BABES-BOLYAI UNIVERSITY
Faculty of Mathematics and Computer Science
(in English)

The story of Harap-Alb – full-stack web game

Graduate,
Vădean Flaviu-Alexandru

Scientific coordinators,
....
....

BABES-BOLYAI UNIVERSITY
Faculty of Mathematics and Computer Science
(in English)

The story of Harap-Alb – full-stack web game

A full-stack pixelated top-down 2D game based on the famous Romanian folklore: "Povestea lui Harap-Alb" using Phaser.js, React.js and Spring Boot.

Graduate,
Vădean Flaviu-Alexandru

Scientific coordinators,
....
....

Summary

The following thesis aims to present the development of a pixelated top-down 2D game that allows children to rediscover the traditional Romanian fairy tale “Povestea lui Harap-Alb“ through gameplay and exposition. From a technological perspective, the application is built using Phaser, a 2D game framework used for making HTML5 games for desktop and mobile, React.js, a popular JavaScript library for building user interfaces, and Spring Boot, a microservice-based Java web framework. The primary motivation for this project is to preserve Romanian’s cultural heritage and promote children’s education through creativity, literacy and entertainment. The application is designed to be easily accessible to a wide audience.

Content

List of tables and figures

Abbreviations

NPM,	Node Package Manager
API,	Application Programming Interface
IDE,	Integrated Development Environment
DOM,	Document Object Model
ERD,	Entity Relationship Diagram
HTML,	HyperText Markup Language
CSS,	Cascading Style Sheets
HTTP,	HyperText Transfer Protocol
JSON,	JavaScript Object Notation
JSX,	JavaScript XML
JWT,	JSON Web Token
JDBC,	Java Database Connectivity
JPA,	Java Persistence API
REST,	Representational State Transfer
SPA,	Single Page Application
UI,	User Interface
SQL,	Structured Query Language
AI,	Artificial Intelligence
XML,	Extensible Markup Language
XSRF,	Cross-site request forgery
RDBMS,	Relational Database Management System
TDD,	Test-Driven Development
URI,	Uniform Resource Identifier
WebGL,	Web Graphics Library

1. Introduction

Every community possesses a fundamental element: history, serving as a cornerstone for comprehending both the past and the present. While it comprises various crucial aspects such as politics and technological progress, our primary emphasis will be on culture, tradition, and folklore.

Culture denotes a diverse array of customs, traditions, and convictions that are collectively shared among a populace, reflecting historical, linguistic, and geographical influences. Ranging from language and music to cuisine and attire, culture constitutes a multifaceted concept that undergoes evolution over time, mirroring societal transformations and global patterns.

Tradition, a vital part of culture, comprises of rituals and practices passed down from one generation to another. These may encompass religious observances and seasonal celebrations like Christmas, Easter, International Women's Day, and Mistletoe Kissing. These traditions symbolize advancement while nurturing a link to the past, uniting individuals to fortify their communal ties.

Folklore acts as a profound manifestation of tradition and culture in diverse formats, encompassing an extensive range of stories, songs, and beliefs. It embodies the collective memory of a community and has been utilized to communicate a society's history, values, and cultural standards, reflecting the viewpoints and encounters of various factions within that society.

As Ion Ghinoiu notes in his book, The Romanian Folk Almanac [1], "Wherever one looks, one knows that the tales and belief embodied in folk wisdom are at the very roots of European civilization. They give us a direct link to the 'Old Ways'. ". Furthermore, Ghinoiu observes that Romania's connection to nature and its forests has remained a central part of the country's cultural identity. "Romania, as I have discovered, never lost its connection with nature and especially to its forests, its 'forest woman spirits', to its wolf, bear and other inhabitants that populate the pantheon governing early animistic beliefs. Villagers, past and present-still, remain faithful to such symbols knowing that 'She' protects and preserves the fertility and creation of all those who inhabit this cosmic space. "

Within this thesis, we shall explore one of the many approaches to fortify the bond with younger generations and Romanian folklore via an interactive and user-friendly web application. This particular application narrates the story of the renowned Romanian fairy tale "Povestea lui Harap-Alb" through a combination of gameplay and exposition, with the objective of reintroducing the traditions and cultural legacy of a nation to forthcoming generations.

From the perspective of software development methodology, the decision was made to adopt Scrum, an Agile development methodology, which involves the execution of tasks in sprints. These tasks are determined at the commencement of the sprint by a Product Owner. Within the realm of specialized literature, it is recognized as a versatile methodology suitable for teams that can swiftly adjust to changes.

Structurally, the document is divided into six chapters, starting with the current introduction, followed by a discussion of the background and rationale for selecting the theme. Chapter II elaborates in depth on the foundation of the application's development as well as the objectives pursued. Subsequently, in Chapter III, a detailed explanation is provided regarding the choice to embrace the Scrum development model, along with the technologies employed in the application's development. Chapter IV focuses on aspects related to the design of the application. Moving on to Chapter V, a depiction is given on the implementation of the application, accompanied by snippets of code and corresponding explanations. Finally, the last chapter delineates the testing process of the application, utilizing principles and tools in the field.

2. The context of the application

In Romania, I have observed a rising trend where youngsters are introduced to smartphones at a very early stage. Oftentimes, children find themselves in situations where they need to wait, leading to boredom, prompting them to resort to smartphones. It is disconcerting how swiftly watching YouTube or playing video games has become a prevalent habit among the youth. Unlike adults, most children exhibit proficiency in transitioning from the lock screen to their desired applications.

In 2018, "Save the Children," an NGO advocating for children's rights in Romania, conducted a survey on internet usage among youngsters aged 12-17. The study, encompassing 1,156 participants, disclosed that 96.3% had a presence on social media platforms. Furthermore, 27% of the respondents spent a minimum of 6 hours online or on their phones during a typical school day, a figure that surged to 48.3% on weekends.

In the year 2022, an investigation was conducted by the organization, encompassing a total of 2,740 children. The results revealed that 78% of the children utilized the internet for a minimum of 3 hours, with 70% engaging in gaming activities and 67% utilizing it for social media purposes. However, only 46% of the children used the internet for the purpose of researching pertinent news.

There have been a bunch of studies done on the new generation, known as generation alpha, that have shown just how short their attention span is. This generation includes people who were born between the years 2010 and 2024. To quote Gloria Mark [2], PhD, who is a chancellor's professor of informatics at the University of California, "I'm a little worried that young children are spending so much time on screens and it's making them think that this is normal behavior. The problem is that when children are very young, there are certain parts of their brain that are not yet mature. So their ability to self-control and a part of the brain called executive function and executive function, you can think of it as the governor of the mind. So it manages things like decision-making, and setting priorities of what we should be paying attention to. It helps manage interference of peripheral stimuli in the environment, and executive function is not yet well developed and children are sitting in front of screens and they're exposed to all kinds of potential distractions. So I think this is not a good idea. Children need self-control for learning, and more and more we're seeing schools having a lot of online learning available for kids, and kids need self-control as a skill to be able to search for information, to do math problems online, to read and write without being distracted. I find it problematic that we're putting children into a digital world before some very critical mental functions are fully developed. I don't think kids are really ready for that. “

It could be argued that dedicating excessive time to trivial pursuits may hinder the use of technology for engaging and educating young individuals. Instead of merely offering passive entertainment, smartphones and computers can be leveraged to introduce children to cultural traditions and narratives. By ensuring this experience is enjoyable and interactive, our objective is to cultivate a sense of pride and connection to their cultural roots. Moreover, this endeavor creates a valuable opportunity to unite individuals of different age groups, allowing both young and elderly family members to partake in the educational journey together.

In his study from 2007 titled "Children's Folklore - Means of Language Education", Iulian Negrilă [3], a professor at Western University "Vasile Goldiş" from Arad, mentioned that “treasures of wisdom and deep meditation, proverbs and sayings, ballads, fairy tales, legends and anecdotes, constituted an inexhaustible source of inspiration for our most representative writers: Dimitrie Cantemir, Mihai Eminescu, Ion Creanga, Ioan Slavici, Tudor Arghezi, etc. “

2.1 Motivation

In my personal observations, I have noted that parents frequently provide smartphones to their children during social gatherings to have uninterrupted interactions. Instances have arisen where children became so engrossed in their devices that they ignored our presence. For instance, I once witnessed a young child on a bus requesting her grandfather's phone, which gave me feelings of sorrow and, at the same time, familiarity. I firmly believe that my web platform presents a purposeful alternative to aimless social media browsing, offering a fun and interactive approach for youths to explore Romanian folklore and cultural heritage.

2.2 The objectives pursued

My primary objectives were centered on ensuring that users are fully engaged in the narrative by seamlessly integrating sound, gameplay, and storytelling while remaining faithful to the original source material. Furthermore, I aimed to develop a game that caters to a wide audience, irrespective of age.

- Accessible Game Design

The game has been intentionally designed to be straightforward and easily understood, featuring pixelated graphics and user-friendly controls. It is optimized to run smoothly on standard home computers with minimal system requirements. Through the incorporation of lifelike scenes, traditional music, and enjoyable challenges, the game strives to captivate users and deliver a delightful experience devoid of any frustrations.

- User-Friendly and Minimalist Interface

Emphasis is placed on enhancing the app's user-friendliness, particularly for a younger demographic. The design maintains simplicity and consistency in components and fonts throughout, enabling the intended audience to navigate the application effortlessly and enjoy a superb user experience.

- Leaderboard and Replayability

To enhance replay value and promote word-of-mouth, the game incorporates a leaderboard. Players engage in mini-games, garnering scores that determine their rankings. The capacity to replay these mini-games inspires users to strive for higher scores, fostering a sense of competition and a drive to ascend the leaderboard. This aspect substantially elevates user engagement and retention.

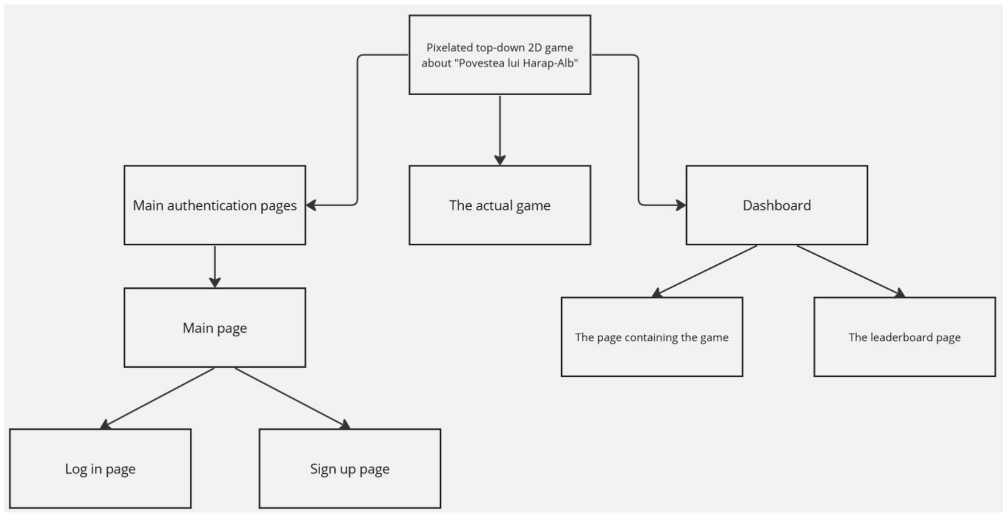

Pixelated top-down 2D game about "Povestea lui Harap-Alb"
Main authentication pages
The actual game
Dashboard
Main page
The page containing the game
The leaderboard page
Log in page
Sign up page

Figure 1. Objective decomposition diagram

Sections are grouped by sub-objectives according to their specifics

3. Software Development Methodology

The success of any project hinges on a set of principles, agreements, or regulations that constitute the software development methodology. Various methodologies have been devised over time to accommodate diverse project intricacies and needs, all rooted in shared principles. In contrast to the conventional model that emphasizes detailed planning with no room for alterations, contemporary models provide flexibility and adaptability to varying circumstances.

The software development procedure encompasses numerous stages that lay the groundwork for developing an application. Before initiating a project, the team typically selects the methodology to employ, which plays a crucial role in defining expenses, client engagement, teamwork strategy, and project deadlines.

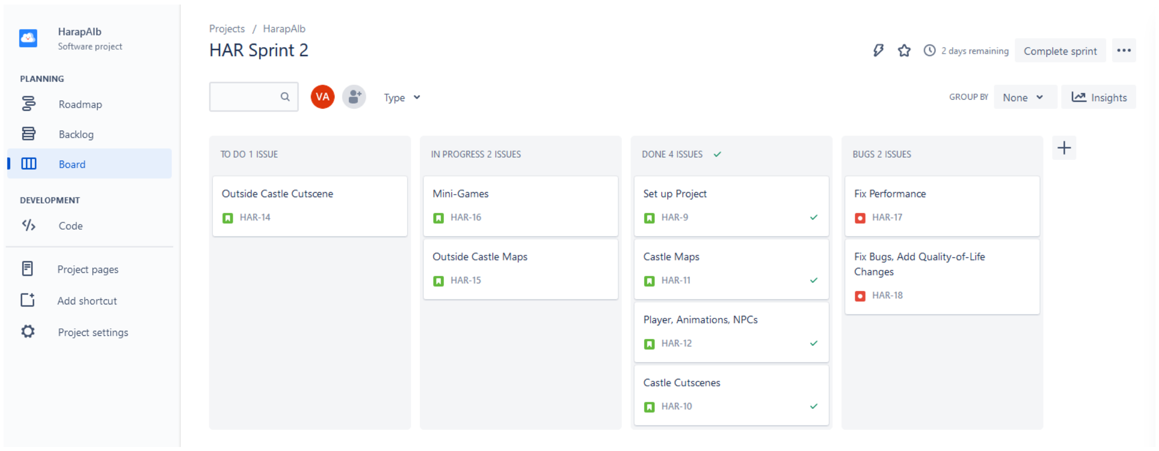
The procedure comprises six stages: documentation and requirement presentation, design, coding, testing, release, and maintenance. During the requirements presentation phase, the team collaborates with the client to understand their requirements and anticipations. In the design phase, the team chooses the architecture and technologies to utilize. Coding actually refers to composing the actual code and constituting a small section of the application. Testing guarantees software quality by confirming the application's functionality and resolving any issues. The release phase involves dispensing the software product based on its specifications, followed by maintenance to ensure uninterrupted operation.

3.1 Scrum

When developing my application, I chose to use the SCRUM Agile methodology. I deeply value how this approach partitions the project into smaller segments that can be addressed gradually. This proves particularly advantageous when unexpected changes arise, as it facilitates prompt adjustments. Typically, this methodology encompasses three pivotal roles: Product Owner, Scrum Master, and team members. Nevertheless, since I was working independently on this project, I oversaw all aspects single-handedly.

SCRUM is centered around sprints, often lasting 1-2 weeks each. Every sprint is devoted to a specific set of tasks that progress through various phases like planning, development, testing, and delivery. This empowers team members to concentrate on specific tasks while ensuring the completion of all steps. Following each sprint, feedback from clients is gathered. Regular meetings are held to oversee progress, tackle any issues, and offer essential support. At the end of each sprint, the team evaluates successes and areas for enhancement.

Initially, I crafted a Product Backlog comprising User Stories and tasks. These tasks were further decomposed and allocated to sprints involving team members. To oversee tasks and defects during development, I made use of a tool called Jira. Jira allows the establishment of distinct boards tailored to the chosen methodology, such as the SCRUM approach I adhered to.



HAR Sprint 2

TO DO 1 ISSUE	IN PROGRESS 2 ISSUES	DONE 4 ISSUES ✓	BUGS 2 ISSUES
Outside Castle Cutscene 🟩 HAR-14	Mini-Games 🟩 HAR-16	Set up Project 🟩 HAR-9 ✓	Fix Performance 🟥 HAR-17
	Outside Castle Maps 🟩 HAR-15	Castle Maps 🟩 HAR-11 ✓	Fix Bugs, Add Quality-of-Life Changes 🟥 HAR-18
		Player, Animations, NPCs 🟩 HAR-12 ✓	
		Castle Cutscenes 🟩 HAR-10 ✓	

Figure 2. The board section from Jira for a sprint within the application
The ones marked in red are bugs and the ones in green are User Stories

During the initial sprint, my main focus was on setting up the project environment and integrating all the necessary dependencies to kickstart the development process. I utilized NPM (Node Package Manager) to download and install the essential libraries required for the project to function smoothly. As part of the setup, I meticulously organized the project in a hierarchical manner, beginning with the creation of fundamental maps and cutscenes to lay the groundwork for the game. This phase provided me with invaluable insights into the syntax and methods needed to effectively operate Phaser, which is a renowned 2D game framework known for its versatility and ease of use. Once I was confident in the functionality of the game engine, I proceeded to work on developing the dashboard and dummy authentication pages using React.js, a popular JavaScript library for building user interfaces. Wrapping up the sprint involved the establishment and configuration of the backend using Spring Boot, a powerful framework that facilitated seamless integration with the front-end components, ensuring a cohesive and efficient overall system.

Moving on to the second sprint, the primary focus shifted towards the actual development of the game itself, diving deep into bringing the envisioned game to life. I extensively referred to online documentation related to the game engine [5] to gain insights and best practices to enhance the game's performance and user experience. I also participated in online forums to read and ask for support on various topics. I started with creating simple scenes that resonated with the underlying theme of the story, I employed tools like Tiled, a widely used 2D level editor in the Phaser community, and utilized free resources such as CharaMEL to craft visually appealing character sprites and backgrounds. These resources helped with the creation and animation of sprites, together with cutscenes, inspired by the initial sketches from the opening pages of the book that served as the basis for the game's storyline and theme. This phase of development aimed to ensure that the initial gameplay experience focused on exploration while also ensuring that users could easily navigate through the controls and mechanics of the game, creating an engaging and intuitive interface for players to interact with.

