UNIVERSITATEA “ALEXANDRU IOAN CUZA” DIN IAȘI

FACULTATEA DE INFORMATICĂ

LUCRARE DE LICENȚĂ

Pub Quiz Iași

propusă de

Duca Alexandru

Sesiunea: februarie, 2024

Coordonator științific

Olariu Florin

UNIVERSITATEA “ALEXANDRU IOAN CUZA” DIN IAȘI

FACULTATEA DE INFORMATICĂ

Pub Quiz Iași

Duca Alexandru

Sesiunea: februarie, 2024

Coordonator științific

Olariu Florin

Table of content

Abstract

The objective of our application is to digitize each aspect of the physical event Quiz Iași, from reservation to a new event and tracking game results based on season and game number to presenting and advertising the event to potential new participants.

The business plan that Quiz Iasi's organizers chose will be covered in detail in the first chapter. We'll learn in-depth information about the fundamental concepts of this competition, the procedure Quiz Iasi uses to create its question sets, the testing that takes place before they are used, and the composition of game rounds.

The Three-Tier Architecture, which separates the system into three layers—presentation, application, and data—will be examined in the second chapter. This architecture is crucial to the web application's design, ensuring maintainability, scalability, and modularity while clearly separating front end and back end components.

On the front end side, the application is developed using React with TypeScript, offering a robust and scalable framework for building user interfaces. For effective state management and handling asynchronous activities, Redux Toolkit and Redux Saga are used, resulting in a flawless user experience. Furthermore, i18n has been utilized for localization, offering support for Romanian and English. We utilized Material-UI and makeStyles for the application's styling, resulting in visually appealing and consistent user interfaces. Smooth navigation inside the application is made possible by React Router, which also makes sure that certain routes are protected. To enhance development workflow and speed up initialization, create-react-app has been replaced with ViteJS as the build tool. The collection of technologies that aided our development process will be discussed further in Chapter 3, Subchapter 3.1.

The application functionality is organized using a model-controller architecture on the back end using Node.js and Express. Maintainability of the code is facilitated and concern separation is guaranteed by this architecture. The database solution is MongoDB, which works with the Mongoose ORM to provide scalability and flexibility for effective data storing and retrieval. More details about the implementation details in Chapter 3, Subchapter 3.2.

As it will be described in the section dedicated to the application’s features, our solution includes two types of users: regular users and administrators. Users can view pages with useful information, such as ‘About’ and ‘Frequently Asked Questions,’ providing a way to promote the Quiz Iași event for those who haven’t participated in this type of event before. The ‘Game Results’ page is the central point of the application, allowing players to

immediately view the results of a game by rounds at the end of a session or any results from the past. Additionally, users can create a team or request entry into an existing team, which will be managed by the team leader. The team leader can view and accept/decline new joining requests, manage existing members (removal, assignment as leader), delete or leave the team. Furthermore, the leader is responsible for registering the team for the next event, choosing the table where the team would like to be seated, and handling payment. On the other hand, the administrator can enter new game results, including the number of the game and season, the teams, the names of the rounds, and the outcomes for each team for every round and also modify existing results, offering consistency for each game.

The final chapter of this documentation will provide a condensed overview of the conclusions drawn from the Pub Quiz Iași application's development process and outline the planned enhancements to further advance its functionality and user experience.

Keywords: Quiz Iași, digitization, three-tier architecture, technology stack, application features, React, Redux, Redux Toolkit, Redux Saga, i18n, Material-UI, React Router, Node.js , Express, MongoDB, Mongoose.

Introduction

In today's quickly changing landscape, the importance of digitization is more evident than ever. As enterprises, organizations, and communities cope with the difficulties of the digital era, the integration of digital technology has emerged as an essential accelerator for innovation, growth, and progress.

Our goal with Pub Quiz Iași is to solve the main flaw of the Quiz Iași event, namely, the absence of a dedicated web application and identity. Given its broad audience reach, we think this event has a lot of potential for growth through digitalization.

The proposed solution aims to start the event's digitization phase. First, we want to eliminate the event's reliance on its Facebook page for communicating information and updates to players. Instead, we advise promoting the web application through the event's page, where both existing clients and possible new players can obtain general information on the event's frequency, round formats, scoring and tiebreaker systems, awards, and more.

At the moment, the game results are only known at the end of the game, when the presenter publicly announces them. However, there is a lack of visual support while reporting the results, and to keep track of the data, participants frequently take down the final scores of all teams on paper. Furthermore, a client who did not attend an event is uninformed of the results, which are not made public due to the lack of a results generation mechanism.

By implementing the administrator module and functionality for adding/modifying game results, we aim to provide transparency and fairness to all participants. We seek to facilitate the organizer's process of creating rankings and offer participants the ability to view the results of previous games at any time, filtered by season and game.

Another identified problem is the reliance on a Google Sheets form for team reservations, which was usually attached to each event via a link. This form specifies the layout of the location where the game will take place, with tables numbered correspondingly. If a participant wants to make a reservation, they must enter their team name above an empty table. When the team later becomes a regular attendee of the event, they may request a recurring reservation of the table they typically occupy from the organizer. To solve this issue, we offer the team registration module, which replicates the previously mentioned process within the Pub Quiz Iași application. This ensures that all resources required for the event's online operations are located within the application.

We have been actively taking part in quizzes that Iași hosts since the fall of 2021. There are only three weekly activities happening at the moment. While every event has its own unique characteristics, Quiz Iași, the biggest and most established pub quiz event in Iași,

stands out for having an intriguing and captivating question structure. Based on our research, we found no other quiz events in Romania with their own websites; instead, every event mostly relies on their Facebook sites to promote themselves. We think the event would benefit greatly from having its own online application, creating a unique brand identity and building credibility and confidence with both existing and potential new participants.

Chapter 1

Business Overview

1.1 History of Pub Quiz

The Pub Quiz is a game of British origin. As the name suggests, it involves an assessment (quiz) in the form of questions answered by customers (grouped in teams) at pub/restaurant venues, where the predominant element should be the entertainment provided to those customers. At the end of the game, based on the results, teams are ranked. This game format gained strong popularity in the 1970s in its country of origin and later spread to the United States and other English-speaking countries. Starting in the 2000s, this format has also reached other countries, including Romania.

The typical Pub Quiz (inspired by the British model) heavily relies on the recall/retrieval of information from memory, without requiring additional mental effort (for example, What is the capital of Thailand? What is the number of championship titles held by Real Madrid FC? How many bones does a healthy adult human body have?). Although this type of game model is well-liked, it does not fully leverage the potential of collective intellectual capacity (as a team). Teams only discuss facts, dates, and information that they recall as part of a larger endeavor to retrieve fixed data (chronological data, for example). In its original form, the Pub Quiz mostly reflects the structure and overall setting in which Quiz Iași decides to conduct its games.

1.2 The Basic Principles of Quiz Iași

The activities of Quiz Iași are focused on the question and how Quiz Iași defines a question. The main criteria that Quiz Iași looks for in a question is interestingness. Although "interesting" is a subjective term, it can be defined more objectively as follows: the question will be current (mostly related to current issues), based on thoroughly verified information, culturally relevant (questions presented for the game will be connected to the linguistic and cultural horizon of the majority of players), have an appropriate level of difficulty (neither too difficult nor too easy), and not rely on specialized knowledge but rather on general culture (ideally, the informational content on which the questions are based will not exceed the high school curriculum).

However, the logical component is what distinguishes a question apart and makes it interesting. Depending on the informational content that the question is built upon, Quiz Iași believes that the game task (the question) inevitably has to have a logical component. Consequently, there should be a chance for the team to determine the correct response through debate even if none of the players know the answer right away (after analyzing the question's wording and the author's clues). The term "Eureka moment" is a conventional term utilized by Quiz Iași to denote the phenomenon of a player (or players) simultaneously determining the correct answer based on the clues provided in the question. These types of questions reward not only the individual knowledge of each player but also the team's capacity for well-structured, productive dialogue.

We believe that an argumentative team discussion centered around a topic with enough hints (such as keywords) to determine the right logical answer (without having to know the answer right away) provides a great deal of intellectual fulfillment for every team member. It is ideal for every player to participate in some way in the discussion of every subject. The entire team benefits intellectually from working together to discover the right answer.

All teams can't answer correctly to all questions (otherwise, it would involve a set of uninteresting questions due to the overly straightforward nature of the questions). Teams will therefore occasionally give incorrect answers. If the question composition model is applied correctly, players will still find satisfaction even if they provide an incorrect answer. They will recognize the logical element once they figure out the proper answer—a concept we may call a "delayed Eureka moment."

It is essential to note that creating questions that satisfy every one of the criteria mentioned earlier is an idealistic objective, but in reality, it is nearly impossible to create every question in this way (especially when it comes to the "logical" criterion). There are numerous causes. The primary reason is the limited amount of time available (possibly, any question can be improved endlessly). Additionally, some of the information underlying the questions might not lend itself well to logical clues (at least not in a way that would make them extremely confusing or overly simplistic), as incorrect and irrelevant indications would be abundant. Taking this into account, the fundamental idea behind Quiz Iași is that the majority of the questions in a set should display the previously mentioned logical component.

1.3 Question testing

A tester comes into play (almost invariably) when creating the questions and sets for Quiz Iași. A tester is a person who has particular experience with pub quizzes and goes through the set questions on their own without knowing the answers in advance. The tester

informs the set editor of their findings and then goes over all of the correct responses. Developing an appropriate set of questions requires a testing phase. A thoroughly tested set suggests questions of good quality, with just the right amount of difficulty.

1.4 Game principles

Pub Quiz is an intellectual game with an established timeframe. Teams of up to six players compete in the "Pub Quiz" game by answering questions of logic, insight, and general knowledge. A presenter reads the questions at the same time to each of the competing teams.

The questions are organized into rounds that are distinguished from one another by their particular themes or formats. A played set typically consists of six rounds, with eight questions in each round. Teams can debate the question and determine the right answer for around a minute (the amount of time may vary depending on complexity). Teams may observe the question wording on screens in the pub for the full minute. Teams turn in their sheets, which contain their written responses, for verification at the end of each round. Because answer verification occurs after each round rather than after each question, teams are encouraged to discuss any possible questions from the current round at any time.

Two points are awarded for a correct answer; one point may be awarded for an answer conceptually close to the correct answer but not quite precise. The game organizers have the final say on whether to give answers two points or one point. Teams are awarded no points for providing a wrong response, but they also aren't given a penalty.

During the game, a "Joker" is available for usage by each team. Regardless of the actual score accumulated, the Joker can be introduced in one of the game's rounds to double the team's score. A team may receive between 0 and 16 points in an 8-question round without a joker and between 0 and 32 points in a round with it. The prize for the right answer is worth two points. In rounds where its usage is desired, the "Joker" is used before the reading of the first question.

Every two successive rounds, there are usually 10-minute breaks in between.

1.5 Round Formats

Although there is a lot of variation in the Quiz Iași round typology, the most common round types may be distinguished.

1.5.1 Matrix round

The requirements for this kind of round are that every right answer must have a continuous three-letter particle (such as CAM, HAI, or LAC), either at the start, middle, or end of the word. When the correct response consists of several words, it is enough for the specified particle to be present in just one word. Teams will be notified of the assigned particle before the round starts. This round stands out as one of the most common uses of the wildcard, often called the "Joker," because of its accessibility and perhaps easier nature.

1.5.2 Music and/or Film Round

There is a weaker logical element in this round format. This round's key elements are sounds (song snippets, movie soundtracks, movie dialogue, etc.) or images (pictures from TV shows, movies, music videos, etc.). The retrieval of movie titles, song titles, or TV show names from memory is essential in this recall-based round. It is a necessity that the movies and music included in this round are well-known and/or up-to-date to keep players interested.

