Silesian
University
of Technology

FINAL PROJECT

Interactive security training platform based on CTF concept

Krzysztof Marek DZIEMBAŁA
Student identification number: 293671

Programme: Control, Electronic, and Information Engineering
Specialisation: Informatics

SUPERVISOR

dr inż. Dominik Samociuk

DEPARTMENT OF COMPUTER NETWORKS AND SYSTEMS

Faculty of Automatic Control, Electronics and Computer Science

Gliwice 2023

Thesis title

Interactive security training platform based on CTF concept

Abstract

(Thesis abstract – to be copied into an appropriate field during an electronic submission – in English.)

Keywords

(2-5 keywords, separated with commas)

Tytuł pracy

Interaktywna platforma do nauki bezpieczeństwa wykorzystująca zadania typu CTF

Streszczenie

(Thesis abstract – to be copied into an appropriate field during an electronic submission – in Polish.)

Słowa kluczowe

(2-5 keywords, separated by commas, in Polish)

Contents

Chapter 1

Introduction

Latest reports show that on average people spend over 6.5 hours daily using the internet [5]. Computers and internet play a huge role in today's world. Unfortunately many services and applications are not secure. Consequences of vulnerabilities may be very different and vary from almost negligible to extremely severe. Some examples include service inaccessibility, data leaks and network takeovers. Sometimes seemingly harmless issues may be exploited in a series of attacks and cause much bigger problems. Like in many areas the best solution to these issues is prevention. Although code review and testing may help with problem detection and fixing, it would be much better if these issues were not created at the first place. According to Michał Sajdak the source of many vulnerabilities and one of the most important problems in security is the lack of knowledge about security issues [13]. A developer, especially if not experienced, may not be aware of potential issues with their code. Education about vulnerabilities may help programmers look on their code from a different perspective therefore making it more secure.

The aim of the thesis is preparing a system useful for teaching cybersecurity. The system is intended to utilize Capture The Flag (CTF) challenges for demonstrating the vulnerabilities. This is a popular challenge type requiring the user to exploit a security issue and find a hidden flag.

The thesis consists of the planning and preparation of the project, creation of architecture capable of running challenges, designing and implementing user interface and a management panel. Additionally, existing solutions and ideas related to the problem were analysed before planning the system. The following chapters are included in this thesis:

- **Problem analysis** - analysis of the problem and available literature, description of existing solutions,

- **Requirements and tools** - formulation of functional and non-functional requirements, use case analysis and selection of third-party tools and technologies, the design and planning process

- **External specification** - specification of hardware and software requirements, installation and setup process, user and administrator manuals, security analysis and recommendations,

- **Internal specification** - description of the system architecture, code organization and sequence diagrams presenting interactions between major components during usage,

- **Verification and validation** - verification of the requirements, general system testing and presentation of bugs which were found and fixed.

Chapter 2

Problem analysis

Traditional capture the flag game requires the player to take a flag or a similar object from opponent base and bring it back to their own. Similarly, in cybersecurity CTF the objective is obtaining a flag from the *opponent*. There are two formats of CTFs - attack-defence and jeopardy. In the first type competing teams have to both attack other teams and protect their own system. The latter requires participants to *steal* the flags from organiser-prepared challenges. The tools developed for this thesis are supposed to help programmers understand security issues, therefore only jeopardy-style are considered.

2.1 Literature research

Using CTF-style challenges for cybersecurity education is not a new idea. Gamification is helpful for increasing user engagement and works in cybersecurity too [8]. An analysis of PicoCTF 2013 competition by Chapman et al. [2] shows that properly prepared CTFs are suitable even for high school students with no security experience. According to the survey answers the teachers observed that the students put more effort into the CTF than usually during lessons.

2.1.1 CTF as university laboratories

CTF challenges are already used during cybersecurity laboratories on some universities. The topic has been approached in various ways by the instructors.

Karagiannis and Magkos [3] use CTFd for managing the challenges. Challenges were divided into subsequent steps to guide the participants through the process and present the knowledge better. VulnHub VMs were used as the challenges and most of them were running locally on students' computers.

In [6] Ksiezopolski et al. present their web application security course based on CTF. Students connect to the infrastructure using individual VPN configurations. A challenge

must be started by the student using a web portal to make it available on the VPN network. This solution allows *cheap* scaling by starting challenges as required.

2.1.2 Technical details of CTF organization

CERT Polska describes their approach to hosting CTF competitions in [7]. Their solution for hosting *web* and *pwn* challenges is based on Docker Compose. This decision allows them using different environments for each challenge, which should behave identically on different devices, therefore eliminating compatibility problems. Routing network traffic to appropriate containers is achieved by using server blocks with `server_name` and `proxy_pass` directives in nginx configuration files.

Authors detail also their remarks regarding task design. The article suggests using NsJail to prevent accidental or purposeful denial of service (DOS) on high-risk challenges such as those leading to RCE. Another helpful idea is using supervisord for managing multiple processes inside a single container.

2.2 Existing solutions

[4]

2.2.1 CyTrONE

[1]

2.2.2 kCTF

2.2.3 CTFd

2.2.4 FBCTF

Chapter 3

Requirements and tools

The project development consisted of multiple stages. One of the most important parts was specifying requirements, which then dictated tool selection and the design process.

3.1 Functional requirements

Functional requirements list functionality available in the system. Each of the requirements contains a description detailing the desired behaviour.

1. **Account creation**: Users must be able to register an account in the system. This operation requires username and password submission from an HTML form in a POST request. Registration is allowed only using unique username. If there already exists an account in the system with the same username, the action must be refused. As a result of a successful registration, a document with the username, cryptographic hash of the user's password and a default role `"user"` is inserted into a collection storing user accounts. After the registration succeeds, the user is redirected to the login page.

2. **Signing in**: Users must be able to log into their accounts. Username and password must be sent in a POST request as HTML form data. The operation must fail if there is no account in the database with the provided username or if the password hash does not match the one stored in the database. If there has been no failure, a session is created and the user is redirected to the home page.

3. **Logging out**: Users must be able to log out of their accounts. It is expected that the user is signed in when they log out. This operation destroys the session (if any) and redirects to the home page.

4. **Changing password**: Users must be able to change their account password. New password must be sent in a POST request as HTML form data. User must be logged-

in in order to change the password. As a result of this operation, user's password hash in the database is updated.

5. **Listing categories**: Users must be able to see a list of categories that exist in the system. List of categories must link to category pages.

6. **Displaying category**: Users must be able to see category details on a category page. The details should include category name, description and a list of related tasks.

7. **Displaying task**: Users must be able to display task details on task pages. The details should include task name, description, challenge URL, hints (if any), and if the user is logged in, a flag submission form. If the task has been solved by the user, instead of the flag submission form, a question is displayed.

8. **Solving challenge**: Users must be able to submit a form with flag from the task page. This action is allowed only for signed-in users. Submitted flag must be verified with the one stored in the database for the given challenge. If the flags match, a success message should be shown to the user and date of solving the challenge by user ought to be saved to the database. Otherwise, a message informing the user about incorrect flag should be presented.

9. **Answering quiz**: User who have solved the main challenge from a task, should be able to see a question on the task page and be able to answer it. Only signed-in users can submit answers. Checkboxes which status indicates whether the user thinks that they are correct must be presented along answers from the database. After the answers are submitted, the quiz should be disabled without a button to submit, checkboxes disabled and reflecting the user's answer and an indication which answers were correct.

10. **Administrator panel**: Users with the `"admin"` role (administrators) should have access to a separate administration panel. The panel should expose additional functionality related to the system management.

11. **Listing users**: Users with access to the administrator panel should be able to get a list of user account in the system. Each entry must contain information about username and user role. The list should be paginated as there may exist many accounts.

12. **Changing user permissions**: Administrators should be able to grant the `"admin"` role to other users, as well as change it back to `"user"`.

13. **Deleting user**: Administrators should be able to remove user accounts from the database. It must not be possible to remove own user account this way.

14. **Creating category**: Administrators must be able to create new categories. The categories must have a name and description. The description must support Markdown input.

15. **Editing category**: Administrators must be able to edit the name and description of existing categories.

16. **Creating task**: Administrators must be able to add new tasks to the system. For each task it must be possible to set the name, description (in Markdown), hints, challenge details, question, answers to the question. Each answer must be marked as correct or incorrect.
    Challenge details must include:

    - Docker image to pull and start,

    - subdomain used for serving the challenge,

    - flag that the users will try to find,

    - an interval specifying how frequently the challenge container should be regenerated

17. **Starting challenges**: The system must be able to pull challenge images, create and containers and direct connections to specified subdomains into appropriate containers. This must be done automatically during system startup and for each new challenge added when creating new tasks.

3.2 Non-functional requirements

1. **Responsiveness**: UI should properly scale across different display sizes. It must be mobile-friendly.

2. **Accessibility**: There should be no errors in the Accessibility section of a webhint scan.

3. **Visual consistency**: A single set of styling rules, such as colours, fonts and icons should be used across whole user interface.

4. **Page load performance**: The system should have a score of over 90 in PageSpeed Insights report for mobile.

5. **Compatibility**: User interface should work in latest (as of January 2023) versions of Firefox, Firefox ESR, Chrome and Safari browsers for desktops and mobile devices. Basic system functionality, except for the administrator panel, should be available in browsers with JavaScript disabled.

3.3   Use cases

The use case diagram presented on Fig. 3.1 shows actions offered to actors using the system. There are two actors, with different sets of allowed interactions. The actor *User* represents anyone with access to the system. Users with special account role `"admin"`, which allows them to perform operations related to management of the system.

3.4   Tools

The implementation of the project significantly benefited from publicly available tools. Used tools are divided into two classes, depending on the way they were used.

3.4.1   Core tools

The system is built on tools, which are regarded to as *core tools*. These are required for operation and are a part of the system architecture.

Node.js + Express + EJS

Node.js is a JavaScript runtime based on V8 engine. There is an enormous ecosystem built around Node.js with over 1.3 million packages available in the npm registry [9]. The server code runs on Node.js and uses the Express framework for request routing and middleware management. Express is a popular web framework with many features and compatible middleware packages. One of Express' features is template rendering support. The project uses EJS templates, which allows creating HTML templates using embedded JavaScript logic.

MongoDB

MongoDB is a NoSQL document-oriented database capable of storing BSON documents. Because BSON stands for Binary JSON, which in turn is JavaScript Object Notation, it integrates nicely with JavaScript. The MongoDB Node.js driver also supports TypeScript, which helps with suggestions and type checking.
During development MongoDB Compass, MongoDB Visual Studio Code extension and mongosh were used. These tools allow database management and, except for mongosh, provide helpful user interfaces.

