MASARYK
UNIVERSITY

FACULTY OF INFORMATICS

Application portal for hackathons

Bachelor's Thesis

MATEJ TARČA

Brno, Fall 2023

MASARYK UNIVERSITY

FACULTY OF INFORMATICS

Application portal for hackathons

Bachelor's Thesis

MATEJ TARČA

Advisor: RNDr. Jaromír Plhák, Ph.D.

Department of Machine Learning and Data Processing

Brno, Fall 2023

MUNI
FI

Declaration

Hereby I declare that this paper is my original authorial work, which I have worked out on my own. All sources, references, and literature used or excerpted during elaboration of this work are properly cited and listed in complete reference to the due source.

Matej Tarča

Advisor: RNDr. Jaromír Plhák, Ph.D.

Acknowledgements

I want to show my appreciation to my advisor, RNDr. Jaromír Plhák, Ph.D., for his attentiveness and professional guidance during writing of this thesis. Additionally, I would like to thank all of the Hack Kosice organizers who were willing to test the application and provided valuable feedback.

Abstract

This thesis details the development of an application portal designed for hackathon events. Hackathons, known for their collaborative and competitive coding environment, necessitate a streamlined process for managing participant applications. This ensures effective selection and invitation of the best participants. The portal, implemented as a web application, emphasizes a user-centric, modern, and responsive design. It adheres to solid software engineering practices, including automated testing and monitoring. Furthermore, the thesis explores the software's design and analytical phase prior to the implementation and concludes by proposing possible future enhancements of the system.

Keywords

hackathon, application portal, web application, Typescript, React, Next.js

Contents

Introduction and motivation 1

1 Analysis and System Design 3
- 1.1 Existing Solutions . . . 3
  - 1.1.1 Quill . . . 3
  - 1.1.2 Myhackupc . . . 4
  - 1.1.3 HackPortal by acmutd . . . 5
- 1.2 Requirements specification . . . 6
  - 1.2.1 Functional requirements . . . 7
  - 1.2.2 Non-functional requirements . . . 9
- 1.3 Use case diagram . . . 10
- 1.4 Data model . . . 10
- 1.5 Analysing system interactions . . . 12
  - 1.5.1 Before the event . . . 14
  - 1.5.2 During the event . . . 14
- 1.6 Application states . . . 16

2 Technologies 18
- 2.1 Technical Stack . . . 18
- 2.2 Next.js . . . 18
  - 2.2.1 Next.js 13 . . . 19
- 2.3 UI components and styling . . . 21
  - 2.3.1 Tailwind CSS . . . 21
  - 2.3.2 Component library - shadcn/ui . . . 22
  - 2.3.3 Storybook . . . 23
- 2.4 Prisma . . . 24
  - 2.4.1 Prisma client . . . 25
  - 2.4.2 Prisma migrations . . . 25

3 Implementation 26
- 3.1 General features . . . 26
  - 3.1.1 Authentication and Authorization . . . 26
  - 3.1.2 Database setup . . . 29
  - 3.1.3 Email Communication . . . 30
- 3.2 Application form . . . 31

3.2.1 Application form structure . . . 31
3.2.2 Editing application form . . . 32
3.2.3 Rendering form and validation . . . 32
3.2.4 File upload . . . 34
3.2.5 Support for unsigned users . . . 35
3.3 Organizer's dashboard . . . 36
3.3.1 Dashboard structure . . . 36
3.3.2 Application judging . . . 38
3.4 Check-in . . . 39
3.5 Other features for hackers . . . 40
3.5.1 Team creation . . . 40
3.5.2 Travel reimbursement requests . . . 40
3.6 GDPR compliance . . . 41
3.7 Deployment . . . 42

4 Quality assurance and monitoring **44**
4.1 Quality assurance . . . 44
4.1.1 Automated testing . . . 44
4.1.2 Static analysis . . . 48
4.1.3 Github Actions . . . 48
4.2 Monitoring . . . 49
4.2.1 Sentry . . . 50
4.2.2 Datadog . . . 50

5 User Testing **53**

6 Conclusion and future work **55**

Bibliography **56**

List of Figures

1 The application form in the Quill application portal 4
2 Organizer dashboard with statistics from the Myhackupc portal employed by Hack Kosice 2023 5
3 Unstyled landing page of the HackPortal by acmutd . . . 6
4 Use case diagram of the hackathon application portal . . 11
5 ERD for the hackathon application portal data model . . 13
6 Sequence diagram for the happy path of hacker filling the application form and getting invited to the hackathon. . . 15
7 Sequence diagram for the check-in process at the start of the event. 16
8 State diagram with statuses of the application object . . . 17
9 The story of Button displayed in the Storybook UI 24
10 The admin's UI for editing a step of the application form . 33
11 The dialog with a form to edit a given form field 33
12 The hacker's UI with application steps 34
13 The structure of the organizer's dashboard 37
14 The page for reviewing an assigned application. 38
15 The cookie consent modal shown upon the first entry to the application portal. 42
16 Example of a test result in Playwright's UI 47
17 Example of a Github pipeline run on a pull request. . . . 49
18 Example of the web vitals analysis visible in the Sentry web interface. 51
19 Example of a log entry in the Datadog web interface. . . . 52
20 Example of a bug reported in the Notion tool. 54

Introduction and motivation

Hackathons are competitions where participants, often called hackers, are invited to solve real-world problems through programming. They are held in a relaxed but fast-paced environment encouraging maximum innovation. The relative easiness of organizing such events led to a rise in the number of hackathons in past years.

Typically attracting student participants, hackathons occur both online and in-person. In-person events have a physical limit on attendance, a constraint also applicable to online versions due to the complexities of managing large numbers of participants and projects. The popularity of hackathons often results in a surplus of interested students, necessitating an application process. Organizers must select and invite only the most suitable candidates. Moreover, the application collection also aids in accessing the event demand and planning accordingly.

Central to this selection process is the application portal for hackathons, typically a web-based platform designed to streamline the collection and evaluation of applications, ultimately leading to inviting the most qualified participants. This thesis focuses on developing a user-friendly and modern application portal, which will be utilized for the organization of the Hack Kosice 2024 hackathon.

Although basic online forms and tools such as Excel can efficiently collect applications for hackathons, developing a custom portal offers notable advantages. A custom-designed portal not only enhances the hackathon's image by offering a seamless signup experience but also plays a crucial role in community building, an integral aspect of hackathon growth. Furthermore, a tailored portal offers practical advantages. It can include specific features that simplify organizational tasks and handle complex application processes more effectively than conventional tools. This approach, while demanding significant investments in initial development, promises long-term benefits in terms of operational efficiency and user engagement.

The initial chapter of this thesis outlines the early phases of the software engineering process. It covers the specification of software requirements and use cases, the design of the data model, and the analysis of user interactions using sequence diagrams. Following this, the

thesis explores the technologies employed in developing the system. A detailed examination of the system's key components and their implementation is provided in the fourth chapter. The final chapters are dedicated to quality assurance, encompassing automated testing, user monitoring, and manual user testing, thereby ensuring the system's robustness and user-friendliness.

1 Analysis and System Design

This chapter focuses on the analysis and design components of the software development process for a hackathon application portal. The examination begins with an overview of existing hackathon application portals, followed by the definition of system requirements, which are illustrated through a use case diagram. The final section addresses the design of the system's data model and the analysis of key use cases using activity and sequence diagrams.

1.1 Existing Solutions

Given the widespread popularity of hackathons, several systems serving as application portals have been developed. This section explores three such popular systems, evaluating their functional aspects and identifying areas where they fall short. The systems were chosen based on their popularity evaluated by the position in search resultes when searching for "application portal for hackathons" and the number of stars and forks on Github. Additionally, the second system - Myhackupc - was chosen because it was already utilized during the organization of the previous Hack Kosice hackathons.

1.1.1 Quill

Quill is a registration portal for hackathons which was initially developed for HackMIT in 2015. It has been utilized by over 60 hackathons globally and has been forked more than 300 times on GitHub. This platform offers a user-friendly interface for hackers, featuring an application form, a dashboard to track application status, and a team creation feature. Additionally, it allows organizers to review applications and invite selected participants. However, Quill falls short in offering advanced functionalities such as participant check-in and collaborative application voting by multiple organizers. Technologically, it relies on outdated systems like MongoDB for storage and is written in vanilla Javascript using the AngularJS framework version 1.8.0, which is no longer actively supported. Furthermore, the lack of recent development activity on the portal, with the last GitHub

commit dating back to 2020, suggests it may not be an ideal choice for use as an application portal.

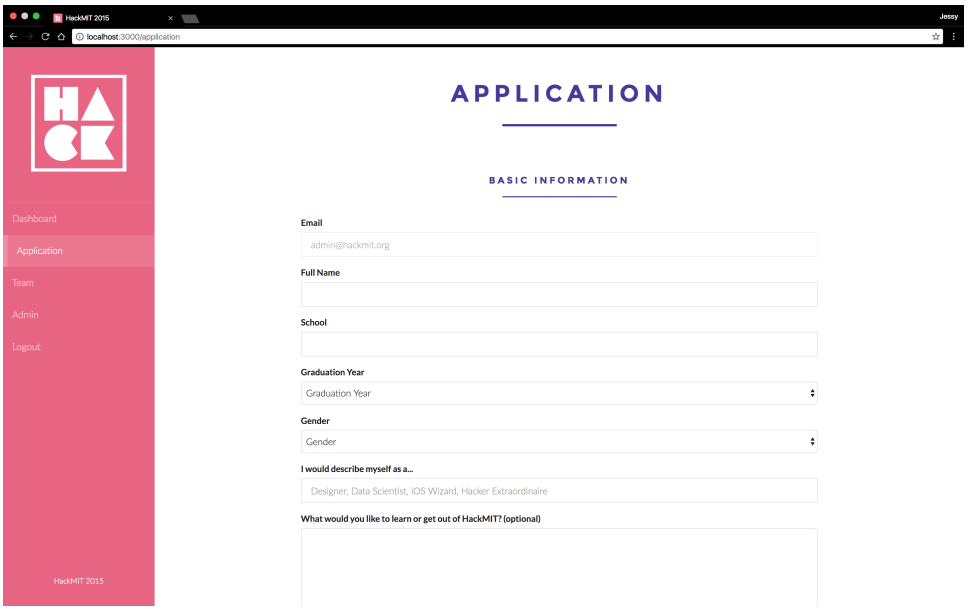

Figure 1: The application form in the Quill application portal

1.1.2 Myhackupc

Myhackupc portal, developed for the renowned Barcelona-based HackUPC hackathon, has actually been used by Hack Kosice for two consecutive years. The system is built using the Python-based framework Django. However, due to unfamiliarity with this less commonly used framework, adapting the portal to meet specific requirements posed challenges. Despite this, the organizers' section of the portal is impressively designed, offering a robust view of various statistics and the ability to vote on applications before sending invitations to hackers. This aspect of the system can provide valuable insights and inspiration for the development of our application portal. Nevertheless, the technology stack used in Myhackupc, coupled with its limited front-end customization and application form adaptability, diminishes its suitability as an optimal choice for an application portal.

Figure 2: Organizer dashboard with statistics from the Myhackupc portal employed by Hack Kosice 2023

1.1.3 HackPortal by acmutd

This application portal is arguably the best written one of the portals that were selected for this comparison. It is developed using Next.js with TypeScript, adhering to good development practices. Key features include a user-friendly application form, fully customizable front-end, QR code-based check-in, and Google OAuth integration. Active development and maintenance is also ongoing. The portal even offers additional functionalities, such as live event support with push notifications, announcements, and a Q&A section, which exceed the requirements of our use case.

However, there are notable deficiencies. Essential features like team creation and application voting are absent. The application form, designed as a single extensive form, lacks the capability to save progress and complete it at a later stage. Additionally, the use of a slightly outdated version of Next.js and the deep integration with Firebase, as opposed to a traditional SQL-based storage system, present fur-

ther challenges. These issues, in combination, render the system less suitable for our specific needs in organizing a hackathon.

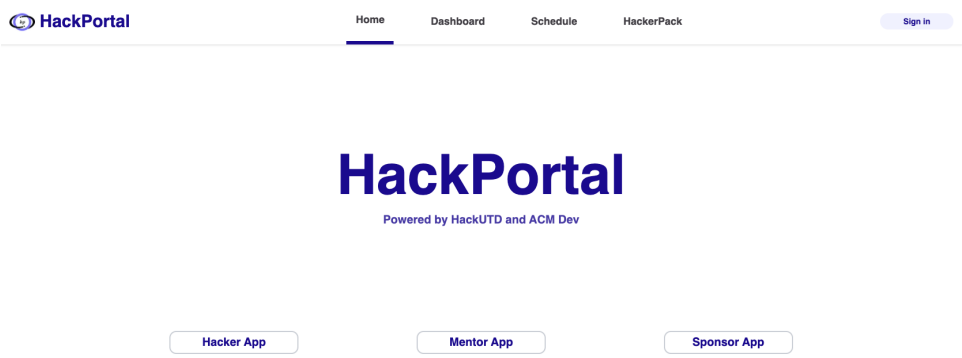

Figure 3: Unstyled landing page of the HackPortal by acmutd

In addition to specific shortcomings, these systems have some shared fundamental limitations. Each is designed for a single-event use, requiring a redeployment with a clean database in order to support subsequent hackathons. Application form modifications require code changes, leading to a cumbersome iteration process. Moreover, they mandate user registration prior to application form access, potentially affecting user retention. Despite these challenges, valuable insights can be obtained from these systems to help with the design process of the proposed hackathon application portal.