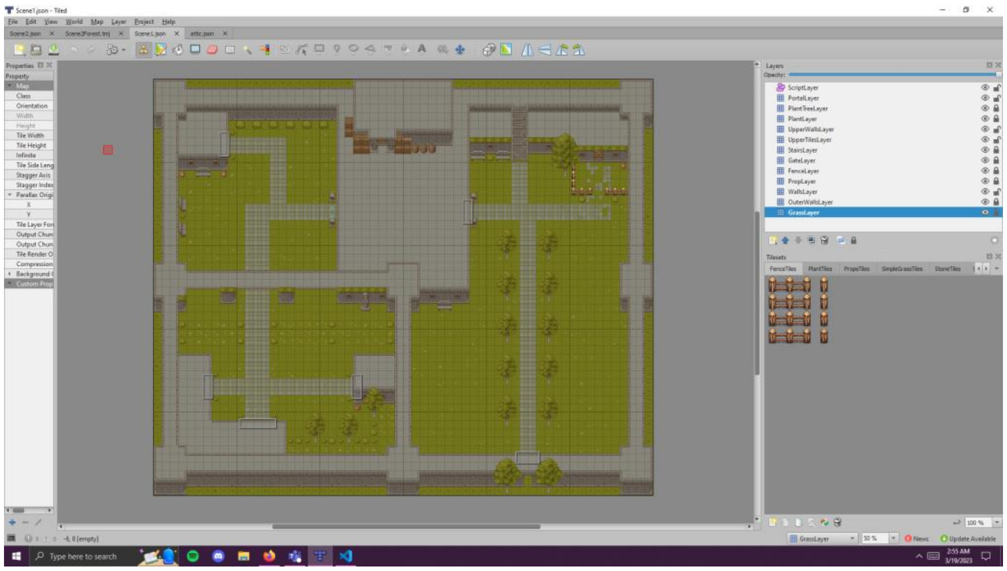


Figure 3. Using Tiled to create the castle scene

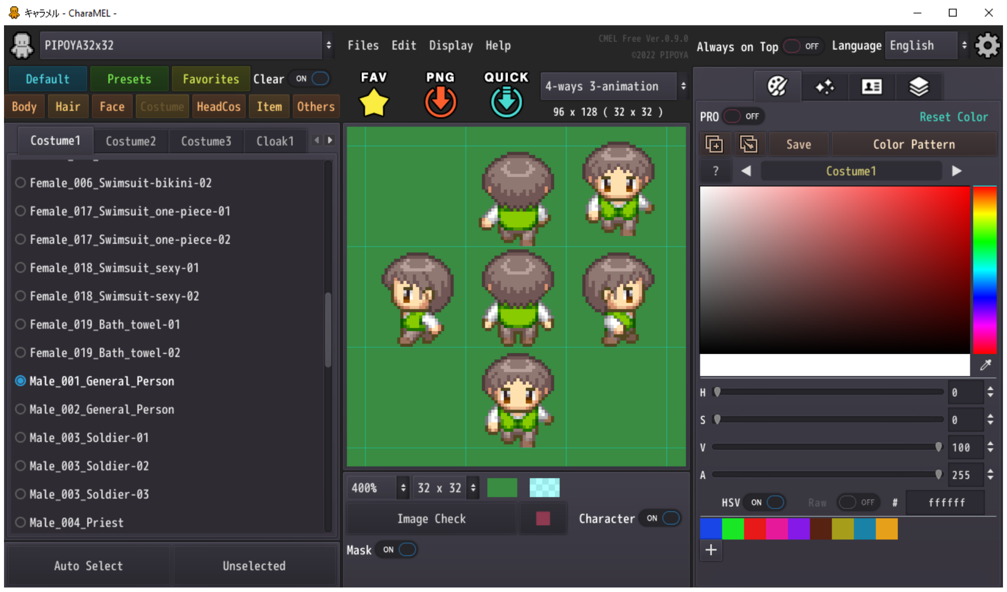


Figure 4. Using CharaMEL to create the main character

During the third sprint, which happened to be the lengthiest period in our development timeline, our main focus was on revamping the game's storyline to better suit the desired simplistic format. This desire was achieved through trial and error. I took the initiative to introduce the initial mini-game, ensuring its seamless integration into the gameplay, all the while designing outdoor settings such as the castle’s props, design, and nature, together with the forest’s landscape. The Concentration mini-game, where players are tasked with uncovering matching pairs of cards within a 15-second time frame, added an exciting challenge to the gaming experience. Striking the right balance between cinematic cutscenes and interactive gameplay proved to be a challenge, especially given the elaborate narrative of the original story. Embracing the concept of simplicity, my goal was to provide players with a smooth and straightforward gaming journey that unfolds in a somewhat linear fashion.

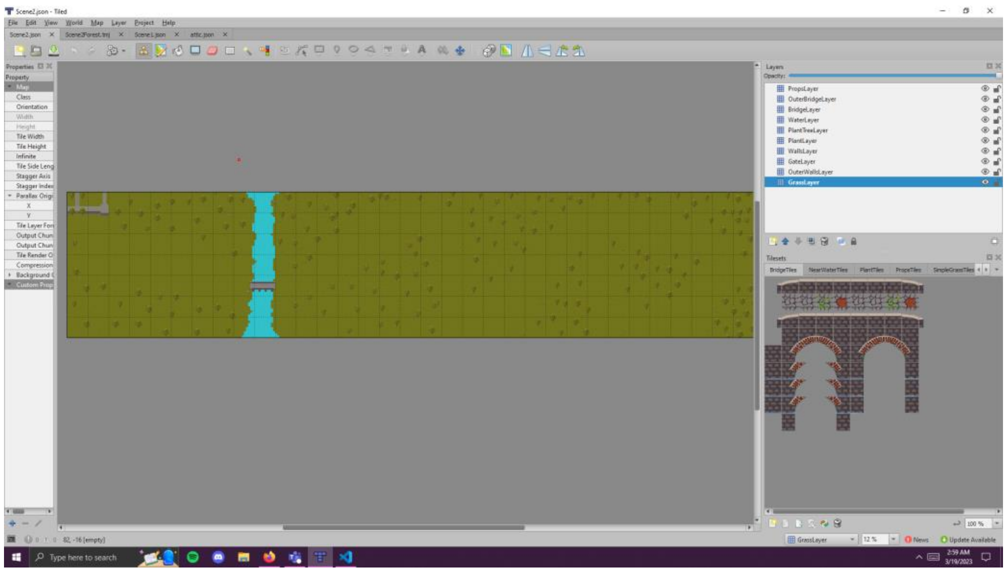


Figure 5. Using Tiled to create the outside of the castle

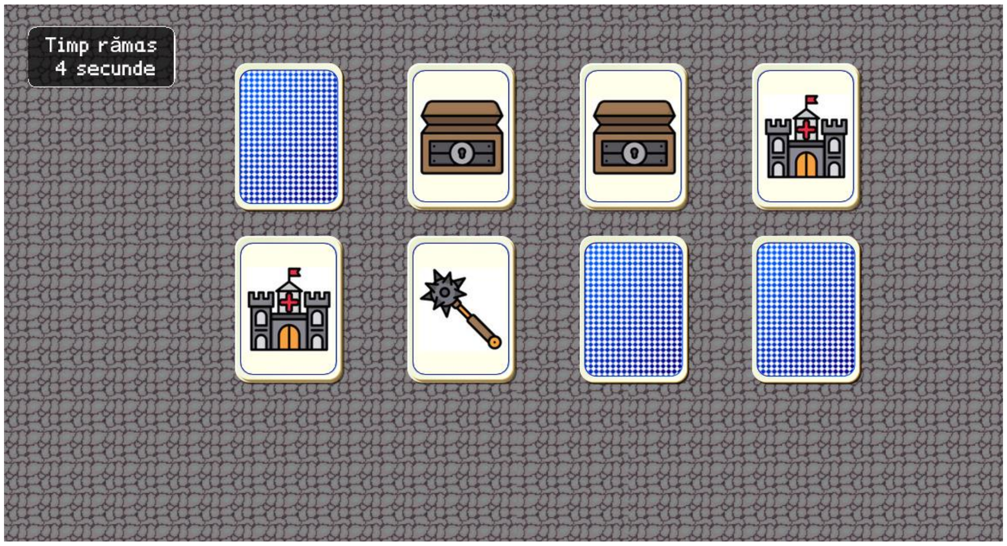


Figure 6. How the first mini-game looks

Transitioning to the fourth sprint, the second most drawn out phase of the project, the demo was successfully concluded by incorporating two supplementary mini-games, finalizing the remaining cutscenes, and refining the simplistic scenes. The second mini-game assessed players' arithmetic abilities under time constraints, while the third mini-game immersed players in a procedurally generated maze tasking them with locating three lettuces within a stipulated timeframe. Moreover, enhancements were made to the opening scene, and a new map with limited visibility was introduced, injecting a fresh layer of thrill and challenge for players to relish. By using existing code snippets, online references, and the aid of Leonardo, an AI toolkit, the process of crafting realistic cutscenes, textures, exploration through gameplay and mini-games was streamlined, thereby enhancing the game's overall visual appeal.

Sayim Aktay, in his paper titled "The Usability of Images Generated by Artificial Intelligence (AI) in Education" [22], reached the conclusion that images crafted by artificial intelligence have the capacity to enhance various teaching activities, circumvent copyright-related challenges, address the issue of insufficient content, and offer numerous other benefits. Although the presence of AI is most evident in the cutscenes, many textures and assets were also enhanced and redesigned using AI, contributing significantly to the game's overall quality.

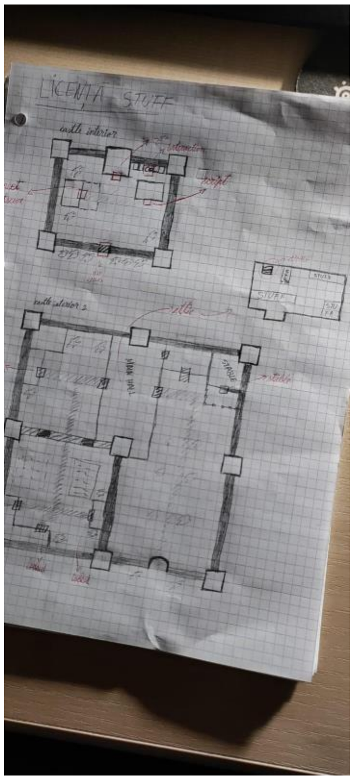


Figure 7. My sketch showing the evolution of the first scene, the castle

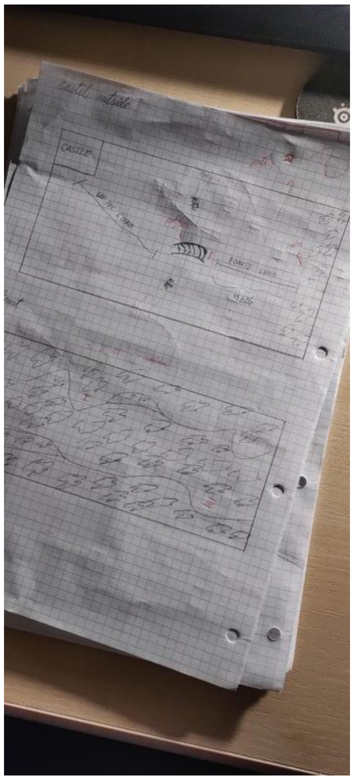


Figure 8. My sketch showing the outside of the castle & the forest

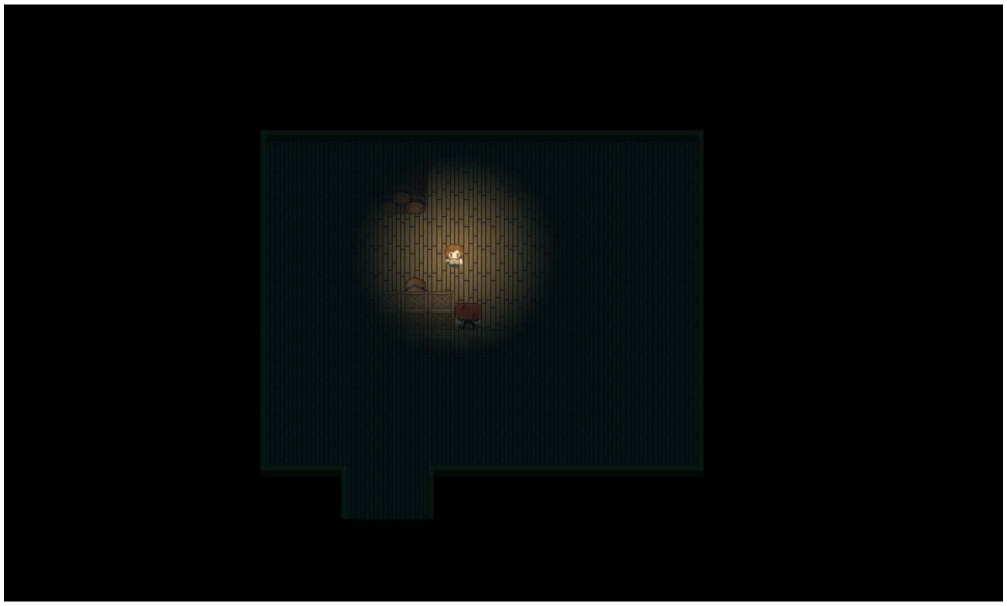


Figure 9. How the first mini-challenge looks with low visibility

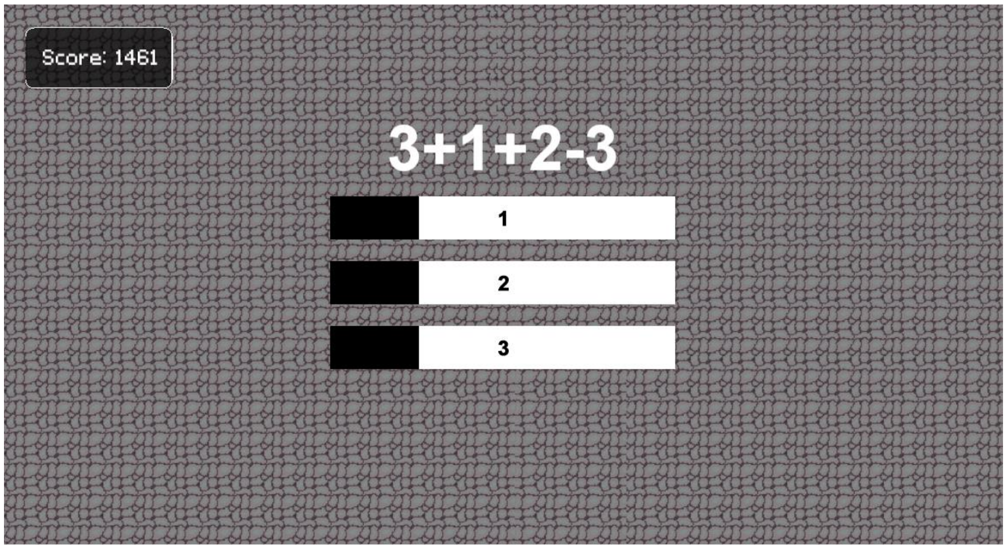


Figure 10. How the second mini-game looks like

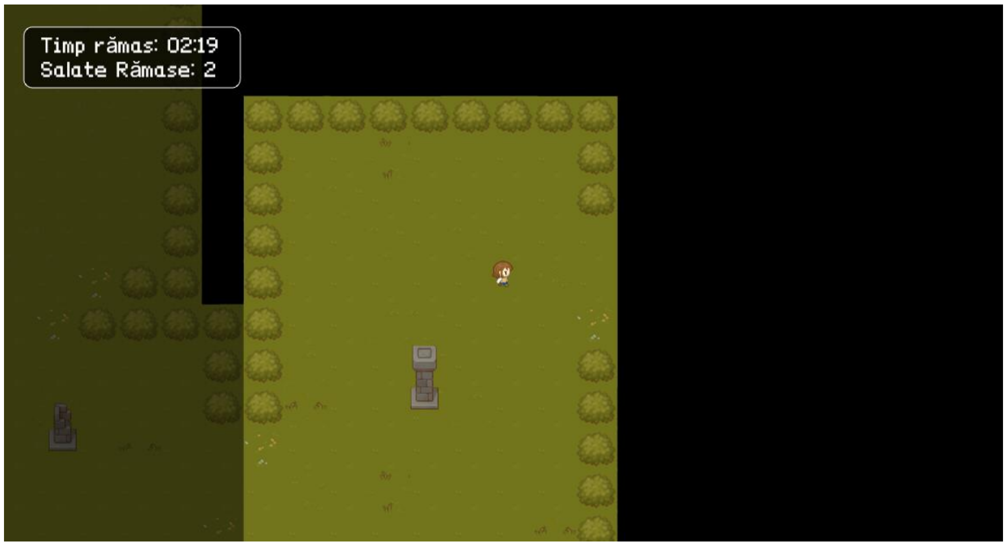
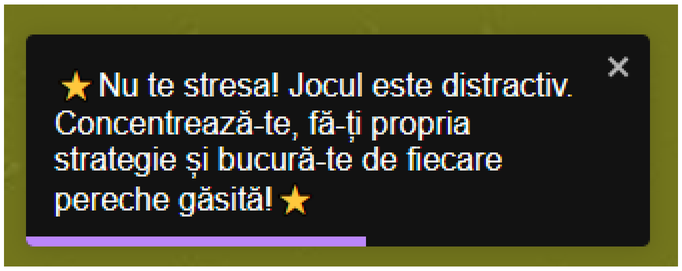


Figure 11. How the third mini-game looks like

During the fifth sprint, which was of comparable duration because of the bugs that occurred, focus was directed towards advancing both front-end and back-end elements, encompassing the design of web pages and a user-friendly dashboard. The decision was made to employ Material UI for utilizing pre-defined components and Pixelify Sans from Google Fonts to maintain a consistent visual appeal. Subsequently, the configuration of the back-end and database infrastructure mirrored this choice, with PostgreSQL being designated as the principal database system, complemented by Spring Boot JPA and JWT for constructing secure and scalable RESTful APIs.

Upon commencing the final sprint, the alpha version of the demo was enthusiastically shared with a select group of friends, eagerly soliciting their valuable feedback. Each piece of feedback was meticulously reviewed, with particular attention given to rectifying any elements of the game that detracted from the overall enjoyment or caused confusion during playtesting. Notably, the integration of a popular notifications package named Toastify was undertaken to help players with reminders and guidance should they opt to skip the cutscenes or get stuck in a mini-game. Furthermore, pieces from the original book were strategically integrated throughout the game to provide additional context in a concise manner. These informative notifications were strategically positioned in the top-right corner of the screen. To ensure the game's efficiency, the code was optimized by segmenting repetitive sections and crafting reusable components to facilitate a smoother development process. In the original story, the book spends a lot of time on exhibition until the main character has the opportunity to jump into the action.



★ Nu te stresa! Jocul este distractiv. Concentrează-te, fă-ți propria strategie și bucură-te de fiecare pereche găsită! ★

Figure 12. How a tip would look like if the player became stuck in the first mini-game

Working with user feedback while also learning new technologies motivated me to keep improving everything. From the gameplay to the overall experience, I went through many trials and errors to get things right. As a study from 2009 [21] on designing educational mini-games noted: “Designing an educational mini-game is more demanding if the mini-game will be played by children unsupervised, as this means that all guidance has to be built into the game. To complicate the task even further, the data from our observations show that experimenting may withhold the children from thinking through a problem more fundamentally”. Because of this, I avoided making the mini-games too complicated, ensuring they were straightforward and didn't leave children guessing the answers. Some mini-games were adjusted to encourage better scores, and I included new external packages like Toastify to provide guidance and make the game more enjoyable without being annoying. Although this process was time-consuming, I believe it successfully addressed all the user concerns.

3.2 Specific Technologies

In this segment, a brief overview of the technologies selected for the application's development will be presented. The fundamental basis of the application's front-end is constructed using the JavaScript programming language. According to a survey conducted in 2023, encompassing 89,184 software developers from 185 nations worldwide, JavaScript emerged as the most favored programming language, with 63.61% of respondents expressing a preference for it.

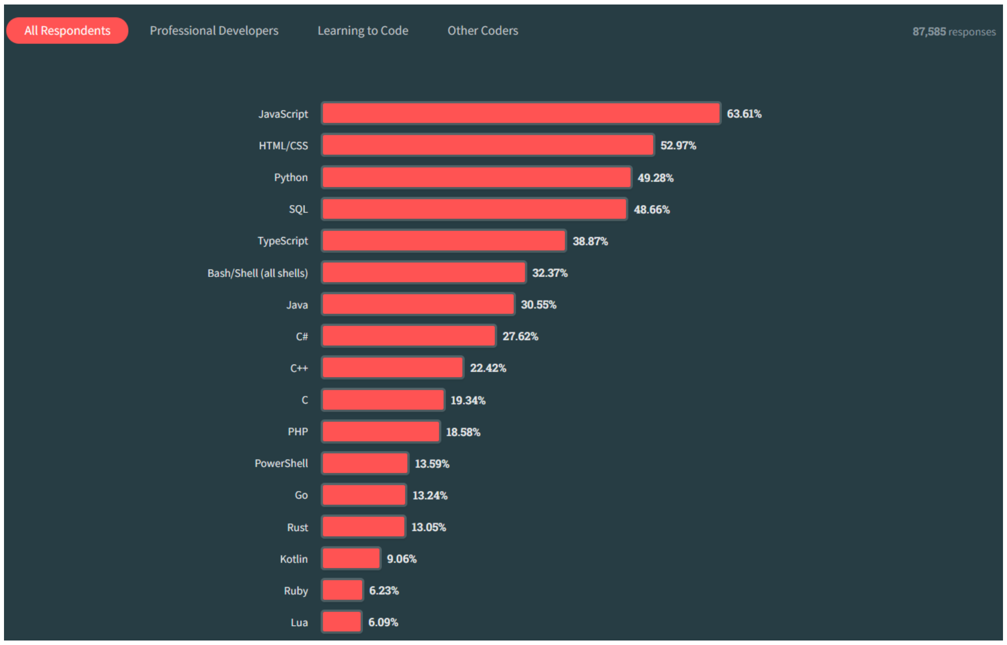


Figure 13. Public opinion about the most popular programming language
Source: StackOverflow 2023 Developer Survey [20]
2023 continues JavaScript's streak as its eleventh year in a row as the most commonly-used programming language

The backend is implemented in Java, the seventh most popular programming language based on the same survey shown before, and the database is SQL-based, with a preference for PostgreSQL. PostgreSQL, a widely-used open-source relational database management system, is chosen for its free availability and popularity. It emphasizes extensibility and adheres to SQL compliance. Spring offers various APIs, such as JDBC and JPA, that work well with PostgreSQL. JDBC is used to interact directly with databases, while JPA simplifies database interactions through object-relational mapping. Both Java and PostgreSQL have excellent community support and documentation, making the implementation process even easier.