1.5.3 Connection Round

In terms of the design of the questions as well as the overall gameplay, it is the most creative round. This round's first seven questions include a traditional format and a fair amount of textual information. The eighth question, "What is the connection?" is formulated the same, regardless. Teams must determine the connection or relationship between the right answers to questions one through seven to solve the eighth question. Teams are specifically told not to look at the questions' textual substance, but to look for the commonality shared by the right answers. Teams try to find the one thing in common that connects all of the right answers.

Teams are aware of what makes the connection Round unique, and they frequently start looking for an association as soon as they receive even a small set of responses (usually two). Teams gather more and more responses as the round goes on, which facilitates the investigation of possible links. Teams are encouraged to capture basic information, including keywords, for questions that are not answered in the time frame given. This procedure aims to make it easier for teams to go back and review the question if they find the "connection," allowing them to determine the right answer or go back and correct any incorrect answers.

Generally speaking, connections can be divided into two categories: form-based and content-based.

- Form-based connections are dependent on the presence of a common pattern in the form of valid responses (e.g., all correct responses begin with the same letter, all correct responses contain only a particular vowel, all correct responses contain only vowels, etc.).
- On the other hand, content-based connections can cover a broad range of categories and are contingent upon the response's thematic content. Here are a few instances of content-based connections:
    - The seven right answers are Somnoroase, Memento Mori (an 18th-century Dutch artwork), Îngeri și demoni (a Dan Brown novel adaptation), O scrisoare pierdută (written by I.L. Caragiale), Noapte, and Doina. These terms are derived from the titles of poems written by Mihai Eminescu.
    - The seven right answers are the names of hotels in Iași: Traian (Roman emperor), Indiana Jones (a character played by H. Ford), and Unirea (historical event).

The round's charm and beauty stem from the round author's deliberate efforts to keep the connection unforced, out-of-date, and monotonous, as well as from not disclosing it too soon—when teams discover the right answers.

1.5.4 Themed Rounds

These rounds consist of questions with a common theme, like football, hunting, plants, animals, Formula 1, Darwin Awards, Nobel Prizes, and more. To keep teams interested and involved in the game, organizers strive to cover a wide range of themes during these rounds and maintain a diverse structure.

Chapter 2

Application architecture

Creating a solid and scalable application architecture is a key component of software development, providing a solid foundation for creating digital solutions. An intelligently planned architecture guarantees an application's robustness, maintainability, and flexibility to changing requirements in addition to controlling the flow of data and processes within it. We explore the complexities of the application architecture used in our project in this chapter, emphasizing the Three-Tier Architecture paradigm.

The Three-Tier Architecture is a software architecture pattern that divides an application into three logically distinct layers, each responsible for managing specific areas of the application's functionality, in order to improve modularity, scalability, and maintainability. Because of the separation of responsibilities, developers can modify one layer without affecting the others, making updates and maintenance simpler. It also makes it possible to work on projects in parallel and encourages code reuse throughout the program.

Web development, enterprise applications, and other software systems where scalability, maintainability, and flexibility are essential are among the many fields where this architecture pattern is applied. It offers an easy-to-follow framework for arranging application code and encourages best practices such as concern separation, which results in software solutions that are more reliable and manageable.

According to this architectural model, an application should be divided into three separate layers: the presentation tier, or user interface; the application tier, where data is processed; and the data tier, where the data associated with the application is stored and managed.

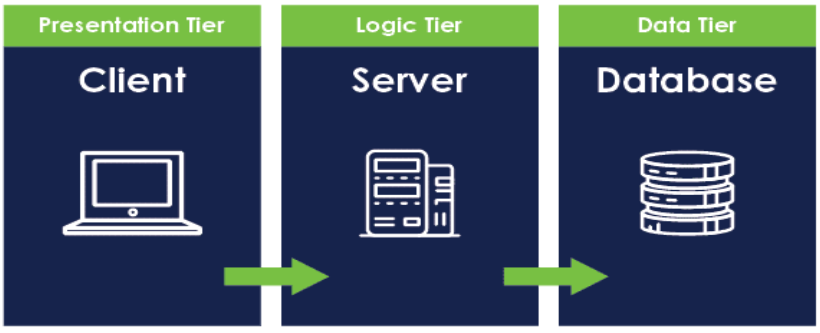

Figure 2.1. Three-tier architecture

The presentation layer, also known as the user interface (UI) layer, is our application's face and serves as the conduit for user interaction with the system. It represents the software's visual and interactive aspects, providing a well-designed interface for users to interact with the underlying technology. It is the application's face, the component that users view and interact with as they navigate through various features and functionalities.

We have used React with TypeScript, a powerful set of tools for creating interactive user interfaces, to construct this layer of our application. Redux is used to manage application state in addition to React, giving our application's data a consistent state container. Redux Saga makes it easier to manage side effects like asynchronous data fetching and intricate state transitions, while Redux Toolkit simplifies Redux setup and improves development efficiency. We use client-side routing with React Router to allow users to navigate between views in our application. When combined, these technologies enable the Presentation Layer to provide a user experience that is dynamic, rich, and responsive. In the upcoming chapter, each technology that makes up this stack will be covered in detail.

The logic layer sits in between the Presentation and Data layers, containing the business logic of the program and acting as a bridge between the user interface and data storage to process data. This layer of the architecture is managed by Node.js and Express.js, which together offer a stable and effective runtime environment for server-side JavaScript development. We can quickly construct routes, manage HTTP requests, and create middleware to process incoming requests by utilizing Express.js. The MongoDB database is accessed with Mongoose ODM (Object Data Modeling), which provides a high-level interface for data modeling and querying. By defining schemas, carrying out CRUD operations, and enforcing data validation rules, Mongoose makes interacting with MongoDB easier and guarantees the consistency and integrity of data.

The data layer is in charge of managing, retrieving, and storing data. The data storage option is MongoDB, a NoSQL database system that is adaptable and scalable. MongoDB's document-oriented model makes it appropriate for a variety of use cases by enabling the storage of organized, semi-structured, or unstructured data. Replication, sharding, and indexing are just a few of the features that may be used to assure high availability and performance while utilizing MongoDB in the data layer. We can easily combine the data layer with the logic layer by using MongoDB in conjunction with Mongoose ODM. This allows data to be exchanged between the application and the underlying database while upholding consistency and dependability.

Chapter 3

Implementation details

Our goal in this chapter is to provide a thorough rundown of the application's features as well as an in-depth look at each of its individual parts.

The front end portion will be covered in detail in the first subsection, along with the technologies used, user interfaces, device compatibility, and the themes of the application.

The back end section will then be shown in the second subsection, which will also highlight the technologies that were used in its creation, how closely it integrated with the front end and the database architecture that was used in conjunction with this component.

Lastly, we will outline the principal features that make up the application, arranged according to their respective user roles, and describe how they were built and used inside the application framework.

3.1 Front-end

A website's front end is an important aspect. When someone visits a website, it is the first thing they encounter and interact with. Additionally, it serves as the primary means of interaction between users and the website. As a result, a website's front end is essential for making a good first impression, establishing credibility, and guaranteeing user satisfaction.
A good front-end design incorporates performance, accessibility, and utility in addition to aesthetics. It ought to have an intuitive user interface that loads quickly, adapts to various devices and browsers, and makes it easy to navigate.

Users' initial perceptions of a web application are shaped by its front end, which also makes it easier for them to interact with the platform. In addition to providing a pleasing appearance, a well-designed front end guarantees clear navigation, which makes it simple for users to explore multiple features including the team module where anyone can create their team or join an existing one, observe previous game results, and know how the organizer develops a set of questions for a new weekly game. The front end, which serves as the quiz event's presentation page, sets the tone for participation by capturing the thrill and intensity of the occasion and offering essential features for both organizers and participants.

Considering the previously mentioned factors, we have placed a strong emphasis on delivering a positive user experience. The application's visual element was intended to be achieved through accessibility on all devices and localization in both English and Romanian. Additionally, we wanted to offer a user-friendly interface that was straightforward, efficient, and predictable by utilizing reusable components. In the case of forms, errors were handled both on the back end and front end to ensure that users have a user-friendly experience that reduces confusion and enhances their overall experience. Request responses will show the appropriate message based on the situation (for instance, the administrator will be alerted if a game they are trying to add already exists in the database, while users will be notified if an account with that username and email already exists).

We will dive into more detail about the technologies used and the implementation details in the upcoming subchapters, which have helped to create a positive user experience.

3.1.1 Utilized technologies

In our project, we chose Vite.js to initialize our React application rather than the more usual Create React App (CRA). It is a quick and lightweight front end build tool that uses modern web development principles to improve development efficiency. By initializing our React application with Vite.js, we gained several benefits that considerably speed up the development process. Vite.js provides extremely fast server startup speeds, allowing us to iterate and test our code with minimum waiting. The seamless integration of Hot Module Replacement (HMR) means that changes are immediately reflected in the browser, enabling a fluid and efficient work environment. Furthermore, Vite.js excels at creating optimal production builds, which result in reduced bundle sizes and faster loading times for end users. Vite.js emphasizes efficiency and performance through the use of modern packaging techniques and a plugin-based architecture.

The front end component of this project was built using React and TypeScript as the major technological stack. This conclusion was reached after thoroughly analyzing several issues, including scalability, maintainability, and developer experience.

React, developed by Facebook, has emerged as a popular JavaScript library for creating user interfaces due to its component-based architecture and declarative approach to UI development. Its appeal is due to numerous fundamental characteristics. React allows developers to develop reusable UI components, which encourages modular and maintainable programming. Furthermore, React employs a virtual DOM, which efficiently updates only the parts of the actual DOM that have changed, resulting in faster speed. React uses unidirectional data flow, which simplifies state management and reduces the risk of issues.

Although JavaScript is flexible and simple to use, it does not have static typing, which can cause difficulties when developing and maintaining large-scale systems. This problem is addressed by TypeScript, a superset of JavaScript, which provides extra language capabilities in addition to static typing. Type safety, better developer tools, and better code documentation are just a few advantages of TypeScript. Developers can construct interfaces and type annotations with TypeScript, which makes code that is easier to comprehend and maintains self-documenting.

React and TypeScript work well together to provide complementary advantages that improve both the software development process and the final product's quality. TypeScript's strong typing guarantees type safety and lowers the possibility of runtime errors in React components. Static typing also promotes the production of stronger, cleaner code, which leads to fewer errors and simpler debugging. By accelerating development workflows and offering helpful feedback, TypeScript's improved tooling and type inference capabilities enhance the

developer experience. Lastly, the application can be scaled more easily as it expands thanks to the combination of TypeScript's static typing and React's component-based architecture, which facilitates easier extensibility and maintenance.

Redux was used as the primary state management solution in conjunction with Redux Toolkit and Redux Saga to handle the state within the front end architecture of this project.

Redux is a JavaScript application predictable state container that offers a centralized store for managing an application's state. Because of its unidirectional data flow architecture, large-scale systems may be simpler to manage and understand state changes. It is composed of three concepts: the store, reducers, and actions. Actions are information payloads that transfer data from the application to the store, whereas the store contains the complete state tree of the application. Reducers define how actions affect the state of the application.

The official set of tools suggested for Redux development is called Redux Toolkit. It offers tools and abstractions to make basic Redux use cases easier, such as immutable update logic and action creation. Redux Toolkit makes it faster and simpler to create Redux applications by abstracting away a significant amount of the boilerplate code that comes with Redux.

Redux Saga is a middleware library for Redux that handles side effects such as complex state changes and asynchronous activities. It leverages ES6 generators to write asynchronous code in a synchronous-like manner, making it easier to understand and test asynchronous logic.

In modern front end applications, Redux, Redux Toolkit, and Redux Saga offer a reliable and effective solution for handling side effects and state management. We aim to guarantee the project's front end architecture's scalability, maintainability, and dependability by utilizing these technologies.

