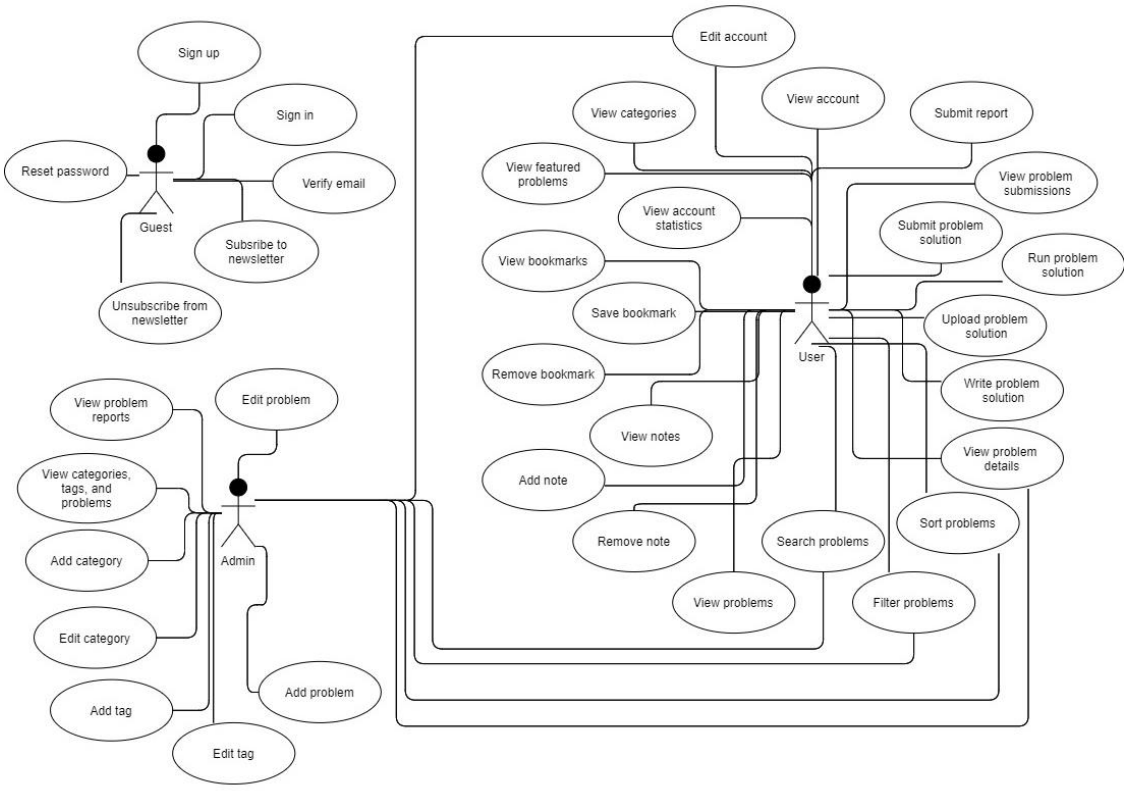


Figure 6.4: Use case diagram

### 6.4.1 **Guest user use-case scenarios**

*UC-1.1: Create account*

**Description:** A guest user can create an account.
**Priority:** High
**Preconditions:** Server is online; Guest user is connected to the internet
**Postconditions:** An account with the user's information is created in the database.
**Basic Flow:**

1. User clicks on Sign up.
2. System loads the Sign-up page.
3. User fills out the necessary data needed to submit.
4. User clicks on the Sign-up button.
5. System creates the account with the given information by adding the information to the database.
6. System redirects the user to the dashboard page.

**Alternate flow:**

1. User clicks on Sign Up with Google Account.
2. System opens a new window to complete the sign-up with a Google Account.

3. User fills out the necessary data needed to log into Google Account and clicks on login.
4. User agrees to give access to Google Account Information.
5. System creates the account with the given information by adding the information to the database.
6. System redirects the user to the email verification page.

**What can go wrong:** The user inputs data in the wrong format in the required fields. The system displays an error message prompting the user to correct the input data.
**Assumptions:** Admin will have their account created by the development team.

*UC-1.2: Verify email*

**Description:** User can verify their email.
**Priority:** High
**Preconditions:** Server is online; Guest user is connected to the internet; User created an account on the sign-up page
**Postconditions:** User's email is verified.
**Basic Flow:**

1. After creating an account, the user will receive an email that contains a verification link.
2. User clicks the verification link.
3. User is redirected to the login page and the system displays a notification message saying that the email is verified.

**What can go wrong:** Users may open the verification link after they already verified the email. In this case, the system will display a message saying that the email is already verified and redirect them to the landing page.

*UC-1.3: Log into account*

**Description:** User logs into their account using the email and password they used when they were signing up.
**Priority:** High
**Preconditions:** Server is online; Guest user is connected to the internet; User is not logged in; Email is verified
**Postconditions:** An account with the user's information is created in the database.
**Basic Flow:**

1. User clicks on Login.
2. System loads the Login page.
3. User enters the necessary information.
4. User clicks the Login button.
5. System checks the information and if it is correct, lets the user log in.
6. System redirects the passenger to the dashboard page.

**Alternate flow:**

1. User clicks on Login.
2. System loads the Login page.
3. Passenger clicks on Login using Google account.
4. User, if not already logged in, enters information to log into their Google account.
5. System logs in the user.