Docker

Docker is a platform for managing containerized software. Challenges are running as Docker containers and managed using Docker Engine API. For debugging, development
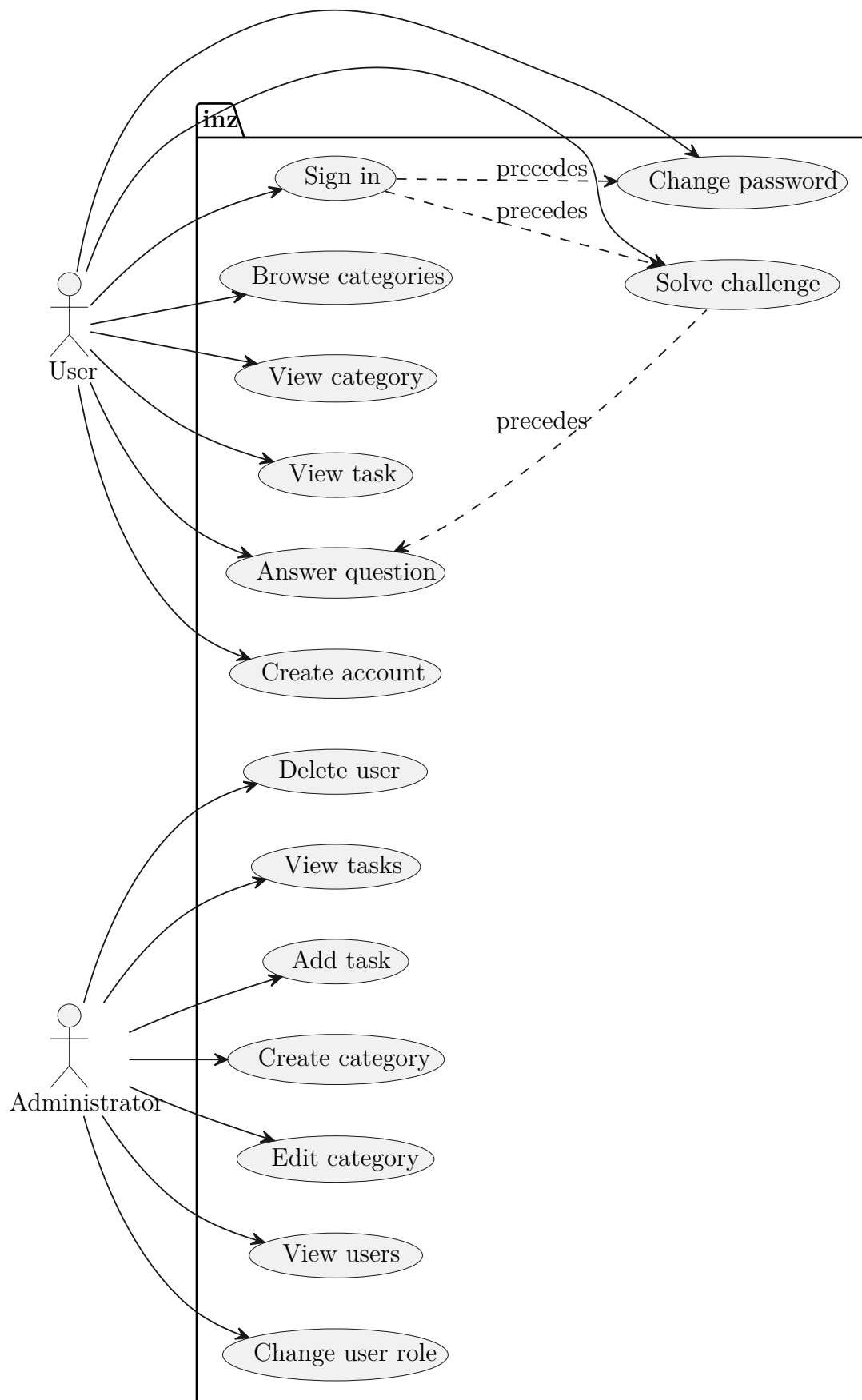
Figure 3.1: Use case diagram

and management standalone Docker tools such as Docker CLI and Docker Desktop can be used.

nginx

To manage multiple domains and subdomain on a single system with one external network interface nginx was used as a proxy. nginx offers multiple functionalities besides proxying. It is also used to serve static files, redirect to HTTPS protocol and terminate TLS connections.

Bootstrap

Bootstrap is a frontend toolkit, which is helpful for designing user interfaces. It provides CSS classes and JavaScript utilities to help with styling and providing interactivity in HTML without the need to write own style sheets and JS scripts.

3.4.2   Development tools

The following tools are in no way required for the systems. These were used to aid development.

Visual Studio Code

Visual Studio Code is a multi-platform code editor that was heavily used for writing the software. It was chosen for multiple reasons, most important of which was familiarity and experience with the tool. This editor supports many languages, especially for JavaScript and related web technologies. Particularly notable is built-in TypeScript support, which can be used with JSDoc comments to improve IntelliSense suggestions. A huge advantage of Visual Studio Code is a broad selection of available extensions.

Git

The project is stored in a Git repository to track changes in the code. The repository is synchronized with GitHub to share it between devices.

ESLint

ESLint is a JavaScript linter, which can be used to detect problems and potential issues in code. It was used with a Visual Studio extension, which automatically analysed open files and provided in-editor warnings and suggestions.

Prettier

Prettier is a popular code formatting tool. It was used to maintain a consistent style in JavaScript files. Thanks to Visual Studio Code plug-in the tool could be used as a default formatter in the editor. A plug-in for ESLint allowed marking improperly formatted code as lint error.

3.5 Methodology of design and implementation

The development process was split into two parts. Before the implementation started the system had to be designed.

3.5.1 Design

The first step in designing the project was defining initial requirements. This resulted in the first (and only) UI concept drawing, which is presented on Fig. 3.2. It was meant to help with defining the exact functionality and the structure of documents in the database. Based on that the structure presented on Fig. 3.3 was created.

After those first ideas were noted, the process of defining the architecture began. The choice of Node.js and Express as a base for the server was easy, as the author had already years of experience with those tools. Similarly, MongoDB was selected as a database server, because of experience and because it can store arrays and objects. To improve experience on low-end devices and enable basic functionality on browsers without JavaScript, server-side rendering was chosen instead of a JavaScript user interface library such as React or Vue. For a template rendering engine two solutions were considered - Handlebars and EJS. Ultimately, EJS was chosen, because of its richer capabilities.

The last problem to solve during the design phase, was coming up with an idea to direct different subdomains to appropriate challenge containers. The simplest solution was using a wildcard DNS record and setting up virtual servers in the proxy (Fig. 3.4). Creation of subdomains could be achieved by creating additional configuration files, which could be automatically picked up by the proxy on reload.

3.5.2 Implementation

The implementation started with setting up a Node.js project with some required dependencies and a basic *Hello world* Express server. Additional tools such as ESLint were also set up at the beginning so that they could be used during the development.

The next step was preparing database connection and setting up code for session support. Once the server could handle sessions, related utilities such as message flashing were prepared. Sessions were required to implement authentication, which was done in
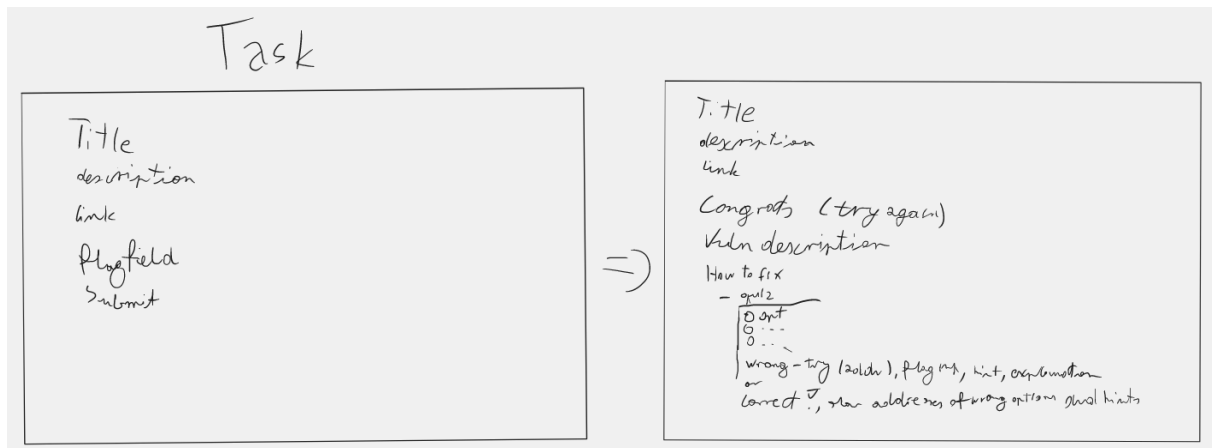
Figure 3.2: Initial task page UI sketch before and after solving the challenge.
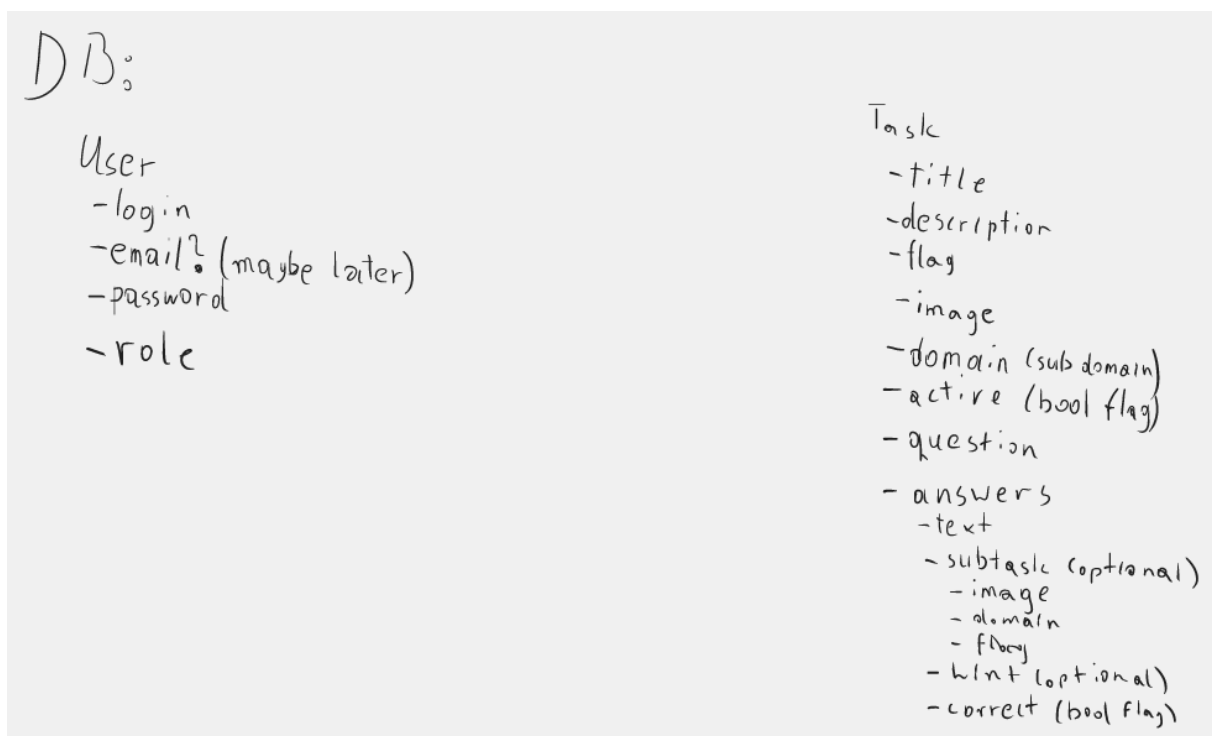

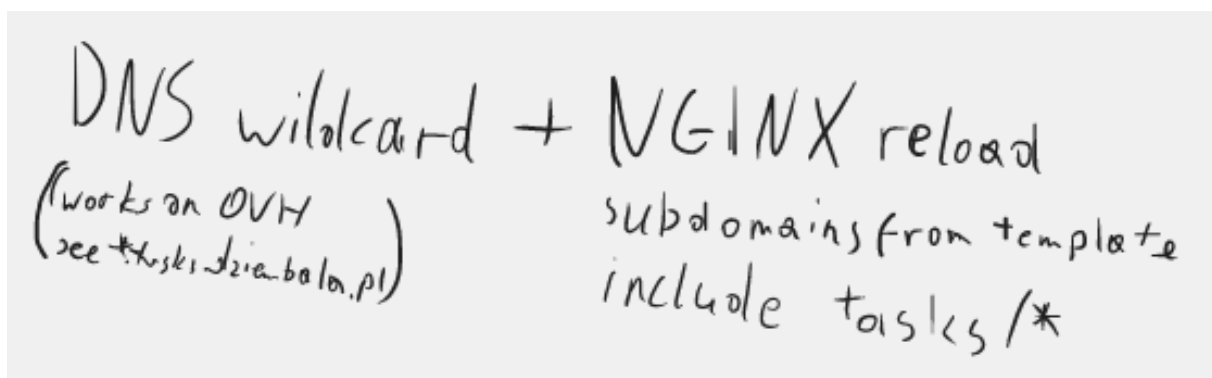Figure 3.3: Initial design of documents representing users and tasks.