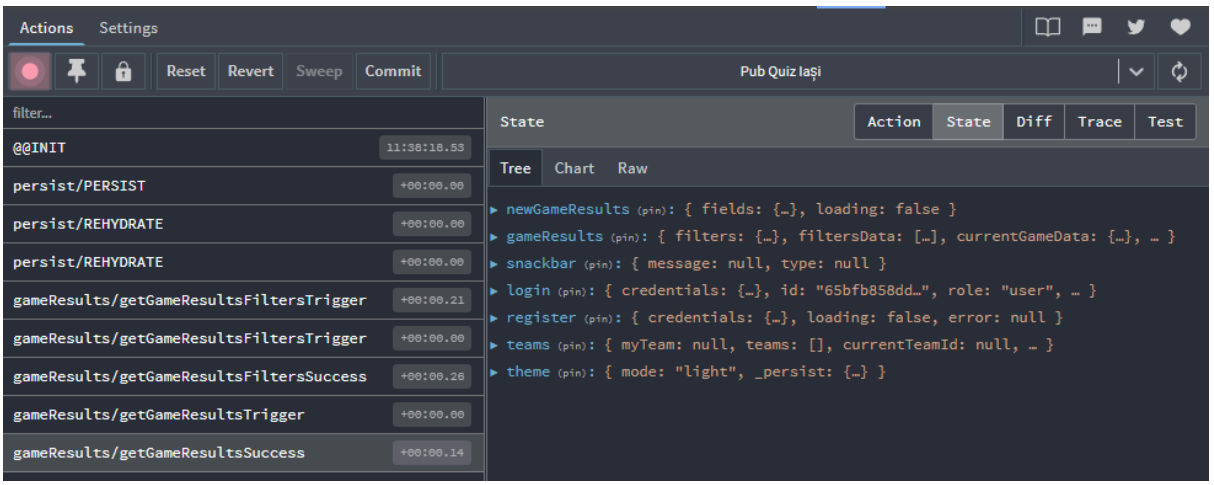

Figure 3.1. Redux store and the actions called when opening Game Results page

To enable the Redux store, we have installed the Redux DevTools Chrome extension. It provides lots of important information, including the actions called and their outcomes (success/failure), as well as the state tree that holds all the data stored in the Redux store.

As we can observe above, persist/PERSIST and persist/REHYDRATE are the actions that help us maintain the login data and theme. These values are preserved in local storage and remain even when the user closes the browser or refreshes the page.

The action getGameResultsFiltersTrigger sends a request to retrieve all the game filters (season and game number), and getGameResultsTrigger returns game information for the currently selected filters (when opening the page, the latest season and game are selected by default).

In the process of creating a single-page application (SPA) for this project, React Router was the main routing technology used. It is a popular library for handling routing in React applications. It gives developers the ability to specify how an application should traverse, allowing users to switch between views or pages without having to reload the entire page. React's component-based architecture is smoothly integrated with React Router, which offers a declarative approach to routing.

React Router offers numerous essential features that make routing in React applications easier. First of all, it makes declarative routing possible, enabling developers to specify routing logic by representing various routes in the application using components. Second, the application may have complex navigation hierarchies by nesting routes inside of each other thanks to React Router's support for nested routing.

As shown below, we also made sure to protect specific paths that are accessible exclusively by administrators by using React Router.

```
const paths = {
  root: '/',
  login: '/login',
  register: '/register',
  about: '/about',
  teamRegistration: '/team-registration',
  gameResults: '/game-results',
  newGame: '/new-game',
  teams: '/teams',
  faq: '/faq',
  contact: '/contact',
};

export const protectedPaths: string[] = [paths.newGame];

export default paths;
```

Figure 3.2. The paths and protected path objects

```
const routes = createBrowserRouter([
  {
    path: paths.root,
    element: <Layout />,
    children: [
      { path: paths.root, element: <Home /> },
      { path: paths.login, element: <Login /> },
      { path: paths.register, element: <Register /> },
      { path: paths.about, element: <About /> },
      { path: paths.teamRegistration, element: <TeamRegistration /> },
      { path: paths.gameResults, element: <GameResults /> },
      { path: paths.newGame, element: <NewGameResults /> },
      { path: paths.teams, element: <Teams /> },
      { path: paths.faq, element: <FAQ /> },
      { path: paths.contact, element: <Contact /> },
    ],
  },
]);
```

Figure 3.3. Router initialization; assigning each path to the corresponding component

```
const Layout = () => {
  const { classes } = useStyles();
  return (
    <div className={classes.body}>
      <Navbar />
      <main className={classes.content}>
        <ProtectedPath>
          <Outlet />
        </ProtectedPath>
      </main>
      <Footer />
    </div>
  );
};
export default Layout;
```

Figure 3.4. Wrapping <Outlet /> with the <ProtectedPath> component

```
const ProtectedPath = ({ children }: { children: React.ReactNode }) => {
  const path: string = window.location.pathname;
  const isAdmin = useAppSelector(selectIsAdmin);

  const navigate = useNavigate();

  useEffect(() => {
    if (protectedPaths.includes(path) && !isAdmin) {
      navigate(paths.root);
    }
  }, [path, navigate, isAdmin]);

  return <>{children}</>;
};

export default ProtectedPath;
```

Figure 3.5. ProtectedPath component that ensures only admins can access protected paths

During the development of the front end components of this project, Material-UI served as a fundamental tool for developing uniform and visually appealing user interfaces. Material-UI, a React component library based on Google's Material Design principles, provided a diverse set of pre-designed and customizable components, allowing for rapid user interface creation, such as buttons, forms, navigation bars, cards, and more, which ensured consistency and accessibility across the application. To customize the visual appearance of Material-UI components and apply specific styles to elements, the makeStyles hook was utilized. Material-UI's makeStyles feature enabled the development of custom CSS styles

within the React component using the JSS (JavaScript Style Sheets) syntax. MakeStyles also provides a responsive and scalable system for maintaining styles, making it easy to maintain and modify the user interface as needed.

3.1.2 Responsiveness

Given the expanding diversity of devices and screen sizes used to access the internet, ensuring responsiveness in a web application has become critical. Responsiveness is essential because it allows customers to have an excellent viewing experience on a variety of platforms, including desktops, laptops, tablets, and smartphones.

According to https://www.similarweb.com/platforms/, as of January 2024, 65.65% of the total web visits are currently mobile, compared to 32.67% coming from desktops. Taking this into account, we ensured flexibility to any sort of screen, as demonstrated in the examples below.

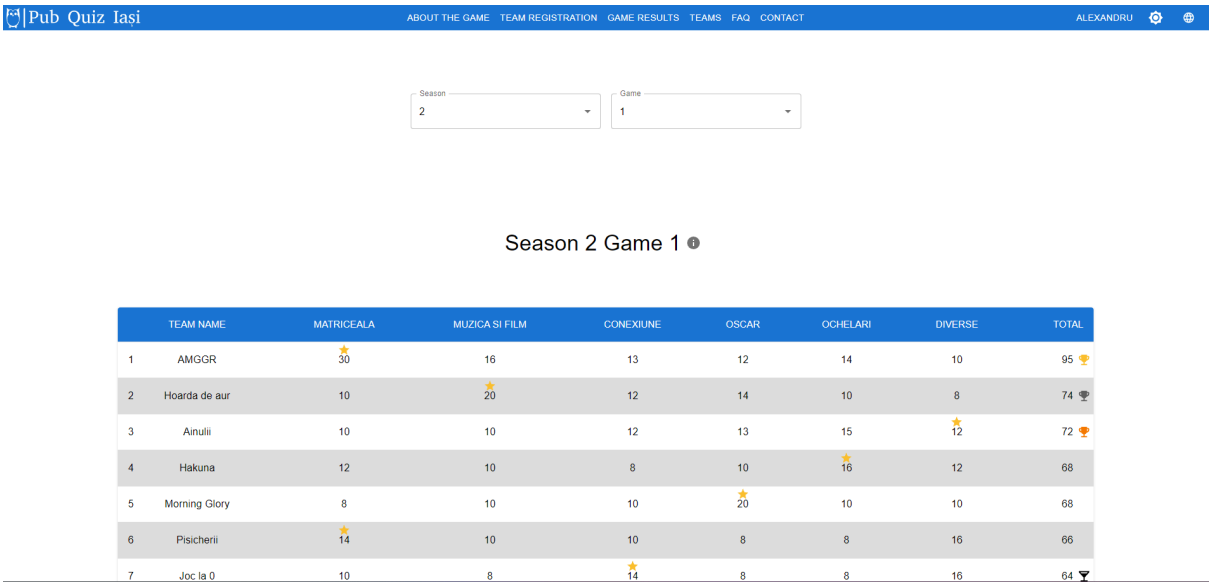

Figure 3.6. Game Results page visualized on desktop

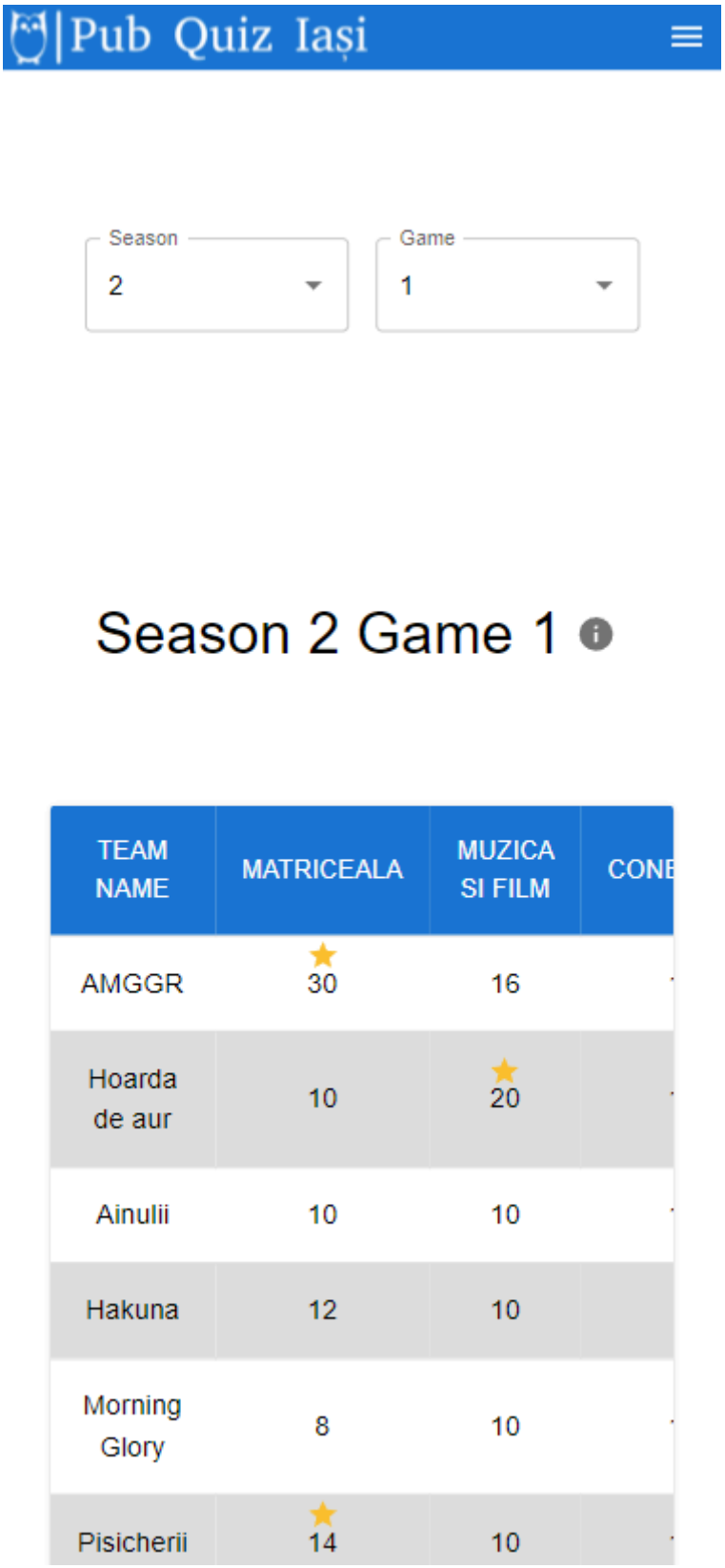

Figure 3.7. Game Results page visualized on iPhone 14 Pro Max

3.1.3 Internationalization and Localization

Internationalization (i18n) and localization (l10n) are two closely connected software development principles that allow applications to be adapted to diverse languages, regions, and cultures.