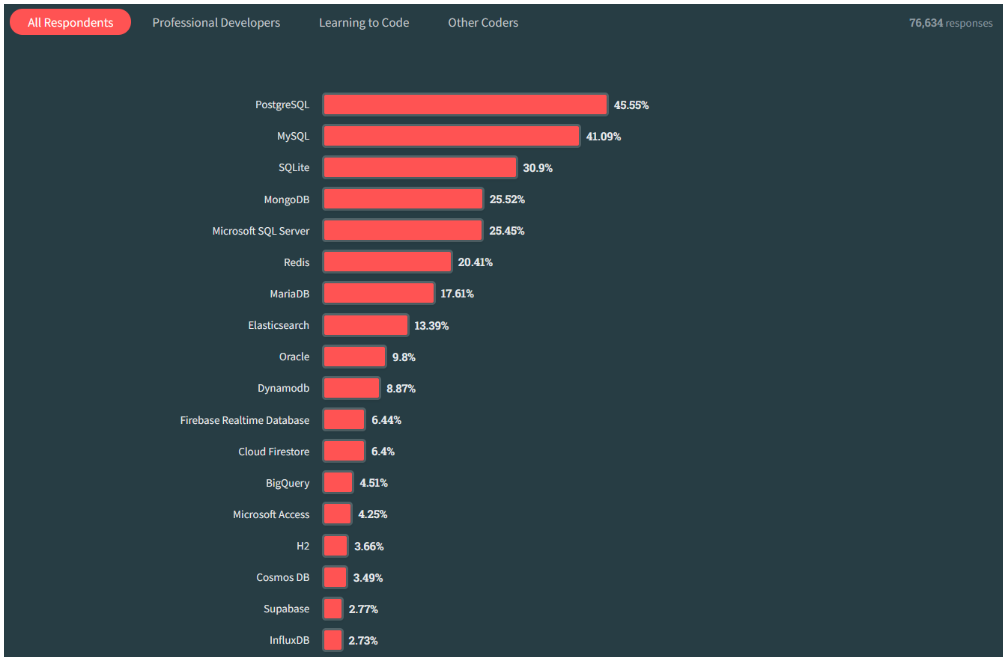


Figure 14. Public opinion about the most popular database
Source: StackOverflow 2023 Developer Survey [20]

3.2.1 Phaser.js

Phaser [4] stands out as an outstanding 2D game framework tailored for crafting web-based games that can operate seamlessly on both desktop and mobile devices. Utilizing JavaScript programming language, Phaser can be easily integrated with technologies like HTML5 and CSS3. This framework provides an array of features and tools to aid game developers in efficiently creating games, encompassing physics systems, sprite management, animation, input handling, and sound management. Owing to its intuitive interface, flexibility, and ability to publish games across diverse platforms, Phaser remains a favored choice among game developers.

3.2.3 Axios

Axios [6], a versatile HTTP client, leverages JavaScript ES6 promises to ensure compatibility with both node.js [7, 8] and web browsers. Through a unified codebase, it seamlessly functions in both environments, utilizing node.js's built-in http module on the server side and XML HTTP requests on the client side. Axios simplifies the manipulation of JSON data, bolsters security against XSRF attacks, and offers capabilities such as intercepting and transforming server requests and responses.

3.2.4 React.js

React.js [9, 10], a library established in 2013 by Facebook (now Meta) in collaboration with individual developers, facilitates the creation of user interfaces based on components. Widely utilized for building Single Page Applications (SPAs) in combination with essential libraries for routing or additional functionalities, React utilizes JSX for embedding HTML code within JavaScript code and updating a virtual DOM based on incoming input. Each component in React encapsulates a portion of the interface and logic internally, with a specialized iteration designed for mobile applications under the name React Native.

3.2.5 Redux Toolkit

The Redux Toolkit [11] functions as a beneficial library that aids developers in efficiently managing the application state, ensuring consistent persistence across its lifecycle. Often paired with React, this library revolves around the concept of a globally accessible "store" within the application, facilitating convenient access to an object's state from any component. While lacking direct support for asynchronous requests, various online packages are available to fulfill this requirement.

3.2.6 React Router

React Router [12] is an exceptional navigation library for React applications, facilitating the creation of dynamic Single Page Applications (SPAs) with a seamless user experience. Developed by React Training and backed by the React community, React Router enables the construction of navigation structures by associating specific components with distinct URLs. By harnessing React's declarative nature, developers can delineate routing rules in a straightforward and lucid manner. Through React Router, developers can incorporate functionalities such as nested routes, route parameters, and navigation guards to enrich user experience fluidly. It stands as an indispensable tool for managing the UI state based on the URL, supporting both client-side and server-side rendering.

3.2.7 Material UI

Material-UI [13] emerges as an excellent React component library crafted to aid in effortlessly creating visually appealing and responsive user interfaces. Developed by Call-Em-All and supported by a dynamic open-source community, Material-UI adheres to Google's Material Design principles, presenting readily usable components for effective UI development. The library smoothly integrates with React applications, offering a wide array of customizable components such as buttons, forms, and navigation elements. By simplifying the application of Material Design principles, Material-UI enhances the overall user experience for web applications.

3.2.8 Spring Boot

Spring Boot [14, 15], an extension of the Spring framework, represents a Java-based tool designed to streamline and expedite the development of Spring applications in a user-friendly manner. It comes with opinionated defaults and standards that simplify configuration intricacies, enabling developers to concentrate more on application logic. Featuring an embedded web server like Tomcat, external deployment becomes unnecessary. Spring Boot facilitates various application architectures, including microservices, and seamlessly integrates with other Spring projects and external libraries. It strikes a fine balance between standardization and personalization, enabling swift development while tailoring solutions to specific requirements.

3.2.9 PostgreSQL

PostgreSQL [16] stands out as an exceptional open-source relational database management system (RDBMS) known for its robustness and adaptability. Developed by the PostgreSQL Global Development Group, it supports SQL and provides advanced functionalities like transactions, foreign keys, and intricate queries. PostgreSQL excels in handling vast datasets and numerous transactions concurrently, making it suitable for a broad range of applications. It adheres to the ACID properties (Atomicity, Consistency, Isolation, Durability) to guarantee data integrity. With support for various data formats such as JSON and spatial data, PostgreSQL accommodates diverse data models. Its active community ensures consistent updates and enhancements, establishing it as a reliable choice for database-centric applications.

3.2.10 JUnit

JUnit [18, 19] is a widely accepted Java testing framework that simplifies the process of writing and executing unit tests effectively. It provides useful annotations for test methods, fixtures, and assertions, aiding developers in creating comprehensive test suites. Originated by Kent Beck and Erich Gamma, JUnit promotes test-driven development (TDD) by facilitating automatic and repeatable code testing. Due to its clear and intuitive framework, JUnit plays a pivotal role in ensuring the dependability and accuracy of Java applications, enabling developers to detect and address issues early on. It has evolved into a standard tool for testing Java applications, contributing to improved code quality and simplified maintenance.

4. Computer system design

In this chapter, we will discuss various aspects concerning the logical presentation of the application, data flow, the application's information database, as well as elements associated with the architecture, components, and their interactions with each other.

4.1 Logical design

During the logical design phase, we utilize the data flow diagram to visualize the movement of data within the system under construction. Each function processes input data, validates and manages it prior to forwarding or executing tasks related to data storage. For example, during authentication, the system validates access credentials, decrypts passwords, and then compares them with user input. A unique token containing user information in base64 encoding is created using a secret key and transmitted to the client. Additional requests are authenticated by middleware, which verifies the presence of the token in the cookie.

Upon successful authentication, users are led to the dashboard where the game commences, accompanied by instructions on gameplay. The dashboard features two buttons - "start" to initiate the game and "menu." Clicking on "start" triggers the game, and progress is monitored. Scores are only sent to the backend if they exceed the previous high score, becoming the new default.

Users can also click on the "menu" button on the dashboard to reveal a sidebar showing their name, email, profile picture, a button for game controls guide, a music mute option, a sign out button, and a button to view the leaderboards. The leaderboards, displayed on a separate page, automatically showcase users who have completed at least one mini-game, arranged by highest score.

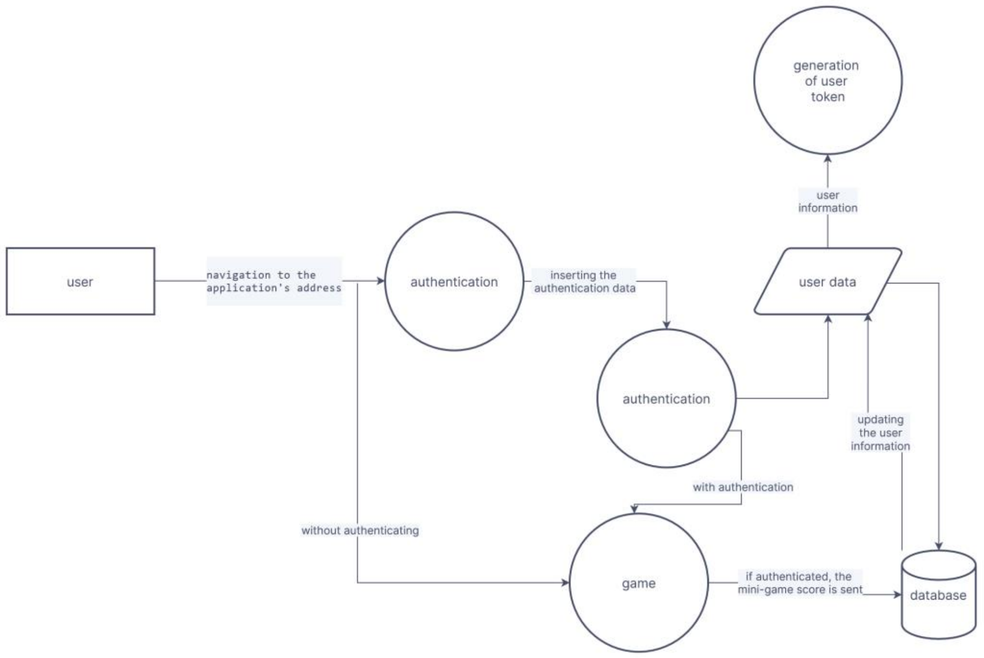

generation of user token
user information
user
navigation to the application's address
authentication
inserting the authentication data
user data
authentication
updating the user information
with authentication
without authenticating
game
if authenticated, the mini-game score is sent
database

Figure 15. Data flow diagram

4.1.1 Information base

The repository of information is where our entire system data is stored and plays a crucial role in the design of our application. It is essential to take into account scalability, especially if there are plans to incorporate additional mini-games. Security also holds significant importance when managing sensitive personal data like names and email addresses. Given the relational nature of the data and the need for structured connections, a SQL database such as PostgreSQL appears to be a fitting choice. PostgreSQL is highly regarded for its versatility and adherence to SQL standards.

Structured Query Language (SQL), originally developed by IBM in the 1970s and later standardized by ANSI in the 1980s, is a commonly utilized language for overseeing relational databases.

PostgreSQL, also known as Postgres, is an open-source relational database system that made its debut in 1989. It is preferred for its extensive features, flexibility, and strong community support. The system follows ACID principles, supports scalability, and prioritizes security. The vibrant community offers assistance, materials, and utilities, making it compatible with a variety of operating systems.

Within a basic project, we have two tables: "user" and "minigame." I have established a one-to-many relationship in the "user" table, allowing a single user to possess multiple mini-games. Meanwhile, in the "minigame" table there exists a many-to-one correlation, indicating that each mini-game is linked to only one user. If a user is deleted, all associated mini-games are also removed, whereas deleting a mini-game does not eliminate the user to whom it belongs.

4.1.2 The architecture of the developed system

The architecture used in the development of the application is the client-server architecture. According to Shklar, co-author (with Rich Rosen) of the popular textbook Web “Application Architecture: Principles, Protocols, and Practices” [17], "client-server applications are groups of distributed programs that run on a network and interact through known communication protocols". Instead of processes being performed on a single system, client-server applications distribute processes between a dedicated server and the rest of the clients. Clients communicate with the server via HTTP requests, and the response sent back by the server is in JSON format. The main advantage of such an architecture is that it is easier to maintain, without having to make constant changes to the client, and through the interface it excludes the need for a terminal.

The server's services delivered via an API (Application Programming Interface) together with the REST architecture, devised by Roy Fielding in 2000. This service architecture is founded on a set of principles. Resources are identified within the application through a URI (Uniform Resource Identifier). These URIs incorporate descriptive terms (e.g., /patients denotes a list of patients), and each endpoint permits CRUD operations via HTTP requests (GET, POST, PATCH, UPDATE verbs). Requests to the server include all requisite information and are autonomous from prior requests. The server furnishes responses in JSON format. Distinct status codes accompany requests to signify their status (e.g., code 200 denotes successful completion, whereas code 400 indicates client-side errors).

Requests are accompanied by specific messages reflecting the progress of the request. In our application, we managed responses and status codes through the Node.js platform. At the server level, the application is constructed on the Spring Boot framework, facilitating interactions with the database for data retrieval and mapping to Spring Boot models.

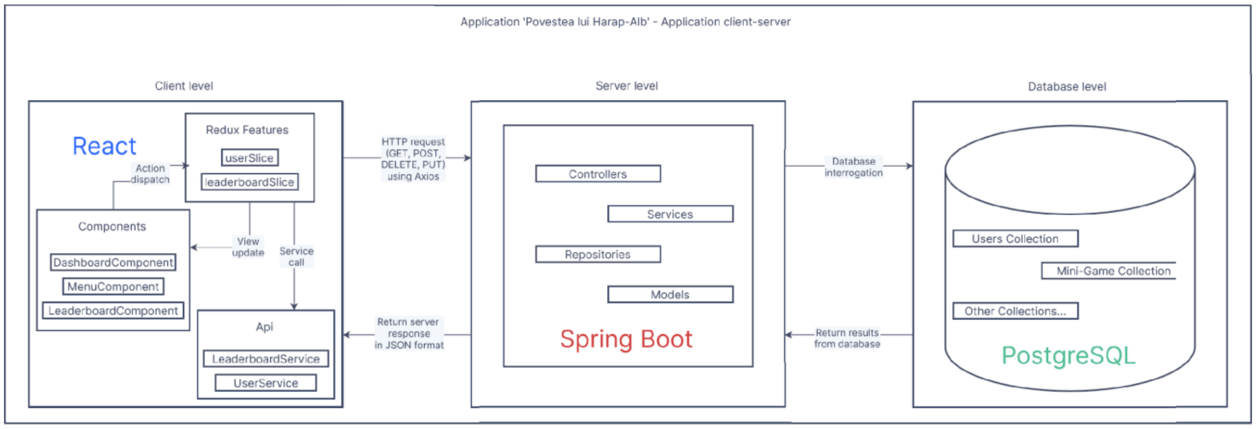


Figure 16. System architecture

4.1.3. Component diagram

The elements of the application were illustrated in a component diagram. Each component assumes a unique role and interacts seamlessly with others in the system. The front-end component was fashioned using the React.js library, renowned for its component reusability throughout the application. React facilitates communication between front-end and back-end components through HTTP requests employing Axios.

The Redux Toolkit oversees the application's state through "dispatch". React dynamically populates components via a virtual DOM tree, ensuring a smooth user experience devoid of page reloads. Requests are directed to specific endpoints, essentially routes dispensing information to the client, triggering functions and middleware interceptors. Application controllers oversee business logic, data processing, and database queries. Data integrity is preserved by Spring Boot JPA, which validates incoming data against the model.

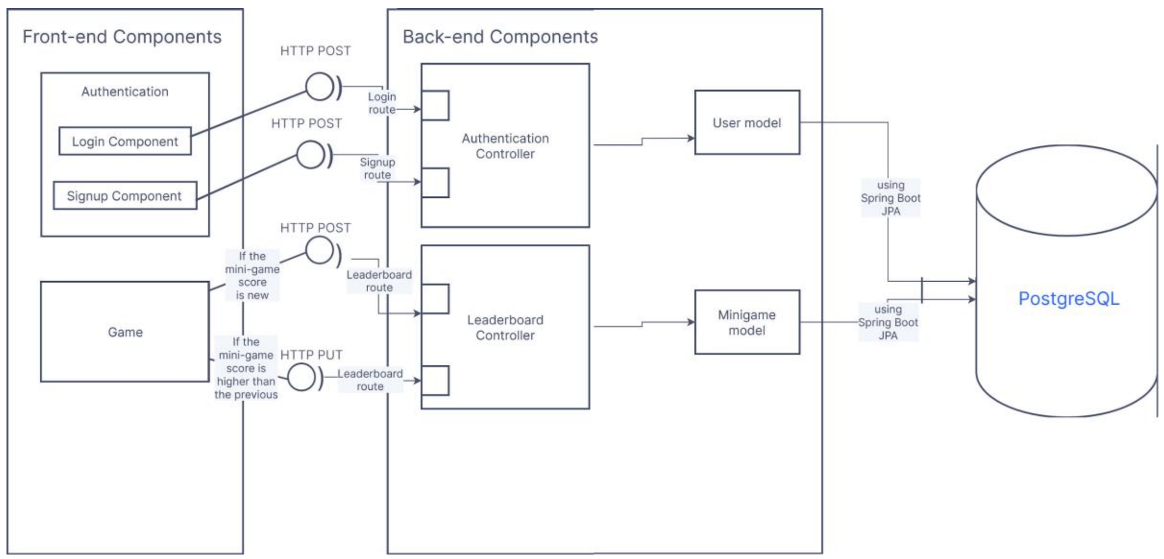


Figure 17. Application diagram on components

4.2 Technical design of the application

In this section, we will take a quick look into the technical design aspects of the application, which emphasizes the structural and relational aspects of the data. We will outline the key components and their interactions, focusing on the entity-relationship diagram that represents our data model.

4.2.1 Entity-relation diagram

In the development of visual representations for our data, we crafted the entity relationship diagram. This diagram offers us insights into the data types present in our application and the connections among the various collections that constitute the entities we interact with. Within our database, we oversee two collections: 'User' and 'MiniGame', with the 'id' field in both acting as the primary key. The link between them is a One-to-Many relationship from 'User' to 'MiniGame', depicted by the 'user_id' foreign key in the 'MiniGame' table.

This configuration permits multiple 'MiniGames' to be linked with each 'User', while each 'MiniGame' is associated with a single 'User'. Through the 'user_id' foreign key in the 'MiniGame' table, we define this association, specifying the respective user for each mini-game.