Figure 3.4: Subdomain management design idea.

the next step. The first part of implementing authentication was enabling user account creation, so that it would be possible to test each element as it was developed. With the code responsible for registration ready, logging in could be worked on. In both cases the server logic was prepared first with the UI following right after to test the whole component.

After the authentication was done and there were some simple pages ready, work on styling the interface could begin. This stage of the development was realized at that moment, because there were already pages which could be tested with the styles and could serve as a base for later content.

Before more public pages could be created, an administration panel had to be made in order to insert data for testing. Unlike all other components, this panel required a lot of interactivity and relied on client-side scripting. To facilitate this requirement, the server code exposes REST-like API, which is consumed by scripts running in the browser. The administrator panel started with user management, continued to category management and finished with task creation.

Since some inputs supported Markdown, a Markdown parser was integrated on both the server and in browser. To allow for syntax highlighting in code blocks, highlight.js was added to the Markdown parser. Syntax highlighting required additional styles, which were available separately in light and dark modes and would not integrate well with Bootstrap's theming. To avoid this problem, a variable-based style sheet with colours from Panda Syntax Light and Dark themes was created with CSS variables set depending on the theme used in Bootstrap.

With the last part of the administration panel, task creation, a system for managing challenge containers had to be prepared. During the implementation of this functionality, the initial database schema shown on Fig 3.3 was recognized as suboptimal and had to be changed.

Finally, user-facing category and task pages were prepared. These were template-based and in case of the task page relatively complicated, so the user interface and server code could be implemented in parallel. As a finishing touch a simple home page was added.

Chapter 4

External specification

4.1 Hardware and software requirements

Depending on whether one wants to use the user interface via a web browser or host the server themselves there are different requirements for each part.

4.1.1 Browser

The user interface requires a modern browser with JavaScript enabled. It is possible to use the system without JS, but theme switching and dropdowns in the navigation bar will not work.

On desktop the latest stable versions of the following browsers are supported:

- Google Chrome

- Microsoft Edge

- Firefox and Firefox ESR

- Opera

- Safari (macOS only)

On Android and iOS support is restricted to the latest stable versions of:

- Google Chrome

- Firefox

- Safari (iOS only)

- Android Browser and WebView (Android only)

Different and older browsers may work, but compatibility is not guaranteed.

4.1.2 Server software

The server software requires the following external software to be installed:

- Node.js 18.x or newer (not as a snap)

- MongoDB 6.0

- Docker Engine 20.10

- nginx 1.23.2 or newer

Supported operating systems:

- Ubuntu 20.04 or newer (x86-64, arm64)

- macOS 11 (x86-64), 12 or newer (x86-64, arm64)

- Windows 10 1809 or newer (x86-64)

- Windows Server 2019 or newer (x86-64)

It may be possible to run the server on other operating systems and architectures as long as the required software can be installed. These unsupported systems, however, will probably require additional installation steps related to `argon2` package setup.

Warning: Challenge Docker images must be compatible with the server architecture. Images presented in this project support only x86-64.

4.2 Server installation

The process of server software installation and configuration can be summarized in the following steps:

1. Install the required software as described in section 4.1.2.

2. Obtain the code from GitHub:
   `git clone https://github.com/krzysdz/inz.git`

3. Navigate to the downloaded project directory:
   `cd inz`

4. Install required npm packages:
   `npm install --omit=dev`

5. Configure MongoDB as a replica set (tutorial).

6. Configure the DNS to point the main domain and the challenges domain to the server. Example DNS configuration:

```
main-ui.com.        60  IN A       127.0.0.1
*.challenges.com.   60  IN CNAME   main-ui.com.
```

7. Obtain TLS certificate for the main domain and the challenges domain. One certificate should cover both domains (the second one with wildcard).
   The key and certificate files should be placed in `nginx/certs` and named according to instructions from the `README` file in that directory.

8. Adjust values in the `config.js` file.

9. Start nginx with prefix set to `./nginx` and configuration `nginx/conf/nginx.conf`:
   `nginx -p ./nginx -c ./nginx/conf/nginx.conf`

10. Set `NODE_ENV` to production:

    Bash/dash/zsh/csh:
    `export NODE_ENV="production"`

    Powershell:
    `$env:NODE_ENV="production"`

11. Start the server:
    `node index.js`

12. *Optional.* Configure services to automatically start the software. Example configuration for systemd based operating systems:

    (a) Modify the default mongod.conf file to specify replica set name as shown on Fig. 1 and enable the mongod service:
    `systemctl enable mongod`

    (b) Create a unit file for the proxy. An example is presented on Fig. 2.

    (c) Create a unit file for the main server. An example is presented on Fig. 3.

    (d) Enable and start the services:

    ```
    # Reload systemd configuration to pick up the new services
    systemctl daemon-reload
    # Enable the services to run at boot
    systemctl enable inz-nginx
    systemctl enable inz
    # Start the server and proxy (required by inz.service)
    systemctl start inz
    ```

17

4.3 Initial configuration

If setting up the server with a fresh database, an administrator account must be created. To do this, one has to:

- Create a user account (`/auth/register`).

- Set the `role` field of the user document to string `admin`. A Node.js script which does that is presented on Fig. 4.

- Log in into the account again.

4.4 Types of users

There are three types of users considered in the system:

- anonymous/not signed in

- regular users - account with role `"user"`

- administrators - account with role `"admin"`

Anonymous users can freely browse the application, but have a limited functionality on the task page.
Regular users gain access to profile page and unlock challenge and quiz submissions.
Administrators expand on the regular user permissions and can use an administration panel for managing the system.

4.5 User manual

This manual is intended for the end users. The functionality available for administrators is described in section 4.6.

4.5.1 Theme selection

The system can use light or dark mode according to user preferences. By default, the browser or OS determines the theme. A dropdown (Fig. 4.1) shown after clicking an icon in the upper right corner can be used to manually change the page theme. The preference is stored in the browser. Three options are available:

- Light - shown on Fig. 4.1b,

- Dark - visible on Fig. 4.1a,
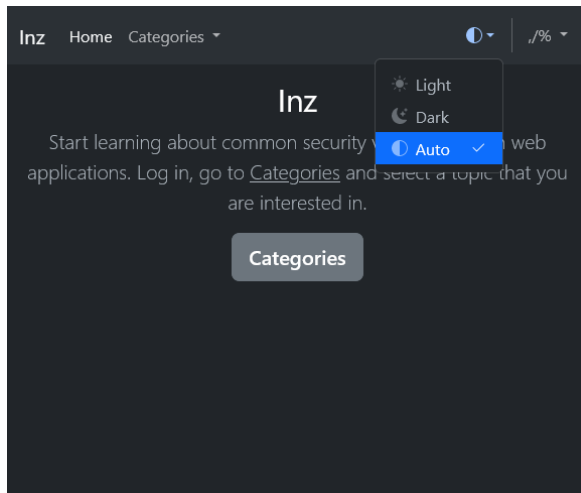
(a) Automatic theme - dark   (b) Light theme

Figure 4.1: Theme selection dropdown

- Auto - determined by browser or OS preferences, presented on Fig. 4.1a with browser theme set to dark.

4.5.2   Account creation

To access all features of the system a user account is required. Without it flag submission and later stages of solving tasks are impossible.

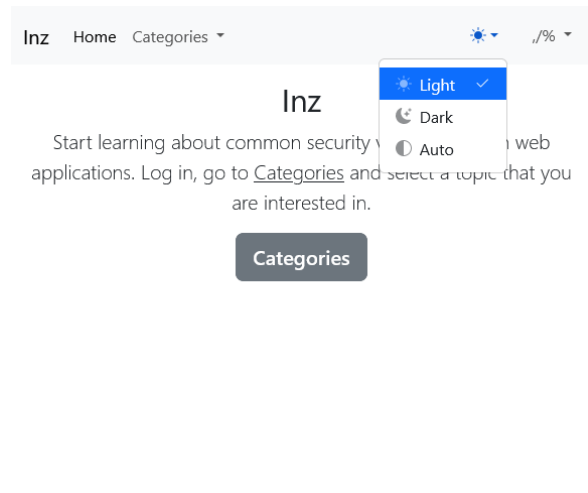The account can be created on the *Register* page which can be accessed from the home page as shown on Fig. 4.2 or by clicking *Create an account* on the login page. On that page the desired username and password have to be filled into the input fields. The password has to be between 8 and 64 characters long. To prevent mistakes in the password it has to be entered twice. If the passwords do not match the browser will prevent form submission and display an appropriate error.

The registration may fail if there already exists an account with the provided username. In such case a different username should be used.

4.5.3   Logging in

To use the account created in 4.5.2 one has to log in into the account. To do that the user has to submit their username and password using a form on the login page (shown on Fig. 4.4). The page can be accessed by clicking a button in the navbar, a button on the home page or a link on the registration page.
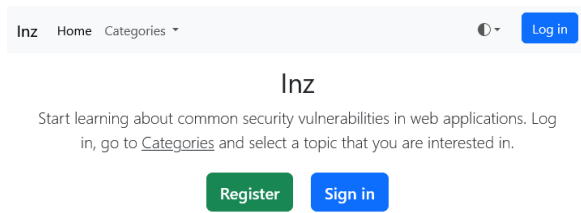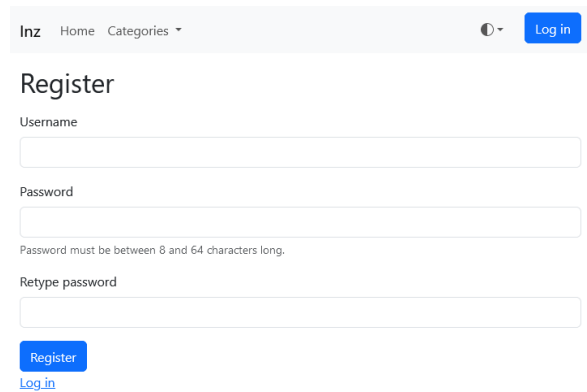
Figure 4.2: Home page - logged out.

Figure 4.3: The registration page.
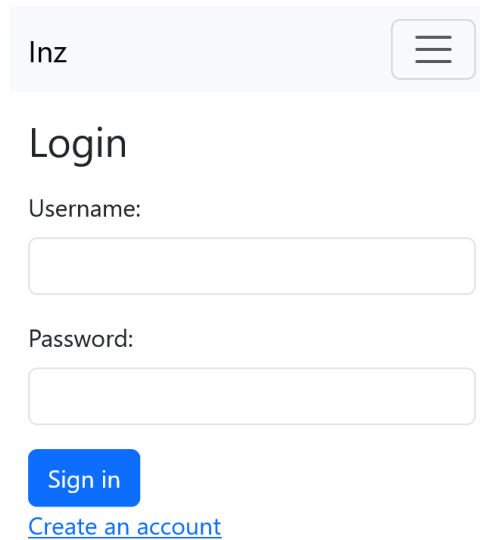
Figure 4.4: Login page viewed on mobile
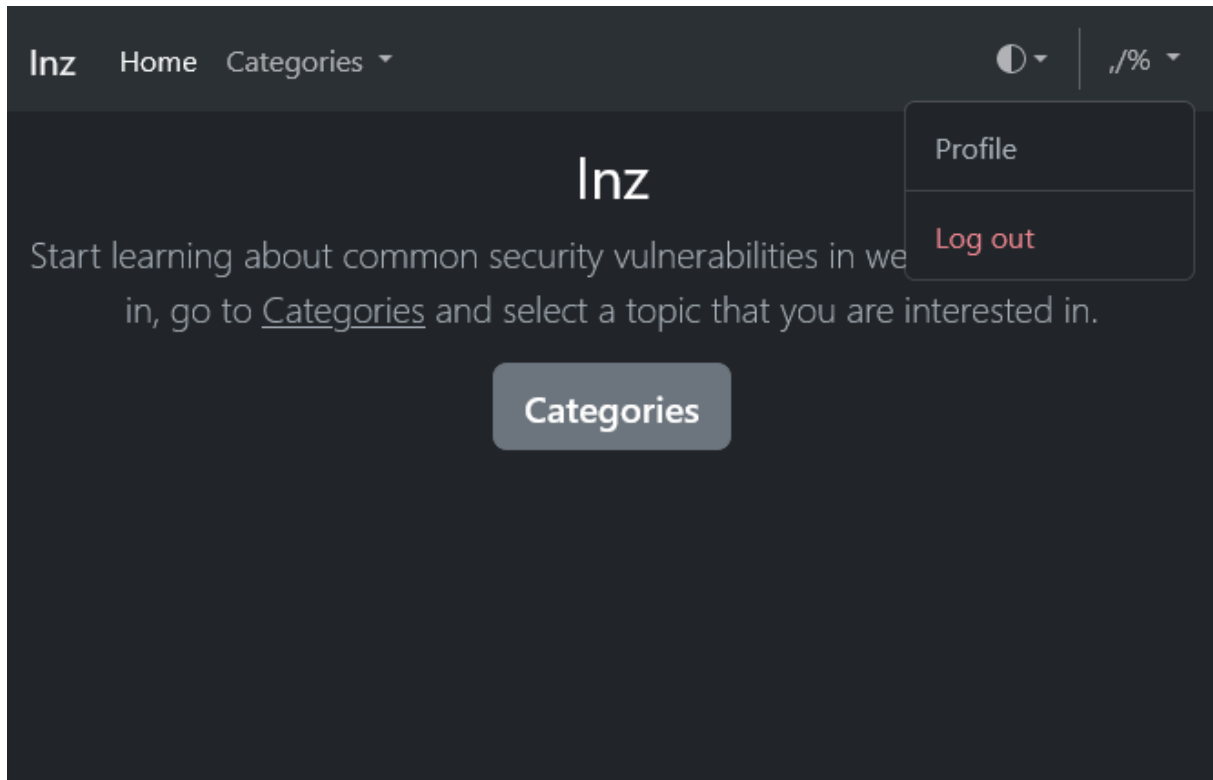
Figure 4.5: Username dropdown on home page for a regular user.