Internationalization is the process of creating and building software in such a way that it can be easily localized for other languages and regions. The purpose of i18n is to make an application adaptable to many languages and cultural conventions without requiring significant changes to the underlying software.

Localization is the process of customizing a software application or product for a given language, area, or culture. It includes translating content, changing graphics and other visual elements, and adjusting functionality to match the linguistic, cultural, and regulatory needs of the target region.

To give users a more customized and linguistically appropriate experience and target a bigger audience, the application has been localized in both English and Romanian.

Adding a new localization only requires adding translations for the new localization and expanding the list of preferences from which the user can choose. This is made possible by the use of the i18n library, which makes internationalization implementation easier. We utilized the LanguageDetector feature for creating the internationalization, which automatically determines the user's preferred language based on factors including browser settings, operating system locale, or other user-specific data. It eliminates the need for users to manually select their preferred language, resulting in a more smooth and intuitive experience. Since the preference is kept in the browser's local storage, the user's choice for localization is maintained, enhancing the experience even if they close the application and return later.

```
i18n
  .use(Backend)
  .use(LanguageDetector)
  .use(initReactI18next)
  .init({
    fallbackLng: Languages.ro,
    detection: {
      order: ['queryString', 'localStorage'],
    },
    ns: 'home',
  });
export default i18n;
```

Figure 3.8. i18n initialization

3.1.2 Application’s theme

Dark themes are essential in web applications because user choice, accessibility, and user experience optimization are becoming more and more important. The conventional color scheme of user interfaces is reversed with dark themes, sometimes referred to as dark mode or night mode, which display light-colored text and objects on a black background. This alternate visual layout improves the overall usability of online applications and provides users with several advantages.

One of the key advantages of dark themes is their potential to reduce eye strain and tiredness, particularly in low-light environments or after extended use. Dark themes make viewing more pleasant by reducing the contrast between the screen and the surrounding environment, which is especially important for users who are sensitive to bright light.

Dark themes are becoming more popular because of their elegant and sleek aesthetic appeal. Many users appreciate the minimalist and elegant style provided by dark themes, which is consistent with current design trends. By offering dark themes as an option, web applications allow users to tailor their experience to their specific interests and preferences.

The proposed solution allows users to customize the application's appearance, including both a dark and light theme. Material-UI includes the ThemeProvider component, which functions as a wrapper for the entire React application. By setting the ThemeProvider at the application's root level, developers may ensure that the defined theme is accessible to all components inside the application hierarchy.

Using Redux's extensive state management capabilities, the transition between themes was made smooth and intuitive, increasing user interaction and satisfaction. Furthermore, the integration of Redux granted the saving of users' theme preferences, ensuring that their preferred theme remained unchanged even if they close the application and return later.

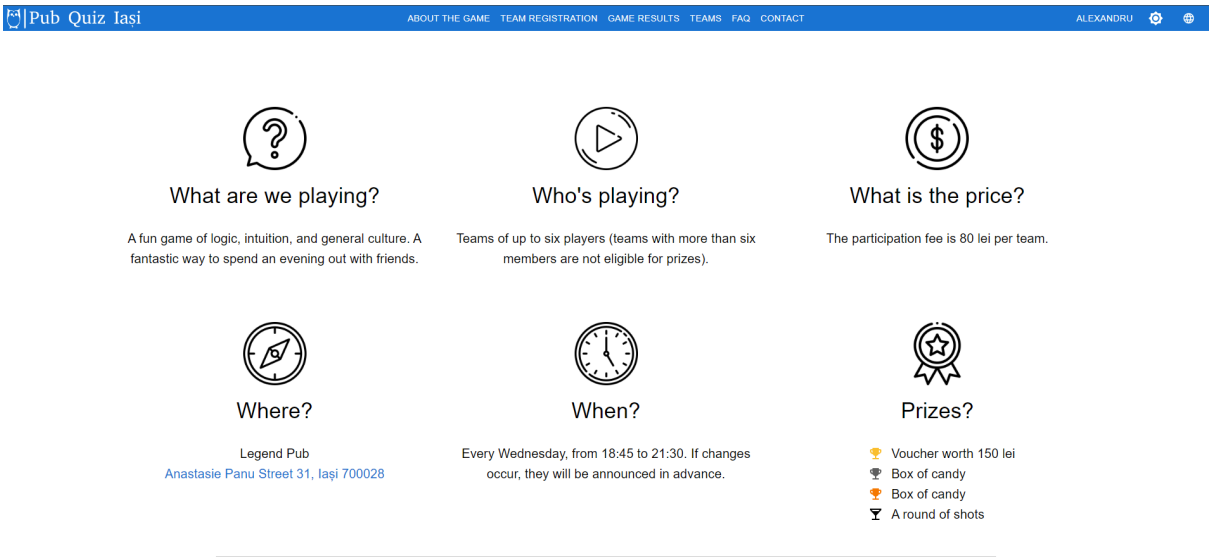

Figure 3.9. About page with the light theme applied

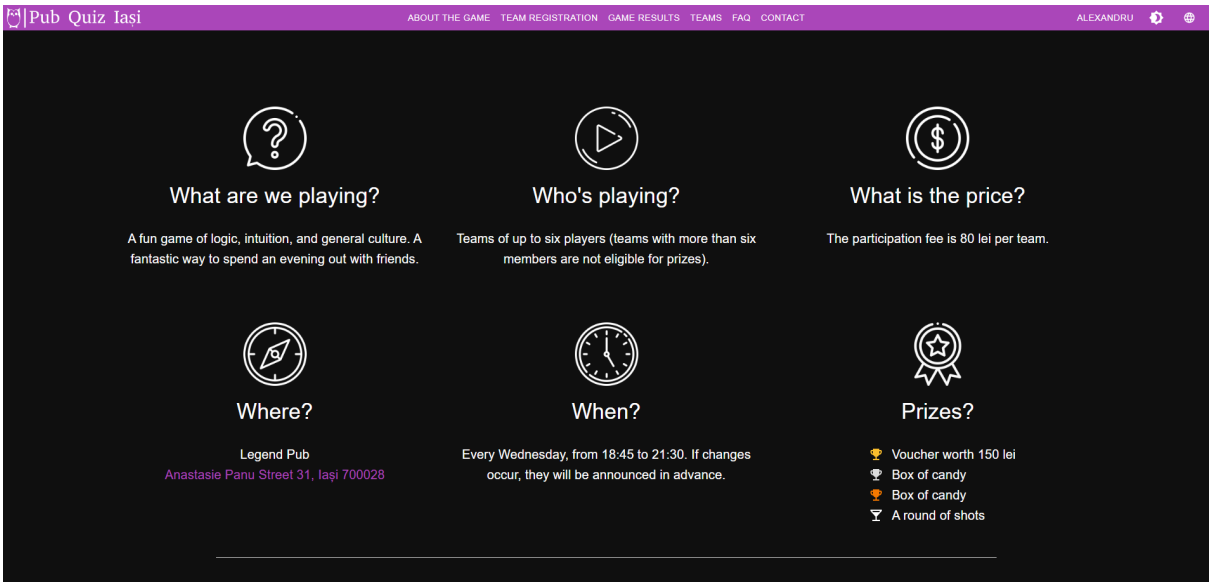

Figure 3.10. About page with the dark theme applied

3.2 Back-end

A web application's backend is its structural core, managing essential tasks including data processing, logic execution, and storing. Its capacity to handle and modify data, validate users, implement security controls, and carry out complex computations or business logic is what makes it so important.

Data necessary for the operation of the application, including user data, content, and application state, are stored and managed by the back end. It guarantees data security, consistency, and integrity, protecting private data from alteration or illegal access. It carries out the business logic of the program by handling queries, executing algorithms, and enforcing restrictions. It carries out the calculations, verifications, and procedures required for the application to run accurately and effectively.

A dynamic and interactive web application is built on the connection between the front end and back end, which allows for smooth data flow and communication between the two layers. The integrity and functionality of the application as a whole depend on this connection. The importance of this link is highlighted by several important factors:

- **API Endpoints**: the front end can request and modify data by interacting with the back end's exposed API endpoints. These endpoints specify the interface that the front end and back end use to communicate data while adhering to the RESTful protocol.
- **Data exchange**: to retrieve or alter data, the front end sends requests to the back end, which receives them, processes them, and returns the requested data or takes the required action. Within the application, real-time updates and interactions are made possible by this bidirectional data flow.
- **Authentication and authorization**: access rules are enforced, user credentials are verified, authentication tokens are generated, and user authentication and authorization are handled by the back end. In order to authenticate users and receive authorization tokens for accessing protected resources, the front end communicates with the backend.
- **Error handling and validation**: the back end handles errors, verifies incoming requests, and maintains consistency and integrity of data. It sends validation results and error warnings to the front end so that it can handle mistakes gracefully and give users feedback.

We will continue this subchapter by delving into the back end implementation details, such as the programming environment chosen and the technologies used. We will also talk about the database that was utilized and how it was integrated with the back end portion.

3.2.1 Utilized technologies

We use Node.js as our programming language to create the application's back end. Based on the V8 JavaScript engine in Chrome, Node.js is an open-source server-side JavaScript runtime environment. It makes it possible for developers to run JavaScript code off of a web browser, which facilitates the creation of scalable, fast network applications.

Considering we chose React for our front end development, we decided to opt for Node.js for our back end part. Using TypeScript across our entire stack also improves our development experience. Static typing and extensive tooling in TypeScript enhance efficiency and safety while completing the picture of a well-rounded full-stack JavaScript solution. We maintain a cohesive and strong tech stack that allows for quick development and smooth interaction between front end and back end components by utilizing Node.js in conjunction with React and TypeScript.

Because Node.js employs an event-driven, non-blocking I/O approach, it can effectively manage several concurrent connections. Node.js is also known for its exceptional performance. It is ideally suited for developing responsive and efficient applications because it is excellent at managing I/O-bound processes, such as network requests. Considering this, it's perfect for developing real-time applications that need to have low latency and high concurrency.

The rich ecosystem of libraries, frameworks, and tools for Node.js speeds up development and expands its capabilities. We've incorporated the Express.js framework into our back end design in addition to Node.js. Simple, adaptable, and feature-rich, Express.js is a Node.js framework for building simple web applications. We leverage Express.js's user-friendly routing system, middleware support, and HTTP utility methods to simplify the process of developing RESTful APIs and web servers. Express.js makes it easier to build middleware functions, allowing us to handle authentication, data validation, error handling, and other interconnected issues effectively. Its modular design and lightweight nature enable us to build scalable and maintainable back end systems effectively.

We have separated the issues in the back end design by dividing the codebase into three separate layers: controllers, models, and routes.

- **Controllers**: these are our application's central processing units, holding the business logic needed to handle incoming requests. In response to requests from clients, each controller is in charge of carrying out particular tasks, such as collecting data from models, performing business functions, and producing the appropriate responses.

Because of this division, code can be made modular and reusable, which improves code structure and maintainability.

- **Models**: the data structures in our application are represented by models. They contain validation rules, database interactions, and data access logic. To guarantee data integrity, consistency, and scalability, we divide data-related issues into models. Models also offer a distinct layer of abstraction between the application and the database that underlies it, making it possible to integrate them easily with a variety of data storage options.
- **Routes**: routes specify our application's endpoints and request-handling logic. Every route maps HTTP requests to the proper controller methods that correspond to a certain resource. We maintain a clear division of responsibilities while promoting code readability by keeping route definitions and controller logic apart. Routes also make API versioning, authentication, and request validation easier, which improves our backend's functionality and security.