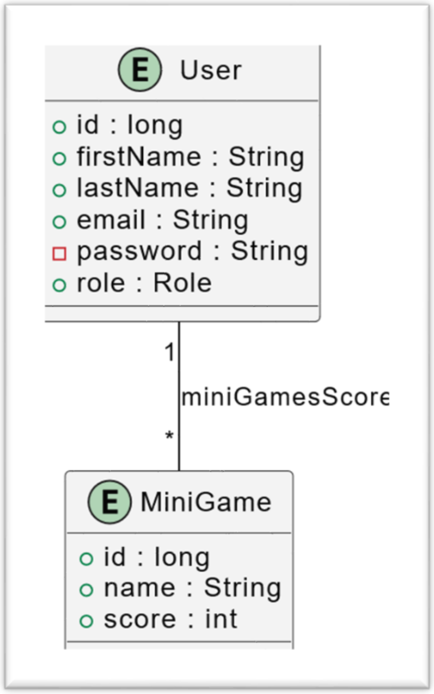


Figure 18. ERD diagram

This simplified entity-relationship diagram ensures ease of development and maintenance, while also providing flexibility for future expansion. New mini-games, regardless of their complexity and features, can be seamlessly integrated without overhauling the existing architecture. Such a design also provides faster query times and easier debugging.

5. Implementation of the IT system

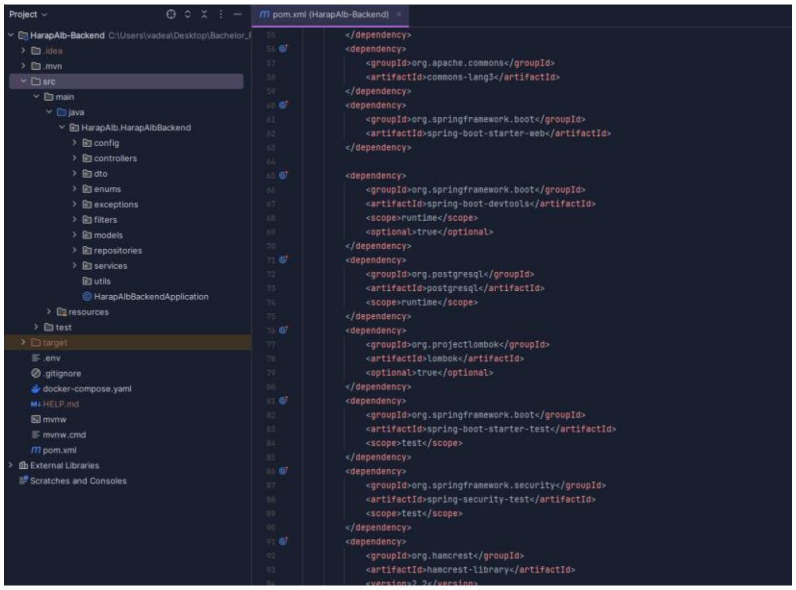
In this segment, we will delve into the specifics of the application's implementation. The utilized development environment consisted of Visual Studio Code and IntelliJ, prominent Integrated Development Environments (IDEs) within the realm of web developers. Web applications commonly make use of various technologies, collectively known as a technology "stack". React.js was employed for the front-end, Spring Boot for the back-end, and PostgreSQL for the database.

The evolution of web applications and their increasing complexity, as underscored by Subramanian in 2019, has led to a shift towards enhancing interactivity. This has resulted in the surge in popularity of SPA (Single Page Application) applications. This shift is credited to the ability for dynamic content updates without complete page reloads, achieved by fetching and displaying information fragments from the server.

5.1 Back-end components

In order to maintain a coherent structure within the project, sub-elements were divided into distinct directories, each fulfilling a specific role. Particularly noteworthy within the server directory are pivotal subcomponents like models (representing entities within the application), controllers (encompassing business logic functions and dispensing data to clients in JSON format through HTTP requests), and routes (outlining endpoints for HTTP requests, each bearing recognizable names corresponding to desired resources and integrating middleware like interceptors for request authorization).

For efficient accessibility to project dependencies, the node package manager was employed for installation through the command line interface. Central to the project lies the pom.xml file, encapsulating crucial details concerning dependencies such as JPA for relational data management, Spring Security for authentication and access regulation, JWT for secure data exchange, Lombok for reducing boilerplate code, alongside prominent testing frameworks like JUnit and Mockito for Java.



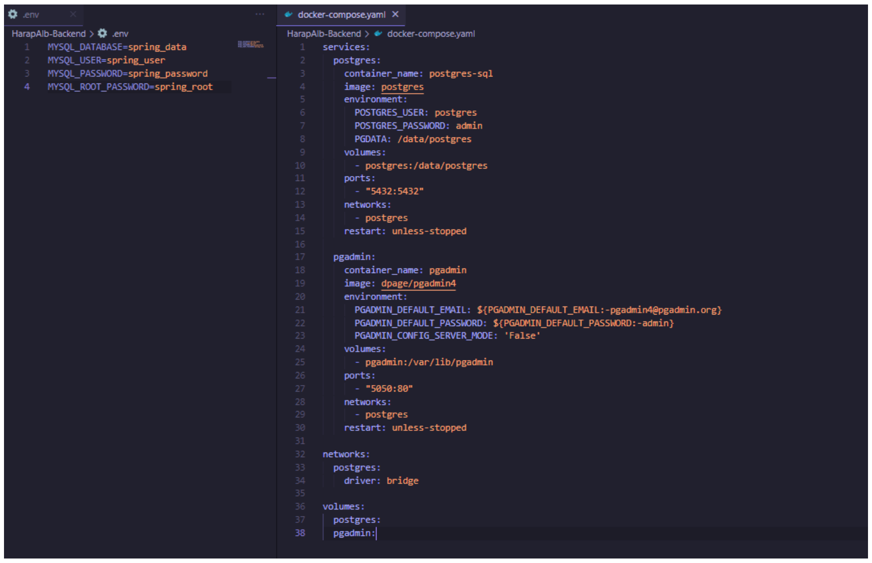
```
Project
HarapAlb-Backend C:\Users\vadea\Desktop\Bachelor_
  .idea
  .mvn
  src
    main
      java
        HarapAlb.HarapAlbBackend
          config
          controllers
          dto
          enums
          exceptions
          filters
          models
          repositories
          services
          utils
          HarapAlbBackendApplication
      resources
    test
  target
  .env
  .gitignore
  docker-compose.yaml
  HELP.md
  mvnw
  mvnw.cmd
  pom.xml
External Libraries
Scratches and Consoles
```

pom.xml (HarapAlb-Backend)

```xml
        </dependency>
        <dependency>
            <groupId>org.apache.commons</groupId>
            <artifactId>commons-lang3</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-web</artifactId>
        </dependency>

        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-devtools</artifactId>
            <scope>runtime</scope>
            <optional>true</optional>
        </dependency>
        <dependency>
            <groupId>org.postgresql</groupId>
            <artifactId>postgresql</artifactId>
            <scope>runtime</scope>
        </dependency>
        <dependency>
            <groupId>org.projectlombok</groupId>
            <artifactId>lombok</artifactId>
            <optional>true</optional>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-test</artifactId>
            <scope>test</scope>
        </dependency>
        <dependency>
            <groupId>org.springframework.security</groupId>
            <artifactId>spring-security-test</artifactId>
            <scope>test</scope>
        </dependency>
        <dependency>
            <groupId>org.hamcrest</groupId>
            <artifactId>hamcrest-library</artifactId>
            <version>2.2</version>
```

Figure 19. Project structure – The back-end components & the 'pom.xml' file

Further included are two additional files: '.env' and 'docker-compose.yaml'. The latter describes the Docker services, networks, and volumes for the application. Docker, a platform empowering developers to automate application deployment, ensures a consistent environment across various systems by encapsulating them into nimble, transportable containers. The former contains environment variables representing configuration parameters for the database, subsequently utilized by services operating within Docker containers.



HarapAlb-Backend > .env

```
MYSQL_DATABASE=spring_data
MYSQL_USER=spring_user
MYSQL_PASSWORD=spring_password
MYSQL_ROOT_PASSWORD=spring_root
```

HarapAlb-Backend > docker-compose.yaml

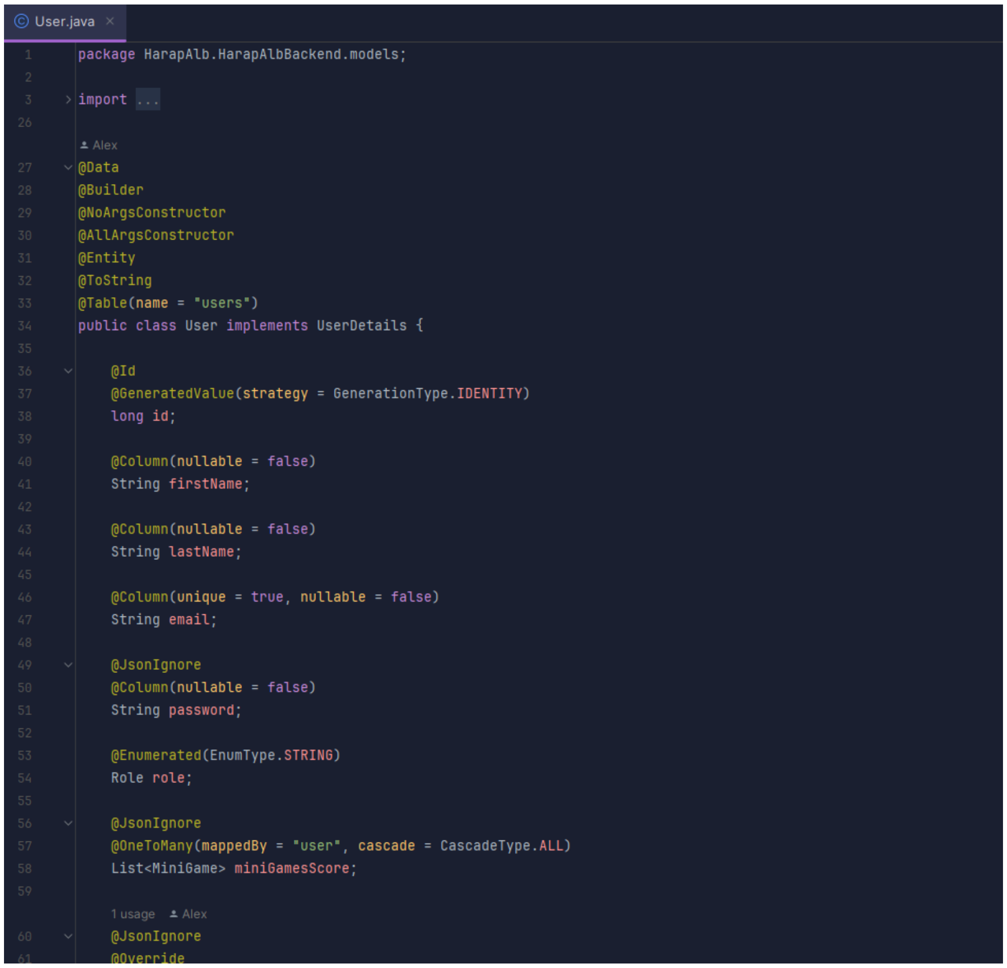
```yaml
services:
  postgres:
    container_name: postgres-sql
    image: postgres
    environment:
      POSTGRES_USER: postgres
      POSTGRES_PASSWORD: admin
      PGDATA: /data/postgres
    volumes:
      - postgres:/data/postgres
    ports:
      - "5432:5432"
    networks:
      - postgres
    restart: unless-stopped

  pgadmin:
    container_name: pgadmin
    image: dpage/pgadmin4
    environment:
      PGADMIN_DEFAULT_EMAIL: ${PGADMIN_DEFAULT_EMAIL:-pgadmin4@pgadmin.org}
      PGADMIN_DEFAULT_PASSWORD: ${PGADMIN_DEFAULT_PASSWORD:-admin}
      PGADMIN_CONFIG_SERVER_MODE: 'False'
    volumes:
      - pgadmin:/var/lib/pgadmin
    ports:
      - "5050:80"
    networks:
      - postgres
    restart: unless-stopped

networks:
  postgres:
    driver: bridge

volumes:
  postgres:
  pgadmin:
```

Figure 20. The '.env' & 'docker-compose.yaml' files

5.1.1 Models

Within a Spring Boot application, models function as crucial Java elements that represent entities or data structures. These classes are frequently enhanced with Spring Data JPA or Lombok annotations to effectively manage database persistence. For instance, the user model utilizes annotations like @Data to automate the generation of standard methods such as getters and setters. Moreover, @Builder aids in implementing a builder pattern, while @NoArgsConstructor and @AllArgsConstructor create constructors with and without parameters, respectively. The presence of the @Entity annotation signifies that the class is a JPA entity, associating its instances with rows in a relational database table.

Furthermore, this class implements UserDetails, an interface provided by Spring Security, to improve user authentication and authorization. Attributes within the class are annotated with @Column to define database column properties, and @JsonIgnore excludes specific fields from serialization. The @Enumerated annotation is utilized to depict fields as enumerated types, and @OneToMany establishes a One-to-Many relationship with the 'MiniGame' entity.

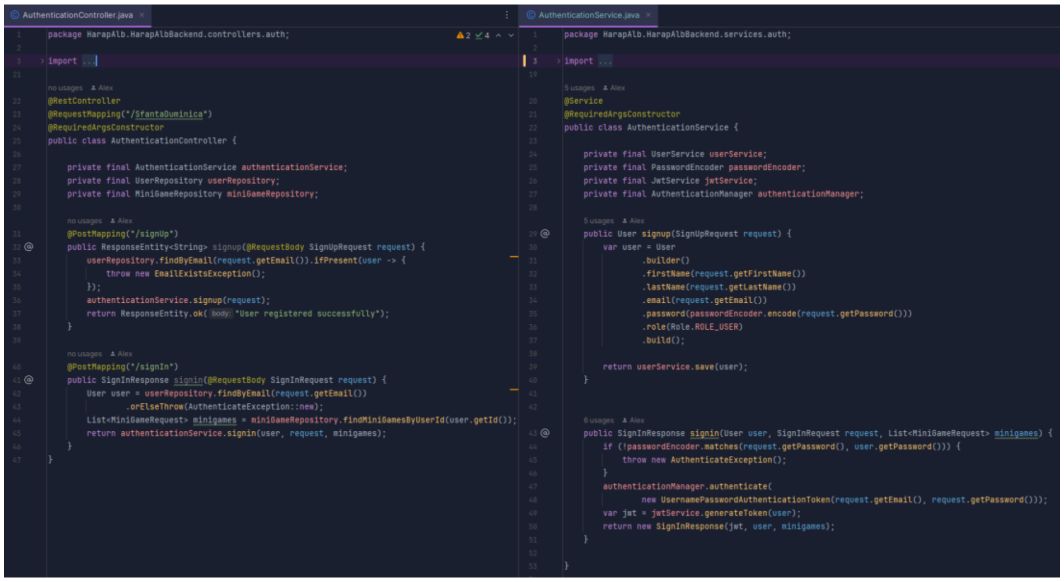


```java
User.java

package HarapAlb.HarapAlbBackend.models;

import ...

Alex
@Data
@Builder
@NoArgsConstructor
@AllArgsConstructor
@Entity
@ToString
@Table(name = "users")
public class User implements UserDetails {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    long id;

    @Column(nullable = false)
    String firstName;

    @Column(nullable = false)
    String lastName;

    @Column(unique = true, nullable = false)
    String email;

    @JsonIgnore
    @Column(nullable = false)
    String password;

    @Enumerated(EnumType.STRING)
    Role role;

    @JsonIgnore
    @OneToMany(mappedBy = "user", cascade = CascadeType.ALL)
    List<MiniGame> miniGamesScore;

    1 usage  Alex
    @JsonIgnore
    @Override
```

Figure 21. The user model

5.1.2 Controllers and services

The application's functionality is connected within the files located in the services directory, a domain easily accessible for exploration through the controllers. These controllers grant us access to data, its manipulation, and various operations for its transformation in a hospitable manner. Every aspect of functionality integrated into the application eagerly anticipates your involvement within this distinguished section of the project.



AuthenticationController.java

```java
package HarapAlb.HarapAlbBackend.controllers.auth;

import ...

@RestController
@RequestMapping("/SfantaDuminica")
@RequiredArgsConstructor
public class AuthenticationController {

    private final AuthenticationService authenticationService;
    private final UserRepository userRepository;
    private final MiniGameRepository miniGameRepository;

    @PostMapping("/signUp")
    public ResponseEntity<String> signup(@RequestBody SignUpRequest request) {
        userRepository.findByEmail(request.getEmail()).ifPresent(user -> {
            throw new EmailExistsException();
        });
        authenticationService.signup(request);
        return ResponseEntity.ok( body: "User registered successfully");
    }

    @PostMapping("/signIn")
    public SignInResponse signin(@RequestBody SignInRequest request) {
        User user = userRepository.findByEmail(request.getEmail())
                .orElseThrow(AuthenticateException::new);
        List<MiniGameRequest> minigames = miniGameRepository.findMiniGamesByUserId(user.getId());
        return authenticationService.signin(user, request, minigames);
    }
}
```

AuthenticationService.java

```java
package HarapAlb.HarapAlbBackend.services.auth;

import ...

@Service
@RequiredArgsConstructor
public class AuthenticationService {

    private final UserService userService;
    private final PasswordEncoder passwordEncoder;
    private final JwtService jwtService;
    private final AuthenticationManager authenticationManager;

    public User signup(SignUpRequest request) {
        var user = User
                .builder()
                .firstName(request.getFirstName())
                .lastName(request.getLastName())
                .email(request.getEmail())
                .password(passwordEncoder.encode(request.getPassword()))
                .role(Role.ROLE_USER)
                .build();

        return userService.save(user);
    }


    public SignInResponse signin(User user, SignInRequest request, List<MiniGameRequest> minigames) {
        if (!passwordEncoder.matches(request.getPassword(), user.getPassword())) {
            throw new AuthenticateException();
        }
        authenticationManager.authenticate(
                new UsernamePasswordAuthenticationToken(request.getEmail(), request.getPassword()));
        var jwt = jwtService.generateToken(user);
        return new SignInResponse(jwt, user, minigames);
    }

}
```

Figure 22. The authentication controller & service

Let's explore the authentication controller, where the sign-up functionality is encapsulated within the @PostMapping("signUp") annotation, making the HTTP POST requests to be directed to the "/signUp" endpoint. This method accepts a SignUpRequest object as the request body, typically containing details like first name, last name, email, and password. Within this method, verification is carried out using the user repository, specifically the method userRepository findByEmail(request.getEmail()), to confirm the presence of a user with the provided email within our database.

Repositories act as dependable entities defining methods for data storage, updating, and retrieval from the database, playing a crucial role in facilitating efficient data management between the application and the underlying database. Upon identifying a matching user, it raises an EmailExistsException to alert the system. Otherwise, it proceeds to register the user by calling authenticationService.signup(request).

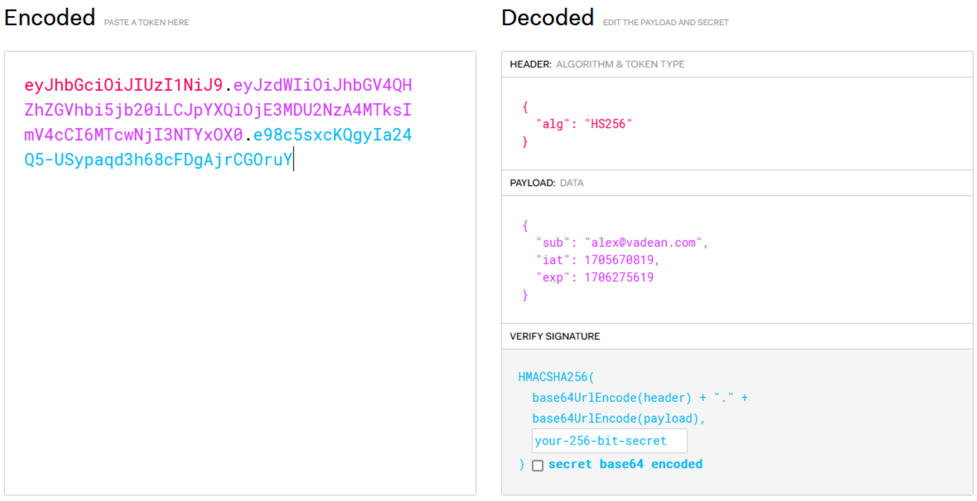
Throughout this process, it extracts attributes and utilizes the builder pattern to create a new User object built with the provided details. Additionally, the user's password undergoes secure encoding through passwordEncoder.encode(request.getPassword()). This robust encoding mechanism is a hallmark of the Spring Security framework, providing features for secure password encoding and authentication.