6. System redirects the user to the dashboard page.

**What can go wrong:** The user does not enter all required information in the input fields or inputs incorrect information. The system displays an error message and prompts the user to enter the needed or valid information. Additionally, the user may not have verified their email and thus will not be able to log in. The system will display the appropriate error message.

*UC-9.1: Subscribe to the newsletter*

**Description:** Guest users can subscribe to the newsletter.
**Priority:** Low
**Preconditions:** Server is online; Guest user is connected to the internet; User is not logged in
**Postconditions:** Guest user is subscribed to the newsletter.
**Basic Flow:**

1. Guest user enters their email in the input field on the landing page.
2. Guest user clicks submit button.
3. System adds the email to the newsletter emails list in the database.
4. System displays a message saying that the guest user is successfully subscribed to the newsletter.

**Alternate flow:** -
**What can go wrong:** Entered email may already be added to the newsletter list. The system will display the appropriate error message.

*UC-9.2: Unsubscribe to the newsletter*

**Description:** User can unsubscribe from the newsletter.
**Priority:** Low
**Preconditions:** Server is online; Guest user is connected to the internet
**Postconditions:** User is unsubscribed from the newsletter.
**Basic Flow:**

1. User clicks on the unsubscribe button in the newsletter email in their email inbox.
2. Link will redirect the user to the landing page of the system.
3. System will display a message saying that the user is successfully unsubscribed from the newsletter.

**What can go wrong:** The user may already be unsubscribed from the newsletter. The system will display an appropriate error message.

*UC-1.4: Reset password*

**Description:** User can reset their password.
**Priority:** Low
**Preconditions:** Server is online; Guest user is connected to the internet
**Postconditions:** User's password is reset.
**Basic Flow:**

1. On the login page, the user clicks the "Forgot password" button.
2. System displays reset password page.
3. User enters their email in the input field.

4. System sends password reset link on their email.
5. User opens a link from the email they received.
6. System displays page for password reset with necessary input fields.
7. User enters a new password.
8. System changes the user's password and redirects the user to the login page.

**What can go wrong:** The user may not enter values in certain required fields or may enter invalid values. The system will display appropriate error messages.

### 6.4.2 **Logged-in user use-case scenarios**

*UC-1.5: View account*

**Description:** User can see their account information
**Priority:** Medium
**Preconditions:** Server is online; User is connected to the internet; User is logged in
**Postconditions:** Account information is displayed.
**Basic Flow:**

1. User clicks on the profile link on the side navigation menu.
2. System displays profile page with account information.

Use case *UC-8: View account statistics* is like *UC-1.5: View account* since user statistics is also displayed on the profile page, below account information.

*UC-1.6: Edit account*

**Description:** User can edit their account which allows them to change the account information.
**Priority:** Medium
**Preconditions:** Server is online; User is connected to the internet; User is logged in
**Postconditions:** The user's account information is changed.
**Basic Flow:**

1. User clicks on the Settings link on the side navigation menu.
2. System displays settings page.
3. User enters the necessary information that they want to change.
4. User clicks the save button.
5. System saves new account information to the database.
6. System displays a message saying that account information is changed successfully.

**Alternate flow:**

1. User clicks on the edit button on the profile page.
2. System displays the settings page and the user continues with normal flow.

**What can go wrong:** The user may not enter required input fields or may enter invalid data. Changes to account information will not be made and the system will display an appropriate error message.

*UC-3.1: View bookmarks*

**Description:** User can see list of bookmarked problems

**Priority:** Low
**Preconditions:** Server is online; User is connected to the internet; User is logged in
**Postconditions:** List of bookmarked problems is displayed
**Basic Flow:**
1. User clicks on bookmarked problems link on the side navigation menu.
2. System displays the list of bookmarked problems.

**Alternate flow: -**
**What can go wrong:** The user may not have any bookmarked problems. The system will display the appropriate message.

*UC-3.2: Save bookmark*

**Description:** User can bookmark a problem.
**Priority:** Low
**Preconditions:** Server is online; User is connected to the internet; User is logged in; Problem is not bookmarked
**Postconditions:** Problem is added to the user's bookmark list.
**Basic Flow:**
1. On the problem page, the user clicks the bookmark button.
2. System adds that problem to the user's bookmark list.

**Alternate flow:**
1. User bookmarks problem by clicking the bookmark button on the list of featured problems.
2. System adds that problem to the user's bookmark list.

*UC-3.3: Remove bookmark*

**Description:** User can remove the bookmark from the problem.
**Priority:** Low
**Preconditions:** Server is online; User is connected to the internet; User is logged in; Problem is bookmarked
**Postconditions:** Problem is removed from the user's bookmark list.
**Basic Flow:**
1. On the problem page, the user clicks the bookmark button.
2. System removes that problem from the user's bookmark list.

**Alternate flow:**
1. User removes the bookmark from the problem by clicking the bookmark button on the list of featured problems.
1. System removes that problem from the user's bookmark list.

*UC-4.1: View notes*

**Description:** User can see a list of their notes.
**Priority:** Medium
**Preconditions:** Server is online; User is connected to the internet; User is logged in
**Postconditions:** List of users' notes are displayed.
**Basic Flow:**
1. User clicks on the notes link on the side navigation menu.
2. System displays a notes page with a list of general and problem notes.