We chose to manage user authentication and authorization using JWTs (JSON Web Tokens), a widely used protocol for sending confidential data as a JSON object between parties. Because they can securely encapsulate information and are stateless, they are frequently used for permission and authentication in web applications.

We create a JWT token with relevant user data, including the user ID, username, and any required responsibilities or permissions, when a user successfully signs in to our application. Afterwards, a secret key that is only known by the server is used to securely sign this token.

The JWT token is sent to the client by the server after successful authentication; this happens usually as part of the login response. After that, the client keeps this token for use in future requests to the server, storing it in a cookie.

```
export const login = async (req: Request, res: Response, next: NextFunction) => {
  const { username, password } = req.body;
  const user = await User.findOne({ username });
  if (user) {
    const passwordMatches = await bcrypt.compare(password, user.password);
    if (passwordMatches) {
      const expiresIn = 60 * 60;
      const jwtToken = jwt.sign({ id: user._id }, process.env.JWT_SECRET as string, { expiresIn });
      res.cookie('jwt', jwtToken, { httpOnly: true, maxAge: expiresIn * 1000 });
      res.status(200).send({ role: user.role, id: user._id });
    } else {
      res.status(401).send(ResponseCodes.WRONG_CREDENTIALS);
    }
  } else {
    res.status(401).send(ResponseCodes.WRONG_CREDENTIALS);
  }
};
```

Figure 3.11. JWT creation

The client adds the JWT token in the request headers for each subsequent request to protected routes or resources. The server then uses the secret key to validate the signature, confirming the integrity and authenticity of the token.

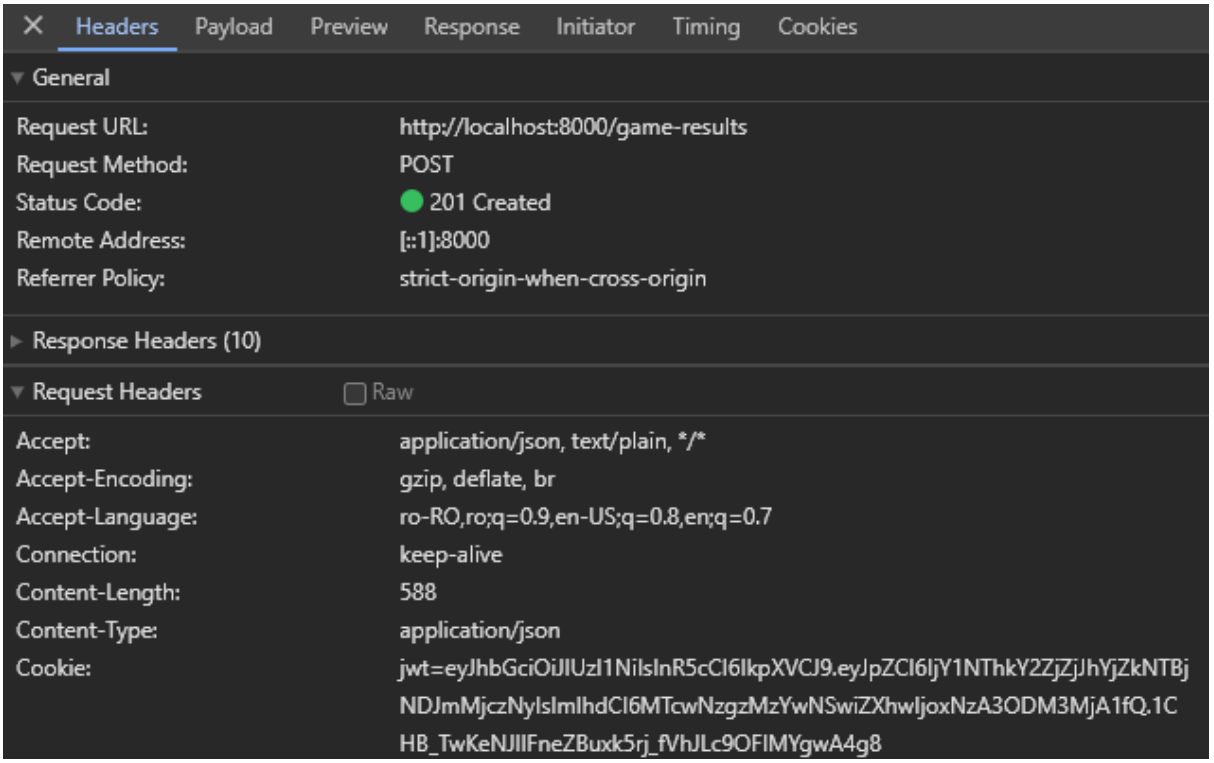

Figure 3.12. Add new game request contains JWT token

The user data encoded in the JWT token is extracted by the server after it has been verified. By using this data, the server can identify the user and assess their level of authorization before approving or rejecting access to the resources they have requested, in accordance with pre-established roles or permissions.

3.2.2 Database

MongoDB is a NoSQL database management system renowned for its adaptability, scalability, and usability. Its document-oriented data model allows for data storage in a flexible, schema-free manner, making it appropriate for scenarios in which data structures grow over time or complex relationships exist between data items. MongoDB's scalability capabilities, such as sharding and replication, enable high availability and fault tolerance, making it excellent for managing significant amounts of data and high-throughput workloads.

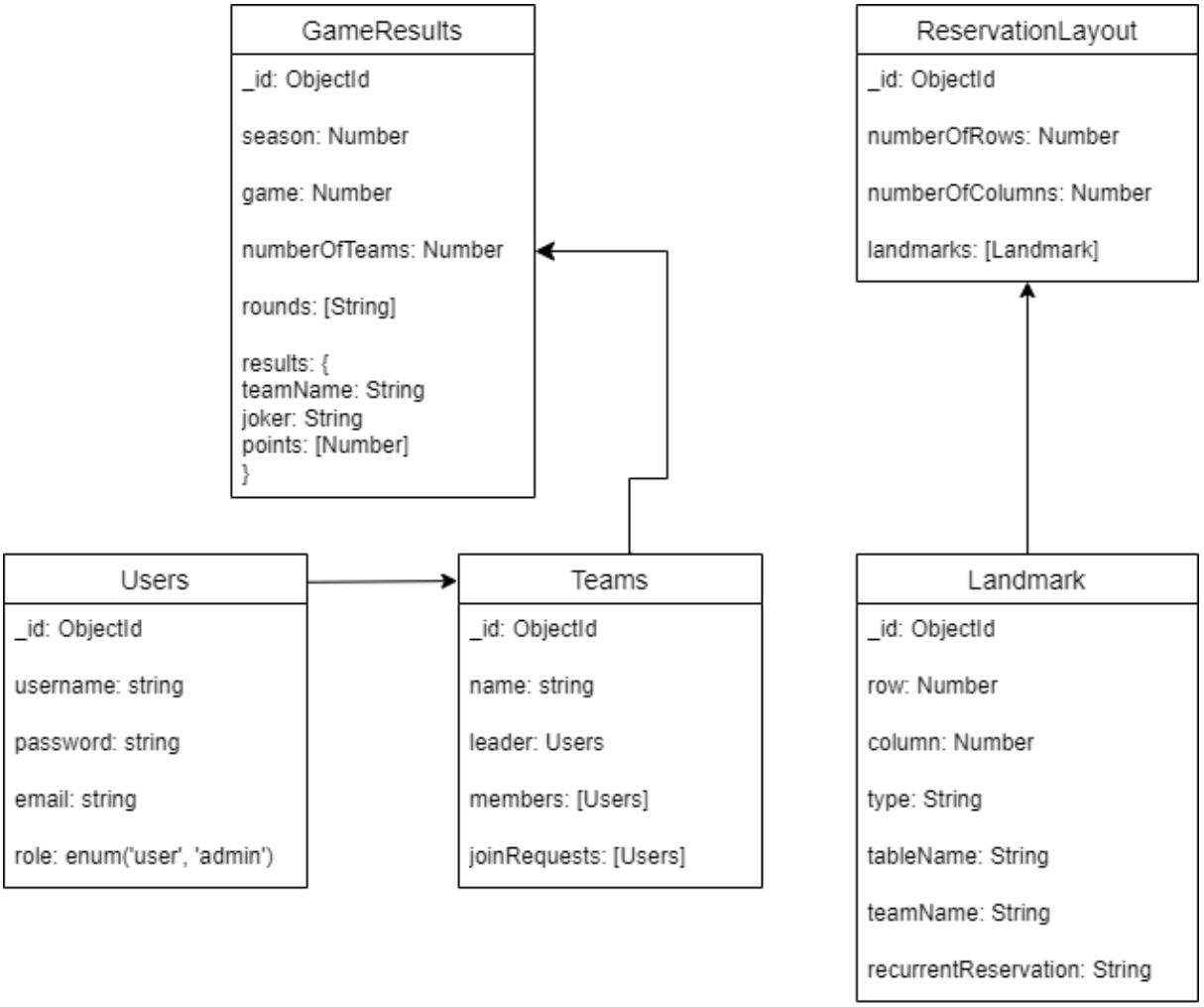

Figure 3.13. Database architecture

We defined our models using Mongoose, an object modeling tool for MongoDB. It provides a higher-level abstraction layer for designing schemas, models, and database interactions. Mongoose allows us to design schemas that represent the structure and validation

criteria of our data entities, assuring data consistency and integrity. These schemas can then be turned into models, which act as an interface between our application and the MongoDB database. Mongoose models offer a wide range of functionalities for querying, updating, and modifying data, making CRUD operations and data management easier.

```
const userSchema = new Schema({
  username: {
    type: String,
    required: true,
    minLength: 3,
  },
  password: {
    type: String,
    required: true,
    minLength: 8,
  },
  email: {
    type: String,
    required: true,
  },
  role: {
    type: Roles,
  },
});
```

Figure 3.14. Users schema initialization using mongoose

3.3 Application's features

This chapter examines the many feature sets that are accessible to both administrators and regular users within our application, each tailored to their specific roles and responsibilities. We'll also look at the general features that all user roles can access.

3.3.1 User features

One important feature in many web applications is the authentication process. Users are given a simple and intuitive login experience, which allows them to quickly authenticate themselves. A simple registration process is accessible for new users, guiding them through the steps to create an account and receive access to other features.