The user's role is designated as ROLE_USER, denoting a standard role for regular users. The User object is then saved in the database by invoking userService.save(user), ensuring its persistence. Subsequently, the method happily returns the saved User object and issues a ResponseEntity with an "OK" status, along with a message confirming the successful registration of the user.

5.1.3 Application security

An important aspect in any application relates to data security. Considering the operation of the application with sensitive data like real names and emails, every user must undergo authentication. Authentication refers to the process where a user, by submitting data associated with their account, gains access to the platform. This process is essential for maintaining security, as depicted in the use-case diagram.

Upon a user's authentication, the relevant data will be provided along with a JWT (JSON Web Token) consisting of three parts: header, payload, and signature. The header includes the algorithm used for token signing, with HS256 being the current choice. The payload contains the user's email, the timestamp of issuance, and the expiration time. The final part encodes the header and payload using Base64, the algorithm specified in the header along with a private key, defined in the environment variable file. This confidential key is vital for decoding the payload, as the user's information remains inaccessible without it.



```
eyJhbGciOiJIUzI1NiJ9.eyJzdWIiOiJhbGV4QH
ZhZGVhbi5jb20iLCJpYXQiOjE3MDU2NzA4MTksI
mV4cCI6MTcwNjI3NTYxOX0.e98c5sxcKQgyIa24
Q5-USypaqd3h68cFDgAjrCGOruY
```

```
{
  "alg": "HS256"
}
```

```
{
  "sub": "alex@vadean.com",
  "iat": 1705670819,
  "exp": 1706275619
}
```

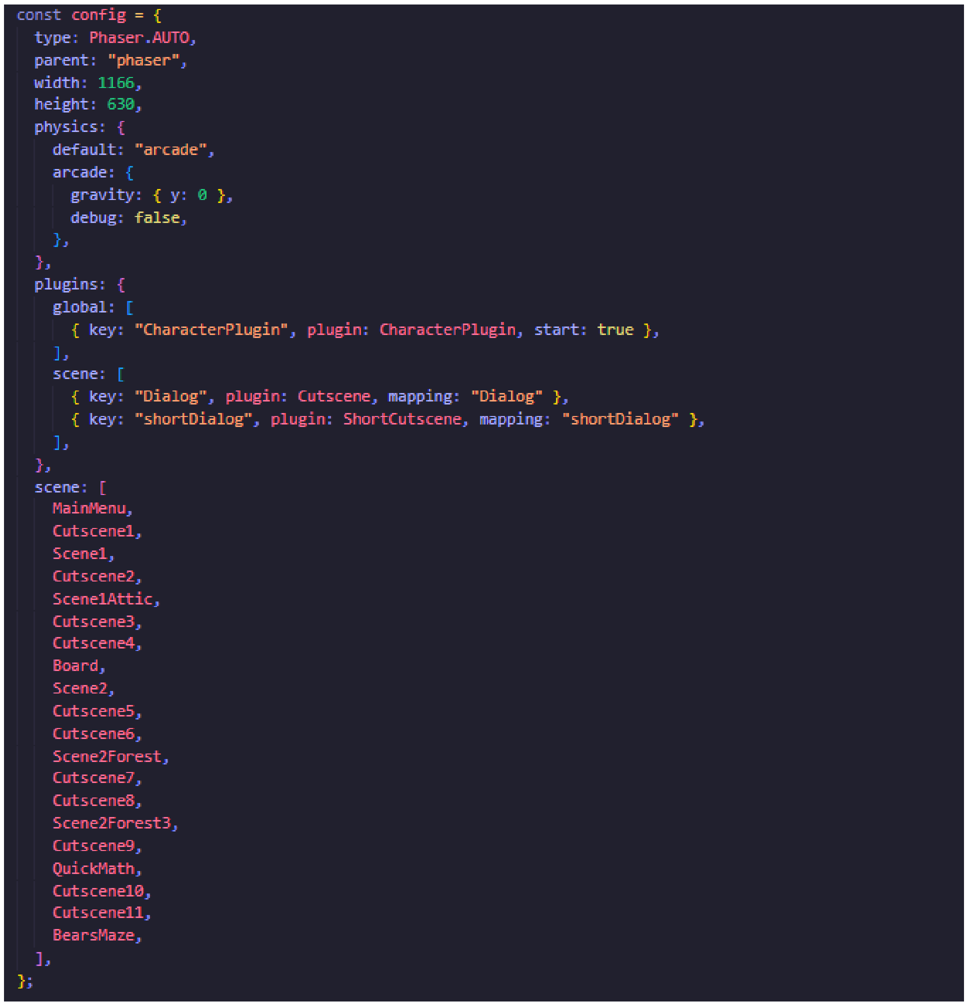
```
HMACSHA256(
  base64UrlEncode(header) + "." +
  base64UrlEncode(payload),
  your-256-bit-secret
) ☐ secret base64 encoded
```

Figure 23. A JWT decoded

5.2 Front-end components

The front-end is established using JavaScript in conjunction with two frameworks. The initial one is React.js, which is grounded on the component principle and offers the primary benefit of code reusability, enabling the segmentation of UI elements into reusable components. Consequently, it ensures code modularity and simplifies the process of modifying interface components. Each of these components was integrated into the project within the "components" directory.

The second framework is Phaser.js, serving as the core framework for the game. It employs a div element as a canvas and scenes to exhibit them on said canvas. A Scene bears resemblance to a theatrical setting in the physical world, containing subordinate elements akin to actors in a real scene. These elements can be sprites, images, and containers.



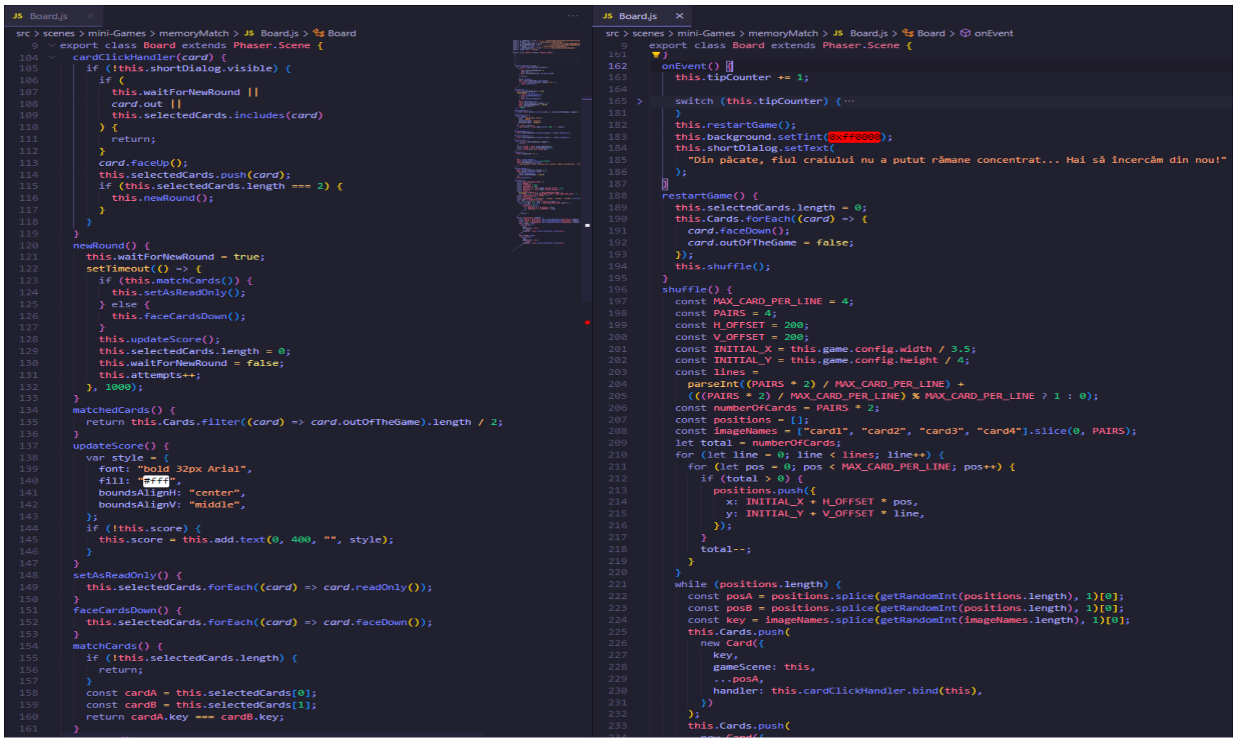
```
const config = {
  type: Phaser.AUTO,
  parent: "phaser",
  width: 1166,
  height: 630,
  physics: {
    default: "arcade",
    arcade: {
      gravity: { y: 0 },
      debug: false,
    },
  },
  plugins: {
    global: [
      { key: "CharacterPlugin", plugin: CharacterPlugin, start: true },
    ],
    scene: [
      { key: "Dialog", plugin: Cutscene, mapping: "Dialog" },
      { key: "shortDialog", plugin: ShortCutscene, mapping: "shortDialog" },
    ],
  },
  scene: [
    MainMenu,
    Cutscene1,
    Scene1,
    Cutscene2,
    Scene1Attic,
    Cutscene3,
    Cutscene4,
    Board,
    Scene2,
    Cutscene5,
    Cutscene6,
    Scene2Forest,
    Cutscene7,
    Cutscene8,
    Scene2Forest3,
    Cutscene9,
    QuickMath,
    Cutscene10,
    Cutscene11,
    BearsMaze,
  ],
};
```

Figure 24. The configuration of the application

The configuration typically consists of the type, indicating the rendering type, parent, the DOM element id where the game canvas will be appended, width, height, physics for establishing the top-down perspective, plugins, and scene, an array encompassing all the scenes utilized in the game.

A scene encompasses the following inherent methods: constructor, init, preload, create, and update. The development process closely resembles web development, allowing for extensive code reuse. The constructor includes the scene key. The init method activates upon scene creation, receiving the Data object that can be transmitted when calling game.scene.add(Data) or game.scene.start(Data). Preload encompasses the assets required to be loaded before the scene and their sources, essentially preparing assets for the scene. Create is invoked upon scene creation, setting the positions of scene elements. Lastly, update is executed with each render frame (typically 60 times per second), commonly involving redrawing, object movement, and similar tasks.

In the project I am working on, scenes are classified into three categories: cutscene, gameplay, and mini-game. It is imperative to divide the scenes as having all elements in a single scene would result in prolonged loading times and execution. A gameplay scene typically involves substantial tasks like loading and positioning numerous assets such as textures and scripts. The maps are created using Tiled, exported in JSON format, and subsequently loaded along with the player. Tiled maps are comprised of layers, with each layer representing a specific aspect of the game world's design. Layers positioned on top display their tiles above those located beneath them.

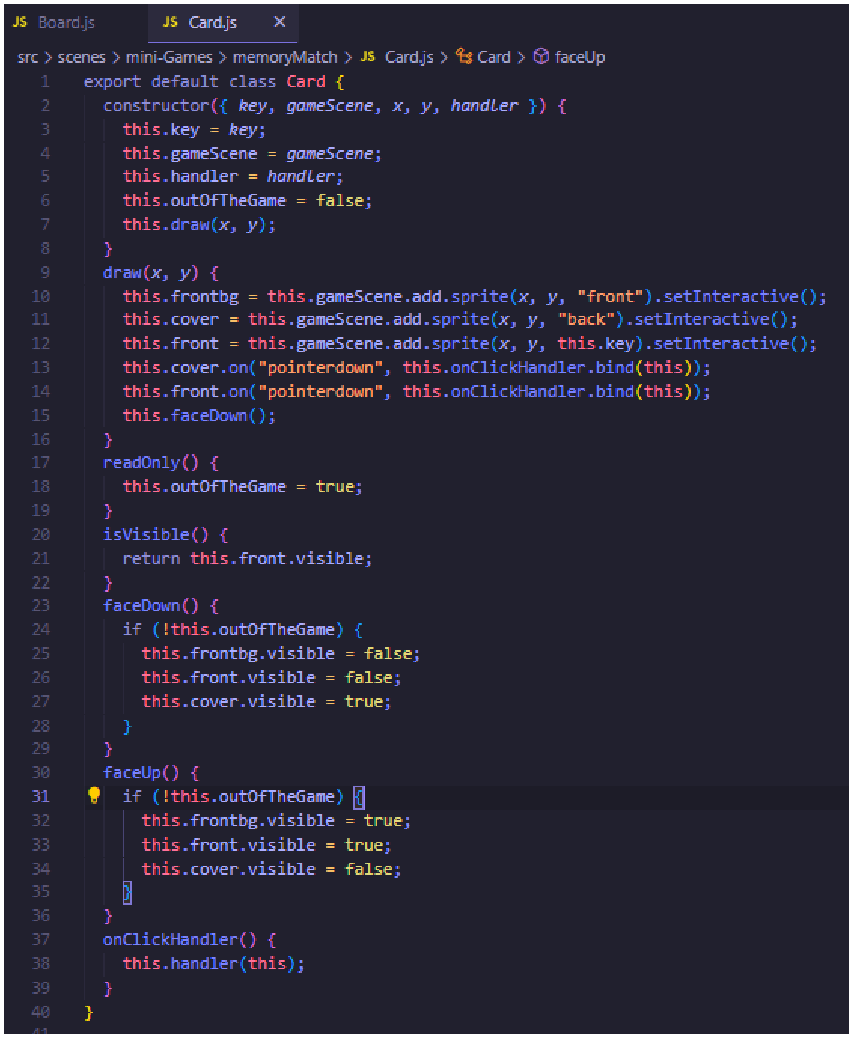


```
export class Board extends Phaser.Scene {
  cardClickHandler(card) {
    if (!this.shortDialog.visible) {
      if (
        this.waitForNewRound ||
        card.out ||
        this.selectedCards.includes(card)
      ) {
        return;
      }
      card.faceUp();
      this.selectedCards.push(card);
      if (this.selectedCards.length === 2) {
        this.newRound();
      }
    }
  }
  newRound() {
    this.waitForNewRound = true;
    setTimeout(() => {
      if (this.matchCards()) {
        this.setAsReadOnly();
      } else {
        this.faceCardsDown();
      }
      this.updateScore();
      this.selectedCards.length = 0;
      this.waitForNewRound = false;
      this.attempts++;
    }, 1000);
  }
  matchedCards() {
    return this.Cards.filter((card) => card.outOfTheGame).length / 2;
  }
  updateScore() {
    var style = {
      font: "bold 32px Arial",
      fill: "#fff",
      boundsAlignH: "center",
      boundsAlignV: "middle",
    };
    if (!this.score) {
      this.score = this.add.text(0, 400, "", style);
    }
  }
  setAsReadOnly() {
    this.selectedCards.forEach((card) => card.readOnly());
  }
  faceCardsDown() {
    this.selectedCards.forEach((card) => card.faceDown());
  }
  matchCards() {
    if (!this.selectedCards.length) {
      return;
    }
    const cardA = this.selectedCards[0];
    const cardB = this.selectedCards[1];
    return cardA.key === cardB.key;
  }
```

Board.js
src > scenes > mini-Games > memoryMatch > Board.js > Board > onEvent

```
export class Board extends Phaser.Scene {
  }
  onEvent() {
    this.tipCounter += 1;

    switch (this.tipCounter) {…
    }
    this.restartGame();
    this.background.setTint(0xff0000);
    this.shortDialog.setText(
      "Din păcate, fiul craiului nu a putut rămane concentrat... Hai să încercăm din nou!"
    );
  }
  restartGame() {
    this.selectedCards.length = 0;
    this.Cards.forEach((card) => {
      card.faceDown();
      card.outOfTheGame = false;
    });
    this.shuffle();
  }
  shuffle() {
    const MAX_CARD_PER_LINE = 4;
    const PAIRS = 4;
    const H_OFFSET = 200;
    const V_OFFSET = 200;
    const INITIAL_X = this.game.config.width / 3.5;
    const INITIAL_Y = this.game.config.height / 4;
    const lines =
      parseInt((PAIRS * 2) / MAX_CARD_PER_LINE) +
      (((PAIRS * 2) / MAX_CARD_PER_LINE) % MAX_CARD_PER_LINE ? 1 : 0);
    const numberOfCards = PAIRS * 2;
    const positions = [];
    const imageNames = ["card1", "card2", "card3", "card4"].slice(0, PAIRS);
    let total = numberOfCards;
    for (let line = 0; line < lines; line++) {
      for (let pos = 0; pos < MAX_CARD_PER_LINE; pos++) {
        if (total > 0) {
          positions.push({
            x: INITIAL_X + H_OFFSET * pos,
            y: INITIAL_Y + V_OFFSET * line,
          });
        }
        total--;
      }
    }
    while (positions.length) {
      const posA = positions.splice(getRandomInt(positions.length), 1)[0];
      const posB = positions.splice(getRandomInt(positions.length), 1)[0];
      const key = imageNames.splice(getRandomInt(imageNames.length), 1)[0];
      this.Cards.push(
        new Card({
          key,
          gameScene: this,
          ...posA,
          handler: this.cardClickHandler.bind(this),
        })
      );
      this.Cards.push(
```

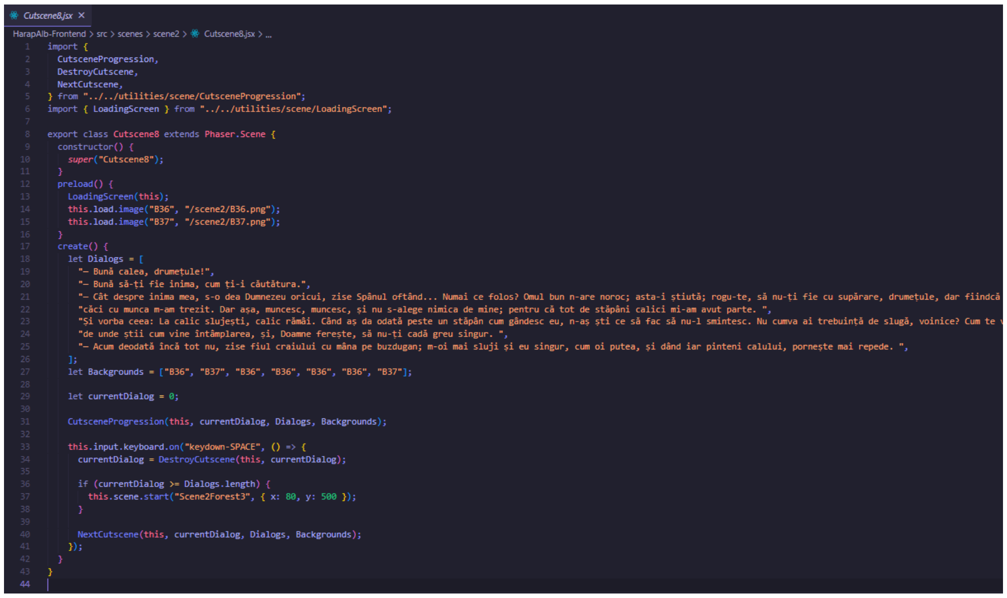
Figure 25. The main logic behind the first mini-game


Board.js
Card.js
src > scenes > mini-Games > memoryMatch > Card.js > Card > faceUp

```javascript
export default class Card {
  constructor({ key, gameScene, x, y, handler }) {
    this.key = key;
    this.gameScene = gameScene;
    this.handler = handler;
    this.outOfTheGame = false;
    this.draw(x, y);
  }
  draw(x, y) {
    this.frontbg = this.gameScene.add.sprite(x, y, "front").setInteractive();
    this.cover = this.gameScene.add.sprite(x, y, "back").setInteractive();
    this.front = this.gameScene.add.sprite(x, y, this.key).setInteractive();
    this.cover.on("pointerdown", this.onClickHandler.bind(this));
    this.front.on("pointerdown", this.onClickHandler.bind(this));
    this.faceDown();
  }
  readOnly() {
    this.outOfTheGame = true;
  }
  isVisible() {
    return this.front.visible;
  }
  faceDown() {
    if (!this.outOfTheGame) {
      this.frontbg.visible = false;
      this.front.visible = false;
      this.cover.visible = true;
    }
  }
  faceUp() {
    if (!this.outOfTheGame) {
      this.frontbg.visible = true;
      this.front.visible = true;
      this.cover.visible = false;
    }
  }
  onClickHandler() {
    this.handler(this);
  }
}
```

Figure 26. The logic behind each way a card can behave

A cutscene scene primarily involves transitioning through frames and text utilizing a specialized plugin designed for displaying text. The reason Phaser does not support animated files is largely due to constraints within the canvas API and WebGL. The frames utilized are produced using Leonardo, an AI toolset that leverages one of its finely-tuned models, Absolute Reality v1.6. This officially sanctioned photorealistic model is ideal for the narrative's requirements, encompassing scenery and dialogues.