**Alternate flow:**

3. User click note button on the problem page
4. System displays modal with the note for that problem.

**What can go wrong:** The user may not have any added notes. The system will display an appropriate message.

*UC-4.2: Add note*

**Description:** User can add a note
**Priority:** Medium
**Preconditions:** Server is online; User is connected to the internet; User is logged in
**Postconditions:** New note is added.
**Basic Flow:**

1. User clicks on the notes link on the side navigation menu.
2. System displays a notes page with a list of general and problem notes.
3. Users click the “Add new note” button.
4. System displays input fields.
5. User enters necessary information in input fields and clicks submit button.
6. System adds a new note to the list of general notes.

**Alternate flow:**

1. On the problem page, the user clicks the note button.
2. System displays modal with input fields
3. User enters necessary information in input fields and clicks submit button.
4. System adds a new note to the list of problem notes.

**What can go wrong:** The user may not fill all required input fields. The system will display the appropriate error message.

*UC-4.3: Remove note*

**Description:** User can remove the specific note.
**Priority:** Medium
**Preconditions:** Server is online; User is connected to the internet; User is logged in
**Postconditions:** Note is removed.
**Basic Flow:**

1. User clicks on the notes link on the side navigation menu.
2. System displays a notes page with a list of general and problem notes.
3. User clicks on the delete button of a specific note.
4. System removes notes from the database.

**Alternate flow:**

1. User click the note button on the problem page
2. System displays modal with the note for that problem.
3. User clicks the delete button.
4. System removes notes for that problem.

*UC-2.1: View categories*

**Description:** User can see a list of available categories.
**Priority:** High
**Preconditions:** Server is online; User is connected to the internet; User is logged in

**Postconditions:** List of available categories for the user is displayed.
**Basic Flow:**

1. User clicks on the home page link on the side navigation menu.
2. System displays the home page of the dashboard with the list of available categories.

**What can go wrong:** The system may not have any added categories. The appropriate message will be displayed.

Use case UC-2.2: *View featured problems* is like *UC-2.1: View categories* since the list of featured problems is displayed below the list of available categories.

*UC-5.1: View problems*

**Description:** User can see a list of problems in the selected category.
**Priority:** High
**Preconditions:** Server is online; User is connected to the internet; User is logged in
**Postconditions:** List of problems for the specific category is displayed.
**Basic Flow:**

1. User selected a category from the list of categories on the dashboard home page.
2. System displays a category page with category information and a list of category problems

**What can go wrong:** The category may not have any problems. This can only be the case for the admin user. The system will display an appropriate message.

*UC-5.2: Search problems*

**Description:** User can search problems in a specific category by problem names.
**Priority:** Medium
**Preconditions:** Server is online; User is connected to the internet; User is logged in
**Postconditions:** Filtered list of problems is displayed.
**Basic Flow:**

1. On the category page, the user enters the problem name in the search input field.
2. System displays the list of problems that match the user input.

**What can go wrong:** There may not be any problem that matches the user's query. The system will display the appropriate message.

Use cases *UC-5.3: Filter problems* and *UC-5.4: Sort problems* are like use case *UC-5.2: Search problems* since these functionalities are displayed on the same page and have a similar purpose.

*UC-6.1: View problem details*

**Description:** User can open a problem page for a certain problem and see problem details such as description, difficulty level, number of upvotes and downvotes, number of total and accepted submissions, and similar.
**Priority:** Medium
**Preconditions:** Server is online; User is connected to the internet; User is logged in
**Postconditions:** User can see details for the selected problem.
**Basic Flow:**

1. User selects problem from problem list on the category page.
2. System displays problem page with problem details.

**Alternate flow:**

1. User selects problem from a list of featured problems on the dashboard page.
2. System displays problem page with problem details.

*UC-6.2: Write problem solution*

**Description:** User can write code solutions for the specific problem.
**Priority:** High
**Preconditions:** Server is online; User is connected to the internet; User is logged in
**Postconditions:** User sees their input displayed in the code editor.
**Basic Flow:**

1. On the problem page, the user clicks on the code editor to focus it.
2. User inputs code.
3. System displays the user's input in the code editor.

*UC-6.3: Upload problem solution*

**Description:** User can upload the file with a code solution.
**Priority:** Medium
**Preconditions:** Server is online; User is connected to the internet; User is logged in
**Postconditions:** User sees their code displayed in the code editor
**Basic Flow:**

1. On the problem page user click the "Upload code" button.
2. Browser displays a popup window prompting the user to select a file with the appropriate file format from their machine.
3. User selects a file from their machine.
4. System displays code from the file in the code editor.

**Alternate flow:** -
**What can go wrong:** The user may select the file that is too large. The system will exit the normal flow and display an appropriate error message.

*UC-6.4: Run problem solution*

**Description:** User can run the code solution with provided sample inputs.
**Priority:** High
**Preconditions:** Server is online; User is connected to the internet; User is logged in
**Postconditions:** User can see the console output of their code execution in the console panel.
**Basic Flow:**

1. User clicks the "run code" button.
2. Code is sent to the server and executed with the code execution engine using the problem's sample inputs.
3. Server responds with console output data.
4. System displays console output in the console panel.