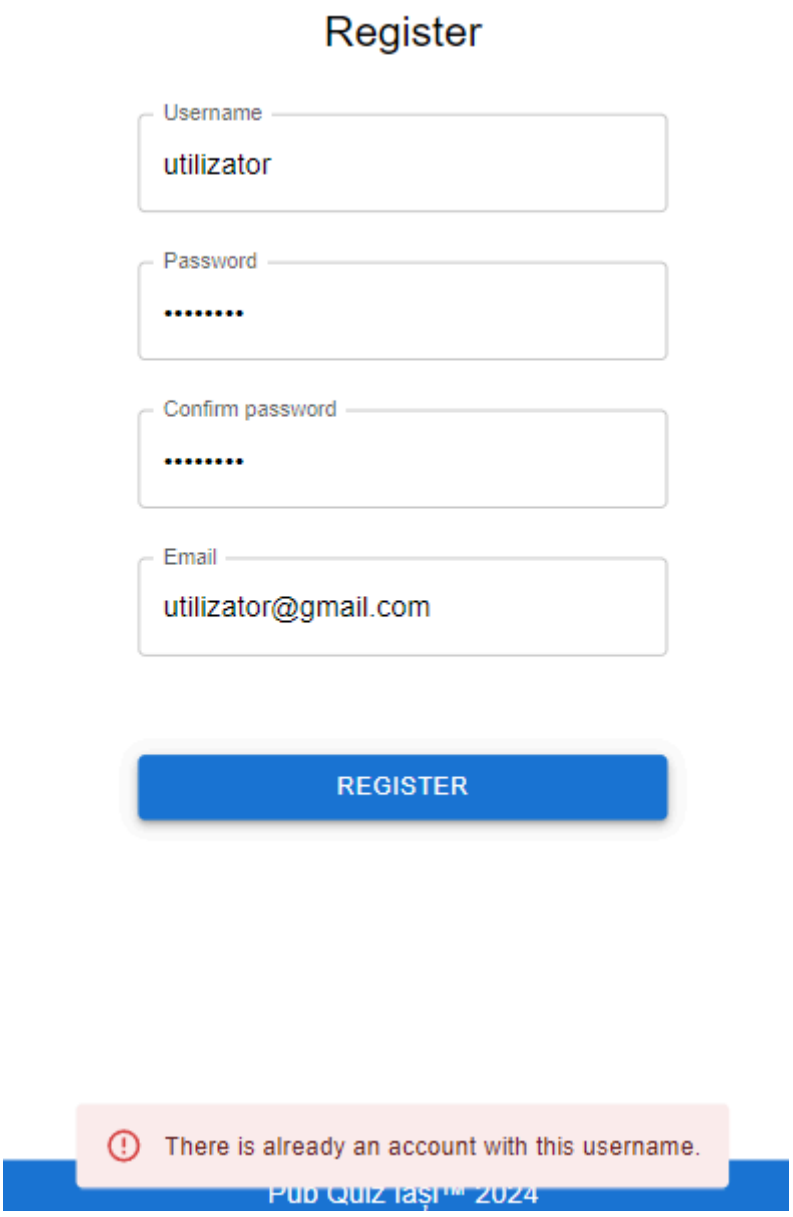

Figure 3.15. Register form

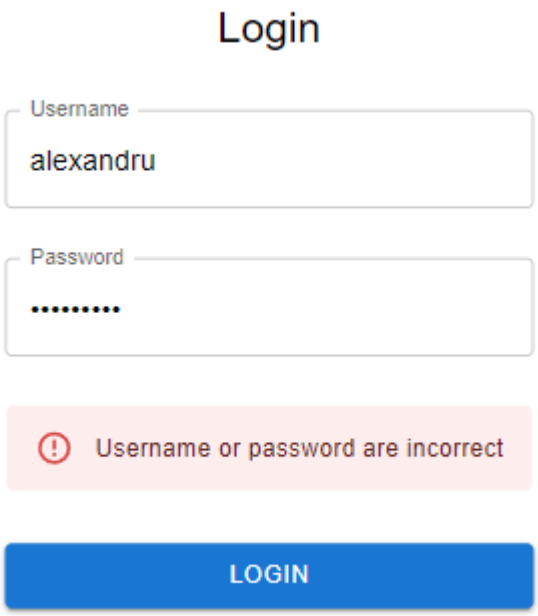

Figure 3.16. Login form

Validation occurs on both the front and back ends of our application. In the register form, we notice two categories of errors: form validation errors (passwords do not match, email format is incorrect) and back end errors, such as the ones listed above.

The game results page is a reusable component that is presented based on the user's role, so regular users will see the uneditable version of the results table. Users can select the season and game filters to view the game results based on these selections. The results table lists the round titles and results for each team, including the joker's round placement. The results are sorted, and the tiebreaker rule is implemented on the back end: if the results are equal across teams, we compare the outcomes from the last to the first round, until we determine a winner.

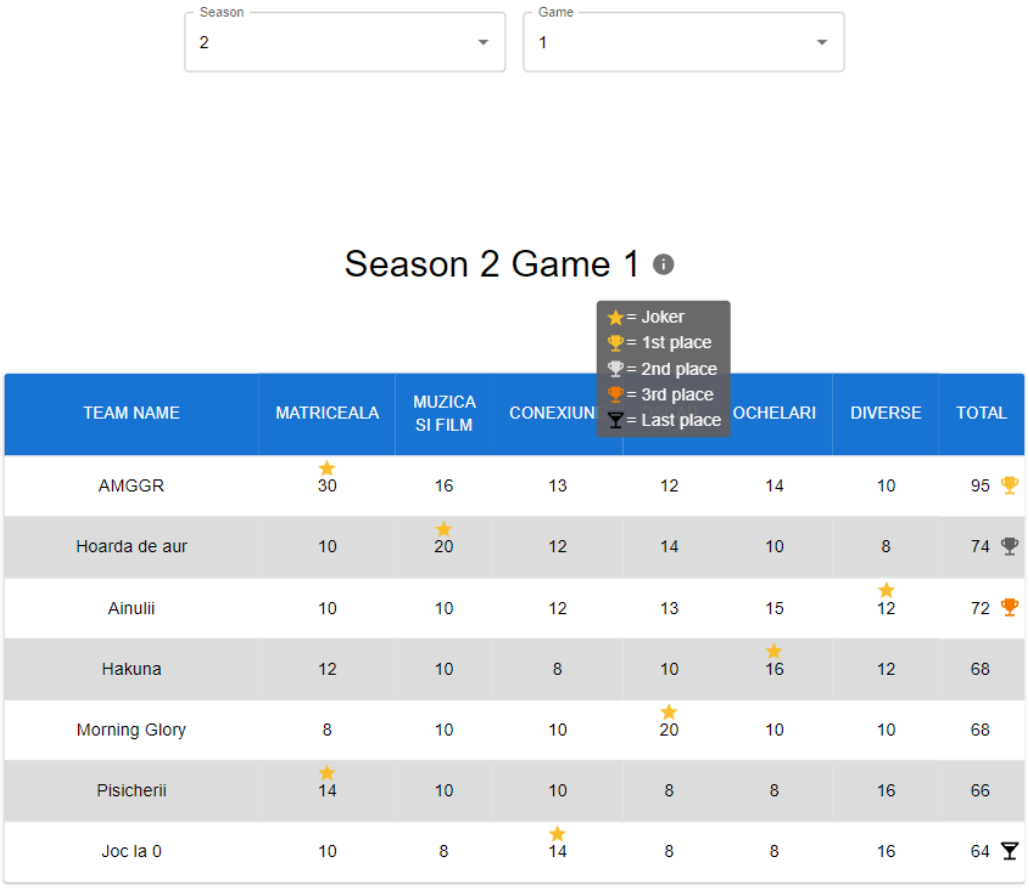

Figure 3.17. Game results page view for regular users

On the teams page, we define three scenarios: the user does not have a team, the user is a team member, and the user is the team leader.

In the first scenario, the following functionalities are available:

- join a team: the user can browse through each team, see the members and team statistics, and request to join the team. The request is sent to the team leader. If the user has previously sent a join request, he will be notified that it has been sent;
- create a team: if the user wants to form a team, a modal will appear, prompting them to enter a team name. The name is verified in the backend, and if the team already exists, the user is notified.

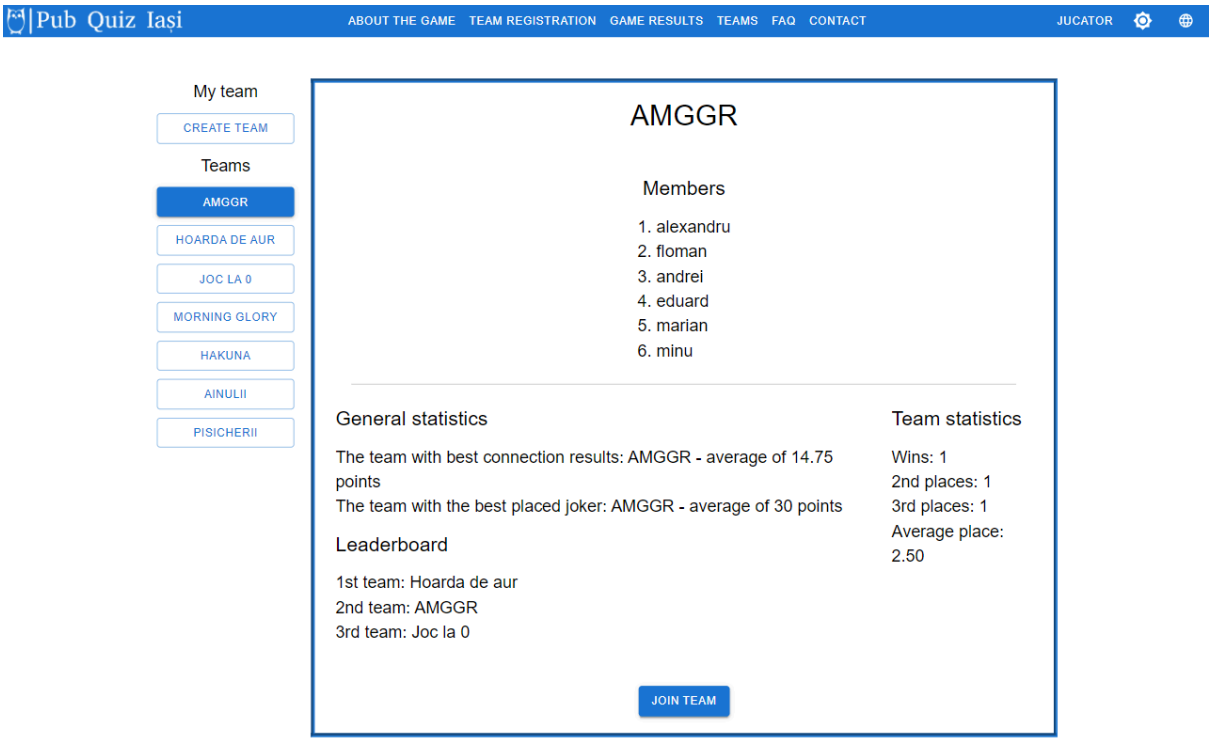

Figure 3.18. Teams page for a user that has no team assigned

A user who is already a member of a team has access to the list of other team members and their statistics, as well as leave the current team. If the user attempts to leave the team, a modal will appear, verifying their decision.

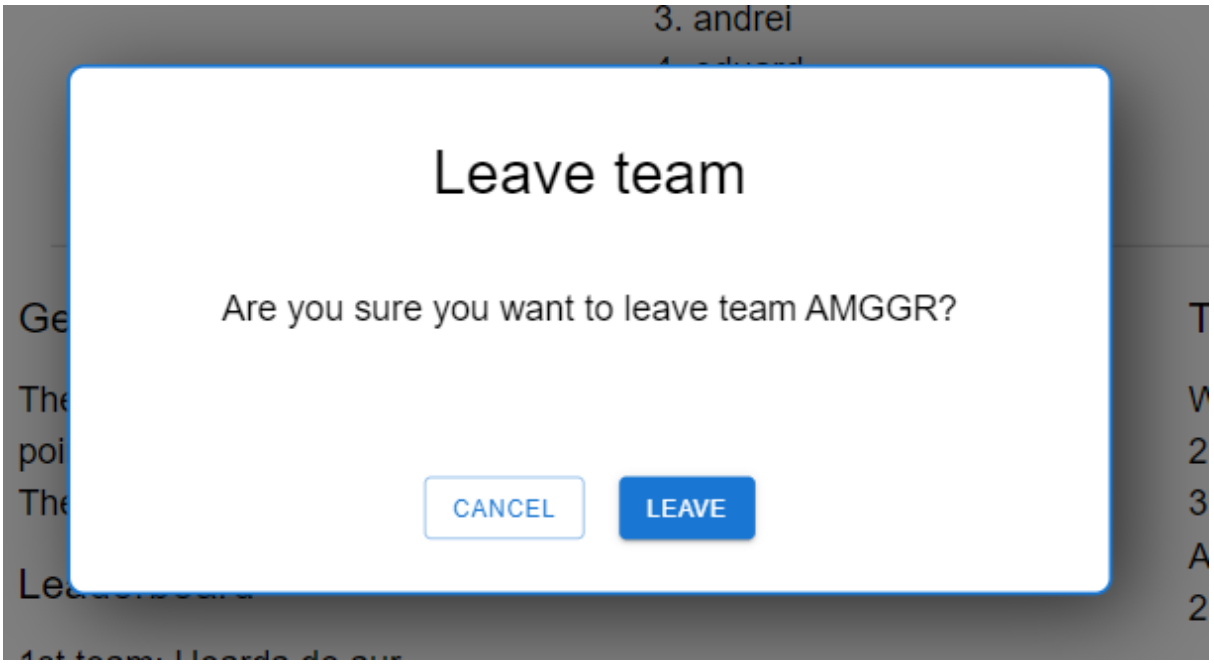

Figure 3.19. The user tries to leave the team

A team's effectiveness is greatly dependent on the presence of a good leader who can provide direction and guidance. In recognition of their essential role, leaders within our platform are offered access to a set of features designed to aid effective team management, as follows:

- handle join requests: the leader is the only person who can see if there are any join requests. He can accept or deny a request, and in each case, a modal appears to confirm their decision.
- manage members: the leader has the authority to either kick a member or delegate leadership to another member. Users that are kicked out of the team will be free to join another team or start a new one. If the user promotes another member to leader, he will lose all of the abilities that came with the position, even if he founded the team.
- delete the team: the leader can make this decision on his own, without the agreement of the other members. If the leader confirms his decision, all members, including himself, will be free to join another team or form a new one.
- register the team to a new event: the leader is the only member of a team who can access and complete the team registration form for an upcoming event.