1.2 Requirements specification

In this section we will focus on specifying user requirements for the system. These type of requirements include the functional and non-functional ones. The requirements are focused solely on the external

interactions of the system and are written in a straightforward natural language. (1, p. 94)

1.2.1 Functional requirements

Functional requirements define the system's expected operations. They detail the services to be provided by the system and its responses to specific inputs.(1, p. 85) A practical approach to defining these requirements involves first identifying the system's stakeholders, followed by a description of their objectives. (1, p. 103)

In the context of the hackathon application portal, the stakeholders can be viewed as all the different types of end-users of the system. These users include: Hackers, Organisers and Admins. Given the extensive range of functional requirements identified for the application portal, an additional categorization by feature types is employed. These feature categories include application, reimbursement, team, and check-in. The following list presents the system's functional requirements, organized by actor and feature type:

- Hacker
  - Application
    - Hacker should be able to fill and submit the application form.
    - Hacker should be able to start filling the form even without creating an account in the system.
    - Hacker should not be able to submit the application without creating an account.
    - Hacker should be able to come back to a previously saved and unfinished application.
    - Hacker should be able to see the status of their application after submission.
    - Hacker should be able to confirm or reject their attendance at the hackathon after getting an invitation.
  - Reimbursement
    - Hacker should be able to request a travel reimbursement.
    - Hacker should be able to see the status of the travel reimbursement after making the request.
  - Team

        * Hacker should be able to create a team.
        * Hacker should be able to invite other hackers to their team.
        * Hacker should be able to join or leave other teams.
        * Hacker should be able to see their team members along with the statuses of their applications.
        * Hacker who created the team should be able to kick out other members.
    - Checkin
        * Hacker should be able to open a screen which is then presented during the check-in at the hackathon, or get a code for the check-in.
- Organiser
    - Application
        * Organiser should be able to review an application, which will assign a score to it.
        * Organiser should be able to see, filter and sort a list of both complete and incomplete applications along with their scores.
        * Organiser should be able to invite hackers to the hackathon.
        * Organiser should be able to see a screen with statistical data and graphs about the applications.
    - Reimbursement
        * Organiser should be able to review and approve reimbursement requests.
        * Organiser should be able to see a screen with statistical data and graphs about the reimbursements.
    - Checkin
        * Organiser should be able to open a screen for the event check-in, where he can scan or input the hacker's check-in code.
- Admin
    - Application
        * Admin should be able to edit the application form structure easily.

    - Admin should be able to set a date when the application portal will start or stop allowing application submission.
    - Admin should be able to edit any application easily.
    - Admin should be able to setup review parameters that are used to score hackers' applications
  - Team
    - Admin should be able to edit any team easily.

1.2.2 Non-functional requirements

Non-functional requirements describe how a system will achieve its intended functionality and set constraints on its behavior. They include aspects such as software performance, security level, flexibility and usability. While testing and evaluating these requirements may be difficult, they are essential for successfully building the desired software system.(2) These non-functional requirements were set for the development of the hackathon application portal:

- The application portal should be open-source and reusable by other hackathons.
- The application portal should be designed in a way that allows easy reusability for multiple events.
- The application portal should have a robust data backup and recovery system in place to ensure that user data is not lost due to system failures, data corruption, or other disasters.
- The application portal must have robust security measures in place to protect the personal information of hackers and organisers.
- The application portal should comply with GDPR regulations.
- The application portal should be accessible to users with disabilities, ensuring that everyone has an equal opportunity to participate in the hackathon.
- The application portal should be designed to handle many concurrent users.
- The application portal should keep logs of all user activities.
- The application portal should be designed in a way that makes it easy to maintain and update over time, including features such as modular code design, code comments, and documentation.

- The application portal should include performance monitoring tools that allow developers to identify and diagnose performance bottlenecks, errors, or other issues in real time.

1.3 Use case diagram

One technique for discovering requirements in software engineering is the application of use cases, which are effectively visualized using use case diagrams, a component of the Unified Modeling Language (UML) standard. (1, p. 106) Use case diagram specifies system's actors using a "stick man" icon, the use cases using ellipses and relationships between them using different lines and arrows. (3)

In the context of an application portal, the primary actor is the 'User', a generalized representation encompassing all actors except the unregistered hacker. The User's primary interactions include sign-in and sign-up functionalities. Additionally, the requirement that states that an unregistered hacker should be able to fill out an application, is depicted through the 'Hacker' actor and its three use cases. These use cases are also applicable to a 'Registered hacker' via generalization. One of the shared use cases across all actors is "updating the status of an application", a feature integral to both the Hacker's responsibilities (such as submitting applications or responding to invitations) and the Organizer's tasks (like inviting or rejecting hackers). It is also noted that the 'Admin' possesses all functionalities of an Organizer, along with the exclusive privileges to modify the application form and user data. The complete use case diagram of the hackathon portal is illustrated in Figure 4.

1.4 Data model

This section focuses on the development of a robust data model for the application portal, which involves the documentation and organization of all data stored by the system. Given the use of a traditional relational database, the data structure will be visualized using an Entity Relationship Diagram (ERD), a widely-recognized method in this context (4).

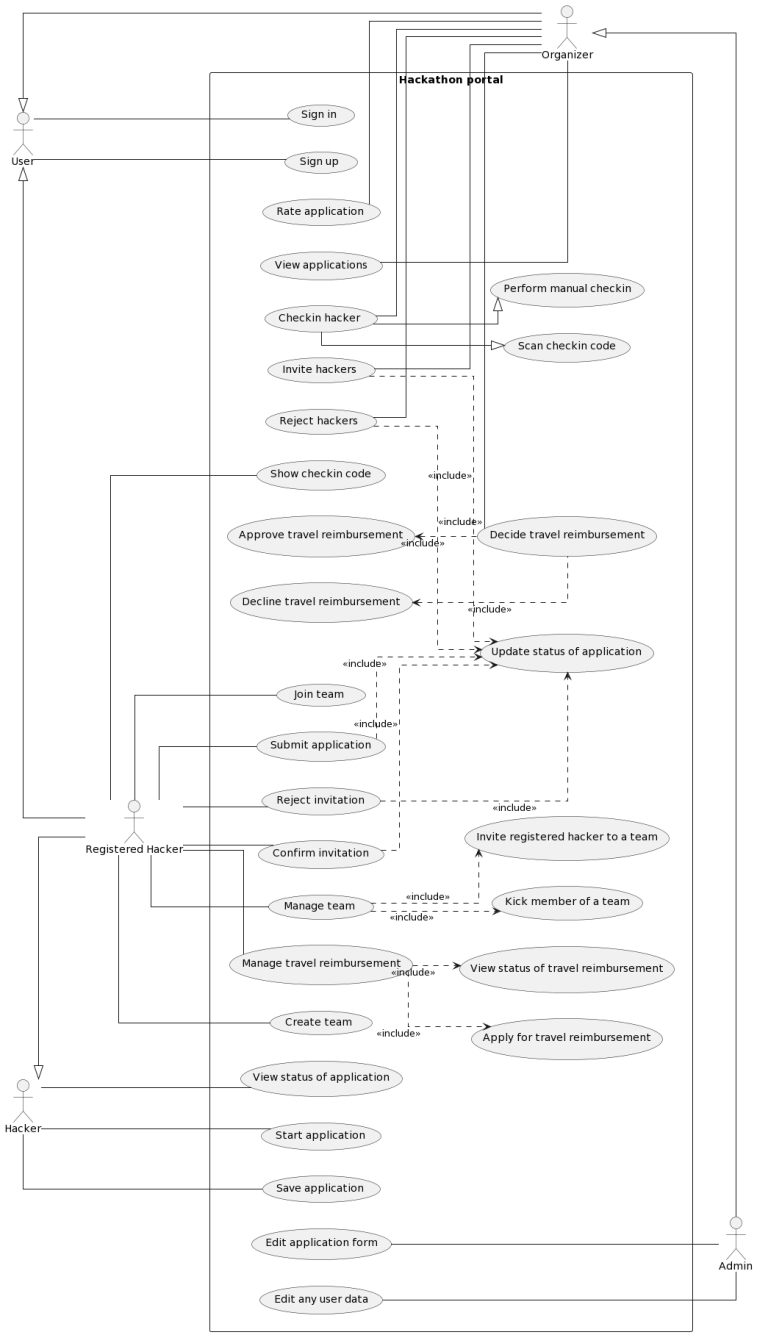

Figure 4: Use case diagram of the hackathon application portal

At the core of the data model is the `User` entity, which is further linked to the `Hacker` and `Organizer` entities to specify the user's role in the system. The `Hacker` entity allows further linking to the `Application`, `Team`, and `TravelReimbursementRequest` entities, whereas for `Organizer`, there is a link to the `Vote` entity.

Another fundamental entity in the model is `Hackathon`, reflecting the system's requirement to support multiple hackathons. Each hackathon features a dynamic application form, defined as a list of `Steps`, each including individual `FormFields`. To accommodate selection-based form field, the `OptionList` entity, containing various `Options`, is associated with the respective `FormField`.

The `ApplicationFormFieldValue` entity is designed to store the responses entered by hackers. It links to the corresponding `Application` and `FormField`, and includes fields for textual input or connections to `Options` or `File` entities, the latter supporting file uploads.

The voting mechanism incorporates `VoteParameter` and `Vote` entities. The `Vote` entity forms a linkage between the `Organizer`, `VoteParameter`, and the actual value of the vote. Notably, the model does not include a direct field for application scores. Instead, scores are calculated as needed, allowing organizers the flexibility to modify voting parameters even during the voting process.

The complete ERD is presented in Figure 5.

1.5 Analysing system interactions

To enhance comprehension of the system and its expected behavior, an examination and visualization of the interactions between the actors and the system, which fulfill defined use cases, is essential. Sequence diagrams, widely recognized as the most prevalent form of interaction diagrams, will be utilized for this purpose. These diagrams emphasize the chronology of messages exchanged among participants. The analysis will concentrate on a critical component of the portal: the journey of a potential hacker from the engagement with the system to the participation in the hackathon. This examination will be limited to the "happy path", where the hacker completes all required fields, submits an application, receives an invitation, confirms attendance, and checks in at the event. To maintain simplicity, the analysis will exclude use

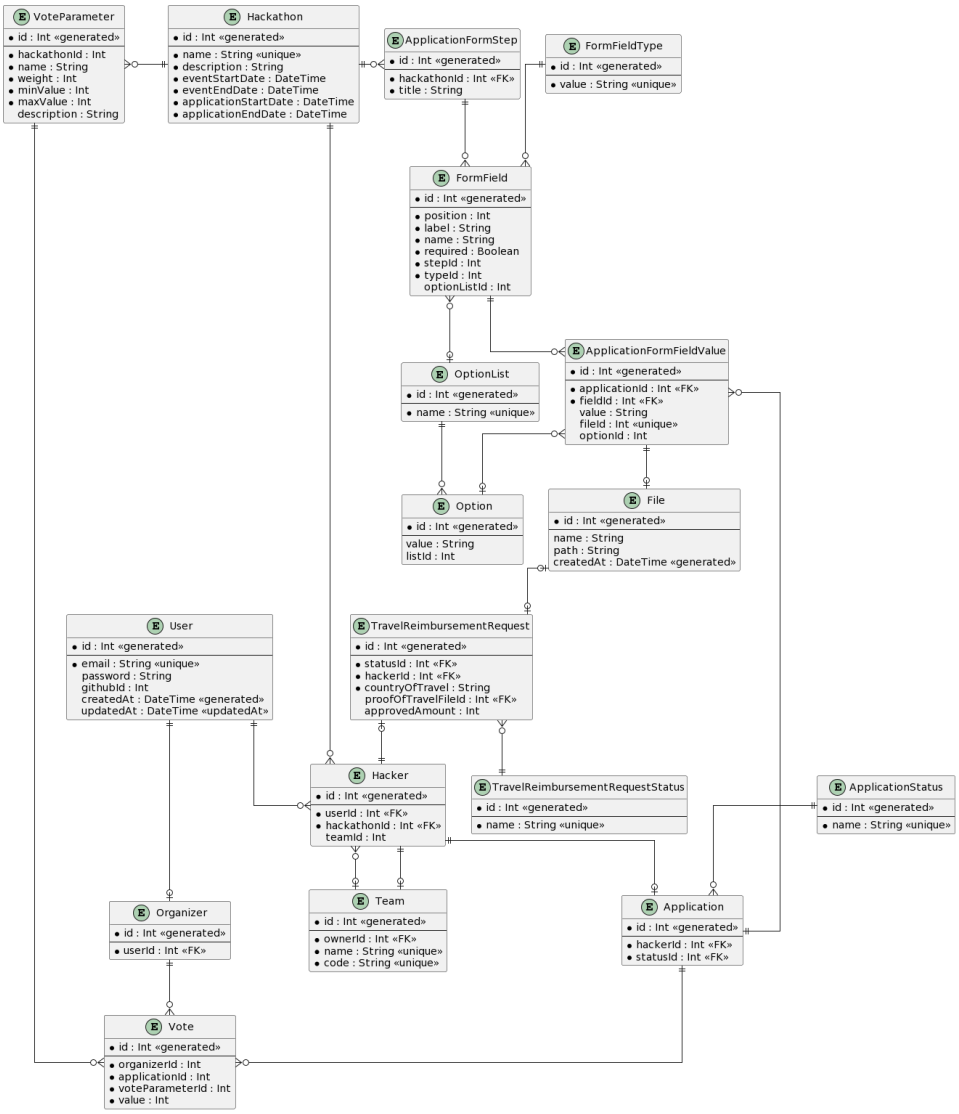

Figure 5: ERD for the hackathon application portal data model

cases related to team formation, joining a team, or requesting travel reimbursement.