*UC-6.5: Submit problem solution*

**Description:** User can submit their code and see test case results.
**Priority:** High
**Preconditions:** Server is online; User is connected to the internet; User is logged in
**Postconditions:** User can see test case results in the results panel.
**Basic Flow:**

1. User clicks the “Submit code” button.
2. Code is sent to the server and executed with the code execution engine against the problem’s test cases.
3. Server responds with outcomes for each test case.
4. System displays test case results on the Results panel.

*UC-6.6: View problem submissions*

**Description:** User can see a list of all problem submissions as well as the list of submissions for the specific problem.
**Priority:** High
**Preconditions:** Server is online; User is connected to the internet; User is logged in
**Postconditions:** List of problem submissions are displayed.
**Basic Flow:**

1. User clicks on the problem submissions link on the side navigation menu.
2. System displays the list of all submissions for all problems with necessary information.

**Alternate flow:**

1. On the problem page, the user selects the submission tab.
2. System displays the list of all submissions for that problem.

**What can go wrong:** The user may not have any submissions. The system will display an appropriate message.

*UC-7.2: Submit the report*

**Description:** User can report the problem.
**Priority:** Low
**Preconditions:** Server is online; User is connected to the internet; User is logged in
**Postconditions:** Report is sent to the admin.
**Basic Flow:**

1. User clicks on the report button on the problem page.
2. System displays report modal with input fields.
3. User enters the necessary information and clicks submit button.
4. System saves the report to the database and closes the modal.

**What can go wrong:** The user may not enter values in certain required fields or may enter invalid values. The system will display appropriate error messages.

### 6.4.3 **Administrator user use-case scenarios**

*UC-7.1: View reports*

**Description:** Admin can see problem reports submitted by users.
**Priority:** Low

**Preconditions:** Server is online; Admin is connected to the internet; Admin is logged in
**Postconditions:** Problem reports are displayed.
**Basic Flow:**

1. Admin clicks on the reports link on the side navigation menu.
2. System displays report page.
3. Admin can see a list of all reports with report title, problem name, date of report, and name of the user that wrote the report.
4. Admin can click on a specific report to expand it.
5. System will display the text of the clicked report.

**What can go wrong:** There may not be any reports. The system displays a message saying that there are no reports.

*UC-8.1: View categories, tags, and problems*

**Description:** Admin can see a list of categories, tags, and problems
**Priority:** High
**Preconditions:** Server is online; Admin is connected to the internet; Admin is logged in.
**Postconditions:** List of categories, tags, and problems are displayed
**Basic Flow:**

1. Admin opens the home dashboard page.
2. System displays the list of categories along with the list of tags for each category.
3. Admin clicks on a specific category.
4. System displays a category page with a list of problems for that category.

**What can go wrong:** There may not be any problems in the category. The system displays a category page and message saying that there are no problems in that category.

*UC-8.2: Add category*

**Description:** Admin can add a new category to the system.
**Priority:** High
**Preconditions:** Server is online; Admin is connected to the internet; Admin is logged in
**Postconditions:** New category is added.
**Basic Flow:**

1. Admin opens the home dashboard page.
2. Admin clicks the “Add new category” button.
3. System displays modal for adding new category.
4. Admin enters necessary information.
5. Admin clicks submit button.
6. System adds a new category to the database and displays a success message.

**What can go wrong:** Admin may not enter values in certain required fields or may enter invalid values. The system will display appropriate error messages.

*UC-8.3: Edit category*

**Description:** Admin can edit information for certain categories.

**Priority:** Medium
**Preconditions:** Server is online; Admin is connected to the internet; Admin is logged in
**Postconditions:** Category information is changed.
**Basic Flow:**

1. Admin opens the home dashboard page.
2. System displays the list of categories.
3. Admin clicks on the edit button of the specific category.
4. System displays edit modal with information of selected category.
5. Admin edit category information and clicks submit button

**What can go wrong:** Admin may not enter values in certain required fields or may enter invalid values. The system will display appropriate error messages.

Use cases *UC-8.4: Add tag* and *UC-8.5: Edit tag* are like *UC-8.3: Edit category* since these functionalities are displayed in the same modal.

*UC-8.6: Add problem*

**Description:** Admin can add new problems to certain categories.
**Priority:** High
**Preconditions:**

1. Server is online.
2. Admin is connected to the internet.
3. Admin is logged in.
4. Category is created.

**Postconditions:** New problem is added to the category.
**Basic Flow:**

1. Admin opens the home dashboard page.
2. Admin clicks the “Add new problem” button.
3. Browser displays popup window prompting the user to select problem file with appropriate file format from their machine.
4. Admin selects a file from their machine.
5. System adds problems to the database and displays a success message.

**What can go wrong:** Admin may select the file that is too large. The system will exit the normal flow and display the appropriate error message.

*UC-8.7: Edit problem*

**Description:** Admin can edit problem information.
**Priority:** Medium
**Preconditions:**

1. Server is online.
2. Admin is connected to the internet.
3. Admin is logged in.

**Postconditions:** Problem information is changed.
**Basic Flow:**

1. On the category page, the admin clicks on the edit button of the specific problem from the problem list.

2. System displays a modal that has input fields with already existing problem information.
3. Admin changes problem information and clicks submit button.
4. System saves new information to the database and displays a success message.