On the team registration page, teams can declare their wish to attend a new event. As previously mentioned, the team leader is in charge of registering the group for the upcoming event by completing the form that is shown below the pub's layout. The leader must select a table from the list and send the request to make a reservation; once made, the reservation is confirmed instantly.

The event organizer's previous method for tracking bookings was a Google Sheets document with the pub's layout. The steps were similar: participants had to choose an empty table and input their team's name. The primary goal of this functionality is to replicate the existing functionality, eliminating the need to rely on another platform to manage reservations.

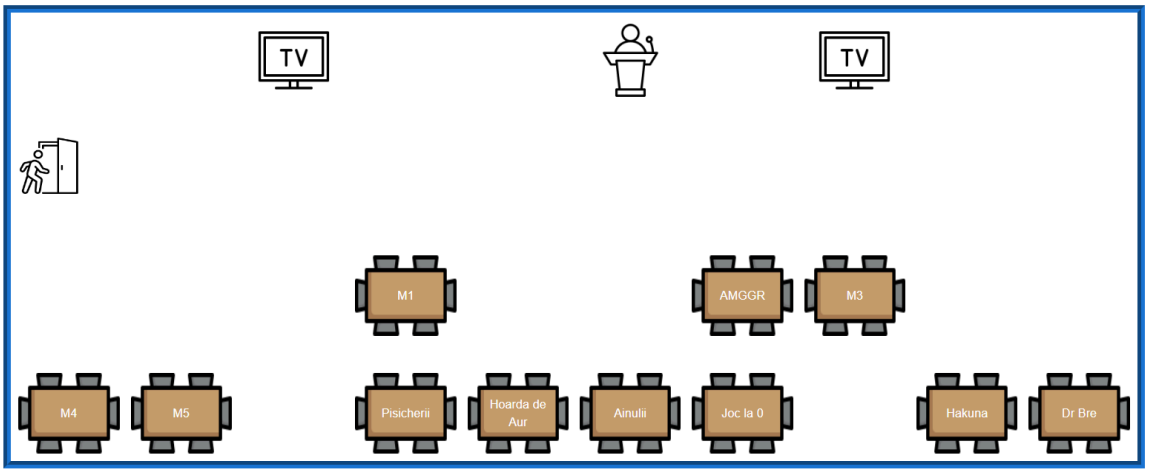

Figure 3.20. Team registration layout

In addition to the features described above, users can communicate directly with the organizer using the contact form. They may submit inquiries, feedback, or additional relevant data by simply entering their name, email address, and message. Following submission, the data is collected and delivered to the organizer's chosen email address.

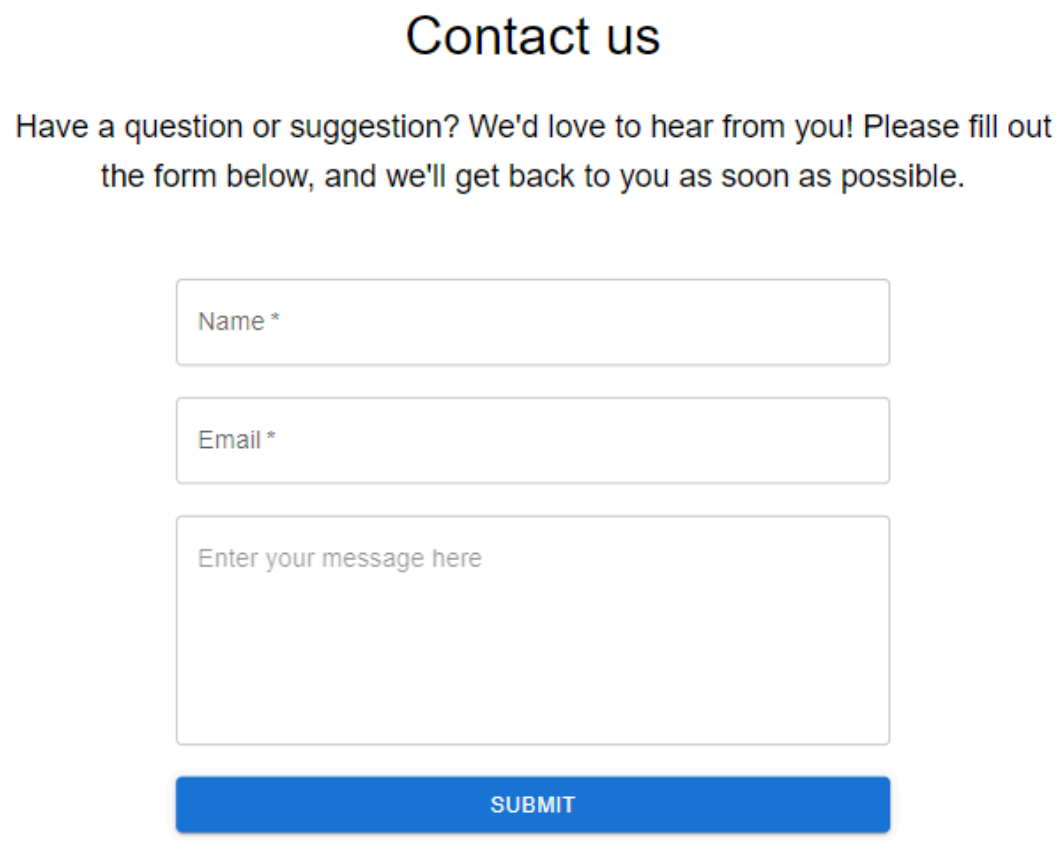

Figure 3.21. Contact form

3.3.2 Admin features

Administrators play an important role in guaranteeing the integrity and correctness of game outcomes, as they have the authority to add, edit, and delete game results as needed. As guardians of this essential component of our platform, administrators have the authority to maintain transparency, arbitrate disputes, and protect the fairness and accountability norms that underpin our game rules.

- add game: this feature is divided into two major components: game info and game results.
    - Game info includes general details regarding the game, such as the season and game number, team count, and round names.

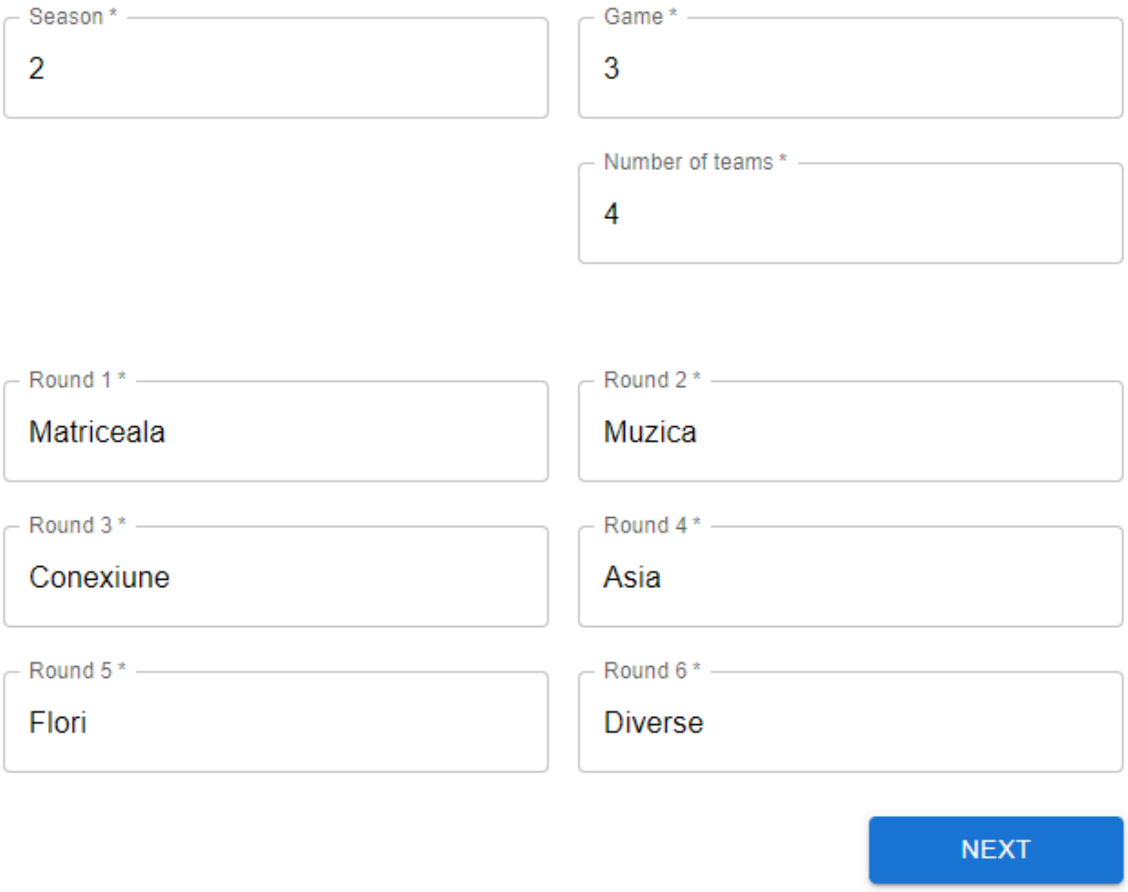

Figure 3.22. Game info component

    - After finishing the first stage of adding a game, the results table dynamically appears, reflecting the data entered in the previous phase. The admin can easily select team names from the team list that we receive from the back end, simplifying the process while ensuring accuracy. We validate the inputs to ensure the data is accurate. We check that no input field is empty and the round

score does not exceed the maximum of 32 points. The submit button is activated after the data is accepted as valid.

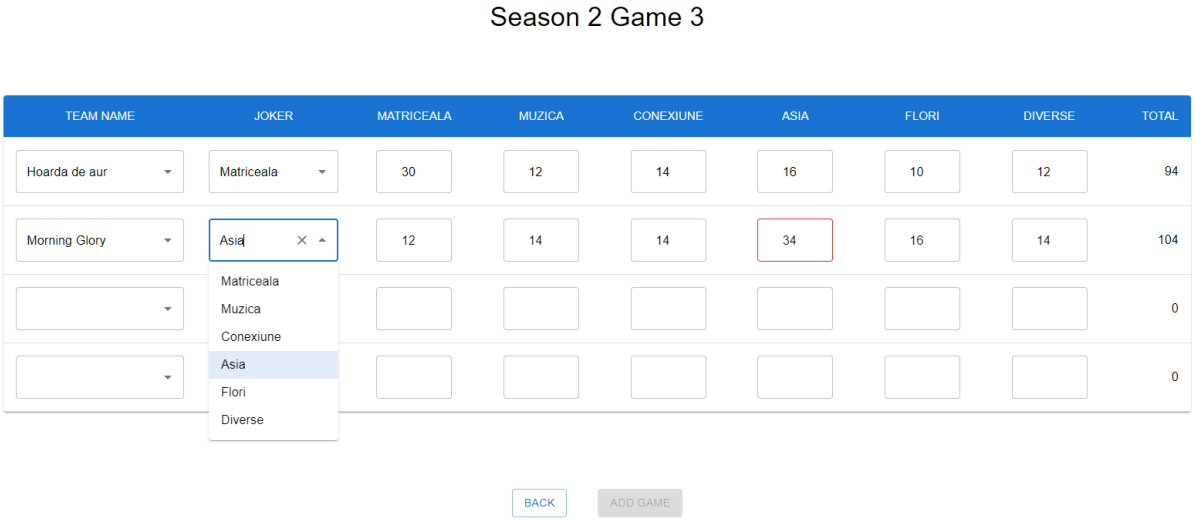

Figure 3.23. Game results component

- update game results: an administrator will see the editable version of the table when he visits the page with the game results. This page displays the components mentioned above, upholding the core idea of React: creating reusable components. The validations are conducted in the same manner as previously during the update process. However, when the administrator wants to change the season and game number, it is checked to see if the game already exists and returns the appropriate message.
- delete game results: when an administrator decides to remove a game, a modal window will show to confirm his choice.