1.5.1 Before the event

Prior to the event, participants access the application portal where a new application is automatically generated upon their initial visit, provided they have a pre-existing account. Participants then proceed to fill out all necessary fields. The system stores this information in a database, linking it to their specific application. Upon completion of all required fields, the system enables submission of the application and confirms successful submission via email.

Subsequently, organizers begin evaluating the applications through a voting process within their section of the system. Once an application gets a sufficient number of favorable votes, an organizer can manually send an invitation to the participant. Upon receiving the invitation, the participant is informed through email and must revisit the application portal to confirm their attendance. This confirmation marks the conclusion of the interaction, making it possible for the hacker to attend the event.

The interaction describe above is visualized in the sequence diagram in Figure 6.

1.5.2 During the event

At the beginning of the event, participants are required to verify their invitation as they enter the venue. This process involves having an organizer scan the participant's personal check-in QR code. If scanning the QR code proves unsuccessful, participants are expected to provide a manual code obtained from the portal. This code should be then presented to the organizer during check-in instead. The scanning process authenticates the participant's application, subsequently displaying the verification status and additional information to the organizer. Once the organizer verifies this information, participants are permitted to enter the event.

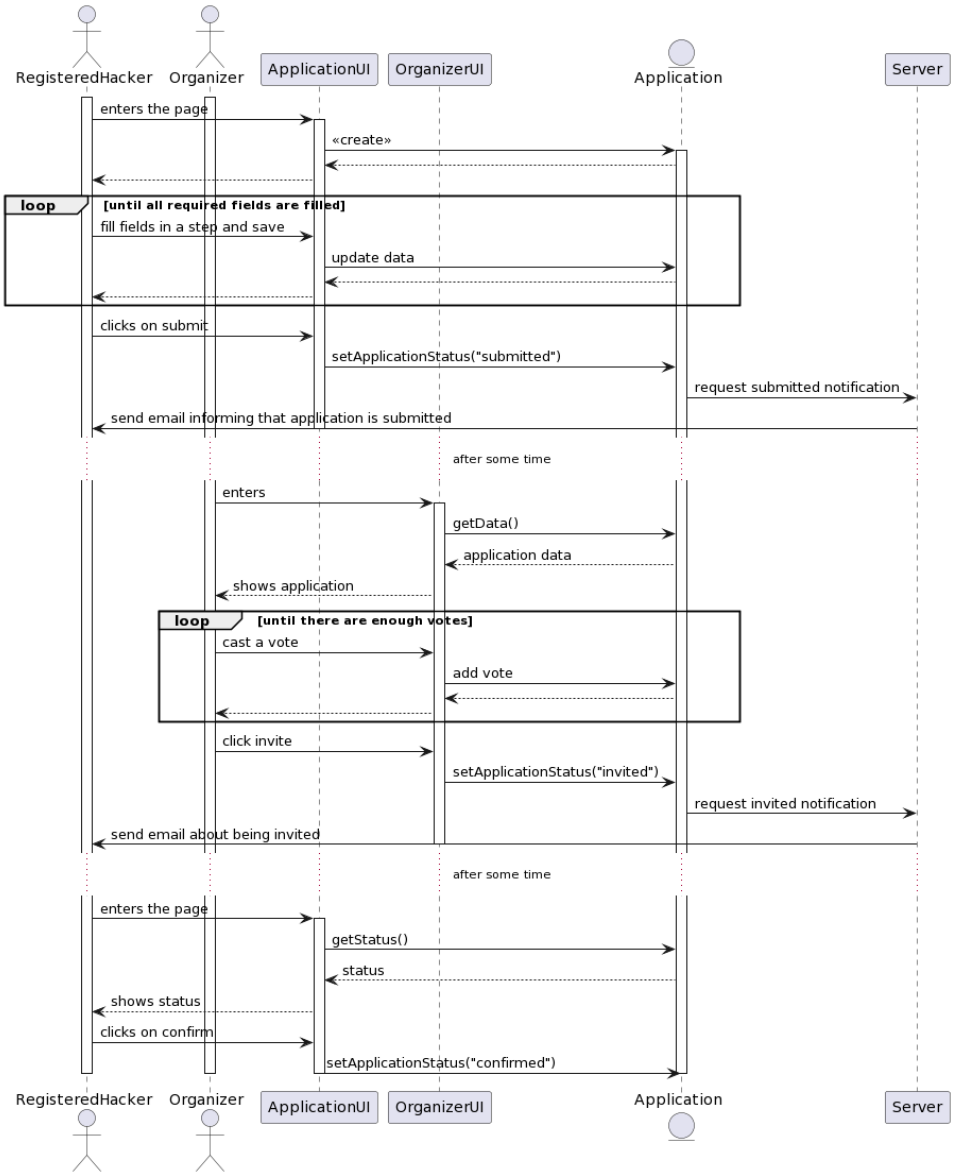

Figure 6: Sequence diagram for the happy path of hacker filling the application form and getting invited to the hackathon.

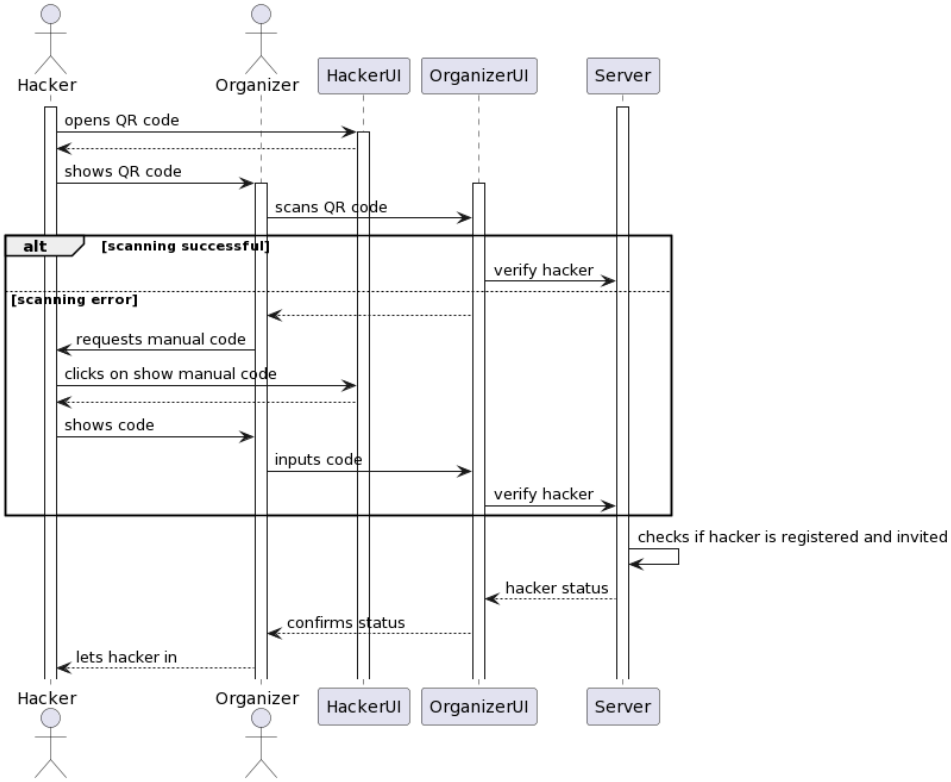

Figure 7: Sequence diagram for the check-in process at the start of the event.

Figure 7 illustrates this process with the use of a sequence diagram.

1.6 Application states

The last part of the analysis focuses on the different states that an application can be in. These states are listed in the ApplicationStatus table and connected to the Application object. They include these states: 'opened' (the starting state), 'submitted', 'invited', 'rejected', 'confirmed', 'declined', and 'attended'. A state diagram in Figure 8 illustrates these states and their changes.

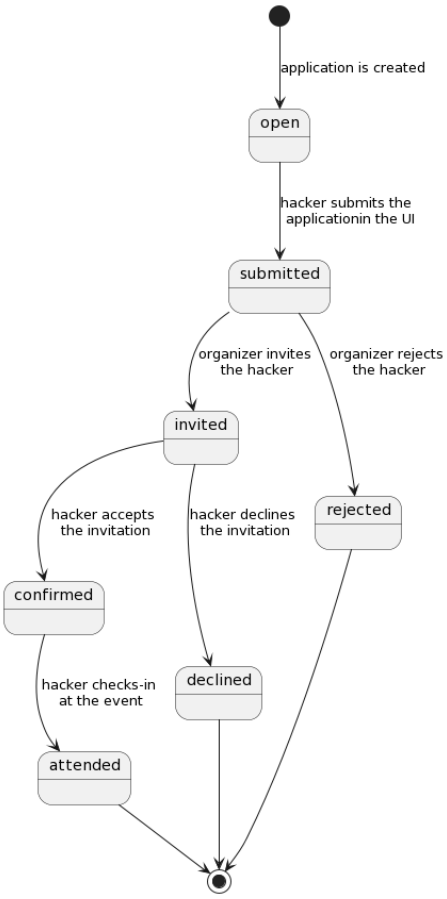

Figure 8: State diagram with statuses of the application object

2 Technologies

This chapter explores the technologies employed in the development of the hackathon application portal. The chapter begins with an overview of the entire technical stack, followed by detailed discussions of key libraries and frameworks in subsequent sections.

2.1 Technical Stack

The requirement for the hackathon portal to be a dynamic web application led to the choice of the popular React library for its development. However, the Next.js framework is preferred over plain React, as it enhances the development process and offers various optimization features, including efficient server-side rendering.

Next.js can be written in both the JavaScript and TypeScript language, with TypeScript being selected for its type-checking features that improve code maintainability and scalability. The development in TypeScript can be supported by powerful tools such as ESLint for static code analysis, Prettier for code formatting, and Jest and Playwright for automated testing. These tools, along with their application, are elaborated in the Section Quality assurance.

The application's UI design incorporates the Tailwind CSS framework and the shadcn/ui component library, facilitating a responsive, friendly and modern user experience. Detailed insights into the choice and application of these styling resources are presented in Section UI components and styling.

For the deployment, the portal employs a Linux-based Virtual Private Server. This setup includes the NGINX web server and PM2 process manager to ensure stable and continous operation.

2.2 Next.js

Next.js calls itself "the React framework", and it is currently the most popular way to build React applications. It is recommended to use when creating a new React application, even by the official React

documentation. (5) The most beneficial feature of Next.js for our system is its server-side rendering capability.

Traditional React renders the application solely on client's device, where the initial load of a pure React web page involves an almost empty HTML file being filled by JavaScript on the client side. However, this approach has many limitations. Slower devices tend to experience longer page loading times, as they must handle rendering and executing the JavaScript code themselves. Additionally, this method can pose challenges for search engine optimization, as web crawlers typically do not execute JavaScript. Next.js offers an elegant solution to these issues by executing React code on the server, within the Node.js runtime, and generating HTML that is then transmitted to the client. After that, pure React can take over and "hydrate" the page, effectively executing JavaScript code that cannot run on the server, such as attaching event handlers.

Furthermore, Next.js brings a host of other features into play, including file-based routing, built-in image optimization, static site rendering, incremental static generation, and more. While an exhaustive description of all these features lies beyond the scope of this thesis, our focus primarily centres on one particular set of features that played a pivotal role in the development of the application portal: the new App router with React server components and server actions.

2.2.1 Next.js 13

In October 2022, Next.js introduced a major change to React application development with the release of version 13. This update brought a new way to create React applications by adding the App router, which allows developers to use cutting-edge React features such as React server components and server actions. [1]

The App router uses a file-based routing system, where each part of the URL path corresponds to a folder in the app directory. These folders can contain several special files that are responsible for rendering the content of a given path. Some of them which are used heavily in our application include:

1. Although the App router was introduced in the Next.js 13 release, it became stable only after the Next.js 13.4 release in May 2023

- page.tsx - this file exports a React component that will be rendered when a user accesses the given URL route
- layout.tsx - this file exports a React component that will wrap all the pages in the given route and its subroutes
- loading.tsx - this file exports a React component that is returned by the server immediately after accessing the route and is rendered while the page.tsx component is loading
- error.tsx - this file exports a React component that is rendered if there is any error while rendering the page.tsx for the route or any subroutes

Most importantly, all components used by the App router are, by default, React server components. These components are rendered solely on the server - they have direct access to server resources such as databases and are executed in the Node.js runtime. They can also use the async/await keywords for fetching the data and return only a Promise of JSX elements (while the server waits for the Promise to resolve the loading.tsx is rendered on the client).