4.5.4 Logging out

A logged-in user can log out using the *Log out* button in the dropdown appearing after the username in upper right corner is clicked. The dropdown is presented on Fig. 4.5.
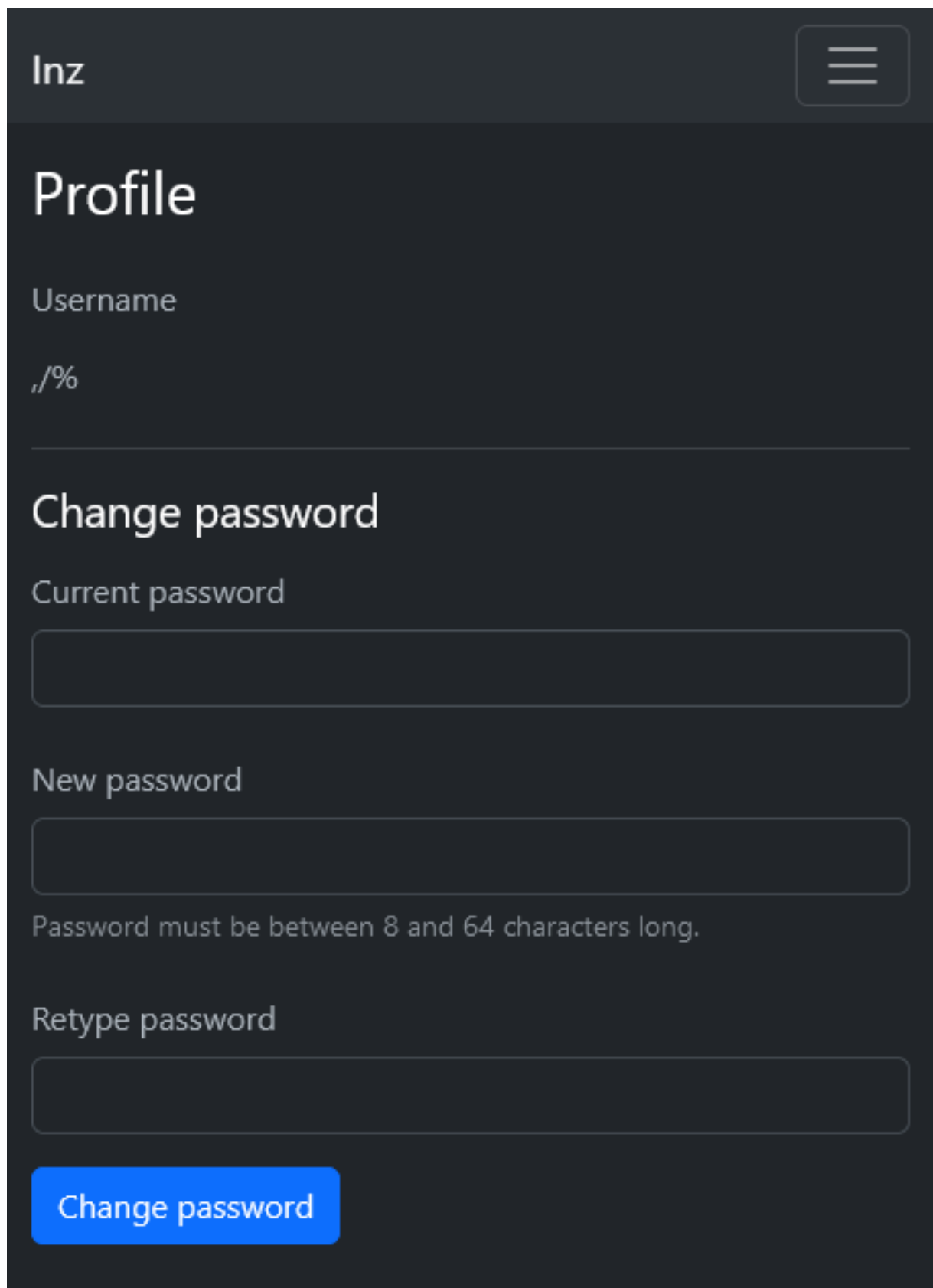
4.5.5 Account management

The profile page (`/profile`) shown on Fig. 4.6 is available for logged-in users through a link in the navbar (see Fig. 4.5). This page allows changing the account password. To change the password one has to provide their current password and enter the new one two times. After submitting the form the system will show a message informing about the success or failure of the operation. Password change does not invalidate existing sessions.

4.5.6 Browsing categories

Categories in the system describe classes of vulnerabilities. A list of available categories can be accessed from the *Categories* dropdown in navbar and from the *Categories* (`/categories`) page. Both lists contain names of the categories which link to appropriate category pages.

Category pages describe classes of vulnerabilities, why these exist and how they can be avoided. Additionally, on the bottom of each category page there is a list of related tasks which demonstrate examples of such issues.

Figure 4.6: Profile page in a narrow window.

4.5.7   Tasks

Tasks show specific examples of vulnerabilities. Each task contains a link to a challenge website which has to be exploited. The process will look differently for each challenge. As a result of a successful attack the user should obtain a flag. The flags should follow the `flag{xxx}` scheme unless specified otherwise in the task description. Tasks can contain hints which may help in case of problems when solving the linked challenge. The hints are collapsed by default to avoid spoilers and can be uncovered by clicking them.

Once the flag is found it should be submitted on the task page. Users who are not logged in will be prompted to log in as demonstrated on Fig. 4.7a. The form is shown only if the challenge is not solved as presented on Fig. 4.7b. After a successful submission a message and a question appear instead of the form.

The second step of completing a task is answering a question like the one presented on Fig. 4.7c. Questions are related to the challenge and usually concern avoiding the problem or fixing it. There are multiple answers, each of which can be marked as true by marking the checkbox next to it. Conversely, if an answer is thought to be false the checkbox should be left empty. Finally, the answers should be sent by clicking the *Submit* button.

If the question has been answered the checkboxes are disabled and reflect user's choices. Answers correctly marked as true or false are displayed in green, while incorrect are shown in red. Additionally, the true answers are listed on the bottom of the page. Below some answers an explanation may appear which expands on the answer. An example of this behaviour with an incorrectly marked first answer is presented on Fig. 4.7d.

Selected incorrect answers may contain additional challenges. Below these answers a link to the challenge will appear. These challenges are modified versions of the original challenge. After solving such challenge the obtained flag should be submitted using a form below the answer. If the additional challenge is solved a success message will replace the form as can be seen on figures 4.7d and 4.7e.

The whole process of solving a task can be observed on Fig. 4.7.

4.6   System administration

The system is managed using and administrator panel `/admin`. It can be accessed from the dropdown in upper-right corner which is presented on Fig. 4.8. JavaScript is strictly required for all functionalities of the panel. There are three tabs for different categories of administrative capabilities.

4.6.1   Managing users

The *Users* tab presented on Fig. 4.9 offers basic account management features. In this tab a list of users registered in the system is presented. Next to each username their role is
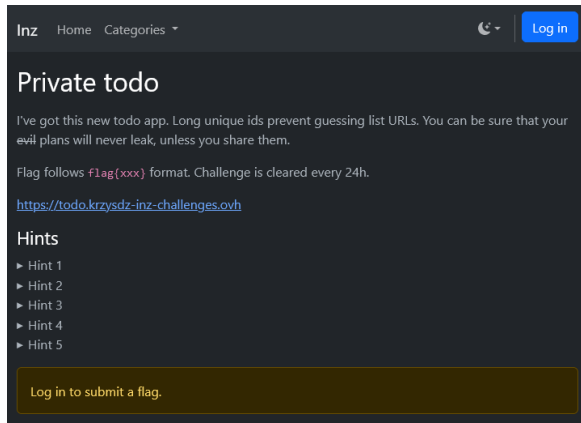
(a) viewed while logged out
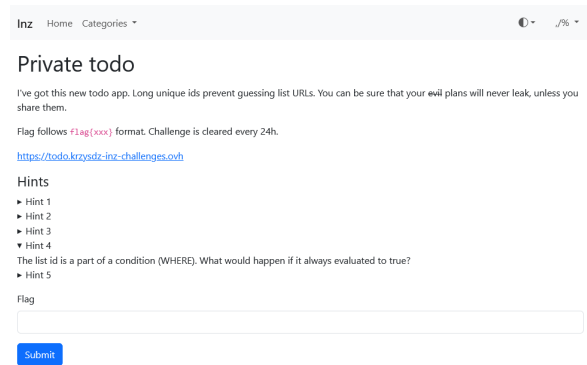
(b) viewed by user
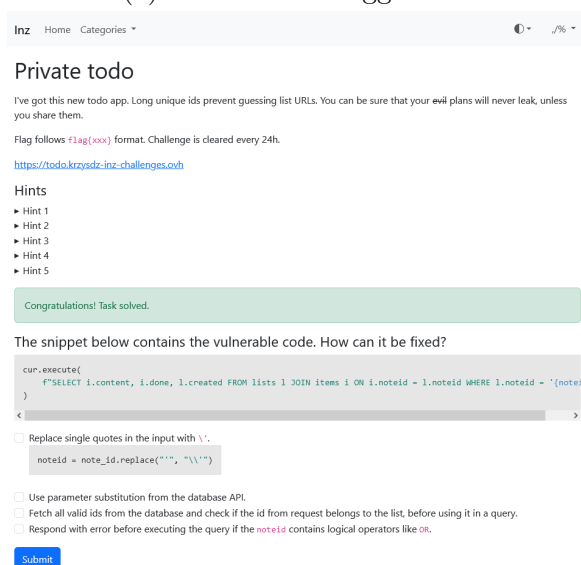
(c) main challenge solved
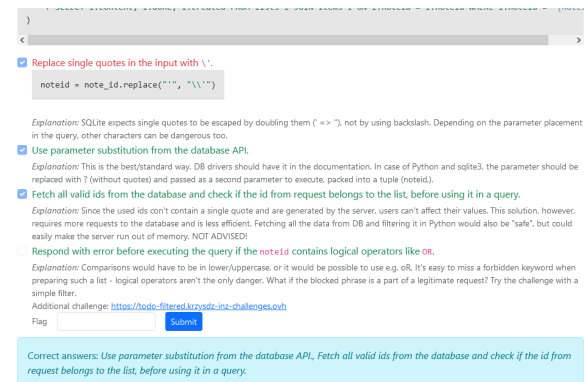
(d) question answered, correct answers FTTF
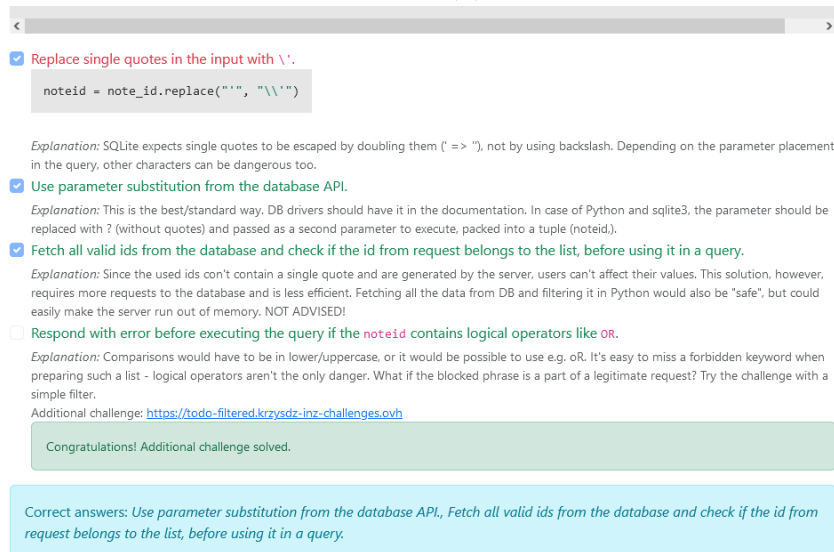
(e) additional challenge solved