**What can go wrong:** Admin may not enter values in certain required fields or may enter invalid values. The system will display appropriate error messages.

### 6.5 User interface

From the end-user point of view, the user interface represents the most important part of the application since it provides human-computer interaction. Kodiraj.ba application provides a simple and consistent user interface, which is achieved by implementing a minimalist design approach and the use of common UI patterns across the application. Moreover, the application aims to use typography and color palette in a way that it can direct or redirect the user's attention to necessary parts but still provide a smooth and sleek design. Additionally, some other rules of interface design are followed, as mentioned in Shneiderman's eight golden rules of interface design: dialogs to yield closure, offering informative feedback, and reducing short-term memory load [41]. In this section, selected representative pages of the application are be presented and explained in terms of mentioned interface rules and guidelines. These selected pages are dashboard, category page, problem page and edit problem modal.

#### 6.5.1 Category page

In Figure 6.5 an example of a category page is shown. The category page is used to display basic information about the category and a list of category problems. As shown in the example of problem difficulties, color coding is used to simplify the user's recognition process and ease the use of the application. Information overload is avoided to make the page clear and not overwhelming. Additionally, on the left side of the page, the navigation side menu is positioned, which is the case with most pages of the

application. This is done with the intention to facilitate user navigation through the application and provide consistency.

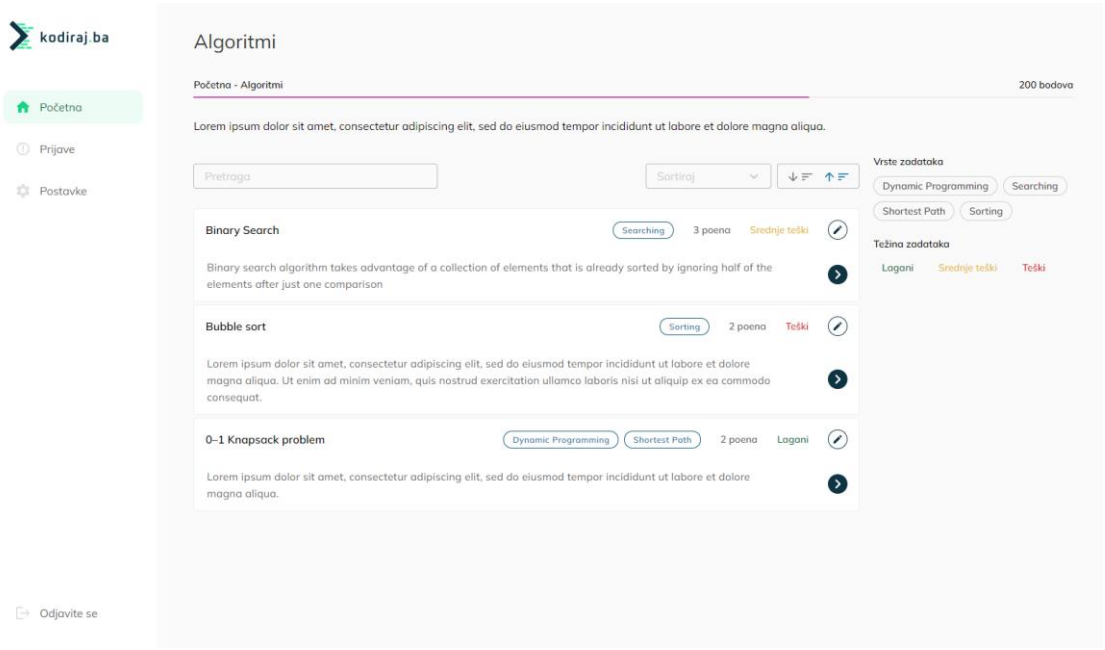


Figure 6.5: Category page

### 6.5.2 Edit problem modal

Edit problem modal, shown in Figure 6.6, is used to edit some basic information about the problem. Overall, through the application, modal windows are used extensively to reduce short-term memory load and user's navigation between pages yet provide a simple and clean design with no information overload. This is also a great example of input field usage, where different types of input fields (inputs, text areas, and dropdowns) follow uniform design patterns but still have enough details to be differentiated from each other. Additionally, buttons that trigger certain actions are emphasized with appropriate accent colors, depending on their type and importance level (primary and secondary), which makes them easy to spot and recognize across the

application and provides nice-looking aesthetics for the overall design of the application.