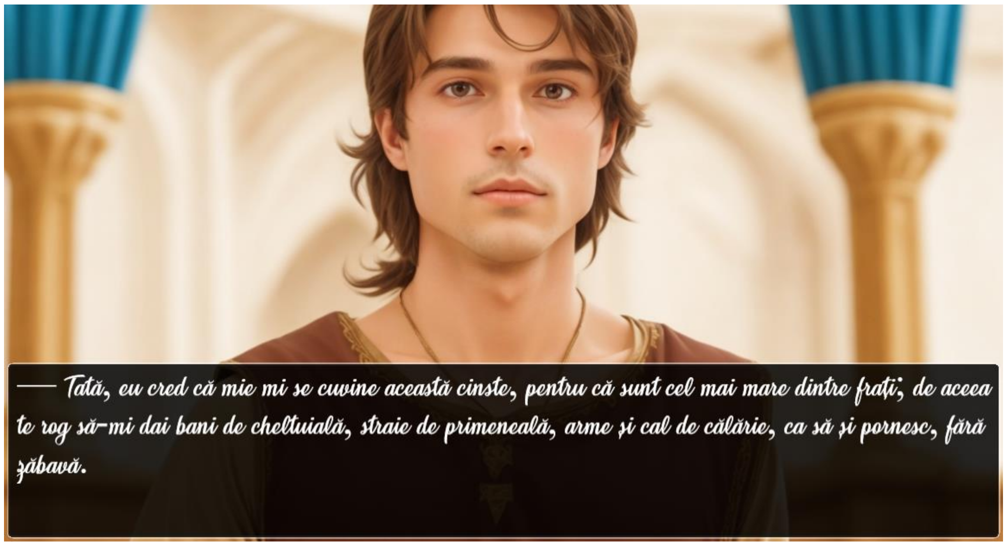


Cutscene8.jsx ×

HarapAlb-Frontend > src > scenes > scene2 > Cutscene8.jsx > ...

```
import {
  CutsceneProgression,
  DestroyCutscene,
  NextCutscene,
} from "../../utilities/scene/CutsceneProgression";
import { LoadingScreen } from "../../utilities/scene/LoadingScreen";

export class Cutscene8 extends Phaser.Scene {
  constructor() {
    super("Cutscene8");
  }
  preload() {
    LoadingScreen(this);
    this.load.image("B36", "/scene2/B36.png");
    this.load.image("B37", "/scene2/B37.png");
  }
  create() {
    let Dialogs = [
      "— Bună calea, drumețule!",
      "— Bună să-ți fie inima, cum ți-i căutătura.",
      "— Cât despre inima mea, s-o dea Dumnezeu oricui, zise Spânul oftând... Numai ce folos? Omul bun n-are noroc; asta-i știută; rogu-te, să nu-ți fie cu supărare, drumețule, dar fiindcă
      "căci cu munca m-am trezit. Dar așa, muncesc, muncesc, și nu s-alege nimica de mine; pentru că tot de stăpâni calici mi-am avut parte. ",
      "Și vorba ceea: La calic slujești, calic rămâi. Când aș da odată peste un stăpân cum gândesc eu, n-aș ști ce să fac să nu-l smintesc. Nu cumva ai trebuință de slugă, voinice? Cum te
      "de unde știi cum vine întâmplarea, și, Doamne ferește, să nu-ți cadă greu singur. ",
      "— Acum deodată încă tot nu, zise fiul craiului cu mâna pe buzdugan; m-oi mai sluji și eu singur, cum oi putea, și dând iar pinteni calului, pornește mai repede. ",
    ];
    let Backgrounds = ["B36", "B37", "B36", "B36", "B36", "B36", "B37"];

    let currentDialog = 0;

    CutsceneProgression(this, currentDialog, Dialogs, Backgrounds);

    this.input.keyboard.on("keydown-SPACE", () => {
      currentDialog = DestroyCutscene(this, currentDialog);

      if (currentDialog >= Dialogs.length) {
        this.scene.start("Scene2Forest3", { x: 80, y: 500 });
      }

      NextCutscene(this, currentDialog, Dialogs, Backgrounds);
    });
  }
}
```

Figure 27. How a cutscene file typically looks like

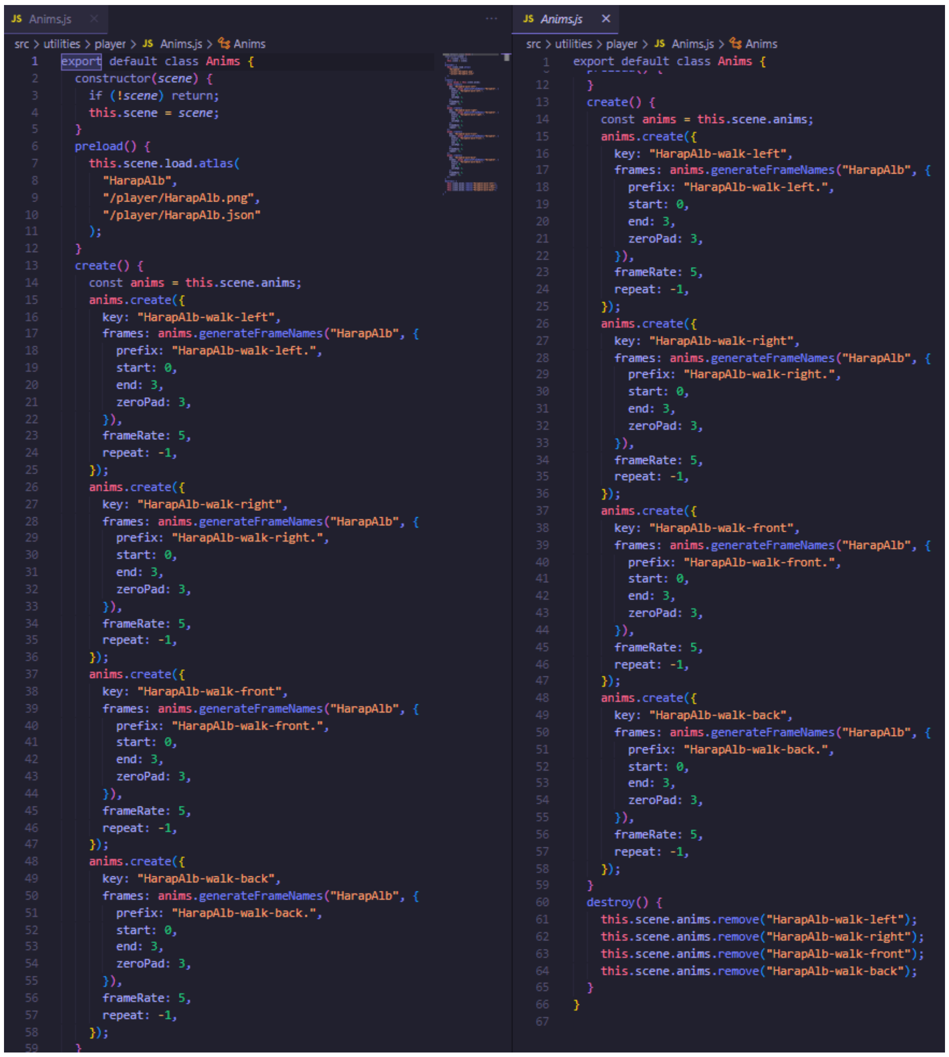


— Tată, eu cred că mie mi se cuvine această cinste, pentru că sunt cel mai mare dintre frați; de aceea te rog să-mi dai bani de cheltuială, straie de primeneală, arme și cal de călărie, ca să și pornesc, fără zăbavă.

Figure 28. How a cutscene looks like

An AI text-to-image generator commonly employs a machine learning approach known as artificial neural networks, which can process textual input to generate an image. Presently, text-to-image AI models are shifting towards diffusion models. These models are trained on vast image datasets, each image accompanied by a descriptive caption to establish a correlation between textual descriptions and images. Throughout this training, the network acquires knowledge about various aspects of the world, such as color schemes and elements that contribute to the overall essence of an image. Once trained, the models can interpret a text prompt from the user, initially creating a low-resolution image and progressively adding details until a full high-resolution image is formed.

Each of the available mini-games - memory matcher, quick arithmetic questions, and procedurally generated maze - possesses its own distinct code and logic, separated from the other scene types: cutscene and gameplay. Given the shared methodologies across scenes, the principle of code reuse can be implemented here. Common methods employed in these scenes involve tasks like centering frames on the canvas, managing the loading screen, positioning the player, handling interactions, player animations, and movement. These shared methods, present in nearly all scenes, have been systematically organized within the project under the 'utilities' directory.


Anims.js

src > utilities > player > Anims.js > Anims

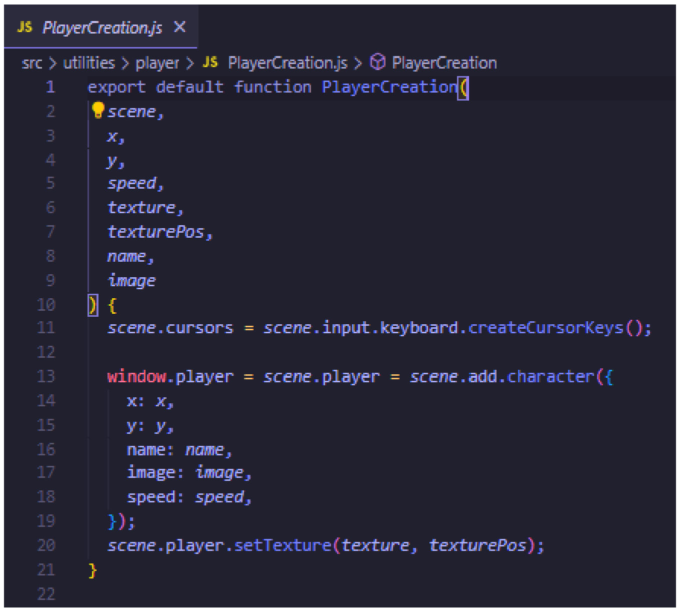
```
export default class Anims {
  constructor(scene) {
    if (!scene) return;
    this.scene = scene;
  }
  preload() {
    this.scene.load.atlas(
      "HarapAlb",
      "/player/HarapAlb.png",
      "/player/HarapAlb.json"
    );
  }
  create() {
    const anims = this.scene.anims;
    anims.create({
      key: "HarapAlb-walk-left",
      frames: anims.generateFrameNames("HarapAlb", {
        prefix: "HarapAlb-walk-left.",
        start: 0,
        end: 3,
        zeroPad: 3,
      }),
      frameRate: 5,
      repeat: -1,
    });
    anims.create({
      key: "HarapAlb-walk-right",
      frames: anims.generateFrameNames("HarapAlb", {
        prefix: "HarapAlb-walk-right.",
        start: 0,
        end: 3,
        zeroPad: 3,
      }),
      frameRate: 5,
      repeat: -1,
    });
    anims.create({
      key: "HarapAlb-walk-front",
      frames: anims.generateFrameNames("HarapAlb", {
        prefix: "HarapAlb-walk-front.",
        start: 0,
        end: 3,
        zeroPad: 3,
      }),
      frameRate: 5,
      repeat: -1,
    });
    anims.create({
      key: "HarapAlb-walk-back",
      frames: anims.generateFrameNames("HarapAlb", {
        prefix: "HarapAlb-walk-back.",
        start: 0,
        end: 3,
        zeroPad: 3,
      }),
      frameRate: 5,
      repeat: -1,
    });
  }
```

Anims.js

src > utilities > player > Anims.js > Anims

```
export default class Anims {
  }
  create() {
    const anims = this.scene.anims;
    anims.create({
      key: "HarapAlb-walk-left",
      frames: anims.generateFrameNames("HarapAlb", {
        prefix: "HarapAlb-walk-left.",
        start: 0,
        end: 3,
        zeroPad: 3,
      }),
      frameRate: 5,
      repeat: -1,
    });
    anims.create({
      key: "HarapAlb-walk-right",
      frames: anims.generateFrameNames("HarapAlb", {
        prefix: "HarapAlb-walk-right.",
        start: 0,
        end: 3,
        zeroPad: 3,
      }),
      frameRate: 5,
      repeat: -1,
    });
    anims.create({
      key: "HarapAlb-walk-front",
      frames: anims.generateFrameNames("HarapAlb", {
        prefix: "HarapAlb-walk-front.",
        start: 0,
        end: 3,
        zeroPad: 3,
      }),
      frameRate: 5,
      repeat: -1,
    });
    anims.create({
      key: "HarapAlb-walk-back",
      frames: anims.generateFrameNames("HarapAlb", {
        prefix: "HarapAlb-walk-back.",
        start: 0,
        end: 3,
        zeroPad: 3,
      }),
      frameRate: 5,
      repeat: -1,
    });
  }
  destroy() {
    this.scene.anims.remove("HarapAlb-walk-left");
    this.scene.anims.remove("HarapAlb-walk-right");
    this.scene.anims.remove("HarapAlb-walk-front");
    this.scene.anims.remove("HarapAlb-walk-back");
  }
}
```

Figure 29. The player’s creation and animation files


PlayerCreation.js
src > utilities > player > PlayerCreation.js > PlayerCreation

```javascript
export default function PlayerCreation(
  scene,
  x,
  y,
  speed,
  texture,
  texturePos,
  name,
  image
) {
  scene.cursors = scene.input.keyboard.createCursorKeys();

  window.player = scene.player = scene.add.character({
    x: x,
    y: y,
    name: name,
    image: image,
    speed: speed,
  });
  scene.player.setTexture(texture, texturePos);
}
```

Figure 30. How the utilities folder structure looks like

From a visual perspective, the pages are designed with a minimalist and intuitive interface in mind. The sidebar option in the dashboard conceals unnecessary buttons, reduces the need to create a separate page for personal profiles, and decreases the number of clicks, considering usability.

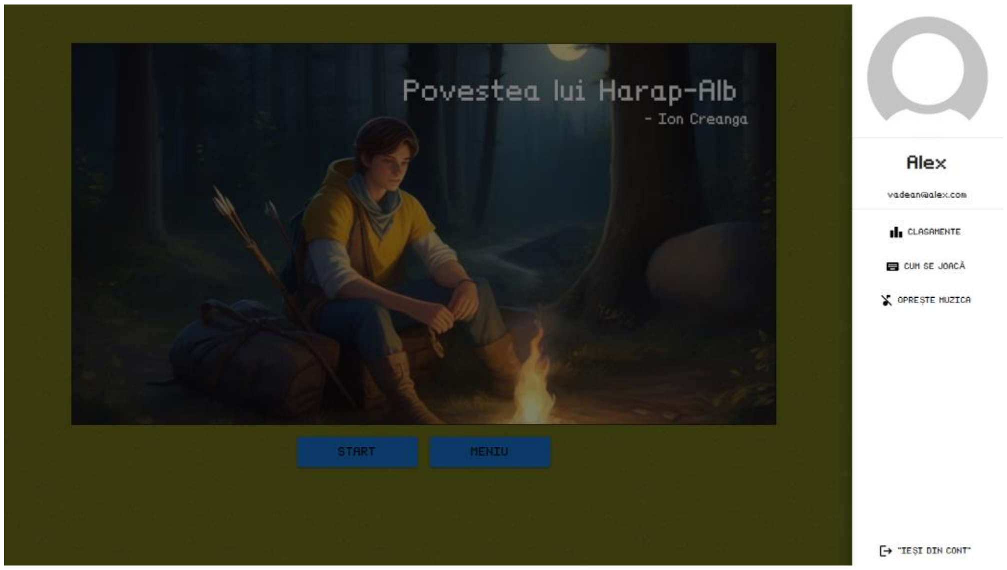
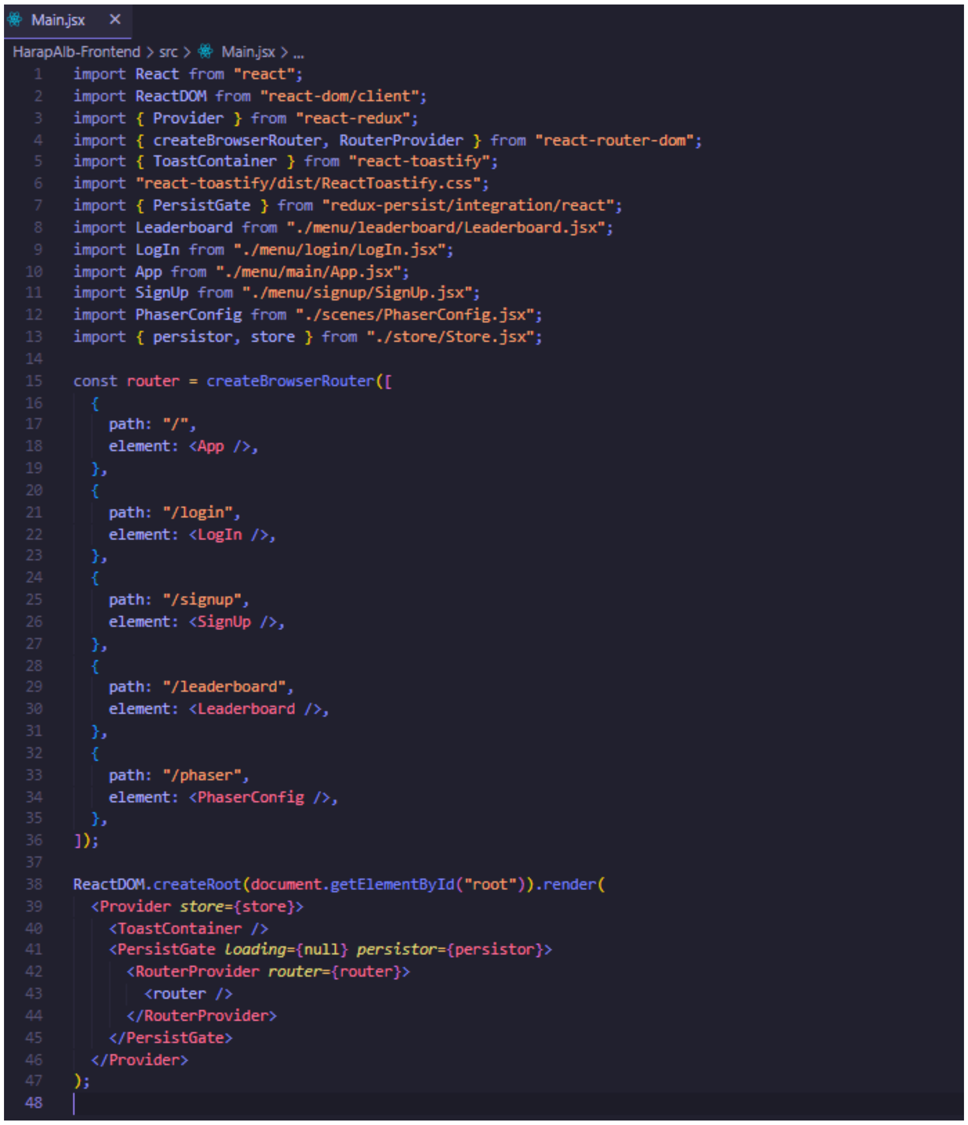


Figure 31. How the sidebar looks like opened in the dashboard

A benefit provided by the React library at the implementation level is associated with the JSX syntax, which allows embedding HTML code directly within JavaScript. This eliminates the need to manually manipulate the DOM tree in the conventional way and enables the utilization of

JavaScript functionalities to directly interact with HTML code. Components in React are essentially functions where constants, variables, and integrated functions like "hooks" can be defined. These hooks are exclusive to function-based React components, which can return HTML code. These components constitute the virtual DOM, stored in memory and synchronized with the actual DOM tree via ReactDOM.