Figure 4.7: Task page at different stages of completion.
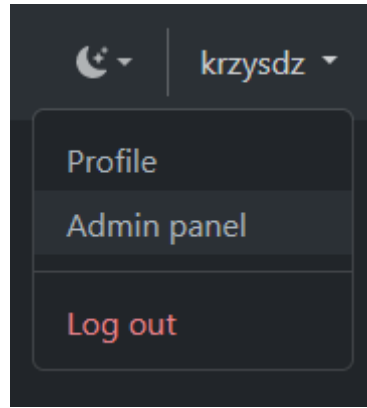
Figure 4.8: Dropdown with admin panel link.
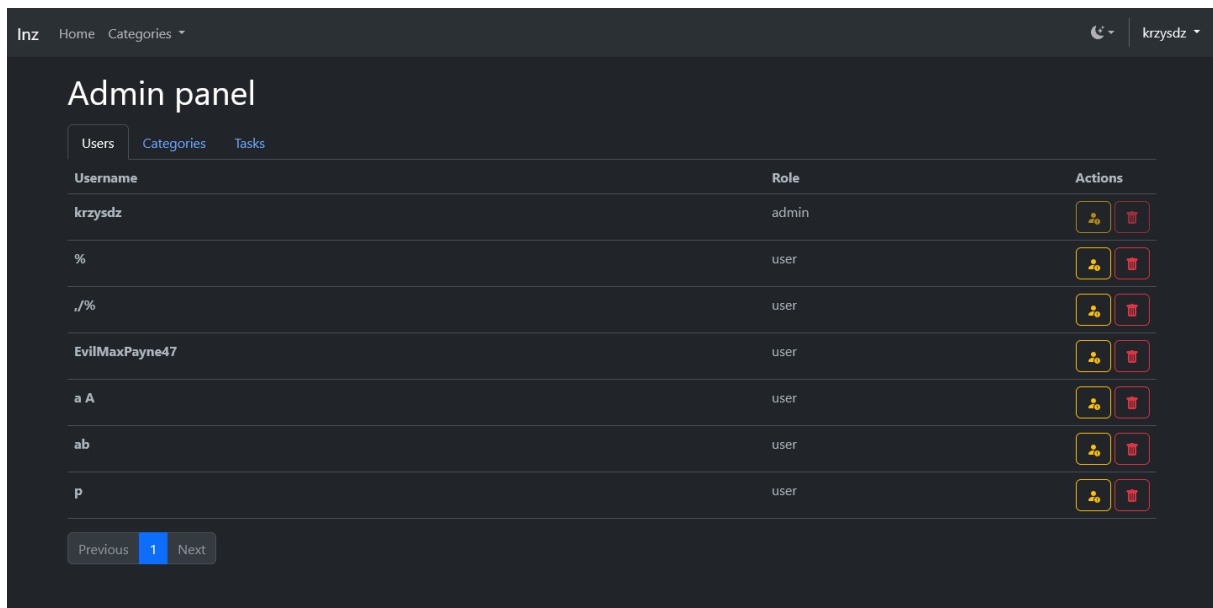
Figure 4.9: Users tab in admin panel.

displayed. Finally, in the third column there are two action buttons. The first one changes user role from `user` to `admin` or the other way. The second one can be used to delete a user account. The list contains at most 10 entries. Pagination buttons below the table can be used to display another page of users.

4.6.2 Managing categories

The next tab *Categories* allows creating and editing category pages. All existing categories are displayed as collapsed accordion items. Clicking on them opens a preview of the category description with an *Edit category* button at the bottom.

New categories can be created by clicking the *New category* button at the top of the tab. The button opens a simple editor which can be used to create new categories. The name and description must be filled before creating the category using the *Create* button. The name must use plain text, since it will always be escaped before displaying.

The description should be entered using Markdown. To help with writing in Markdown a preview mode can be toggled by clicking the *Preview* button on the right above the description input.

If the *Edit category* button at the bottom of category description the editor will be populated with the stored details. The editor is the same as the one for new categories, but instead of the *Create* button, a *Save* button will appear.

Both creating a new category and editing an existing one causes reloading of the categories list. The description editor uses Marked for compiling Markdown content. The supported specification is listed at the bottom the discussion markedjs/marked#1202.

4.6.3 Managing tasks

The tab *Tasks* show an overview of existing tasks and allows creating new. Like categories the existing tasks are listed as expandable accordion items.

New tasks can be added using a form which appears after clicking the *New task* button at the top of the tab. The following data should be entered in the form:

- **Task name:** The task name. It is displayed on the top of the task page and as a tile title on the bottom of relevant category page.

- **Category:** The category of vulnerabilities this task is assigned to. Must be chosen from one of the existing categories.

- **Description:** Task description displayed on the task page. The editor supports markdown and can be used the same way as the category description editor.

- **Challenge details:** The details for the main challenge. These options include:

  - **Docker image:** Docker image to use for the challenge container. The image will be pulled after creating the task.

  - **Subdomain:** Subdomain of the domain specified in the config file, which will be used to hosting the challenge.

  - **Flag:** Flag which should be found by the users. Flags submitted by users will be compared to this value.

  - **Expose flag as `FLAG` environmental variable:** Whether the flag should be exposed in the container as an environmental variable `FLAG`. This option can be used to avoid hardcoding flag values in challenge images.

  - **Reset interval:** How often the challenge container should be removed and recreated. The value should be given in seconds. Set to 0 to disable automated restarts.

- **Hints:** A list of hints for the main challenge. Hints are rendered inside a collapsed `<details>` tag on the task page, because they are supposed to contain *spoilers*. New hints can be added by clicking *Add a hint*. All hint inputs must be filled. The X button on the right of a hint input can be used to remove it. It is possible to create a task with no hints.

- **Question:** The question shown after solving the challenge. The question is rendered without escaping, so inline HTML can be used, but the content is placed inside an `<h4>` tag.

- **Answers:** These are the answers to the question from the field above. Additional answers can be added using the *Add an answer* button. It is not possible to remove an answer. An answer has the following properties:

  - **Answer:** The answer text that is presented to the users. The text is not escaped and can include HTML.

  - **Answer is correct:** Whether the answer is correct and users are expected to check it.

  - **Add challenge:** Shows an additional set of fields related to an extra challenge. **Available only if the answer is not marked as correct.**

  - **Explanation:** An **optional** explanation that is shown after answering the question. It may underline the reasons why an answer is right or wrong.

  - **Challenge details:** Challenge details for an additional challenge related to the answer. Options are the same as for the main challenge. **Shown only if the *Add challenge* option is checked.**

An expanded task accordion item presents the task properties. Challenge details other than the *expose flag…* option are shown in collapsed `<details>`. Checkboxes next to answers indicate whether these answers are marked as correct. Optional explanations are shown in italics below respective answers.

4.7   Security issues

Secure design is an important part of the project. The target audience is expected to be or become experienced with exploitation, so the system had to be carefully created and must be responsibly managed.

4.7.1   Project author's considerations

Security of the system in a huge part depends on the main server design and implementation details.

Data flow analysis

The data sources provided by regular users are:

- username set in the registration process - a string of at most 255 characters, always escaped in templates, inserted as `textContent` in admin panel, encoded with `encodeURIComponent()` in URLs,

- category name - a string (part of the URL), used for comparison in `$match` stage of an aggregation pipeline as `{ $match: { name: name } }`,

- task id and challenge id in challenge and quiz submissions - always a conversion to `ObjectId` is attempted, only the converted value is used later,

- flag in challenge solution submission - trimmed using `String.prototype.trim()` and compared using `===`,

- `answer[n]` properties in quiz submission - iterated from 0 to the number of answers in the task and compared to `"on"`.

The data coming from these sources is always used in a safe way. Data entered by administrators can be *dangerous* by design, since they must be able to insert raw HTML and run any software using containers.

Known vulnerabilities

Express version used in the project depends on a vulnerable version of qs library (CVE-2022-24999). qs package is used in Express only if the `query parser` setting is set to `"extended"` or the `urlencoded` middleware's `extended` option is a truthy value. The `query parser` option is `"simple"` by default and the `urlencoded` middleware is created with `{ extended: false }`, so this vulnerability should not affect the project.

4.7.2   Administrator's responsibilities

Regardless of the project design it has to be configured and used properly so as not to introduce new security issues. System administrators should keep in mind the following warnings:

- Challenges should use a different domain than the main website so that users who exploit a challenge will not be able to set cookies valid for the main domain.

- Special care must be taken when designing challenges, especially ones leading to RCE. Since challenges are running as Docker containers a misbehaving container can bring the whole system down. To provide isolation and limit resources use of NsJail is suggested.

4.8 Covered categories of issues

The design of the project separates the system from the content. Categories and tasks can be added at any time as described in sections 4.6.2 and 4.6.3. The example deployment covers some of the common security vulnerabilities found in web applications.

4.8.1 SQL injection

This category covers basics of SQL injection exploitation and prevention. Small code snippets and examples are provided to help users understand the reason behind this class of vulnerabilities. At the bottom of the description a *Read more* section contains links to additional resources connected to this topic.

An example of a vulnerable application is presented in the *Private todo* task. The target is a simple website offering to-do list creation and sharing. These lists are accessed using an `id` parameter, which is insecurely concatenated with the database query allowing for an injection attack. The question shown after solving the challenge presents the vulnerable line and answers show suggested methods of fixing the problem. One of them which proposes filtering some SQL keywords has an additional challenge.

Chapter 5

Internal specification

5.1   System architecture

The system is managed by a server running on Node.js. As presented on Fig. 5.1, this process manages the MongoDB database, challenge containers via Docker Engine API and the nginx proxy configuration. Challenges are run as Docker containers to enable easy configuration by just puling an appropriate image and each of them is bound to a different subdomain of a dedicated challenge domain thanks to the single proxy server.

The nginx process is the only one exposed publicly. Other software can listen only on localhost for security reasons.

5.2   Database structure

The use of a NoSQL database, allowed for a more flexible schema, based on the desired access to the data. A mix of embedded data model and normalized data model is used. The used database schema is presented on Fig. 5.2. Despite preferring the denormalized model, it was necessary to keep some relations between documents. This is achieved by storing fields with `_ids` of referenced documents. There is also a relation between keys of embedded documents in the `users` collection, which are stored as stringified `ObjectIds` and documents from `tasks` and `challenges`. This relation, however, is not used in queries to the database, but managed completely by the server code.