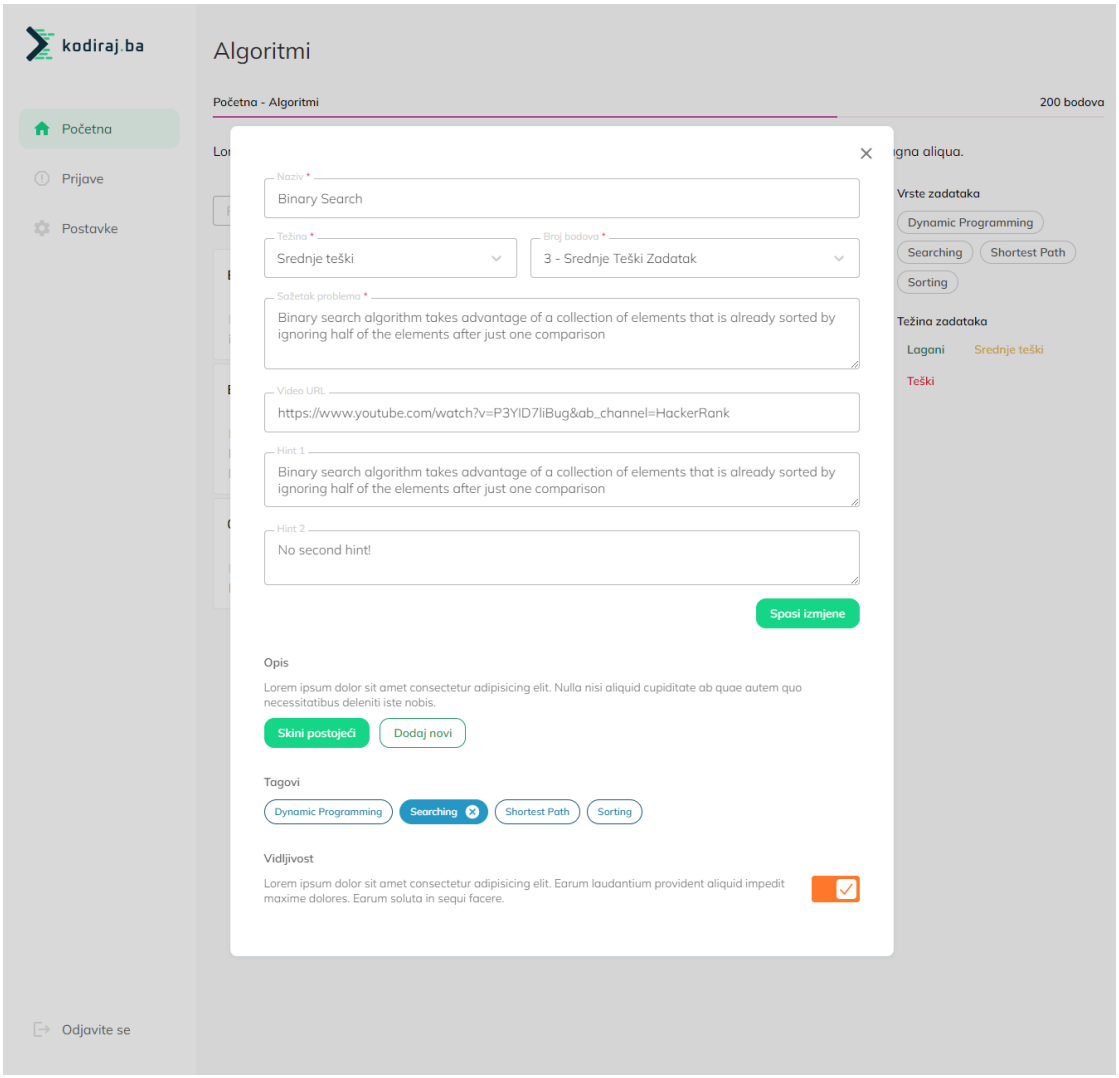


Figure 6.6: Edit problem modal

### 6.5.3 Problem page

Although the problem page, displayed in Figure 6.7, provides more information compared to some other pages of the application, it still represents a clean design. This page has the necessary information needed for the user to understand the problem, code editor, console, and results tabs as well as other required functionalities defined in system features. Due to numerous details and components on this page, it has

responsive functionalities to improve the user experience. Thus, the left side of the page, with the problem information, can be resized or completely minimized to expand the code editor area. Additionally, console and results tabs can be expanded or collapsed, depending on the user's needs and preferences. Similar to other pages, color coding is used for problem difficulties and category names. Moreover, the code editor has syntax highlighting which provides color-coding of different patterns and keywords in the code that, in turn greatly improves code readability. Another feature of the code editor is line numbering and code collapsing and expanding, which helps users navigate the code and find the parts of code that they are interested in. All these features are implemented in the code editor to provide the best coding experience.