One of the benefits of this approach is that there is almost no need for any client-side fetching libraries - the fetching of APIs is executed in the server component, and data are either rendered directly on the server or passed to client components using props. Or, in the case of this system, there is no need to implement any API in the traditional sense. The database can be directly accessed in the server components to get the required data.

The next integral part of the new Next.js 13 approach are server actions. Web applications normally want to respond to the user's interaction by executing some server-side code (such as writing to a database). In the App router, this is achieved by invoking a function defined as a server action from the client component handlers. If a function is marked with the "use server" directive, Next.js understands that it should compile to invoke a generated POST API endpoint instead of calling the function directly when accessed in the client-side code.

This new approach presents certain limitations: the backend and frontend code is closely integrated. Initially, this integration facilitates

quicker application setup. However, it poses challenges for subsequent modifications, such as switching to alternative frontend or backend technologies or utilizing a single backend across multiple clients, including mobile applications. Although these features are not required for the current system, they represent a potential area of concern. To mitigate this, the system's code organization can be improved in the following ways:

- All "getters" - functions that query data in server-side components are in separate files inside the `src/server/getters` folder and have a manually typed interface.
- All server actions are defined in separate files inside the `src/server/actions` folder and have a manually typed interface.

Separating out a Data Access Layer like this makes the code not only easier to maintain but also strengthens the app's security. During security audit, checking this layer is a key for making sure access permissions are set up right. Importantly, this method of separation is suggested by the official Next.js guidelines, too. (6)

2.3 UI components and styling

A key attribute of the web application, as perceived by users, is its visual appeal and user-friendliness. An interface that is easy to navigate not only facilitates smoother user interaction but also significantly improves the user experience by making the application accessible and intuitive. To realize these objectives, considerable effort was put into selecting robust UI components and libraries that emphasize both accessibility and functionality. The subsequent section elaborates on the criteria and methodology used in choosing, customizing, and organizing a UI component library that aligns with the aesthetic and functional requirements of the hackathon application portal.

2.3.1 Tailwind CSS

For the hackathon application portal's styling, the Tailwind CSS framework was chosen over more traditional methods of writing plain CSS.

Tailwind CSS provides HTML utility classes that directly map to CSS attributes, eliminating the need for writing custom CSS. Conventional CSS implementation typically involves assigning a semantic class (for example ".primary-button") to an element and attaching all styles to it, which can lead to a lot of redundant code.

Tailwind's approach, which associates classes with distinct CSS properties such as ".text-white," ".bg-red-500," or ".font-medium," enables developers to bypass the need to write CSS entirely. Tailwind also optimizes project bundle size by including only the classes used in the application, ensuring a small CSS bundle size (7). One of the drawbacks of Tailwind is that applying intricate styles requires stringing together many classes, which can compromise code readability. For example this button component:

```
<button class="cursor-pointer inline-flex items-center
    justify-center rounded-md text-md font-default ring-
    offset-white transition-colors focus-visible:outline-
    none focus-visible:ring-2 focus-visible:ring-slate-950
    focus-visible:ring-offset-2 disabled:pointer-events-none
     disabled:opacity-50 dark:ring-offset-slate-950 dark:
    focus-visible:ring-slate-300">Save</button>
```

However, this issue can be easily addressed by a proper component abstraction and keeping all UI components with many tailwind classes in separate files, which essentially means you will not encounter the long class names lists during normal development.

2.3.2 Component library - shadcn/ui

While the styling of the UI components on web page is important, achieving a seamless functionality of these components is also vital. Creating well-functional web UI components is a challenging task, requiring a solid understanding of various HTML tags and their attributes. Moreover, it is critical to support the essential accessibility features of the web. For example, in the case of text input, these features may involve "aria-label" or "aria-error" properties to convey the correct message to users using screen readers. More complex elements like dropdowns also benefit from solid keyboard navigation options. That is why utilising a library with well-crafted components that take all these requirements into consideration is often the better choice.

To simplify the development of the hackathon application portal, we adopted and customised a ready-made UI component library called shadcn/ui. Shadcn/ui is a wrapper around the Radix UI headless component library, which introduces sensible styling to the Radix components using Tailwind CSS. It is a relatively new library, first released in March 2023, but thanks to its minimalistic and modular approach, it is quickly gaining popularity.

An advantage of shadcn/ui is that it does not follow the traditional dependency installation; instead, you can directly integrate the components' code into your project and start using them. This approach provides an exceptionally high level of customizability. Moreover, shadcn/ui components already utilize other popular React libraries, such as @tanstask/react-table and react-hook-form, delivering a comprehensive suite of well-functioning components.

By customising shadcn/ui components and including them in the codebase, a special component library was built for the project. All the components reside in the `src/components` folder, while the basic ui components are in the `src/components/ui` folder. These UI components include: "alert-dialog, data-table, tabs, card, popover, toaster, scroll-area, label, tooltip, alert, combobox, heading, stack, calendar, radio-group, command, Button.test, dialog, table, button, text, toast, checkbox, collapsible, dropdown-menu, select, textarea, input, form". Other more complex components are in the `src/components/common` folder - e.g. the Navbar component.

2.3.3 Storybook

To ensure comprehensive documentation for our custom UI library and facilitate the development and testing of the components, the Storybook framework was adopted. Storybook's fundamental element is a "story", designed to present a single UI component with all its properties and variations through the Storybook UI 9. These stories are created as separate TypeScript files located in the `src/components/stories` folder. You can access all the stories by launching the Storybook development server using the `npm run storybook` command.

The integration of Storybook into the hackathon application portal represented a task with low effort but substantial benefits. It will

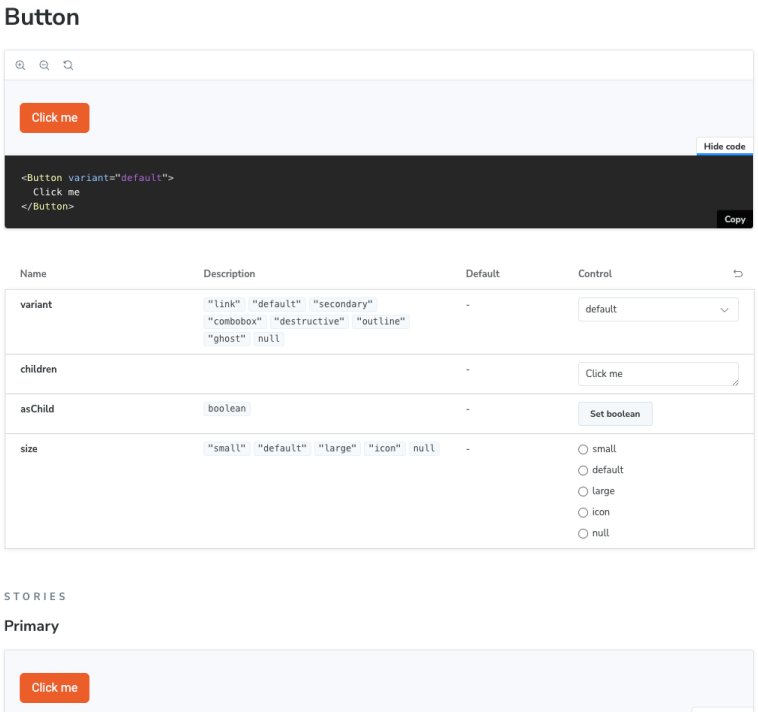

Figure 9: The story of Button displayed in the Storybook UI

greatly assist new developers joining the project, enabling them to easily identify the right components and explore all their properties without the need for extensive code exploration.

2.4 Prisma

Prisma is an open-source framework utilizing Typescript, serves as an Object-Relational Mapping (ORM) tool. Its primary function is to handle database queries through the Prisma client, which contains a fully typesafe interface. This feature completely reduces the necessity for writing raw SQL queries, which can be prone to security vulnerabilities. At the core of Prisma is its schema file, written in a custom data modeling language. This schema defines all application models along with a 'datasource' attribute. This attribute details the specific database provider and outlines their connection parameters. Prisma's

compatibility extends to a wide range of popular SQL databases, such as SQLite, PostgreSQL, MySQL, or Microsoft SQL Server. The schema plays a pivotal role in both main features of Prisma framework - the generation of the Prisma client and the management of database migrations. (8)

2.4.1 Prisma client

Prisma client is a separate Typescript library that is dynamically generated based on the Prisma schema file. The library exposes all defined models with their CRUD[2] operations which are fully typed. Internally, Prisma converts these functions to SQL queries which are executed in the database specified in the datasource of the prisma schema. Notably, Prisma functions can be invoked both in server-side and client-side environments. The utilization of this library significantly enhances the developer's experience and the overall safety in database operations.

2.4.2 Prisma migrations

Prisma also includes a comprehensive support for managing database migrations. When modifications are made to the Prisma schema, executing the prisma migrate dev command prompts the library to create a migration SQL script. This script makes the transition of the database from its existing state to the updated one. Additionally, the migration process includes checks for potentially risky changes, such as adding a new required non-nullable field to an existing model. The SQL migration files are version-controlled and can be utilized for updating the database in a production environment through the prisma migrate deploy command.

2. Create, Read, Update, Delete

3 Implementation

Chapter four focuses on the system's implementation. The initial section examines the implementation of essential general features, including authorization, authentication, email client configuration, and database connectivity. Subsequent sections discuss the application portal's primary features, namely the application form, organizer's dashboard, and additional hacker functionalities. The final section explores the system's deployment on the production server.

3.1 General features

3.1.1 Authentication and Authorization

Authentication involves verifying a user's identity, whereas authorization concerns determining a user's access rights. (9) These processes are fundamental for the successful operation of a hackathon application portal. Hackers need the ability to log in and manage their application information, while organizers require authorization to access the dashboard containing all applications for a specific hackathon. Additionally, organizers with administrative access should be capable of editing the application form and managing various portal settings.

Authentication

Developing an effective authentication solution is a complex and critical task, demanding absolute precision and error-free operation. For the hackathon application portal, a pre-existing authentication library designed for Next.js applications was selected. Among the various options available, the chosen library met several key criteria: widespread use, adherence to security standards, active maintenance, customizability, extensibility, and independence from any commercial company offering cloud-based authentication solutions. NextAuth.js emerged as the optimal choice, being a popular, open-source authentication solution for Next.js, community-maintained, and ensuring full data ownership. It also supports multiple OAuth clients natively.

The library was configured with two OAuth providers - Google and GitHub - and a basic credentials provider. Google was selected for its widespread popularity and adoption. GitHub was chosen mainly due to the target audience of software developers, who are likely to have GitHub accounts and may prefer this sign-in method. The credentials provider is a simple email and password form for user authentication. User engagement with these sign-in methods will be continually monitored and evaluated, allowing for future adjustments to the provider options.

Additionally, the JWT strategy was selected for session management. NextAuth.js handles token operations automatically and offers a user-friendly API for retrieving current sessions, either on the client side (using the useSession hook) or server side (with the getServerSession function).

Finally, storing user information in a database is essential for associating records with users and maintaining their data. For OAuth logins, information is automatically stored using NextAuth.js's adapter feature. The library includes a pre-made adapter for the Prisma framework, which handles database read and write operations during OAuth sign-ins. However, there was a requirement to define the authentication related database tables - User and Account - in accordance with the NextAuth.js documentation.

For the credentials provider, user information is manually stored in the User table during the signUp server action. Additionally, email verification is required for users signing up with this method. The user's email is considered unverified until they visit a link sent via email containing a custom token. Until then, the user can not submit their application.

The configuration of NextAuth.js is located in the src/app/api/auth/[...nextauth]/route.ts file. Within this file, the selected authentication providers are specified, along with the custom callbacks that fire during the creation of JWT and session objects. These callbacks perform database queries to append extra information to both the JWT token and session object, making these data readily accessible when requesting the session from the library. Utilizing TypeScript required extending the JWT and session object type definitions with custom properties, which was done in the `types/next-auth.d.ts` file. This

extension ensures accurate typing of the return value for the session retrieval functions.

Authorization

The application portal defines three access levels: hacker, organizer, and admin. A user with a verified email is classified as an organizer if they have a corresponding entry in the Organizer table. Additionally, if the 'isAdmin' property of this record is set to true, the user is granted admin status. The record in the Organizer table is automatically created upon signup if the user's email domain is listed in a predefined whitelist (configurable via an environment variable). The whitelist for this project includes the domains @hackkosice.com and @hackslovakia.com. Importantly, the definitive access to the dashboard requires user to have verified email, based on the premise that only organizers possess emails from whitelisted domains. Admin privileges can be assigned directly in the database by authorized personnel or through the user interface by an existing admin. On top of the basic organizer rights, admin can also edit the application form, setup voting parameters, manage option lists, create new hackathons or change general setting of the hackathon.