In the Main.jsx file, an external routing library called React Router Dom was employed, as React, not being a framework, lacks certain essential features for developing a comprehensive application. Within this file, routes along with their respective paths and components to be displayed upon navigation are defined.



```jsx
import React from "react";
import ReactDOM from "react-dom/client";
import { Provider } from "react-redux";
import { createBrowserRouter, RouterProvider } from "react-router-dom";
import { ToastContainer } from "react-toastify";
import "react-toastify/dist/ReactToastify.css";
import { PersistGate } from "redux-persist/integration/react";
import Leaderboard from "./menu/leaderboard/Leaderboard.jsx";
import LogIn from "./menu/login/LogIn.jsx";
import App from "./menu/main/App.jsx";
import SignUp from "./menu/signup/SignUp.jsx";
import PhaserConfig from "./scenes/PhaserConfig.jsx";
import { persistor, store } from "./store/Store.jsx";

const router = createBrowserRouter([
  {
    path: "/",
    element: <App />,
  },
  {
    path: "/login",
    element: <LogIn />,
  },
  {
    path: "/signup",
    element: <SignUp />,
  },
  {
    path: "/leaderboard",
    element: <Leaderboard />,
  },
  {
    path: "/phaser",
    element: <PhaserConfig />,
  },
]);

ReactDOM.createRoot(document.getElementById("root")).render(
  <Provider store={store}>
    <ToastContainer />
    <PersistGate loading={null} persistor={persistor}>
      <RouterProvider router={router}>
        <router />
      </RouterProvider>
    </PersistGate>
  </Provider>
);
```

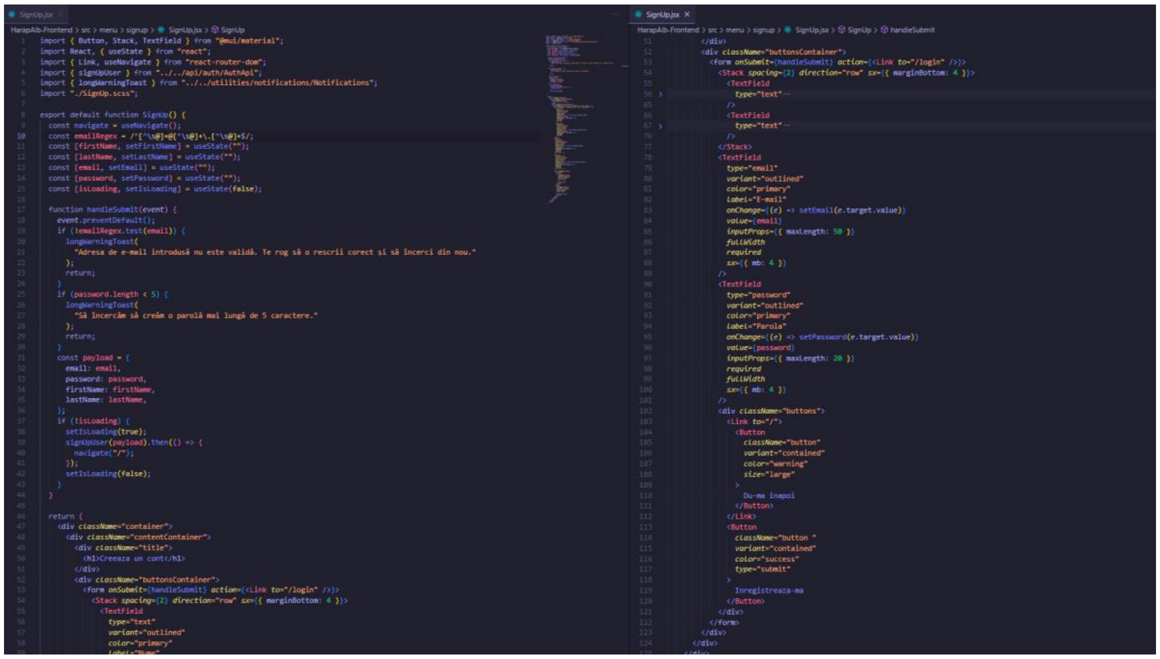
Figure 32. The application’s routes

5.2.1 React component

Components encompass both visual interface elements and programming logic aspects of the application. Within these components, additional functions are implemented to be executed when specific events occur, such as form submission. These functions are predominantly asynchronous, implying a delay in receiving the actual response post request initiation. For instance, the user registration form exemplifies the implementation of a component.

The user registration component consists of form fields for first name, last name, email, and password, with navigation buttons for submission and returning. It utilizes the ‘useState’ hook to manage state variables for the mentioned fields, preventing duplicate submissions. This hook defines pairs of constants and functions responsible for modifying the constant values, including the option to set default values. Additionally, it incorporates the ‘useNavigate’ hook from a navigation library to direct users to the home page post successful registration.

The ‘handleSubmit’ function is activated upon submission of the registration form. It conducts client-side validation for email format and password length, showcasing appropriate warning messages if necessary. Upon validation, it constructs a payload object and invokes the ‘signUpUser’ function to process the registration.



```
HarapAlb-Frontend > src > menu > signup > SignUp.jsx > SignUp
import { Button, Stack, TextField } from "@mui/material";
import React, { useState } from "react";
import { Link, useNavigate } from "react-router-dom";
import { signUpUser } from "../../api/auth/AuthApi";
import { longWarningToast } from "../../utilities/notifications/Notifications";
import "./SignUp.scss";

export default function SignUp() {
  const navigate = useNavigate();
  const emailRegex = /^[^\s@]+@[^\s@]+\.[^\s@]+$/;
  const [firstName, setFirstName] = useState("");
  const [lastName, setLastName] = useState("");
  const [email, setEmail] = useState("");
  const [password, setPassword] = useState("");
  const [isLoading, setIsLoading] = useState(false);

  function handleSubmit(event) {
    event.preventDefault();
    if (!emailRegex.test(email)) {
      longWarningToast(
        "Adresa de e-mail introdusă nu este validă. Te rog să o rescrii corect și să încerci din nou."
      );
      return;
    }
    if (password.length < 5) {
      longWarningToast(
        "Să încercăm să creăm o parolă mai lungă de 5 caractere."
      );
      return;
    }
    const payload = {
      email: email,
      password: password,
      firstName: firstName,
      lastName: lastName,
    };
    if (!isLoading) {
      setIsLoading(true);
      signUpUser(payload).then(() => {
        navigate("/");
      });
      setIsLoading(false);
    }
  }

  return (
    <div className="container">
      <div className="contentContainer">
        <div className="title">
          <h1>Creeaza un cont</h1>
        </div>
        <div className="buttonsContainer">
          <form onSubmit={handleSubmit} action={<Link to="/login" />}>
            <Stack spacing={2} direction="row" sx={{ marginBottom: 4 }}>
              <TextField
                type="text"
                variant="outlined"
                color="primary"
```

```
HarapAlb-Frontend > src > menu > signup > SignUp.jsx > SignUp > handleSubmit
        </div>
        <div className="buttonsContainer">
          <form onSubmit={handleSubmit} action={<Link to="/login" />}>
            <Stack spacing={2} direction="row" sx={{ marginBottom: 4 }}>
              <TextField
                type="text" ···
              />
              <TextField
                type="text" ···
              />
            </Stack>
            <TextField
              type="email"
              variant="outlined"
              color="primary"
              label="E-mail"
              onChange={(e) => setEmail(e.target.value)}
              value={email}
              inputProps={{ maxLength: 50 }}
              fullWidth
              required
              sx={{ mb: 4 }}
            />
            <TextField
              type="password"
              variant="outlined"
              color="primary"
              label="Parola"
              onChange={(e) => setPassword(e.target.value)}
              value={password}
              inputProps={{ maxLength: 20 }}
              required
              fullWidth
              sx={{ mb: 4 }}
            />
            <div className="buttons">
              <Link to="/">
                <Button
                  className="button"
                  variant="contained"
                  color="warning"
                  size="large"
                >
                  Du-ma inapoi
                </Button>
              </Link>
              <Button
                className="button "
                variant="contained"
                color="success"
                type="submit"
              >
                Inregistreaza-ma
              </Button>
            </div>
          </form>
        </div>
      </div>
```

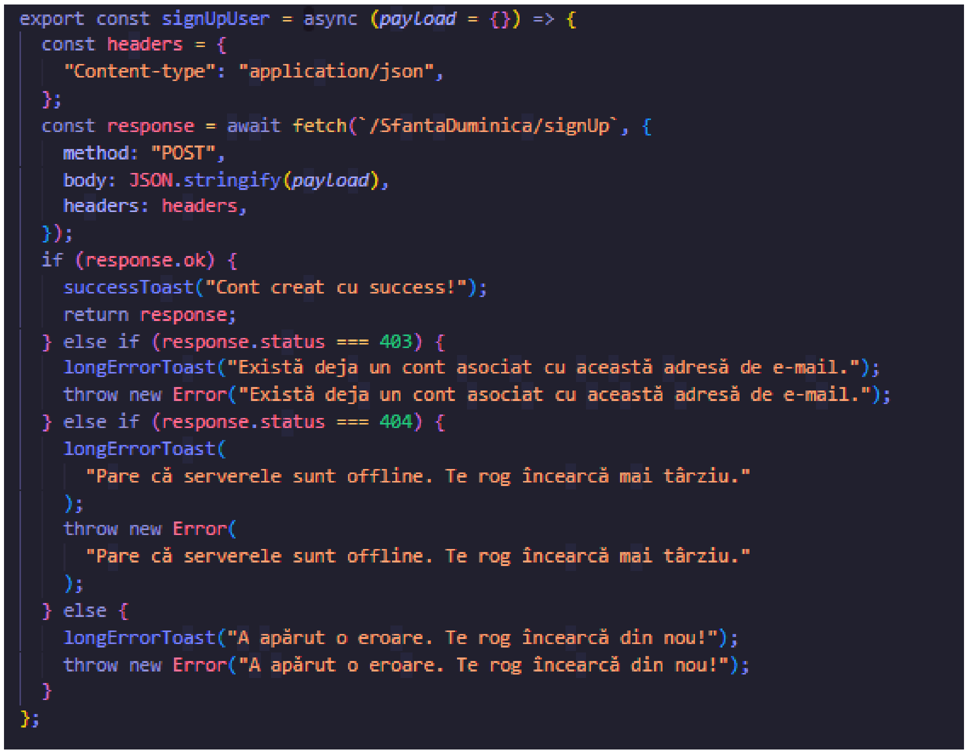
Figure 33. React component for the sign up

The left side contains JavaScript functions and the right side returns JSX (HTML code embedded in JavaScript)

5.2.2 Calling the API from the front end

In the folder named "api" located at the project's root, the services have been implemented to facilitate communication with the backend. Each file is dedicated to a specific service, containing a set of functions to ensure the separation of responsibilities and enhance code modularity. Within the respective file, the environment variables for each endpoint have been specified, enabling the utilization of the axios library for making HTTP requests to the backend.

Illustratively, when considering the process of registering a user, the function is designed to accept a payload object as an input parameter. It effectively employs the fetch function to dispatch a POST request to the /SfantaDuminica/signUp endpoint, along with the provided payload formatted in JSON, while also configuring the content type in the headers. Following the request transmission, an evaluation of the response status is conducted. Depending on the received error code, a tailored toast message is dispatched; conversely, in case of success, the response is returned.



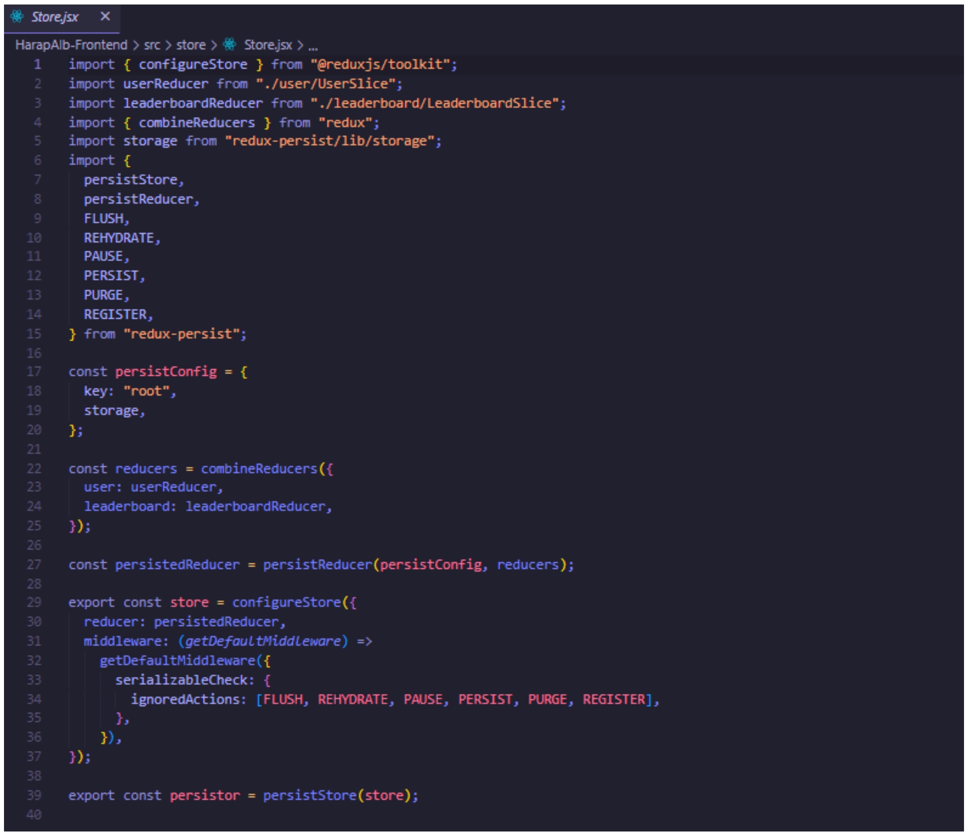
```
export const signUpUser = async (payload = {}) => {
  const headers = {
    "Content-type": "application/json",
  };
  const response = await fetch(`/SfantaDuminica/signUp`, {
    method: "POST",
    body: JSON.stringify(payload),
    headers: headers,
  });
  if (response.ok) {
    successToast("Cont creat cu success!");
    return response;
  } else if (response.status === 403) {
    longErrorToast("Există deja un cont asociat cu această adresă de e-mail.");
    throw new Error("Există deja un cont asociat cu această adresă de e-mail.");
  } else if (response.status === 404) {
    longErrorToast(
      "Pare că serverele sunt offline. Te rog încearcă mai târziu."
    );
    throw new Error(
      "Pare că serverele sunt offline. Te rog încearcă mai târziu."
    );
  } else {
    longErrorToast("A apărut o eroare. Te rog încearcă din nou!");
    throw new Error("A apărut o eroare. Te rog încearcă din nou!");
  }
};
```

Figure 34. The signup HTTP request

5.2.3 Managing application state using the Redux Toolkit

The challenge of accessing data across multiple components that interact with one another has long been a prevalent issue encountered by the React library. Traditionally, to utilize the state of variables within subcomponents, a hierarchical passing mechanism was imperative – beginning from the root component and traversing down the application tree to reach the "children" components. Given the frequent necessity to manipulate numerous variables within the application, the process of passing variable states down the hierarchy eventually becomes cumbersome. To establish global access, the implementation of a utility library named Redux Toolkit is recommended, thereby facilitating the establishment of a universal "store" permitting accessibility from any component within the application through a function known as "useSelector".

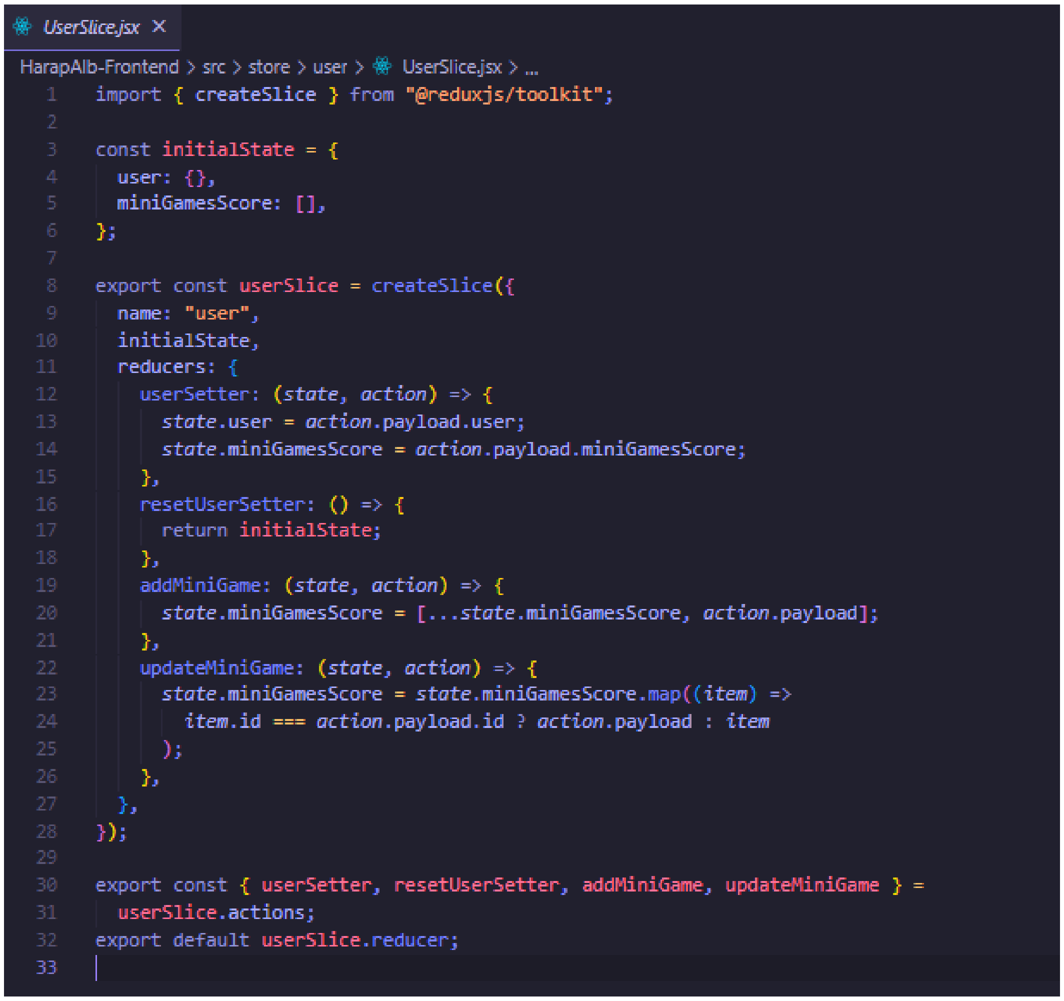
Within the file named store.js, a universally accessible store has been defined, consolidating all the "reducers" present in the application. Fundamentally, a reducer is a function that, when provided with a function and the preceding application state as inputs, produces a fresh state as output.



```
import { configureStore } from "@reduxjs/toolkit";
import userReducer from "./user/UserSlice";
import leaderboardReducer from "./leaderboard/LeaderboardSlice";
import { combineReducers } from "redux";
import storage from "redux-persist/lib/storage";
import {
  persistStore,
  persistReducer,
  FLUSH,
  REHYDRATE,
  PAUSE,
  PERSIST,
  PURGE,
  REGISTER,
} from "redux-persist";

const persistConfig = {
  key: "root",
  storage,
};

const reducers = combineReducers({
  user: userReducer,
  leaderboard: leaderboardReducer,
});

const persistedReducer = persistReducer(persistConfig, reducers);

export const store = configureStore({
  reducer: persistedReducer,
  middleware: (getDefaultMiddleware) =>
    getDefaultMiddleware({
      serializableCheck: {
        ignoredActions: [FLUSH, REHYDRATE, PAUSE, PERSIST, PURGE, REGISTER],
      },
    }),
});

export const persistor = persistStore(store);
```

Figure 35. Global store for maintaining state using Redux

For instance, a reducer denoted as userReducer – specified in the userSlice.jsx file – governs the state pertaining to the presently logged-in user. The initial state of this slice encompasses an empty user object alongside an empty array labeled miniGamesScore. It incorporates various reducers such as userSetter, responsible for configuring user and mini-games scores; resetUserSetter, designed to restore the state to its original values; addMiniGame, facilitating the addition of a new mini-game score; and updateMiniGame, enabling the modification of an existing mini-game score.