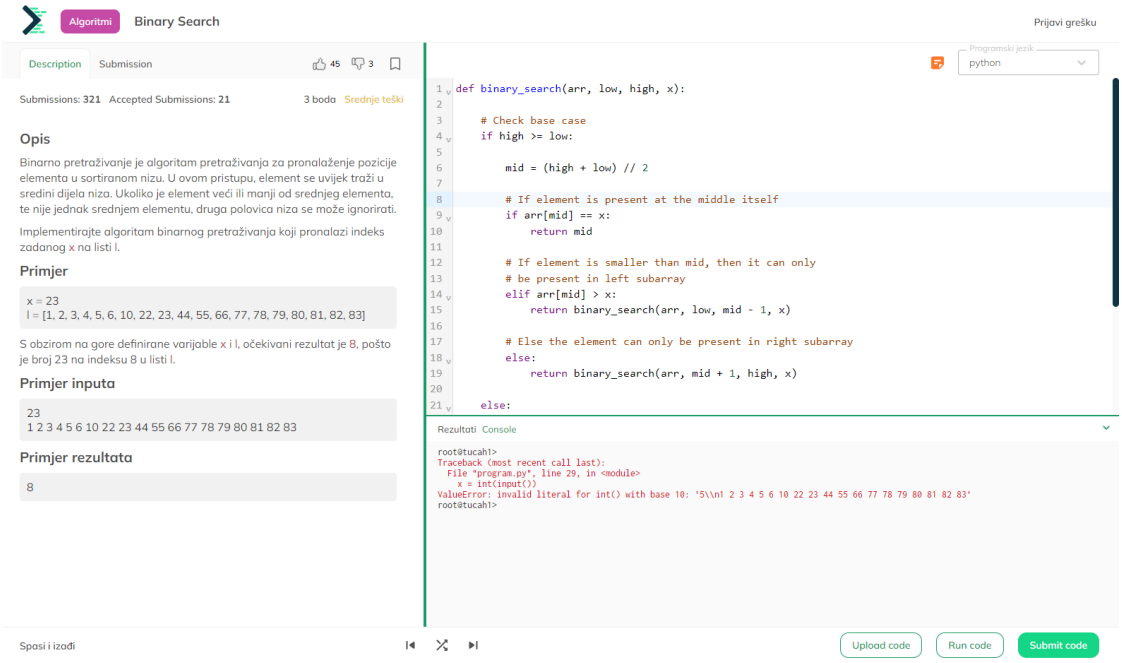


Figure 6.7: Problem page

# CHAPTER 7 **CONCLUSION**

Kodiraj.ba web application provides students and young people in Bosnia and Herzegovina, that strive to learn more and improve their skills, with a convenient tool that is easy to use and in their native language. Additionally, this paper presents one of the architectures, toolsets, and implementations that can be used to develop such a system. Users of the Kodiraj.ba web application can create and manage their accounts, browse categories and tags, search, filter, and sort problems in categories, solve problems, run, and submit those solved problems. While administrator users can manage content in the application, which consists of categories, tags, and problems. Online judge systems and web applications have become a crucial part of education in the IT industry. They provide a convenient way to practice and develop programming and problem-solving skills. To further improve the architecture and implementations of existing online judge web applications and positively influence the development of new systems in the field, it is important to discuss these topics and share ideas. Moreover, by developing new online judge web applications for smaller communities and markets in local languages, many people can be provided with a helpful and useful tool for growth and improvement in this industry. Kodiraj.ba web application was developed with these goals and ideas in mind.

# REFERENCES


[1] S. Wasik, M. Antczak, J. Badura, A. Laskowski, and T. Sternal, “A Survey on Online Judge Systems and Their Applications,” ACM Comput. Surv., vol. 51, no. 1, pp. 1–34, Jan. 2019, doi: 10.1145/3143560.

[2] “Codewars - Achieve mastery through coding practice and developer mentorship,” Codewars. https://www.codewars.com (accessed Jun. 10, 2022).

[3] “LeetCode - The World’s Leading Online Programming Learning Platform.” https://leetcode.com/ (accessed Jun. 10, 2022).

[4] “Coderbyte | Code Screening, Challenges, & Interview Prep.” https://coderbyte.com/ (accessed Jun. 10, 2022).

[5] “HackerRank,” HackerRank. https://www.hackerrank.com/ (accessed Jun. 10, 2022).

[6] “Learn to Code - for Free,” Codecademy. https://www.codecademy.com/ (accessed Jun. 10, 2022).

[7] “Learn R, Python & Data Science Online | DataCamp.” https://www.datacamp.com/ (accessed Jun. 10, 2022).

[8] I. M. Wirawan, A. R. Taufani, I. D. Wahyono, and I. Fadlika, “Online judging system for programming contest using UM framework,” in 2017 4th International Conference on Information Technology, Computer, and Electrical Engineering (ICITACEE), Oct. 2017, pp. 230–234. doi: 10.1109/ICITACEE.2017.8257708.

[9] J. Wu, S. Chen, and R. Yang, “Development and application of online judge system,” in 2012 International Symposium on Information Technologies in Medicine and Education, Aug. 2012, vol. 1, pp. 83–86. doi: 10.1109/ITiME.2012.6291253.

[10] H. Sharma, “Secure Online Judge in Cloud Environment,” IOSR J. Comput. Eng., vol. 19, no. 3, pp. 102–106, May 2017, doi: 10.9790/0661-190302102106.

[11] R. Kasahara, K. Sakamoto, H. Washizaki, and Y. Fukazawa, “Applying Gamification to Motivate Students to Write High-Quality Code in Programming Assignments,” in Proceedings of the 2019 ACM Conference on Innovation and Technology in Computer Science Education, New York, NY, USA, Jul. 2019, pp. 92–98. doi: 10.1145/3304221.3319792.

[12] H. Z. Došilović and I. Mekterović, “Robust and Scalable Online Code Execution System,” in 2020 43rd International Convention on Information, Communication and Electronic Technology (MIPRO), Sep. 2020, pp. 1627–1632. doi: 10.23919/MIPRO48935.2020.9245310.

[13] A. Hafiz and K. Jin, Architecture of a Flexible and Cost-Effective Remote Code Execution Engine. 2021.

[14] M. J. Stammerjohann, “Sandboxing remote code execution in the distributed system RCE,” p. 84.

[15] G. Derval, A. Gego, P. Reinbold, B. Frantzen, and P. V. Roy, “Automatic grading of programming exercises in a MOOC using the INGInious platform,” p. 6, 2015.