Access as a hacker is indicated by a record in the Hacker table. A user can hold multiple Hacker records linked to different Hackathons. For the current active Hackathon, a Hacker record is created when a user first visits the main application page and is not an Organizer. Concurrently, an Application record for the respective Hacker is also created.

User authority is verified for each server action and path rendering through manual checks using the `requireOrganizerSession` or `requireHackerSession` functions. These functions validate the existence of an active session (confirmed via a callback from the NextAuth.js library). The user's session ID is then used to query the Organizer or Hacker tables in the database. Absence of a corresponding record results in an error, which is managed according to the context, either by displaying an error page or initiating a redirection.

A specialized subset of hacker rights includes team ownership. Users automatically become the owner (indicated by the 'ownerId'

field in the Team record) upon creating a team. Team owners have the authority to rename their team and remove members.

3.1.2 Database setup

In the preceding chapter, the use of Prisma ORM for database manipulation within the system was already discussed. During the implementation, SQLite was selected as the system's database due to its compactness and efficiency, characterized by a single file representation on disk. This feature enables effortless deployment and management, even in Docker container environments. SQLite is notably effective for websites with low to medium traffic, typically handling fewer than 100,000 daily visits (10). This capacity aligns well with the expected traffic for the hackathon application portal, which is unlikely to exceed these numbers.

The primary task in database setup involved creating the Prisma schema file. This process involved incorporating all models identified during the data model design phase, as well as defining the datasource and the Prisma client generator. Presented below is an excerpt from the Prisma schema, specifically the Hackathon model and both the generator and datasource definitions:

```
generator client {
  provider = "prisma-client-js"
}

datasource db {
  provider = "sqlite"
  url      = env("DATABASE_URL")
}

model Hackathon {
  id                    Int                    @id @default(
      autoincrement())
  name                  String                 @unique
  title                 String?
  description           String?
  eventStartDate        DateTime
  eventEndDate          DateTime
  applicationStartDate  DateTime
  applicationEndDate    DateTime
  hackers               Hacker[]
```

```
  sponsors                Sponsor[]
  applicationFormSteps    ApplicationFormStep[]
  voteParameters          VoteParameter[]
  createdAt               DateTime                @default(now())
  updatedAt               DateTime                @updatedAt
}
```

Furthermore, Prisma features a "seed" functionality. The seed script is executed during the initial setup of the database, potentially populating tables with necessary default data. This capability proved invaluable for filling the ApplicationStatus, FormFieldType, and TravelReimbursementRequestStatus tables with their essential values. The script can be found within the `prisma/seed.ts` file.

3.1.3 Email Communication

The capability to send notifications to users via email is essential for the proper functioning of the application portal. To address the complexities of crafting rich, styled emails with reliable delivery, the decision was made to integrate an external email service provider. The chosen provider was Brevo, as it was already in use by the Hack Kosice organization. However, since all the code for email dispatch is separated in the `src/services/emails` directory, it is easy to transition to any other email service, by modifying only to the functions in the `sendEmail.ts` file.

Brevo's features are accessed through its API. For enhanced type-safe interaction, the `swagger-typescript-api` library was utilized. This library generates types and interfaces for APIs based on the Swagger with OpenAPI specification, applicable in Brevo's case. Additionally, Brevo's API enables the use of its sophisticated templating engine. Email templates are designed within the Brevo web application, and the system only references the template IDs. The API also supports injecting dynamic content into these templates, a feature essential for easy customization of the content in the email.

Example of sending an email through the Brevo API with dynamic content:

```
await emailClient.smtp.sendTransacEmail({
  templateId: 41,
  to: [
```

```
    {
      email: "recipient@email.com",
    },
  ],
  params: {
    confirmationLink: "token",
  },
});
```

3.2 Application form

The primary objective of the hackathon application portal is to gather user applications efficiently. Consequently, significant attention has been devoted to ensuring a seamless user experience for participants during the application process. This also includes enabling organizers to edit the form with ease, while retaining all advanced features required for a comprehensive application form.

3.2.1 Application form structure

To enhance user experience, the application form is divided into multiple steps. These steps can be completed in any order and are automatically saved as the application progresses. Each step is fully customizable and has a title, an optional description, and various form fields. The form supports multiple field types, including text, textarea, radio, select, file, checkbox, and combobox. Radio, select, and combobox fields necessitate an option list for proper functionality, with options that can be dynamically defined by the organizers. Moreover, each form field can be designated as required and may include a brief description rendered as a tooltip adjacent to the field's label. An application is considered ready for submission when all required fields in every step are filled and stored in the database, provided the user has verified their email or is using an OAuth sign-in method.

Furthermore, responding to user feedback during testing, the application form also includes "custom visibility" form fields. These fields are linked to another field and become visible only if that field has a specific value entered. Currently, the system supports linking these fields only to selection-based fields (radio, select, or combobox). This

feature can help to significantly enhance the user experience. For instance, the form can initially check if the user has previously attended a hackathon using a simple yes/no radio input. Subsequently, if the user selects 'yes', a textarea appears prompting them to detail their experience at prior hackathons.

All data necessary for rendering the application form is stored in the ApplicationFormStep and FormField tables, along with the OptionList and FormFieldVisibilityRule tables for certain fields. When a user saves their data, an entry is created in the ApplicationFormFieldValue, linking to the corresponding form field.

3.2.2 Editing application form

The application form can be edited in the Organizer's dashboard if the current user has admin privileges. This functionality includes the creation of new steps, modification of their titles and descriptions, and the ability to delete or reorder them. Within each step, administrators can manage form fields by adding, editing, or deleting them, as well as rearranging their order. A live preview of the form is displayed on the right side of the screen during the editing process, providing immediate feedback on changes made as seen in Figure 10

The process of creating and editing steps is conducted through a dialog box, which is detailed in Figure 11. Additionally, the dialog includes an option to select a checkbox labeled "Should be shown in application list and detail." This feature is particularly useful in the application list display within the main dashboard area. It ensures that only fields marked as true under this option are visible in the applications table, effectively filtering out non-essential fields such as consent checkboxes. This functionality enhances the clarity and usability of the application list by focusing on important information.

3.2.3 Rendering form and validation

In the hacker application interface (visible in Figure 12), the steps of the application form are presented as interactive cards. Each card visually indicates the completion status of a step. Selecting a card opens a dedicated page displaying the form fields for that step. These pages utilize the Form component from shadcn/ui, which incorporates

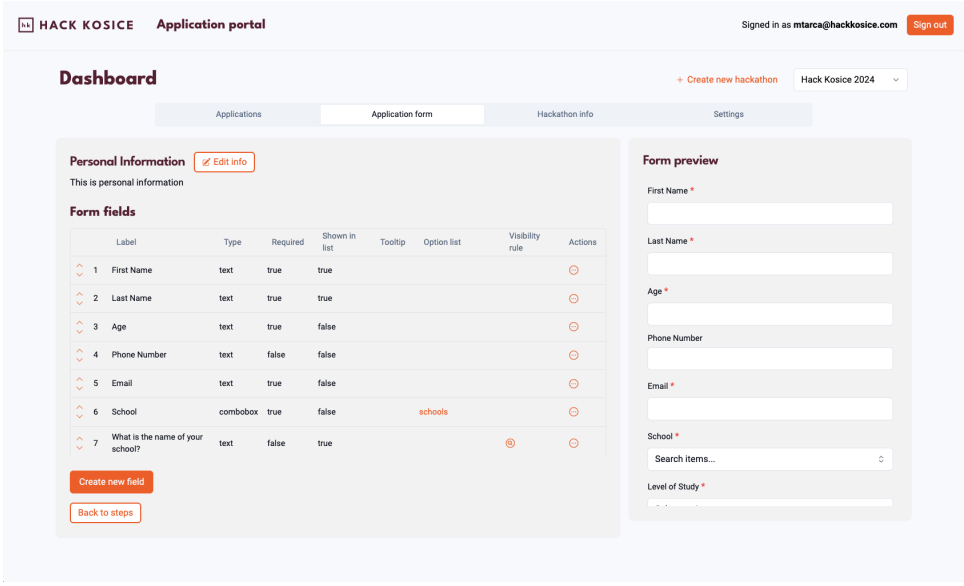

Figure 10: The admin's UI for editing a step of the application form

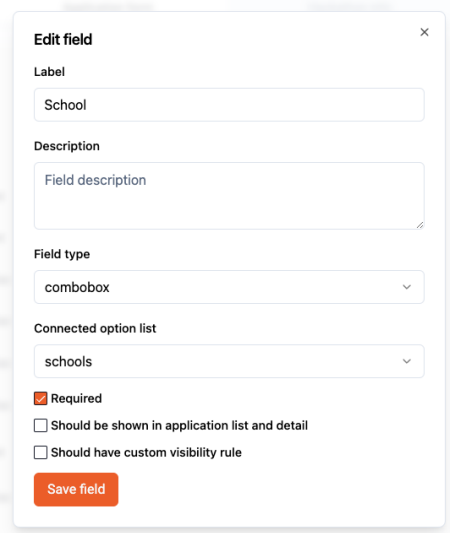

Figure 11: The dialog with a form to edit a given form field

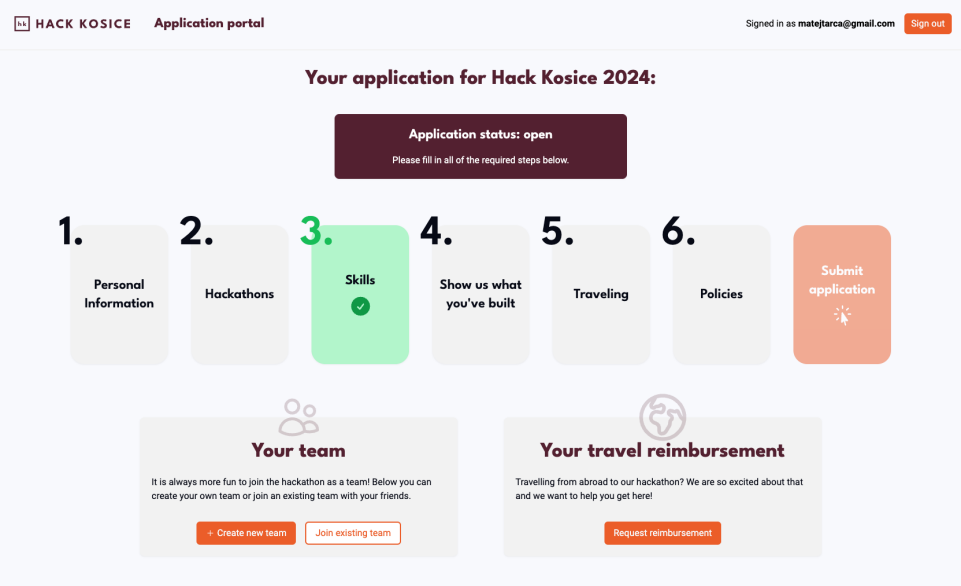

Figure 12: The hacker's UI with application steps

the react-hook-form library to achieve the desired client-side form interactivity possible in React. This interactivity encompasses dynamic checks for values in fields targeted by "custom visibility" rules and real-time client-side validation, facilitated by the zod library. Notably, the zod validator is dynamically generated based on form field data received from the server. The rendering of these forms is handled by the FormRenderer component, whose adaptability is demonstrated by its application in both the hacker application interface and the organizer dashboard, as previously discussed. The submit application button at the right side of the screen is disabled until the check for completeness of all steps does not pass.

3.2.4 File upload

The file upload field is a special type of form field which has a separate handling than other fields. During the implementation a decision was made to use cloud storage service instead of storing the uploaded files on the same server as the application portal, leading to enhanced security and improved performance. Cloudflare R2 storage was selected

for its compatibility with the S3 API of Amazon's widely-used storage service, facilitating easier migration to other S3 API-compatible providers while offering cost benefits over the traditional Amazon S3. Cloudflare's free tier provides substantial resources, including 10GB of monthly storage, along with a allowance of 1 million A class (state-mutating) and 10 million B class (state-reading) operation requests, without egress bandwidth charges.

The data storage system is implemented using the "@aws-sdk/client-s3" library, which offers a native TypeScript SDK for S3 API interactions. A key feature utilized are the presigned URLs, which allows clients to upload or download files directly from the bucket, bypassing the server. This mechanism includes the server generating a URL that restricts client operations to specific objects and passing it down to client. This approach optimizes performance while maintaining robust security. (11) To facilitate portability to a different file upload solution, all file upload functionalities are located in the `src/services/fileUpload` folder.

Upon successful file upload to the bucket, the server records this event in the File table of the database, which includes a handler for later file retrieval. This record is associated with the ApplicationFormFieldValue corresponding to the related file upload form field.

3.2.5 Support for unsigned users

One of the important functional requirements for the portal was that hackers can start filling up the form without creating an account, while being able to save their progress. However, saving progress for unregistered users differs from registered users, as their inputs cannot be linked to a specific account in the database. To address this, the implementation phase explored various options, ultimately selecting the use of the browser's localStorage API to store application data. This approach restricts data storage to a single device, an essential limitation for this functionality.

When a user saves a form step, the system checks for user authentication. If the user is not signed in, the data, formatted as a JSON string, is saved to local storage instead of being sent to the server. Functions for managing local storage are centralized in the `src/services/helpers/localData` directory. The data in local stor-

age are continuously updated as the unregistered user interacts with the form.