UserSlice.jsx ×
HarapAlb-Frontend > src > store > user > UserSlice.jsx > ...

```
import { createSlice } from "@reduxjs/toolkit";

const initialState = {
  user: {},
  miniGamesScore: [],
};

export const userSlice = createSlice({
  name: "user",
  initialState,
  reducers: {
    userSetter: (state, action) => {
      state.user = action.payload.user;
      state.miniGamesScore = action.payload.miniGamesScore;
    },
    resetUserSetter: () => {
      return initialState;
    },
    addMiniGame: (state, action) => {
      state.miniGamesScore = [...state.miniGamesScore, action.payload];
    },
    updateMiniGame: (state, action) => {
      state.miniGamesScore = state.miniGamesScore.map((item) =>
        item.id === action.payload.id ? action.payload : item
      );
    },
  },
});

export const { userSetter, resetUserSetter, addMiniGame, updateMiniGame } =
  userSlice.actions;
export default userSlice.reducer;
```

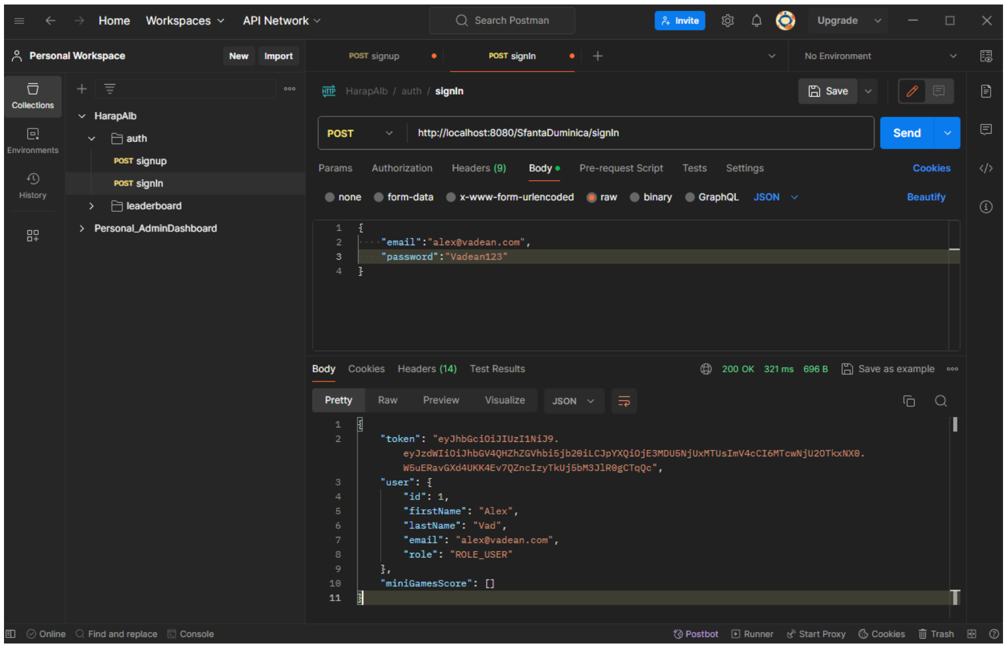
Figure 36. How the userSlice file looks like

The userSetter action is invoked during the login process when a user effectively authenticates their credentials. To activate an action stipulated within a reducer, the "useDispatch" function from React Redux is utilized. Subsequently, upon dispatching an action, access to the variable's state is facilitated through an additional imported function named "useSelector".

6. Testing the application

Software quality assurance is a process that underlies any high-quality software product. The testing phase is of utmost importance as it ensures that the implemented product complies with beneficiary requirements and functions properly. This process plays a crucial role in timely error detection before delivering the software product to the customer or production side. Testing of the application was conducted during each sprint when backlog functionalities were implemented. Additionally, testing was carried out upon completion of all functionalities to ensure the previous product's functionality and compliance with accumulated requirements.

API testing of the back-end was performed using a tool known as Postman. Postman allows testing of each created endpoint through various HTTP requests (GET, POST, PUT, PATCH) without the need for a visual interface or a form. Consequently, a series of tests were executed at the API level to verify their correct functionality upon deploying each route.



```
POST http://localhost:8080/SfantaDuminica/signIn
```

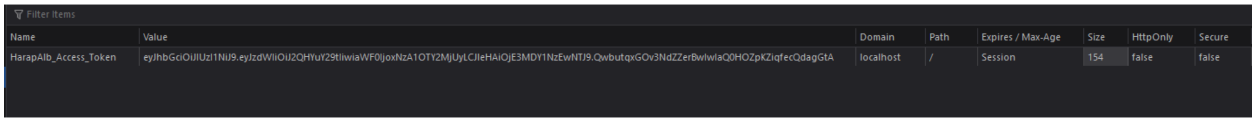
```
{
    "email":"alex@vadean.com",
    "password":"Vadean123"
}
```

```
{
    "token": "eyJhbGciOiJIUzI1NiJ9.
        eyJzdWIiOiJhbGV4QHZhZGVhbi5jb20iLCJpYXQiOjE3MDU5NjUxMTUsImV4cCI6MTcwNjU2OTkxNX0.
        W5uERavGXd4UKK4Ev7QZncIzyTkUj5bM3JlR0gCTqQc",
    "user": {
        "id": 1,
        "firstName": "Alex",
        "lastName": "Vad",
        "email": "alex@vadean.com",
        "role": "ROLE_USER"
    },
    "miniGamesScore": []
}
```

Figure 37. Testing the application’s login using Postman

HTTP requests in a collection were named based on the specific service intended to be called. An example of an authentication request test is shown in the figure above, utilizing the POST method with the path /SfantaDuminica/signIn. The request data was sent in JSON format rather than as form-data. Following the request, a 200 OK response was received from the server, confirming the success of the request. The response, in JSON format, contained the JWT and user information. The server responded in 320 milliseconds, indicating a swift request processing time.

The same request was repeated traditionally by logging in directly from the React client. To monitor the request progress, Chrome browser's Developer Tools were used. The request was successful, yielding the expected response within 304 milliseconds. Subsequently, the JWT was stored in the local storage using the popular, simple, and lightweight JavaScript API, js-cookie.

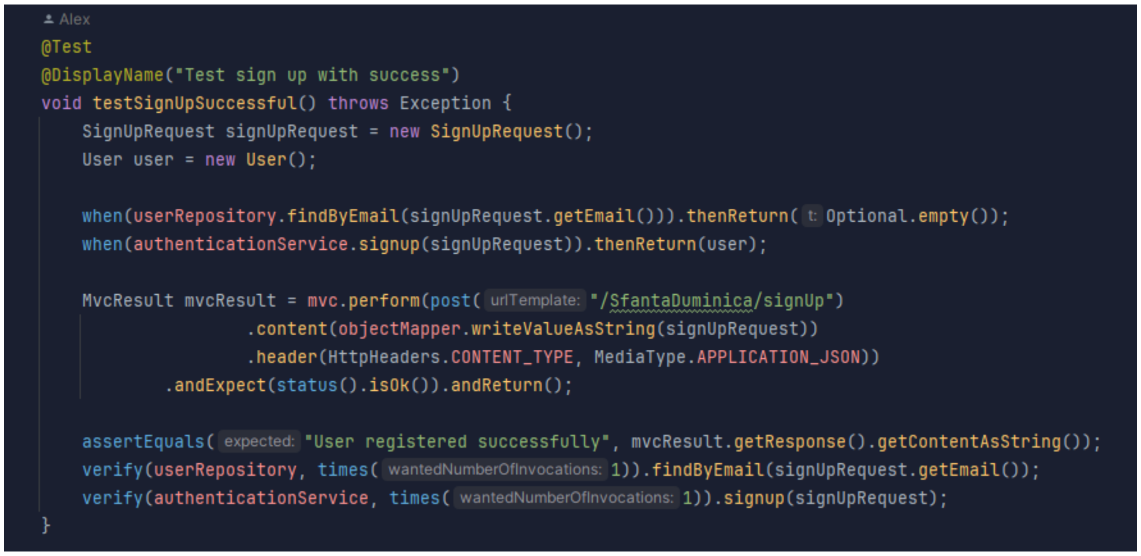


Name	Value	Domain	Path	Expires / Max-Age	Size	HttpOnly	Secure
HarapAlb_Access_Token	eyJhbGciOiJIUzI1NiJ9.eyJzdWIiOiJ2QHYuY29tIiwiaWF0IjoxNzA1OTY2MjUyLCJleHAiOjE3MDY1NzEwNTJ9.QwbutqxGOv3NdZZerBwlwlaQ0HOZpKZiqfecQdagGtA	localhost	/	Session	154	false	false

Figure 38. The JWT inside the browser's storage

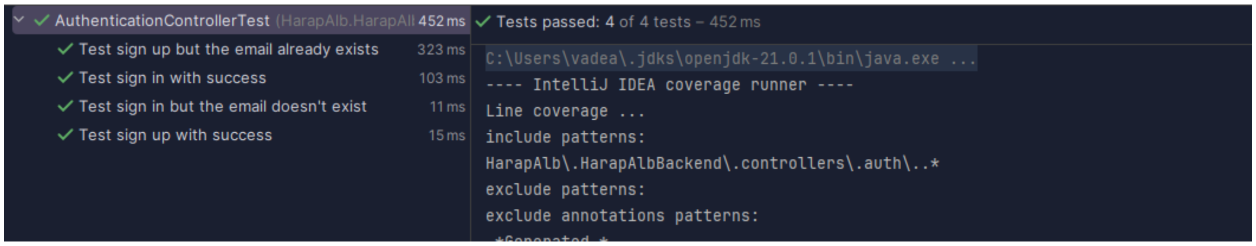
When writing unit tests for Java applications, developers commonly choose between two popular testing frameworks: JUnit and Mockito. JUnit facilitates test creation and execution, while Mockito simplifies the generation and utilization of mock objects. These mock objects are often used in services to replicate real objects' behavior within controlled environments. MockMvc, a component of the Spring Framework, offers a testing infrastructure enabling developers to write controller tests in a simulated environment without actual HTTP requests.

Unit tests concentrate on the isolation and validation of the accuracy of individual components or units, often utilizing mocked dependencies to ensure autonomy. Conversely, integration tests assess the interactions and cooperation among multiple components or systems, evaluating how they operate collectively to achieve higher-level functionality. When writing tests, it is imperative to encompass every possible scenario or circumstance.



```
Alex
@Test
@DisplayName("Test sign up with success")
void testSignUpSuccessful() throws Exception {
    SignUpRequest signUpRequest = new SignUpRequest();
    User user = new User();

    when(userRepository.findByEmail(signUpRequest.getEmail())).thenReturn( t: Optional.empty());
    when(authenticationService.signup(signUpRequest)).thenReturn(user);

    MvcResult mvcResult = mvc.perform(post( urlTemplate: "/SfantaDuminica/signUp")
                    .content(objectMapper.writeValueAsString(signUpRequest))
                    .header(HttpHeaders.CONTENT_TYPE, MediaType.APPLICATION_JSON))
            .andExpect(status().isOk()).andReturn();

    assertEquals( expected: "User registered successfully", mvcResult.getResponse().getContentAsString());
    verify(userRepository, times( wantedNumberOfInvocations: 1)).findByEmail(signUpRequest.getEmail());
    verify(authenticationService, times( wantedNumberOfInvocations: 1)).signup(signUpRequest);
}
```

Figure 39. Testing the application's registration with success using JUnit and MockMvc



```
✓ AuthenticationControllerTest (HarapAlb.HarapAll 452 ms
    ✓ Test sign up but the email already exists    323 ms
    ✓ Test sign in with success                    103 ms
    ✓ Test sign in but the email doesn't exist      11 ms
    ✓ Test sign up with success                     15 ms
```

```
✓ Tests passed: 4 of 4 tests – 452 ms
C:\Users\vadea\.jdks\openjdk-21.0.1\bin\java.exe ...
---- IntelliJ IDEA coverage runner ----
Line coverage ...
include patterns:
HarapAlb\.HarapAlbBackend\.controllers\.auth\..*
exclude patterns:
exclude annotations patterns:
```

Figure 40. Authentication Test Results for the Application

Testing frameworks implementation guarantees the reliability of Java applications by eliminating the requirement for real HTTP requests. Developers can ensure the correct and effective functioning of their applications in various scenarios by testing all possible cases with both unit and integration tests.

Conclusion

In this project, my main aim was to tackle the issue of the absence of educational games and offer a fun alternative for Romanian children who are fed up with aimlessly scrolling through content. The target audience I had in mind is quite young, so I made sure that the solution I came up with was extremely easy to grasp, approachable, and not complex at all in terms of gameplay and navigation within the game environment. Additionally, the choice to adapt "Povestea lui Harap Alb" is advantageous because it is in the public domain and is part of the Romanian literature curriculum.

After going through numerous rounds of revisions, playtests, and receiving a plethora of feedback, the current game, despite being just a demo at this point, turned out to be incredibly enjoyable, even for individuals in their early adulthood. Initially starting as a mere proof of concept, it gradually transformed into the current demo version that exists today. The app exudes a very inviting vibe, with everything being translated into Romanian. Players have the option to effortlessly skip the account creation process or watching the cutscenes and can directly immerse themselves in playing the game. The exploration aspect and the mini-games seem to be the crowd favorites.

By utilizing freely available online assets and limited resources, I was able to achieve a relatively simple theme and ambiance for the game that effectively creates the desired charming impact. Moreover, leveraging free online AI tools, the inclusion of cutscenes featuring AI-generated real characters has been warmly welcomed by the audience.

The exploration feature allows players to engage with the game world, providing them with a sense of control while staying true to the original narrative. Balancing the elements of fun, immersion, and interest was quite challenging given the extensive exposition in the game. Unfortunately, certain fantasy elements had to be removed to maintain a balance between gameplay and cinematic sequences. Despite working with constraints in terms of resources and time, I believe I succeeded in making the game quite captivating, as evidenced by the positive feedback, jokes, and anecdotes shared by close acquaintances and individuals who heard about the game.

The mini-games underwent rigorous testing with individuals possessing varying computer proficiency levels, both slow and fast users. I consider this feedback to be highly valuable as it aided me in refining the games to be both challenging and entertaining simultaneously, by adjusting aspects like logic and requirements.

We are currently in an era where everyone, particularly young children influenced by social media personalities, is constantly active across different platforms. Introducing a pixelated top-down game based on a familiar tale can serve as a delightful change of pace, whether the players are already familiar with the story or are yet to discover it in the future.

Bibliography

[1] Ion Ghinoiu, *Romanian FOLK ALMANAC,* Lulu Press, Inc., 19 January 2020 (accessed on 5 February 2023)

[2] Gloria Mark, PhD, *Speaking of Psychology: Why our attention spans are shrinking*, American Psychological Association, February 2023 (accessed on 1 May 2023)

[3] Iulian Negrila, *Folclorul copiilor – mijloc de educare a limbajului*, Western University "Vasile Goldis" from Arad, Faculty of Psychology and Educational Sciences, December 2007 (accessed on 22 March 2023)

[4] Phaser.js, *Making your first Phaser 3 game* taken over from phaser.io: https://phaser.io/tutorials/making-your-first-phaser-3-game/part1 (accessed on 30 August 2022)

[5] Pello Xabier Altadill and Richard Davey, *Phaser by example*, 19 April 2024 (accessed on 27 April 2024)

[6] Axios.js, taken over from axios-http.com: https://axios-http.com/docs/intro (accessed on 10 April 2023)

[7] Node.js, taken over from nodejs.org: https://nodejs.org/en/about (accessed on 9 April 2023)

[8] Alex Young, Bradley Meck, Mike Cantelon, Tim Oxley, Marc Harter, T.J. Holowaychuk, and Nathan Rajlich, *Node.js in Action Second Edition*, Manning, *17* September 2017 (accessed on 10 April 2023)

[9] React.js, taken over from react.dev: https://react.dev/learn (accessed on 11 April 2023)

[10] Cory Gackenheimer, *Introduction to React*, Apress, 8 September 2015 (accessed on 3 July 2023)

[11] Redux Toolkit, taken over from redux-toolkit.js.org: https://redux-toolkit.js.org/introduction/getting-started (accessed on 14 April 2023)

[12] React Router, taken over from reactrouter.com: https://reactrouter.com/en/main/start/concepts (accessed on 14 April 2023)

[13] Material UI, taken over from mui.com: https://mui.com/material-ui/getting-started/ (accessed on 14 April 2023)

[14] Spring Boot, taken over from spring.io: https://spring.io/guides/gs/rest-service (accessed on 14 April 2023)

[15] Craig Walls, *Spring Boot in action*, Simon and Schuster, December 2015 (accessed on 24 August 2023)

[16] PostgreSQL, taken over from postgresql.org: https://www.postgresql.org/about/ (accessed on 14 April 2023)

[17] Leon Shklar and Rich Rosen, *Web Application Architecture: Principles, Protocols and Practices*, Wiley; 1st edition, 27 April 2009 (accessed on 13 February 2024)

[18] JUnit, taken over from junit.org/junit5/: https://junit.org/junit5/docs/current/user-guide/ (accessed on 14 April 2023)

[19] Petar Tahchiev, Felipe Leme, Vincent Massol and Gary Gregory, *JUnit in Action, Second Edition*, Manning, 7 August 2010 (accessed on 4 December 2023)

[20] StackOverflow, *StackOverflow Developer Survey* taken over from: https://survey.stackoverflow.co/2023/ (accessed on 1 March 2024)

[21] Frans Van Galen, Vincent Jonker, and Monica Wijers, *Designing educational mini-games*, 5th Annual Conference of the International Society for Design and Development in Education, Cairns, Australia, January 2009 (accessed on 2 May 2024)

[22] Sayim Aktay, *The usability of images generated by artificial intelligence (AI) in education*, Mugla Sitki Kocman University, Turkey, 31 December 2022 (accessed on 5 May 2024)