[16] H. Yibo, Z. Zhang, B. Yuan, H. Bi, M. N. Shahzad, and L. Liu, “An Experimental Online Judge System Based on Docker Container for Learning and Teaching Assistance,” in 2019 IEEE SmartWorld, Ubiquitous Intelligence & Computing, Advanced & Trusted Computing, Scalable Computing & Communications, Cloud & Big Data Computing, Internet of People and Smart City Innovation (SmartWorld/SCALCOM/UIC/ATC/CBDCom/IOP/SCI), Aug. 2019, pp. 1462–1467. doi: 10.1109/SmartWorld-UIC-ATC-SCALCOM-IOP-SCI.2019.00264.

[17] Node.js, “Node.js,” Node.js. https://nodejs.org/en/ (accessed Jun. 10, 2022).

[18] “Express - Node.js web application framework.” https://expressjs.com/ (accessed Jun. 10, 2022).

[19] P. G. D. Group, “PostgreSQL,” PostgreSQL, Jun. 10, 2022. https://www.postgresql.org/ (accessed Jun. 10, 2022).

[20] “What is sequelize.js?” https://www.educative.io/edpresso/what-is-sequelizejs (accessed Jun. 10, 2022).

[21] “Linux Containers.” https://linuxcontainers.org/ (accessed Jun. 10, 2022).

[22] “Messaging that just works — RabbitMQ.” https://www.rabbitmq.com/ (accessed Jun. 10, 2022).

[23] “Figma: the collaborative interface design tool.,” Figma. https://www.figma.com/ (accessed Jun. 10, 2022).

[24] “React – A JavaScript library for building user interfaces.” https://reactjs.org/ (accessed Jun. 10, 2022).

[25] “Redux - A predictable state container for JavaScript apps. | Redux.” https://redux.js.org/ (accessed Jun. 10, 2022).

[26] “Sass: Syntactically Awesome Style Sheets.” https://sass-lang.com/ (accessed Jun. 10, 2022).

[27] “Contabo Quality VPS & Dedicated Servers At Incredible Prices.” https://contabo.com/en/contabo.com/en/ (accessed Jun. 10, 2022).

[28] “PM2.” https://pm2.keymetrics.io/ (accessed Jun. 10, 2022).

[29] “Advanced Load Balancer, Web Server, & Reverse Proxy,” NGINX. https://www.nginx.com/ (accessed Jun. 10, 2022).

[30] “Automate your workflow from idea to production,” GitHub. https://github.com/features/actions (accessed Jun. 10, 2022).

[31] “OWASP Top Ten Web Application Security Risks | OWASP.” https://owasp.org/www-project-top-ten/ (accessed Jun. 10, 2022).

[32] P. Sriramya and R. A. Karthika, “Providing Password Security by Salted Password Hashing Using Bcrypt Algorithm.”

[33] S. Kumar, R. Mahajan, N. Kumar, and S. K. Khatri, “A study on web application security and detecting security vulnerabilities,” in 2017 6th International Conference on Reliability, Infocom Technologies and Optimization (Trends and Future Directions) (ICRITO), Sep. 2017, pp. 451–455. doi: 10.1109/ICRITO.2017.8342469.

[34] F. Holik and S. Neradova, “Vulnerabilities of modern web applications,” in 2017 40th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO), May 2017, pp. 1256–1261. doi: 10.23919/MIPRO.2017.7973616.

[35] “SQL Injection Prevention - OWASP Cheat Sheet Series.” https://cheatsheetseries.owasp.org/cheatsheets/SQL_Injection_Prevention_Cheat_Sheet.html (accessed Jun. 10, 2022).

[36] G. Nakagawa and S. Oikawa, “Fork Bomb Attack Mitigation by Process Resource Quarantine,” in 2016 Fourth International Symposium on Computing and Networking (CANDAR), Nov. 2016, pp. 691–695. doi: 10.1109/CANDAR.2016.0124.

[37] O. Flauzac, F. Mauhourat, and F. Nolot, “A review of native container security for running applications,” Procedia Comput. Sci., vol. 175, pp. 157–164, Jan. 2020, doi: 10.1016/j.procs.2020.07.025.

[38] F. Ponce, G. Márquez, and H. Astudillo, “Migrating from monolithic architecture to microservices: A Rapid Review,” in 2019 38th International Conference of the Chilean Computer Science Society (SCCC), Nov. 2019, pp. 1–7. doi: 10.1109/SCCC49216.2019.8966423.

[39] K. Gos and W. Zabierowski, The Comparison of Microservice and Monolithic Architecture. 2020, p. 153. doi: 10.1109/MEMSTECH49584.2020.9109514.

[40] “What’s the typical flow of data like in a React with Redux app ?,” GeeksforGeeks, Jul. 18, 2021. https://www.geeksforgeeks.org/whats-the-typical-flow-of-data-like-in-a-react-with-redux-app/ (accessed Jun. 10, 2022).

[41] E. Wong, “Shneiderman’s Eight Golden Rules Will Help You Design Better Interfaces,” The Interaction Design Foundation. https://www.interaction-design.org/literature/article/shneiderman-s-eight-golden-rules-will-help-you-design-better-interfaces (accessed Jun. 10, 2022).