However, users must create an account to submit the application. This requirement is prominently communicated through a red banner at the top of the application form, urging users to register at their earliest convenience. Once a user logs in and accesses the application page, the locally stored data is synchronized with the server. This synchronization involves sending data to the server, which then creates records in the `ApplicationFormFieldValue` table, provided no existing records are present (this measure prevents accidental data overwriting). Upon successful data transfer to the database, the local storage is cleared. This process provides a seamless transition from filling the application locally without an account to continouing as a registered hacker.

Unfortunately, one additional limitation is the inability to upload files through the form without an account, due to the specific nature of file uploads and storage.

3.3 Organizer's dashboard

The previous section highlighted the form editing capabilities of the organizer dashboard. Beyond this, the dashboard offers a comprehensive suite of tools essential for organizers and administrators. These include a detailed application list complemented by statistical analysis, an application judging interface, a system for reviewing travel reimbursement requests, management of option lists, functionalities for creating new hackathons, and general hackathon settings management. Since most of these features predominantly involve basic CRUD operations, this section will primarily concentrate on the dashboard's overall technological structure and the application judging feature.

3.3.1 Dashboard structure

The implementation of the organizer's dashboard strongly demonstrates the capabilities of Next.js, especially its nested layouts feature. The dashboard's URL always begins with /dashboard. At this route, the App router folder structure includes a `layout.tsx` file, which

presents the dashboard title and a selector component for hackathons currently available in the database. Upon selecting a hackathon, users are redirected to /dashboard/[hackathonId]/. Importantly, since the previous `layout.tsx` file resides in the parent directory of the whole /dashboard route, both the selector and the title remain visible even when visiting the nested route. In the directory `/app/dashboard/[hackathonId]` is an additional `layout.tsx` file, which houses a tab menu for navigating the different sections of the dashboard. Clicking on a tab redirects the user to a specific feature URL, such as /dashboard/[hackathonId]/form-editor, where the relevant `page.tsx` file for that feature is located. This setup provides an efficient abstraction for the navigation elements that are consistently present on every page. Moreover, the use of Next.js's Link component and router for client-side navigation ensures smooth transitions within the dashboard, maintaining a seamless user experience even as the actual page URL changes. The layout of the dashboard structure is visualized in Figure 13.

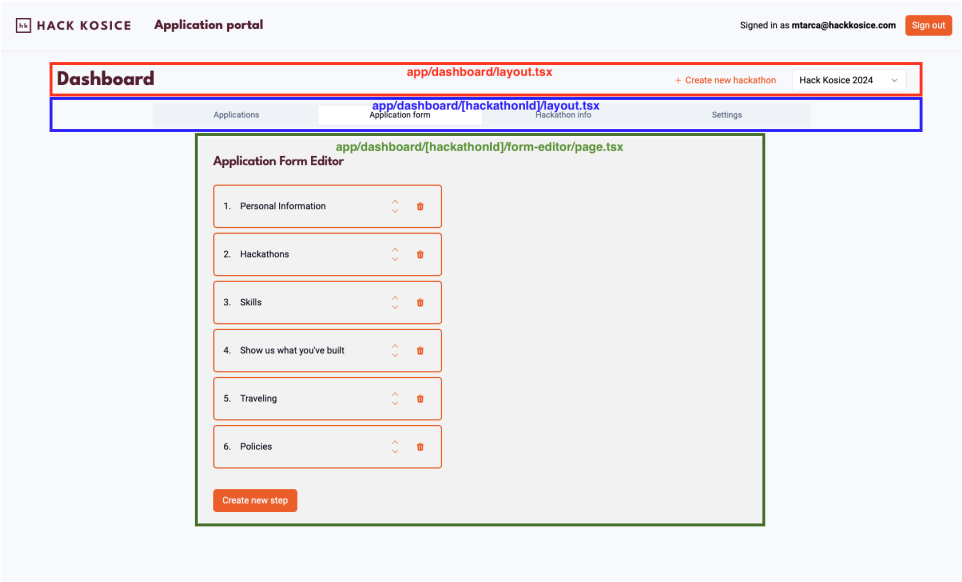

Figure 13: The structure of the organizer's dashboard

3.3.2 Application judging

The judging process in the hackathon application portal is designed to assign scores to applications, thereby facilitating the selection of top candidates. This process begins with the portal administrator defining voting parameters, each characterized by a title, a range for minimum and maximum values, and a weight. Organizers can then access the judging section to vote on applications. Upon entering the judging page the system automatically assigns an application with the fewest votes to the organizer to ensure even distribution of votes. This assignment is recorded in the database to maintain consistency across multiple page visits. Organizers then review applications via the user interface and cast their votes within the predefined voting parameter ranges.

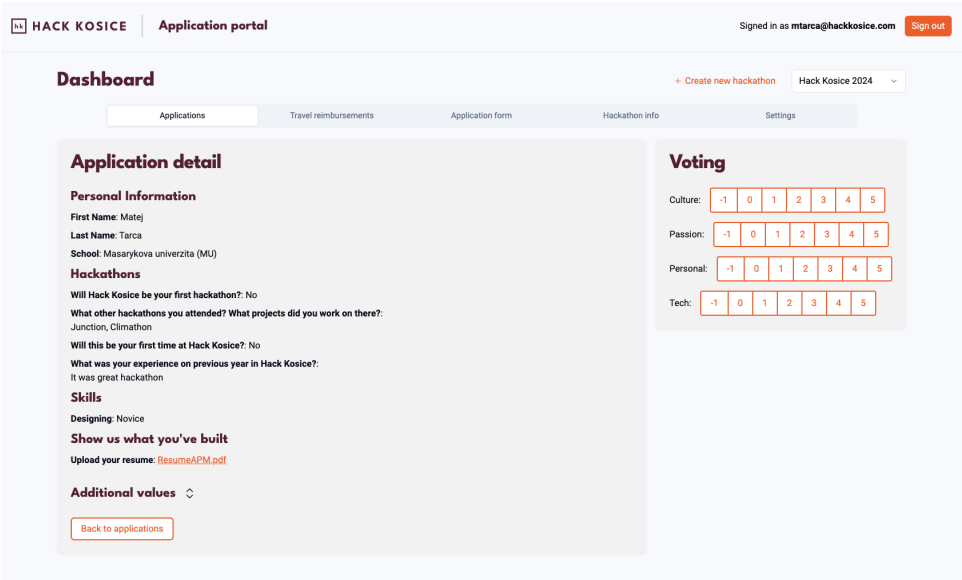

Figure 14: The page for reviewing an assigned application.

The scoring mechanism involves first calculating the score given by each organizer as a weighted sum of the voting parameter values. The final score of the application is then calculated as an average of all scores of organizers for given application. Given O_A as a set of all organizer that voted for application A with o_v being the value of

organizer's vote and given V as a set of all vote parameters with v_w being the weight of the parameter the final score is calculated as:

$$score_A = \frac{\sum_{o \in O_A} \sum_{v \in V} v_w . o_v}{|O_A|}$$

Additionally, the score is visually differentiated in the UI by color to indicate its relative significance, based on the number of votes received. Once an application achieves a sufficiently high and relevant score, organizers can invite the hacker. This invitation is conveyed through an email, requiring the hacker to confirm attendance through their portal account.

3.4 Check-in

One of the functional requirements for the application portal was being able to check-in hackers at the event. This feature encompasses two distinct interfaces: one displaying a QR code for participants, and another including a QR code scanner or manual input option of the check-in code for organizers. The QR code is generated using the "qrcode.react" library and the "@yudiel/react-qr-scanner" library supplies the QR scanner component.

Each QR code encapsulates a unique check-in code, created on the server. This code is formed by encrypting the string "hackathon-{hackathonId}-hacker-{hackerId}", with dynamic values replaced appropriately, using the symmetric AES-256-CBC cipher. The cipher utilizes a 32-byte key, set as a server environment variable, and a 16-byte initialization vector. The initialization vector, converted to hexadecimal, is appended to the check-in code, enabling later decryption.

Upon scanning by an organizer, the code is transmitted to the server for decryption. The decrypted information undergoes validation to confirm its format and the accuracy of embedded values. The security of this process is upheld by the AES-256-CBC cipher; correct format in the decrypted data implies that it was indeed generated by the server, given its exclusive access to the secret key. Notably, check-in codes are issued solely for confirmed applications.

In instances where QR scanning is impractical, participants can provide their email address, also displayed on the check-in screen, to

the organizer. This allows the organizer to retrieve the participant's information manually and complete the check-in process.

3.5 Other features for hackers

On top of being able to submit the application for a hackathon, hackers can use the application portal to form and join teams as well as request travel reimbursement.

3.5.1 Team creation

In the "Team" section, hackers have the option to form their own teams with their own unique names. Upon creation of a team, the current user becomes the owner of the team. This role grants the ability to rename the team and manage its membership, including the removal of members if necessary. Each newly formed team is assigned a randomly generated 12-character code, which serves as a private access key for other hackers wishing to join. Moreover, all team members have visibility of each other's application statuses within the team interface. The maximum team size is determined by hackathon admins and can be modified to suit the event's requirements. Allowing team formation during the application phase can significantly help organizers in selecting participants, with a preference for complete teams, and in organizing seating arrangements for onsite events.

3.5.2 Travel reimbursement requests

The final, yet crucial feature for participants is the travel reimbursement request process. Given that hackathon participants, typically students, often travel from various global locations, hackathons commonly offer financial assistance for travel. Participants traveling from foreign countries can apply for travel reimbursement through the portal by indicating their country of departure. When initiating a reimbursement request, a prompt appears, displaying details set by the organizers, including eligibility criteria and the potential reimbursement amount. Once submitted, organizers can review these requests in the dashboard, deciding on approval or rejection and manually determining the reimbursed amount. Approved requests then move

to the status "approved and waiting for documents." Before payment, participants must upload evidence of travel and their financial details (such as IBAN) into the portal. The travel document upload process is implemented the same way as in the application form, utilizing Cloudflare R2 storage. Organizers then reassess the uploaded documents and, upon verification, can designate them as ready for payment. Once the payment is processed, the request can be marked as "paid," thereby concluding the process.

3.6 GDPR compliance

The Application Portal for Hack Kosice, operating primarily within Europe, must adhere to the European Union's General Data Protection Regulation (GDPR). Compliance with GDPR involves several critical aspects: storing only essential data, maintaining transparency about data processing, and obtaining user consent that is both informed and freely given. (12) This information is detailed in the site's Privacy and Cookie policies.

The portal processes various personal data essential for the application process, from which users cannot opt out as they are essential for the primary functionality of the system (but users can still request to have them deleted after the hackathon via the contact address found in Privacy policy). Additionally, as outlined in the section Monitoring, the portal tracks user interactions to enhance user experience and analyze performance. This tracking involves storing cookies in the user's browser. Under GDPR, explicit user consent is required for such actions.

To ensure GDPR compliance, the Privacy and Cookie policy pages were created to provide comprehensive information on data processing. Furthermore, the portal displays a Cookie Consent banner before initiating any tracking. This banner allows users to opt in or out of performance and analytics cookies. If a user rejects tracking, it is not initiated at all. This functionality is managed by the `CookieConsentDialog` component and communicated to the rest of the app through the `CookieConsentContext`. However, certain cookies, like those set by NextAuth for user authentication and those remembering cookie preferences, are considered essential and cannot be opted out of.

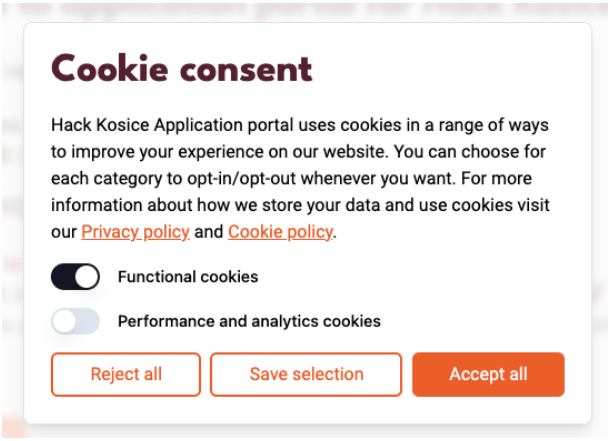

Figure 15: The cookie consent modal shown upon the first entry to the application portal.

3.7 Deployment

The traditional method for deploying Next.js applications often involves using the Vercel serverless cloud platform. However, for the deployment at Hack Kosice, a custom approach was adopted, utilizing a Linux-based Virtual Private Server (VPS). This choice was driven by the pre-existing use of the VPS by the Hack Kosice organization, offering a more cost-effective solution compared to Vercel. Additionally, this method afforded greater control over the deployment process.

Deployment of a Next.js application begins with building the application using the `next build` command, followed by starting the Node server from the generated `server.js` file. This process is managed on the VPS using PM2, a widely-used daemon process manager for Node.js. PM2 ensures continuous operation of the application in the background and offers valuable features such as log management and process reloading.