There are two additional indexes:

- a unique index on the `subdomain` field in the `challenges` collection to make sure that there are no duplicate subdomains,

- a unique index with a collation on the `name` field in the `categories` collection to accelerate case-insensitive search by category name and prevent duplicates.
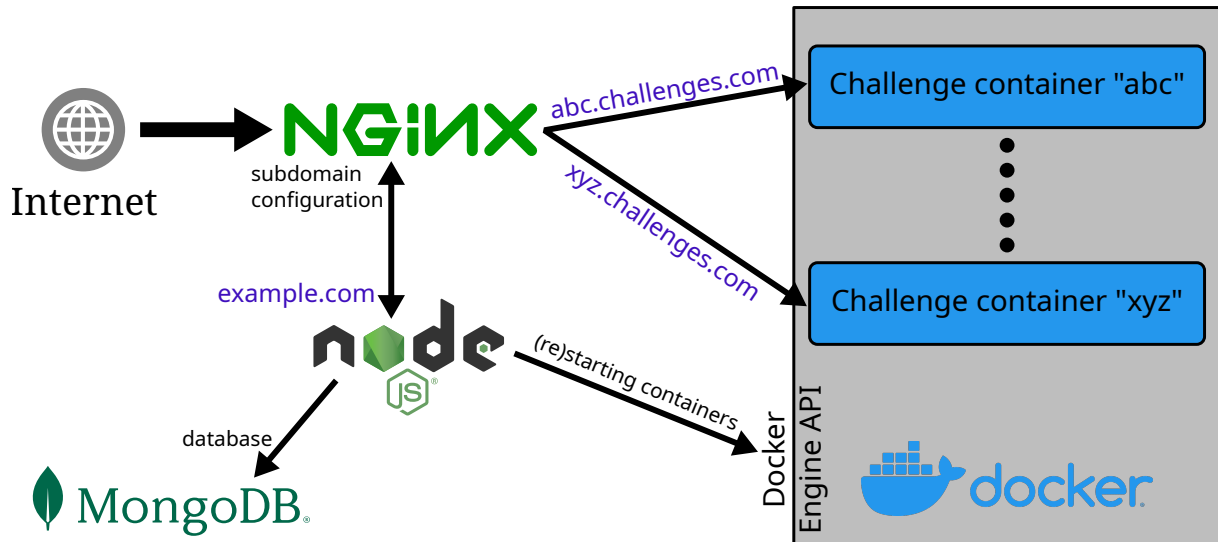
Figure 5.1: Visual representation of system architecture.

The view `fullTasks` is a view collection on the `tasks`, which uses an aggregation pipeline to embed documents from the `challenges` collection using `$lookup` where necessary. The pipeline is presented on Fig. 5.

5.3   Code organization

The code is divided into ECMAScript modules. The main one is `index.js`, which imports all the other required modules, some external tools, configures and starts the server. Each module serves a separate function.

5.3.1   Middleware

Two modules are responsible for middleware functions. Middleware is a function called during request processing, which can act on the request and response objects and pass execution to functions declared later in the router stack.

One of them, `src/middleware.js`, contains general utility middleware functions:

- `authenticated` - a middleware which returns 401 if the user is not logged in,

- `adminOnly` - a middleware which returns 403 if the user is not an administrator,

- `addCategoriesList` - a middleware executed before any routers, which fetches all category names and makes them available as response locals to use within templates.

The other module was separated, because it alters the request object and has an associated `.d.ts` typings file to help with type checking and code suggestions. It is `src/flash.js` and adds the following methods to each request object:

Collection "users"

UserDoc
```
{
  _id: string,
  hash: string,
  role: "user" | "admin",
  solves?: {
    [challengeId: string]: Date | undefined
  },
  answers?: {
    [taskId: string]: boolean[]
  }
}
```

Collection "categories"

CategoryDoc
```
{
  _id: ObjectId,
  name: string,
  descriptionRaw: string,
  descriptionFormat: "html" | "md",
  descriptionProcessed?: string | undefined
}
```

Collection "tasks"

TaskDoc
```
{
  _id: ObjectId,
  name: string,
  categoryId: ObjectId,
  descriptionMd: string,
  descriptionRendered: string,
  challengeId: ObjectId,
  hints: string[],
  question: string,
  answers: ({
      text: string,
      correct: true,
      explanation?: string
  } | {
      text: string,
      correct: false,
      explanation?: string,
      challengeId?: ObjectId
  })[],
  visible: boolean
}
```

Collection "challenges"

ChallengeDoc
```
{
  _id: ObjectId,
  taskImage: string,
  subdomain: string,
  flag: string,
  flagInEnv?: boolean | undefined,
  resetInterval: number,
  challengeKind: "task" | "subtask"
}
```

View "fullTasks"

FullTaskDoc
```
{
  _id: ObjectId,
  name: string,
  categoryId: ObjectId,
  descriptionMd: string,
  descriptionRendered: string,
  challenge: ChallengeDoc,
  hints: string[],
  question: string,
  answers: ({
      text: string,
      correct: true,
      explanation?: string
  } | {
      text: string,
      correct: false,
      explanation?: string,
      challengeId: ObjectId,
      challenge: ChallengeDoc
  } | {
      text: string,
      correct: false,
      explanation?: string
  })[],
  visible: boolean
}
```
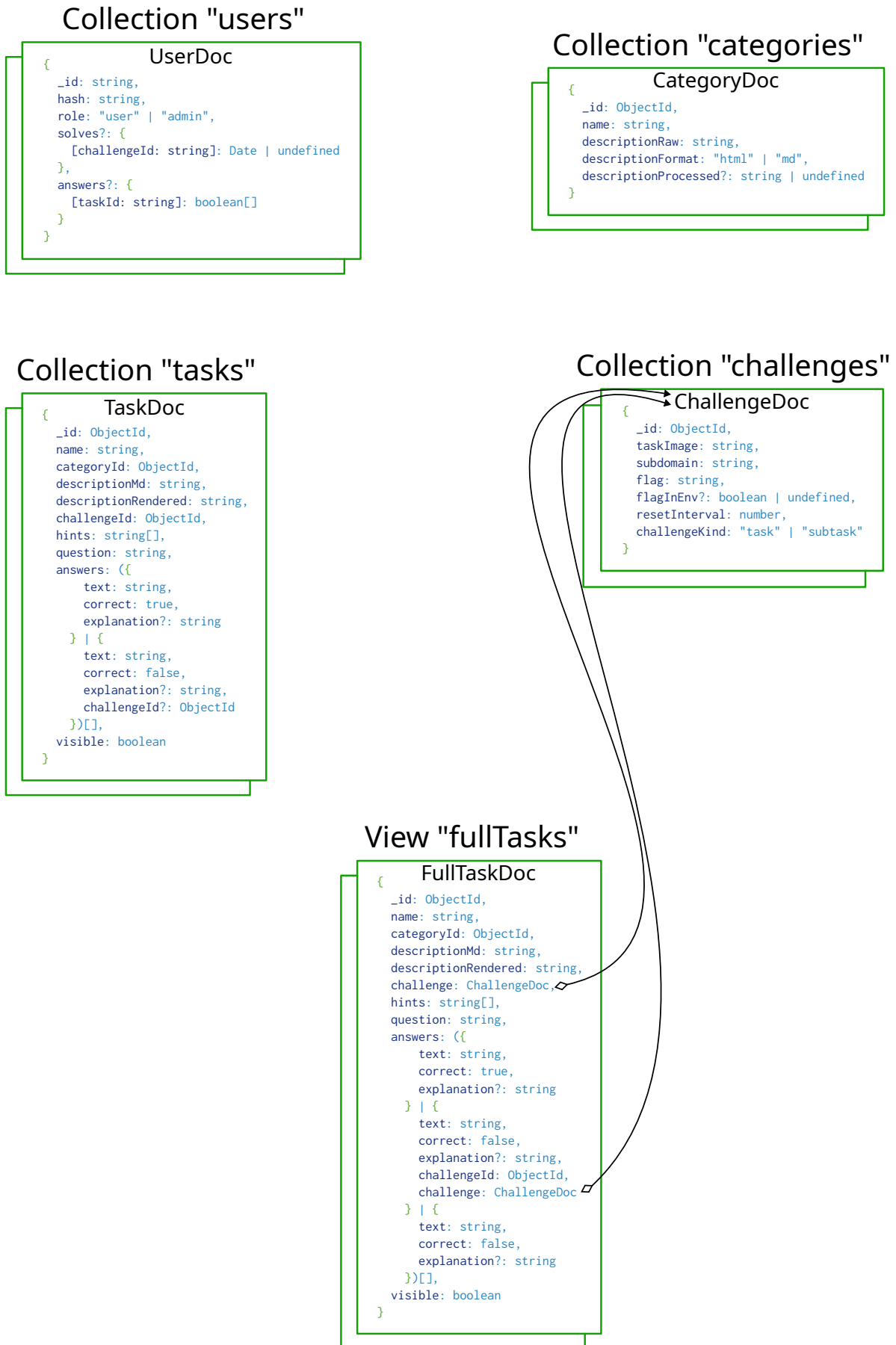
Figure 5.2: Visual diagram of the database schema.

- `flash(message, category = "info")` - a function which adds a message with category to a list of flash messages connected with the session

- `getFlashedMessages(options)` - a function which returns the flashed messages with associated categories. This function can also filter the returned flashes by category. It is also available in response locals for direct use in templates.

The behaviour of this middleware is supposed to mimic the message flashing functionality of the Flask framework.

5.3.2 Routers

The `index.js` module provides only the request handler for the root path `/`. Other paths are delegated to separate routers, which live inside modules under `src/routes`. The following paths are registered:

- `/auth` handled by `authRouter` from the `auth.js` module. It serves the register, login and logout pages, as well as handles password change requests.

- `/profile` guarded by the `authenticated` middleware, handled by `profileRouter` from the `profile.js` module. It serves only the profile page.

- `/admin` guarded by the `authenticated` middleware, handled by `adminRouter` from the `admin.js` module.
  The router uses additional middleware `adminOnly` and `express.json()`. It renders only a single page - the admin panel. It provides a REST-like JSON API, which is used by a script on the admin panel page.

- `/category` handled by `categoryRouter` from the `category.js` module. It serves category pages with lists of related tasks and a category list page `/categories/`.

- `/task` handled by `taskRouter` from the `task` module. It serves task details pages and handles challenge and quiz submissions. All submissions are guarded with `authenticated`. The details page is handled separately for anonymous and other users as described in section 4.4.

5.3.3 Markdown parser

To parse descriptions in Markdown the `marked` parser is used together with syntax highlighting library `highlight.js`. Because the processing can take some time and the execution runtime is single-threaded, it is offloaded to a worker thread. The thread is terminated after a timeout passes to avoid blocking the thread pool. [11, 10]

The first module responsible for this functionality `src/markedThread.js` listens to messages on `parentPort` and responds to them with an HTML string containing parsed Markdown.

The second module `src/markdown.js` exports a `processMarkdown` function, which manages the worker thread. It returns a `Promise`, which is resolved after the worker sends the rendered string. If the worker emits an error event, the promise is rejected. After a specified time passes the worker is terminated and the promise rejected, unless it had finished successfully earlier.

5.3.4 Database connection

The module `src/db.js` does not export any functions. When it is imported it creates a `MongoClient` connected to the database server and opens the database specified in the configuration file. It calls then an `async` function `setupDb`, which prepares necessary indexes and views. Both the client (`client`) and database (`db`) are exported. All modules importing either exported variable reuse the same client connection.

5.3.5 Challenge management

Challenges are managed using functions from the `src/challenges.js` module. The module has the following functions:

- `getChallengeContainers(all = false)` - returns challenge containers (only running, unless `all` is `true`). It relies on label `pl.krzysdz.inz.challenge-kind` being present.