We automatically retrieve the seasons and games that are available and provide the most recent results when an admin modifies or removes a game.

3.3.3 General features

The landing page, about the game, is meticulously created to provide visitors with detailed information about our event. The page is divided into two sections, offering a summary of the event's essential aspects.

The first part acts as a friendly introduction, giving newcomers the information they need to engage with our event by briefly presenting key information such as the location, participation fee, date and time, and prizes. The design for this section was previously shown in figure 3.9 of subchapter 3.1.2.

In the final section of this page, we present a more thorough examination of the unique format of the quiz rounds, hoping to provide participants with a full idea of what to expect during the event.

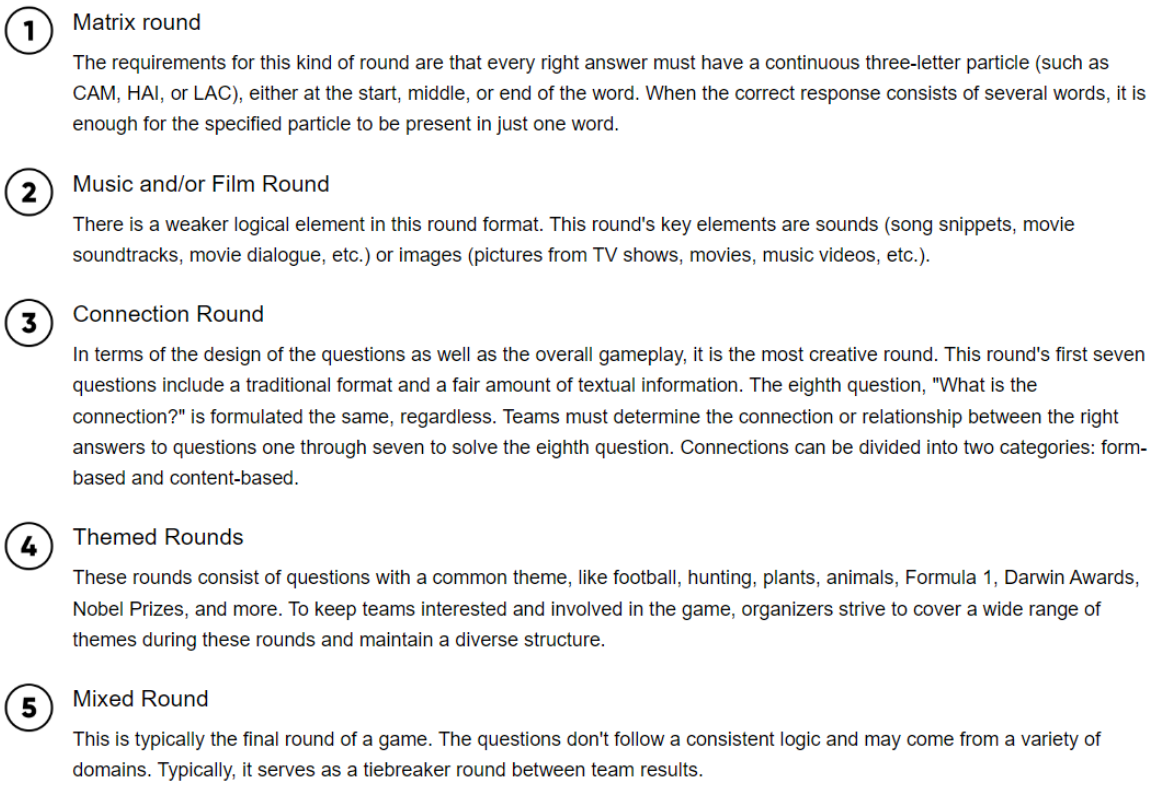

Figure 3.24. Round formats on about page

Considering this page is the first point of contact, it establishes credibility and trust, which motivates potential participants to learn more and eventually take an active role in its success.

The last section of our application answers frequently asked questions and addresses concerns new players may have about the event, making it a useful tool for onboarding. Our goal is to provide participants confidence and clarity by means of well-chosen questions and

insightful responses, allowing them to get familiar with the event. The FAQ section serves as a guide, ensuring that all participants are motivated and prepared to fully engage in the event.

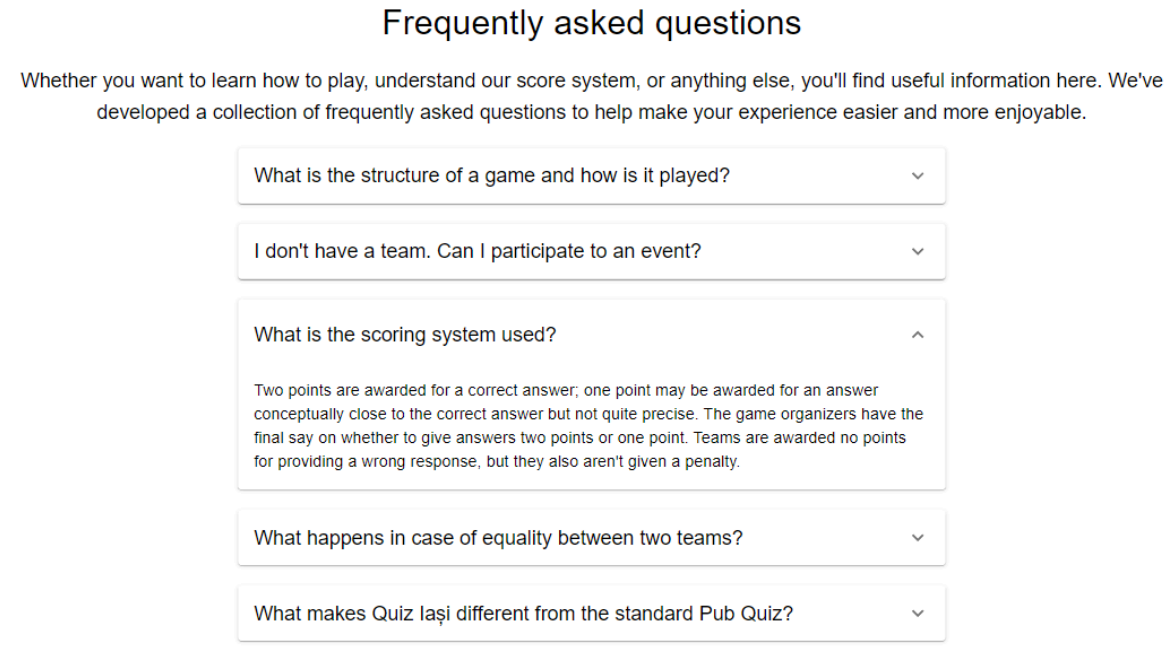

Figure 3.25. FAQ page

Conclusions

With a variety of features for users and organizers, the Pub Quiz Iași application effectively advances the digitization of the physical Quiz Iași event. We have successfully separated the three layers of the program by implementing the three-tier design, which enhances scalability, maintainability, and flexibility.

Through the implementation of the selected technology stack on the Pub Quiz Iași application's front end and back end, we have made notable progress in improving the system's functionality and user experience.

On the front end, React's integration with TypeScript, Redux, Redux Toolkit, Redux Saga, and React Router has allowed us to create a highly responsive, interactive, and intuitive user interface. React's component-based architecture, along with TypeScript's strong typing system, has enabled us to create reusable and type-safe UI components, encouraging code maintainability and scalability. Redux has made centralized state management and asynchronous side effect handling possible, along with Redux Toolkit and Redux Saga, guaranteeing a reliable and effective data flow inside the application. Furthermore, React Router has made it possible to navigate between views with ease, which has improved user satisfaction and engagement overall. We have greatly improved the accessibility and customization possibilities of the Pub Quiz Iași application by giving our users a customized experience with two different localizations, English and Romanian, and two contrasting themes, light and dark theme.

On the back end, a stable and effective framework for server-side development and data administration has been made possible by the combination of Node.js with Express.js and Mongoose ODM for MongoDB. Because of the event-driven, non-blocking architecture of Node.js, we are able to handle multiple requests at once with optimal performance and scalability. The development of RESTful APIs has been made easier by Express.js, allowing the front end and back end components of the application to communicate with each other flawlessly.

Pub Quiz Iași application is a significant step toward modernizing the Quiz Iași event. It takes the event into the digital sphere while upholding its fundamental principles and enhancing accessibility and functionality. Future updates and additions to the application's features will make it even more useful for attendees and event planners, and it will cement the application's position as a crucial part of Quiz Iași’s digital shift.

Future improvements

We want to add more features to the application soon in order to give users an even more thorough and interesting experience. A number of new additions and features that will improve the Pub Quiz Iași application and strengthen its value proposition are on our roadmap. These are a few of the key functionalities we intend to focus on:

- increasing the social aspect of the application by including a chat section for each team. Team members can share information and discuss strategy for the upcoming games;
- completely digitalize the event by hosting a game on the website. These games could be played if the physical event is not possible, or as a stand-alone game. We could establish two different leaderboards: one for the physical event and one for the online event.
- developing a chat bot to help users through our application, ensuring quick learning of the capabilities and a better user experience.

Bibliography

[1] S. Vlasic, “Three-Tier Architecture Approach for Custom Applications,” Zirous. https://www.zirous.com/2022/11/15/three-tier-architecture-approach-for-custom-applications-2/

[2] “What is Three-Tier Architecture | IBM.”, https://www.ibm.com/topics/three-tier-architecture

[3] “SPA (Single-page application) - MDN Web Docs Glossary: Definitions of Web-related terms | MDN.”, https://developer.mozilla.org/en-US/docs/Glossary/SPA

[4] “Vite.”, https://vitejs.dev

[5] “React.”, https://react.dev/

[6] “JavaScript.”, https://developer.mozilla.org/en-US/docs/Web/JavaScript

[7] “JavaScript With Syntax For Types.”, https://www.typescriptlang.org/

[8] “Redux - A predictable state container for JavaScript apps. | Redux.”, https://redux.js.org/

[9] “Redux Toolkit | Redux Toolkit.”, https://redux-toolkit.js.org/

[10] “Redux-Saga - An intuitive Redux side effect manager. | Redux-Saga.”, https://redux-saga.js.org/

[11] “MUI: The React component library you always wanted.”, https://mui.com/

[12] “React Router.”, https://reactrouter.com/en/main

[13] “react-i18next.”, https://react.i18next.com/

[14] “Mobile vs. Desktop Traffic Market Share [January 2024],” Similarweb. https://www.similarweb.com/platforms/

[15] “Node.js — Introduction to Node.js.”, https://nodejs.org/en/learn/getting-started/introduction-to-nodejs

[16] “Express - Node.js web application framework.”, https://expressjs.com/

[17] auth0.com, “JSON Web Tokens.”, http://jwt.io/

[18] “MongoDB: The Developer Data Platform,” MongoDB. https://www.mongodb.com

[19] “Mongoose ODM v8.1.2.”, https://mongoosejs.com/