Application connection to the outside world is facilitated through the Nginx web server, which functions as a reverse proxy for the Next.js Node server running on localhost on the VPS. It is crucial to properly configure Nginx to serve both as the reverse proxy and to ensure public accessibility of the generated Next.js static files. The Nginx con-

figuration is managed through a file located in the `sites-available` directory.

Example of the Nginx config for the Next.js application follows:

```
server {
    server_name apply.hackkosice.com;

    error_log /var/log/nginx/error_log_portal;
    access_log /var/log/nginx/access_log_portal;

    location = /favicon.ico { access_log off; log_not_found
        off; }

    location /_next {
        alias /home/hackkosice/HackPortal/.next;
    }

    location / {
      root /home/hackkosice/HackPortal/public;
      try_files $uri @proxy;
    }

    location @proxy {
        proxy_pass http://localhost:3005;
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
    }
}
```

To simplify the redeployment process of the application, a bash script named `deployLocally.sh` was developed, located in the `scripts` directory. This script automates several key tasks: it reinstalls dependencies, regenerates the Prisma client, applies database migrations, and builds the Next.js application. This streamlining of the redeployment procedure ensures efficiency and consistency in the application's maintenance.

4 Quality assurance and monitoring

4.1 Quality assurance

During the development of the hackathon application portal, a strong emphasis was placed on maintaining a high software quality. While meeting the functional requirements is crucial for user satisfaction, it is essential not to overlook other aspects of the software quality such as maintainability and performance. (13) The quality assurance tools that were implemented for the hackathon application portal primarily focused on assessing functionality and code maintainability. To ensure ongoing software quality throughout the application portal's lifecycle, the development employed methods of automated and continuous quality assurance, which involved the creation of automated tests, the configuration of static analysis tools, and their seamless integration into the CI/CD pipeline via Github Actions.

4.1.1 Automated testing

Testing plays a crucial role in software development and quality assurance. Its primary purpose is to detect and address system flaws before they impact end-users. Typically, software testing involves running the program with predefined inputs and verifying whether it produces the expected outputs or behaves correctly. (14) Tests can be either manual or automated. In this section, we focus on describing the automated tests developed for the hackathon application portal at different granularities: unit, and system (end-to-end) tests.

Unit tests

Unit tests are part of the initial testing phase of the system. These tests evaluate individual components in isolation, aiming to replicate or mock external dependencies while concentrating on the component's internal behavior. (1, p. 211)

For unit testing, the Jest testing framework alongside the React Testing Library was utilized. This combination is a popular choice for unit testing JavaScript web applications built with React. It offers a comprehensive API for mocking external dependencies, accessing

rendered components, and interacting with them. Jest also permits the monitoring of test coverage and setting coverage thresholds, ensuring that every component is thoroughly tested.

Below is an example of a test case in the unit test for the `FormRenderer.tsx` component, showcasing the features of Jest and React testing library mentioned above.

"src/scenes/ApplicationFormStep/components/__tests__/FormRenderer.test.tsx"

```
    it("should render simple form with a text field", async
        () => {
    render(
      <FormRenderer
        formFields={[textFormFieldObject]}
        onSubmit={onSubmitMock}
        actionButtons={actionButtons}
      />
    );

    const input = screen.getByLabelText("Name");
    const button = screen.getByRole("button", { name: "Save"
        });
    expect(input).toBeVisible();
    expect(button).toBeVisible();

    await act(() => userEvent.click(button));
    expect(onSubmitMock).not.toHaveBeenCalled();
    await waitFor(() =>
      expect(screen.getByText("This field is required")).
          toBeVisible()
    );

    await act(() => userEvent.type(input, "John"));
    expect(input).toHaveValue("John");

    await act(() => userEvent.click(button));
    await waitFor(() => expect(onSubmitMock).
        toHaveBeenCalledTimes(1));
  });
```

All unit tests are present next to the components or services they are texting inside a special *__tests__* folder and follow a naming structure `NameOfTheFile.test.ts(x)`. To execute all unit tests, there is a custom

npm script invoked by calling either `npm run test:jest` or `npm run test:jest:coverage`.

End-to-end tests

The second category of testing applied to the application portal consists of system tests, which evaluate the application in a comprehensive manner that mirrors its production environment. (1, p. 219) Within this category, end-to-end tests specifically examine the user's complete experience, tracking their journey from start to finish within the application. These tests are designed to assess the application from the user's viewpoint, ensuring a seamless operational experience.

End-to-end tests were developed using the Playwright library - a versatile framework for testing web applications that accesses them through a simulated browser environment. Playwright supports multiple browser rendering engines, including Chromium, Firefox, and Webkit, and offers a cross-language API. (15) Given the Typescript code-base, the Playwright tests are written using Typescript as well.

As mentioned previously, end-to-end tests require an environment that replicates the production setting, including a database that mirrors the live setup. Creating such an environment poses the challenge of facilitating local development and debugging of the tests. This problem is addressed by employing a Docker container tailored for test execution and a bash script designed to aid interaction with the container for development purposes.

The Docker image is configured with Ubuntu to meet the specific demands of Playwright for browser engine installation. The responsibilities of the container include installing Node.js and Playwright's dependencies, building the application, and setting up the database. A bash script located in the project's `scripts` folder called `runE2ELocally.sh` manages the Docker container operations. Running this script without any arguments triggers the startup of the Docker container, if it is not active, and proceeds to execute all end-to-end tests in a headless mode, thereby streamlining the testing process. However, the script's behaviour can be modified by following properties:

- –build - if passed the image and container is rebuilt from the `e2e.Dockerfile` file before starting the container

- –copy-tests - if passed all tests from the `e2e` folder and the `playwright.config.ts` are copied to the running container before executing them

- –ui - if passed the Playwright's UI 16 server will start in the container instead of running the tests in the headless mode. This UI allows developer to execute the tests manually and observe their runnings and errors.

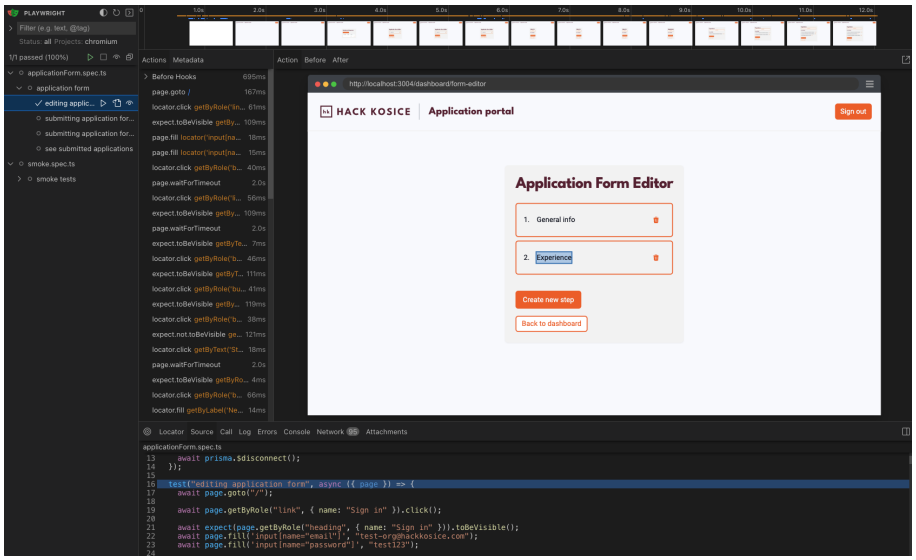

Figure 16: Example of a test result in Playwright's UI

Addressing the challenge of database management during end-to-end testing posed another significant hurdle. It is imperative to start each test suite with the database in a specific, known state to ensure consistency and to prevent any overlap of data from concurrently running test suites. A tailored script was implemented to clean the database and preload it with necessary records before the start of every test suite, using the 'beforeAll' hook for initial setup. To eliminate simultaneous database access, the Playwright configuration was modified to restrict the worker count to just one, effectively queuing test suite executions to run one after the other.

All the e2e tests reside in the `e2e` folder in the codebase and can be executed by running the bash script mentioned above.

4.1.2 Static analysis

Static analysis is a process of quality assurance done by verifying correctness of the software without its execution. It works by identifying common error patterns by parsing the program's source. (1, p. 395) The primary component of automated static analysis in our project is the ESlint library, a standard tool in modern TypeScript and JavaScript ecosystems. ESlint's configuration is defined in the ".eslintrc.json" file, where multiple plugins and rulesets specific to the project are established. Few noteworthy ones are:

- @typescript-eslint/recommended
- next/core-web-vitals
- prettier/recommended

Furthermore, the Typescript compiler itself can be viewed as a type of static analyzer too. When the compiler is run with the "–noEmit" flag, it will not create any compiled JavaScript files but will report all the type errors it finds in the code.

Both of these analyzer can be run by executing commands `npm run lint` and `npm run types`.

4.1.3 Github Actions

To ensure that the quality assurance measures mentioned in the previous sections are actively employed during the development process they are included in the Github pipeline via the Github Actions feature. The project's GitHub repository is configured to prohibit direct pushes to the main branch, allowing only the creation of pull requests. Furthermore, these pull requests cannot be merged until the pipeline successfully completes. This rule can only be bypassed by users with administrative permissions in exceptional circumstances.

The GitHub pipeline is comprised of several actions, defined using YAML files within the `.github/workflows` folder. These actions include static checks, which involve running ESLint and TypeScript;

unit testing, conducted using Jest in coverage mode; and end-to-end (E2E) tests, executed using Playwright in both desktop and mobile environments. This setup ensures that any code merged into the main branch is thoroughly tested and unlikely to disrupt the system upon deployment.

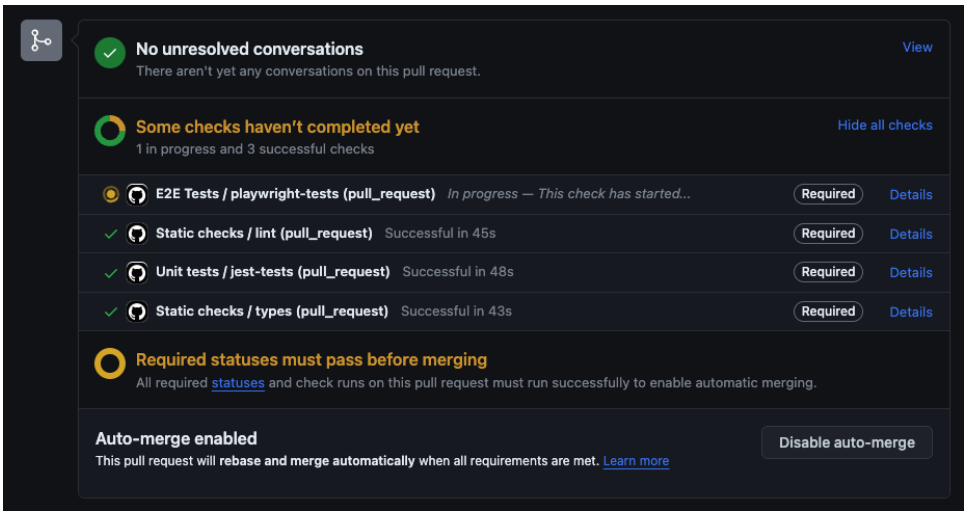

Figure 17: Example of a Github pipeline run on a pull request.

4.2 Monitoring

Effective monitoring of system performance and health in a production environment is critical for ensuring a seamless user experience. Additionally, monitoring user interactions on the site is valuable for gaining insights into actual user behavior and needs. Consequently, incorporating monitoring capabilities was a fundamental aspect of the Application Portal's development. Sentry and Datadog, two popular monitoring solutions, were selected for this purpose.

These platforms offer robust monitoring capabilities, but typically at a considerable cost. However, the Github Education pack enabled their integration into our system at no cost. Nevertheless, the application portal's logging implementation is designed for flexibility, allowing easy transition to other, more cost-effective providers if necessary.

This is facilitated by a global useLog hook, providing a custom API for logging that can transmit logs to any chosen provider.

4.2.1 Sentry

Sentry is a monitoring platform specializing in identifying and understanding uncaught errors within the application. It includes a native @sentry/nextjs library for error tracking in Next.js applications on both client and server sides. Server-side initialization occurs in the `sentry.server.config.ts` file, while client-side initialization takes place in the CookieConsentDialog after acceptance of analytics cookies (required by Sentry). Additionally, Sentry offers manual error reporting through its Sentry.captureException function, which is widely used throughout the system.

Importantly, Sentry can also track core web vitals on the client side, such as Largest Contentful Paint, First Contentful Paint, First Input Delay, Cumulative Layout Shift, and Time To First Byte. These metrics offer valuable insights into the end-user's performance and experience.

4.2.2 Datadog

Datadog provides a comprehensive monitoring solution for a lot of different types of systems and applications. The features from Datadog which are employed in our system include Real User Monitoring and Infrastructure Monitoring. The latter is conducted via the Datadog agent installed on the Virtual Private Server (VPS) hosting the system. This agent automatically gathers and reports server metrics like CPU usage, memory usage, disk operations, and network activities, along with logs from the Nginx web server.