- `startupChallenges()` - removes all nginx subdomain configuration files, stops and removes all challenge containers and calls `addChallenge` with each challenge document from the database.

- `addChallenge(challengeDoc, reload = false)` - verifies the image tag, then calls functions `startChallengeContainer` and `createNginxConfig`. If `reload` is `true`, calls a function responsible for reloading nginx configuration.

- `checkTag(challengeDoc)` - checks if the document contains an image with a tag. If there is no tag, `:latest` is appended to the image name. Returns the `challengeDoc` with corrected `taskImage` field.

- `startChallengeContainer(challengeDoc)` - pulls the image specified in the argument, stops and removes containers with the same name if they exist, creates a new container and starts it. If `resetInterval` is not 0, `setInterval` is created, which uses `restartChallengeContainer` to periodically restart the container.

- `restartChallengeContainer(port, challengeDoc)` - restarts the container specified in `challengeDoc`. This function is not exported and called only in interval, which is created after starting the container for the first time. It stops and removes the container including volumes, creates a new container reusing the same port and starts the new container.

- `createNginxConfig(subdomain, port)` - creates an nginx configuration file in `nginx/conf/subdomains` directory, which defines an HTTPS server proxying request made to the subdomain of domain specified in config to the specified port on localhost.

- `reloadNginx()` - executes the nginx binary with `-s reload` and prefix set to `./nginx` from the project directory to instruct the proxy to reload its configuration.

5.4 Sequence diagrams

Basic system functionality in a simplified form can be presented in form of UML sequence diagrams. Figures 5.3, 5.4 and 5.5 demonstrate flows related to authentication and account management. The next three diagrams 5.6, 5.7 and 5.8 present operations connected with tasks. The diagrams are simplified, to avoid repeating the process of fetching task and checking user progress before rendering the page.

Actions available for the administrators involve an additional participant - a script running in the browser which is responsible for managing the UI and communication with the server. Because the panel is not rendered on the server, the process of loading it is more complicated as can be seen on Fig. 5.9. The most complicated operation is adding a new task. The flow is presented on 5.10 and requires interactions with even more system components.
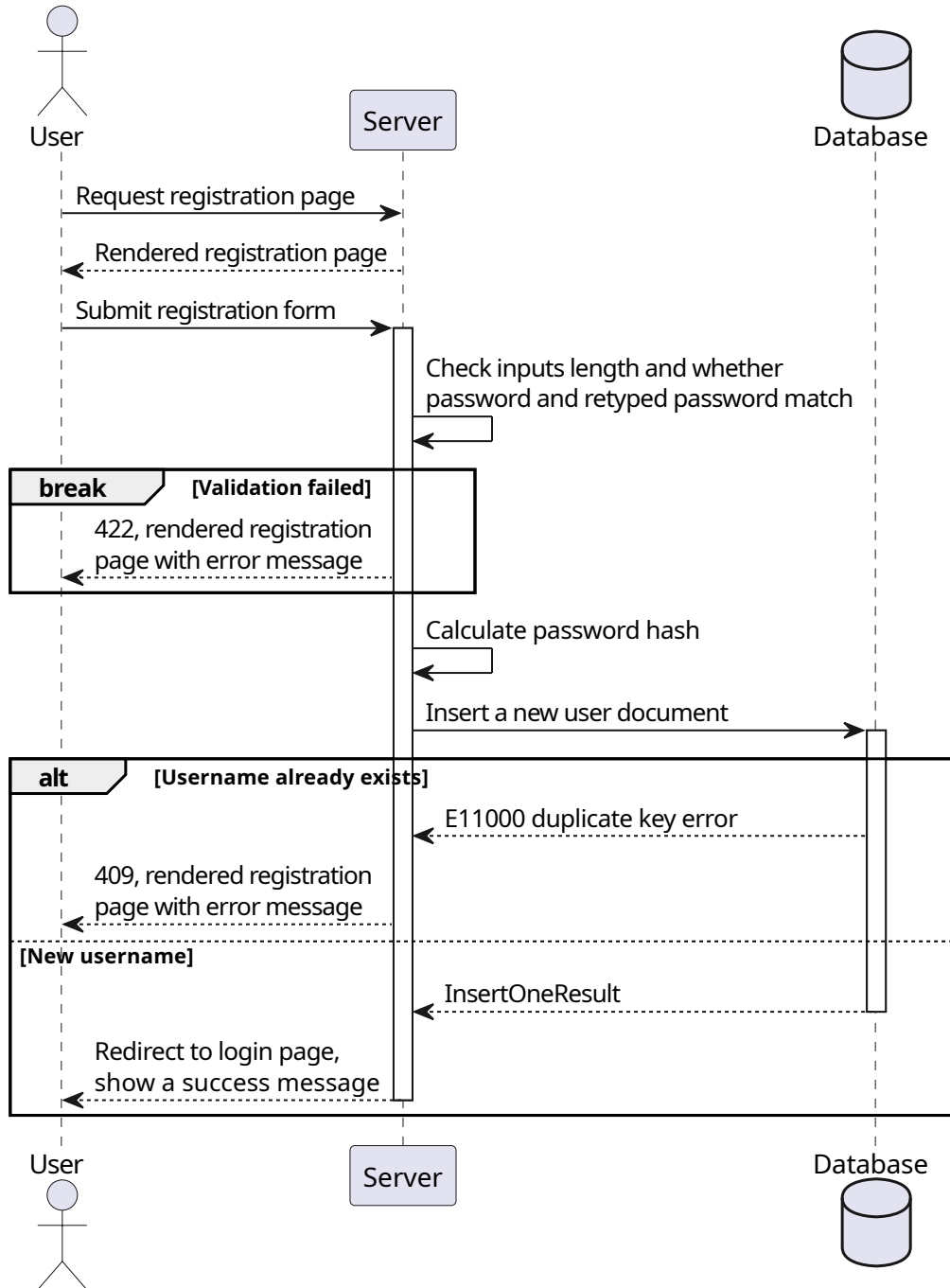
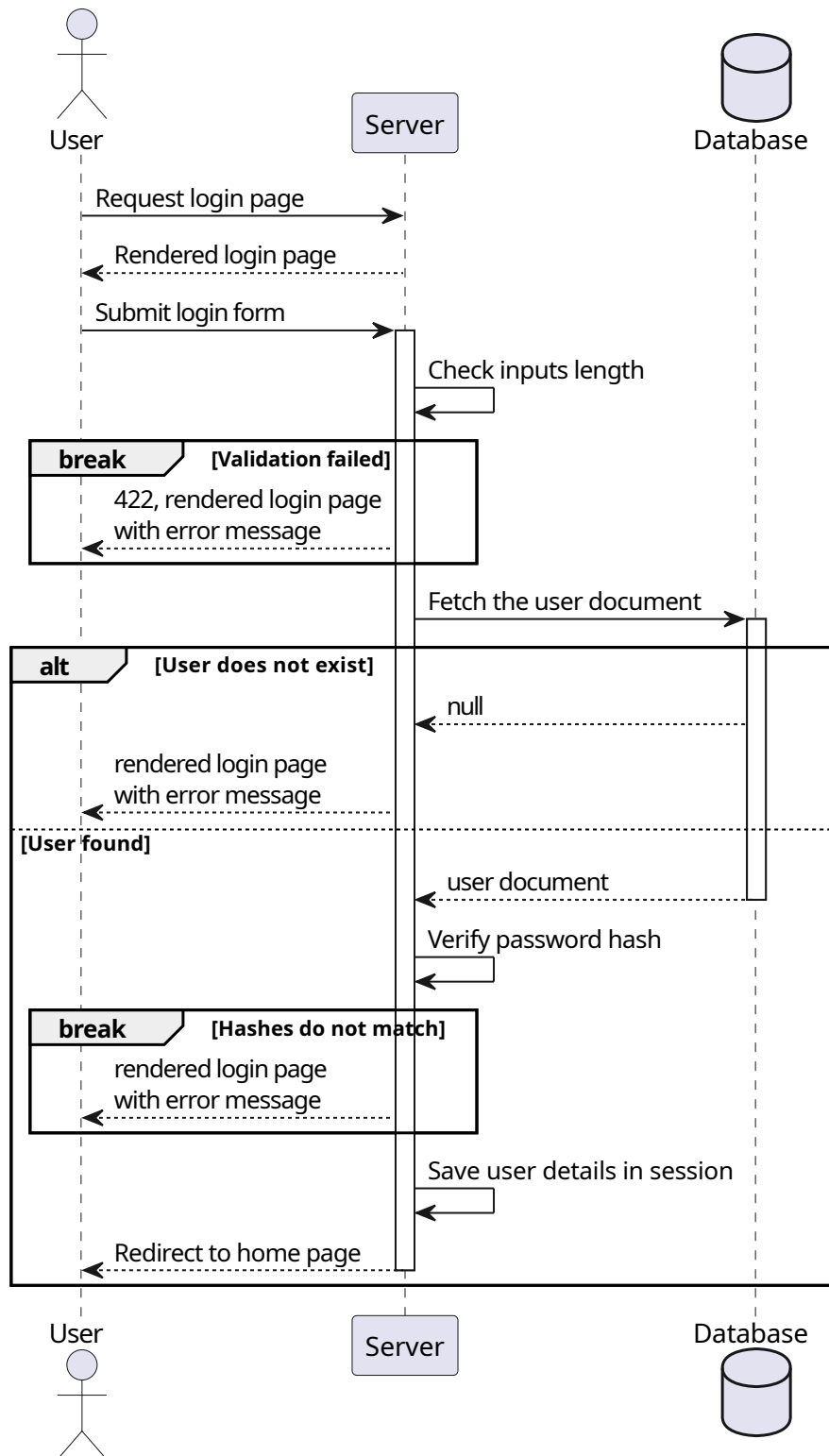Figure 5.3: Registration sequence diagram