Real User Monitoring is achieved through the @datadog/browser-logs library. Following user consent for cookies and tracking, the library is initialized, allowing the use of the datadogLogs.logger.log function to transmit logs to Datadog. The library automatically appends user data such as current URL, IP address, and user agent to the logs. The Datadog library also features an easy-to-use API for setting global user information, enabling attachment of session details like email and user ID to all logs for a given session.

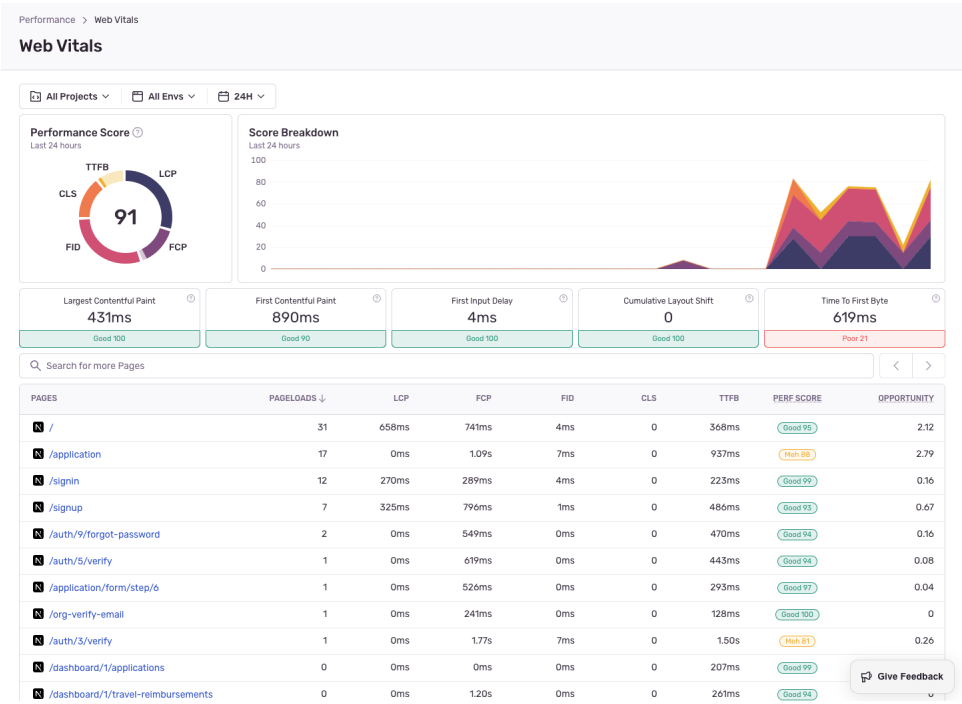

Figure 18: Example of the web vitals analysis visible in the Sentry web interface.

The client-side Datadog logs are manually triggered in response to specific user interactions, such as page visits or actions performed on the page, including button clicks. Furthermore, each log entry contains an additional information relevant to the action. For instance, when a user visits the application form step page, the log captures and records the page's title.

Datadog's web interface features a comprehensive log overview and allows for the creation of customizable dashboards that display specific live data views. Additionally, it supports the development of monitors with alerts, which can track particular types of logs or infrastructure data and notify developers when unexpected values are detected.

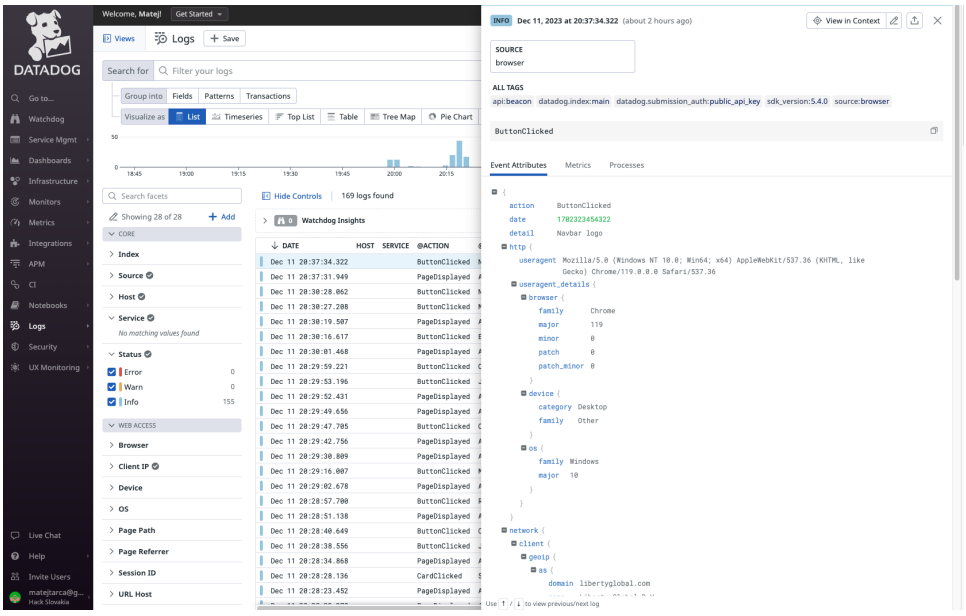

Figure 19: Example of a log entry in the Datadog web interface.

5 User Testing

This chapter outlines the multi-phase user testing process applied to the application portal, complementing the automated testing outlined in the previous chapter. The objective was to minimize system issues and ensure compliance with stakeholders' requirements. The portal was continuously deployed in a production-like environment, enabling hackathon organizers to conduct manual tests.

The user testing unfolded in three phases, commencing in October 2023 and recurring monthly until the portal's release in December 2023. The portal was tested by seven individuals, consisting of six members from the Hack Kosice organizing team and one external participant. These testers did not receive a detailed step-by-step guidance, but rather were provided with high-level objectives, including tasks like defining the application form, submitting applications, and creating teams. Feedback and issues were systematically documented using the Notion note-taking tool, a resource already utilized by the Hack Kosice team. This documentation included the issue's type and severity, detailed descriptions of each issue, steps for replication, and, where relevant, screenshots. Additionally, Notion also served to accumulate suggestions for potential feature enhancements.

Manual testing revealed several user experience challenges. These included the positioning of back buttons, the readability of the application review screen, the creation of option lists in new form fields, excessively wide tooltips or the impracticality of select components with a lot of options. User feedback also led to the incorporation of several beneficial features. These enhancements included the ability to duplicate form fields, sort applications by scores or customize the visibility of specific fields. Furthermore, this process was instrumental in identifying numerous bugs, such as issues with reordering form steps, visibility of teams in application lists, creation of circular custom visibility dependencies, and various authentication problems.

The user testing phase also provided an opportunity to evaluate the portal's monitoring capabilities. Data collected during this period aided in the development of the custom dashboards in Datadog. Continuous monitoring of real user interactions and issues will per-

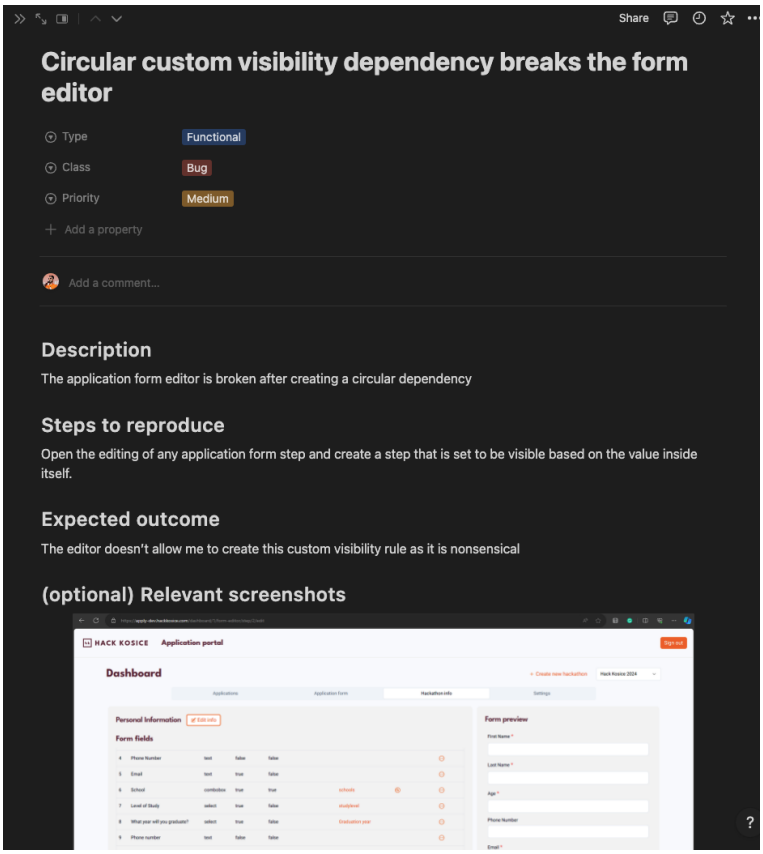

Figure 20: Example of a bug reported in the Notion tool.

sist throughout the system's lifecycle, with an emphasis on increased traffic after the post-production release.

6 Conclusion and future work

The thesis focused on developing an application portal for hackathons, specifically designed to streamline the application process for participants. The project began with an analysis of existing solutions, followed by the design of software requirements, data models, and use case analysis through sequence diagrams. The implementation phase encountered several challenges, particularly in working with the latest version of Next.js (which was released during the writing of the thesis) and establishing effective end-to-end tests in a simulated browser environment.

Despite these challenges, the project was successfully completed and the system was deployed for practical use at the Hack Kosice 2024 hackathon. The developed system not only handles application submissions but also facilitates team creation and travel reimbursement requests. Its user-friendly interface allows organizers to modify application forms and information with ease. Furthermore, the system includes features for reviewing and voting on applications and inviting top candidates. On top of that, the development was done in a way that ensured GDPR compliance, and provided comprehensive user and performance monitoring.

Looking ahead, there are multiple opportunities for further enhancements. The system could be expanded to include features for in-event engagement, such as automated seating arrangements for teams. Integration with a payment provider could automate the distribution of travel reimbursements. Additionally, leveraging the full server-less and edge deployment capabilities of the Next.js framework could further optimize the system's deployment.

In summary, this thesis has successfully created a valuable tool for managing hackathon applications, greatly benefiting the Hack Kosice team. The system's effective blend of technology and user-friendly features sets a new standard for future events. While there is room for further enhancements, the project's achievements mark a significant step forward in simplifying and improving hackathon organization.

Bibliography

1. SOMMERVILLE, Ian. *Software Engineering (9th Edition)*. Pearson Education, Inc., 2010. ISBN 0137035152.
2. CHUNG, Lawrence; PRADO LEITE, Julio Cesar Sampaio do. On non-functional requirements in software engineering. *Conceptual modeling: Foundations and applications: Essays in honor of john mylopoulos*. 2009, pp. 363–379.
3. COOK, Steve; BOCK, Conrad; RIVETT, Pete; RUTT, Tom; SEIDEWITZ, Ed; SELIC, Bran; TOLBERT, Doug. *Unified Modeling Language (UML) Version 2.5.1*. 2017-12. Standard. Object Management Group (OMG). Available also from: `https://www.omg.org/spec/UML/2.5.1`.
4. LI, Qing; CHEN, Yu-Liu. Entity-relationship diagram. In: *Modeling and analysis of enterprise and information systems*. Springer, 2009, pp. 125–139.
5. META. *Start a New React Project*. 2023. Available also from: `https://react.dev/learn/start-a-new-react-project`. Accessed: November 5, 2023.
6. MARKBÅGE, Sebastian. *How to Think About Security in Next.js*. 2023. Available also from: `https://nextjs.org/blog/security-nextjs-server-components-actions`. Accessed: December 6, 2023.
7. TAILWIND LABS INC. *Utility-First Fundamentals*. 2023. Available also from: `https://tailwindcss.com/docs/utility-first`. Accessed: November 6, 2023.
8. *What is Prisma?* Prisma, 2023. Available also from: `https://www.prisma.io/docs/concepts/overview/what-is-prisma`. Accessed: December 6, 2023.
9. METZ, C. AAA protocols: authentication, authorization, and accounting for the Internet. *IEEE Internet Computing*. 1999, vol. 3, no. 6, pp. 75–79. Available from DOI: `10.1109/4236.807015`.
10. *When To Use SQLite*. SQLite, 2023. Available also from: `https://www2.sqlite.org/cvstrac/wiki?p=WhenToUseSqlite`. Accessed: December 6, 2023.

11. CLOUDFLARE. *Presigned URLs*. 2023. Available also from: `https://developers.cloudflare.com/r2/api/s3/presigned-urls/`. Accessed: December 7, 2023.

12. WOLFORD, Ben. *What is GDPR, the EU's new data protection law?* Proton Technologies AG, 2023. Available also from: `https://gdpr.eu/what-is-gdpr/`. Accessed: December 17, 2023.

13. O'REGAN, Gerard. *Introduction to software quality*. Springer, 2014.

14. YOGESH, Singh. *Software Testing.* Cambridge University Press, 2011. ISBN 9781107012967. Available also from: `https://search.ebscohost.com/login.aspx?direct=true&db=e000xww&AN=465756`.

15. PLAYWRIGHT TEAM. *Installation* [`https://playwright.dev/docs/intro`]. 2023. Accessed: November 7, 2023.