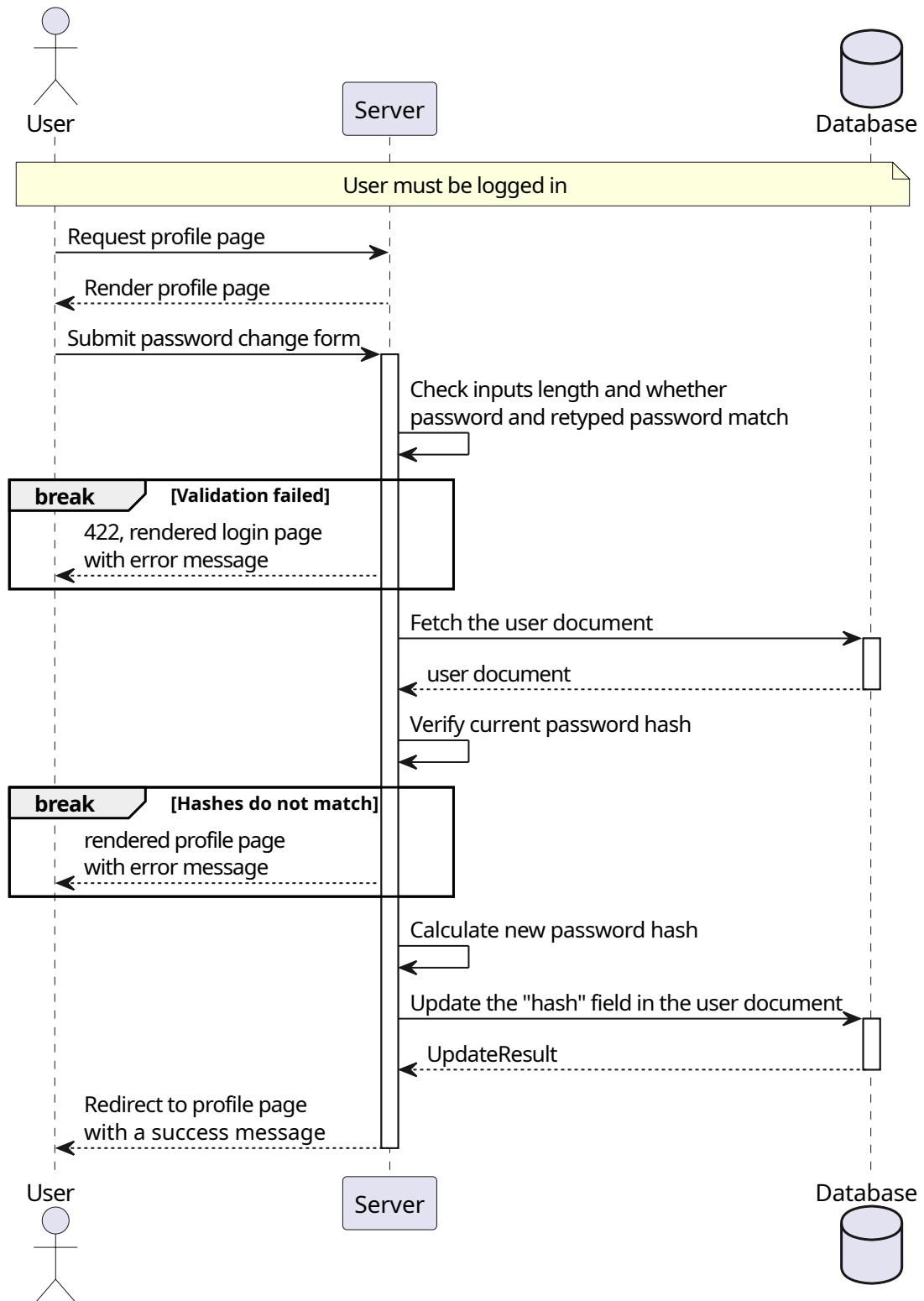Figure 5.4: Logging in sequence diagram
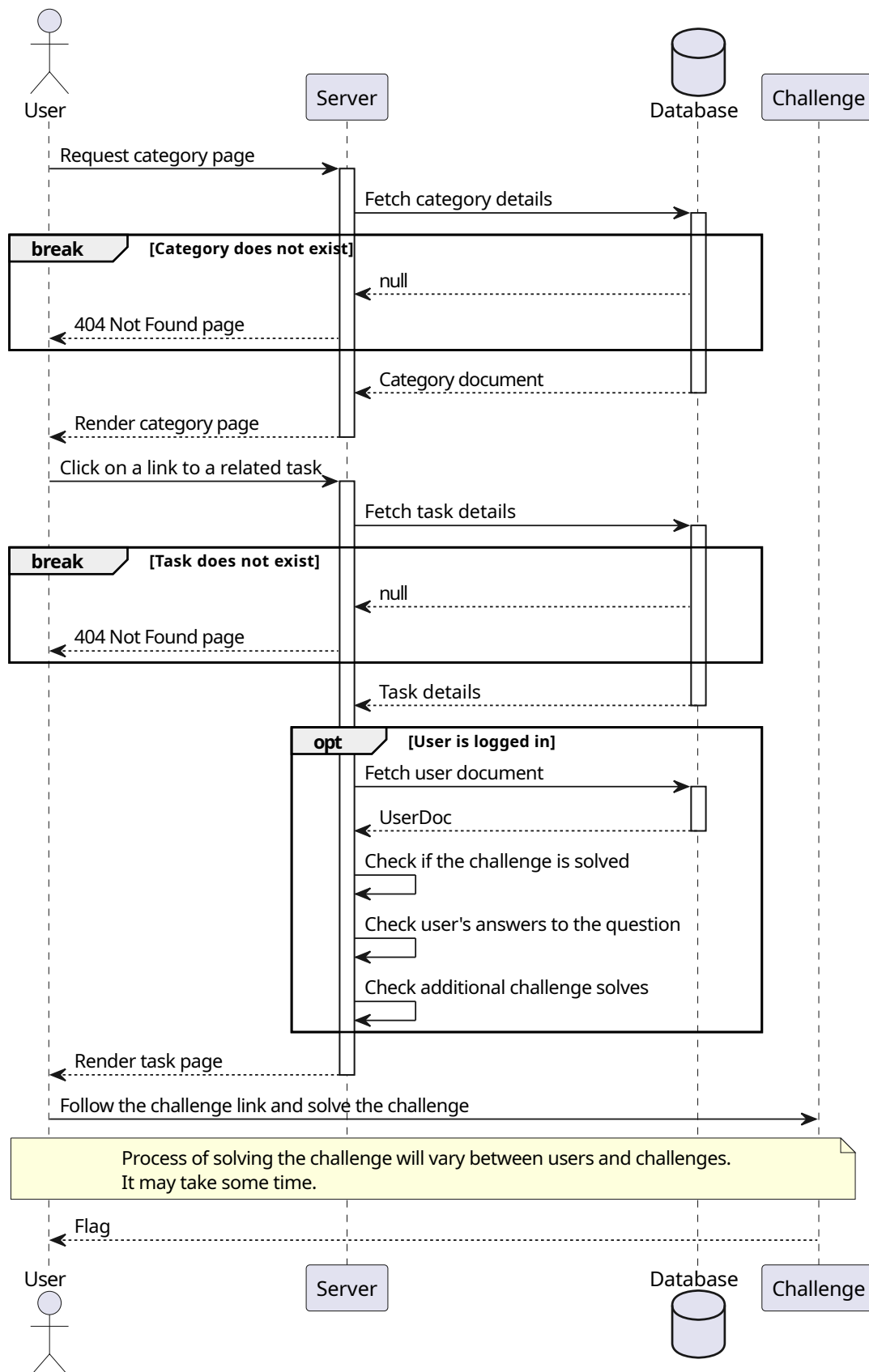
Figure 5.5: Password change sequence diagram

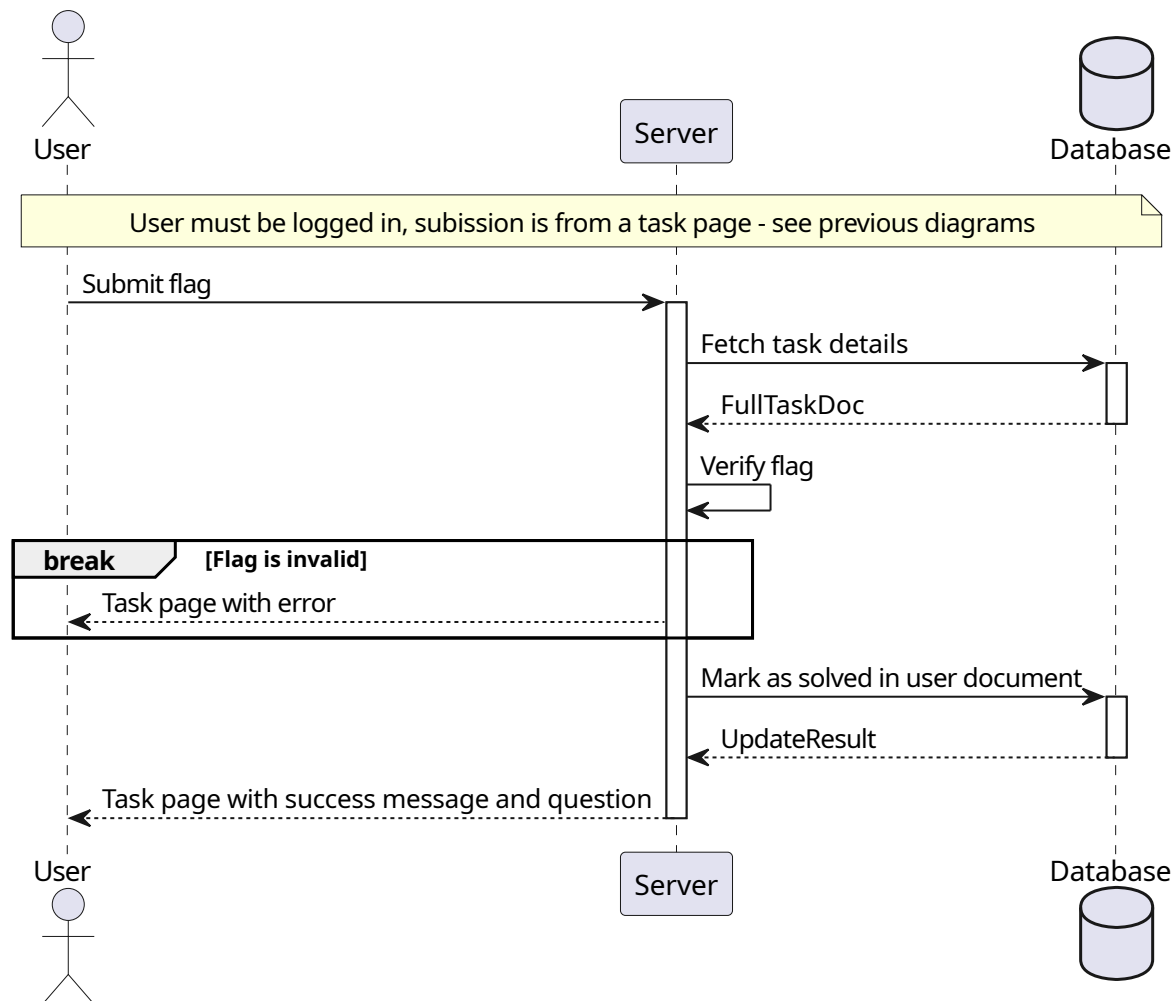Figure 5.6: Navigation and solving a task
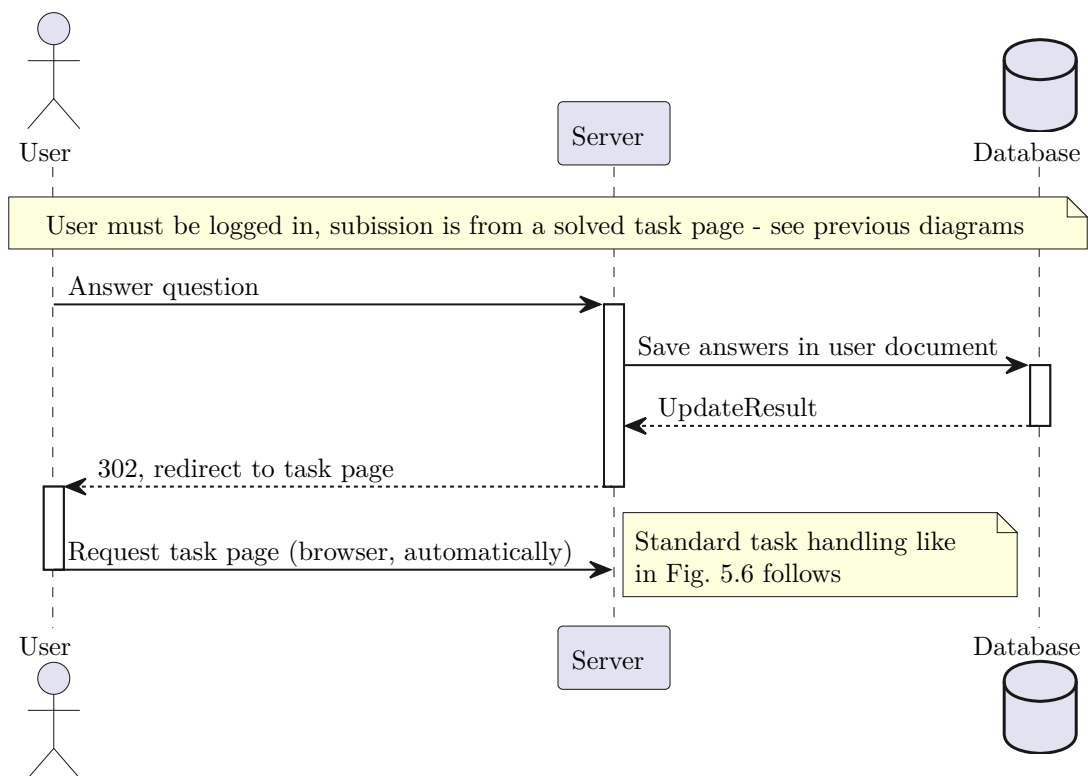
Figure 5.7: Submitting challenge flag
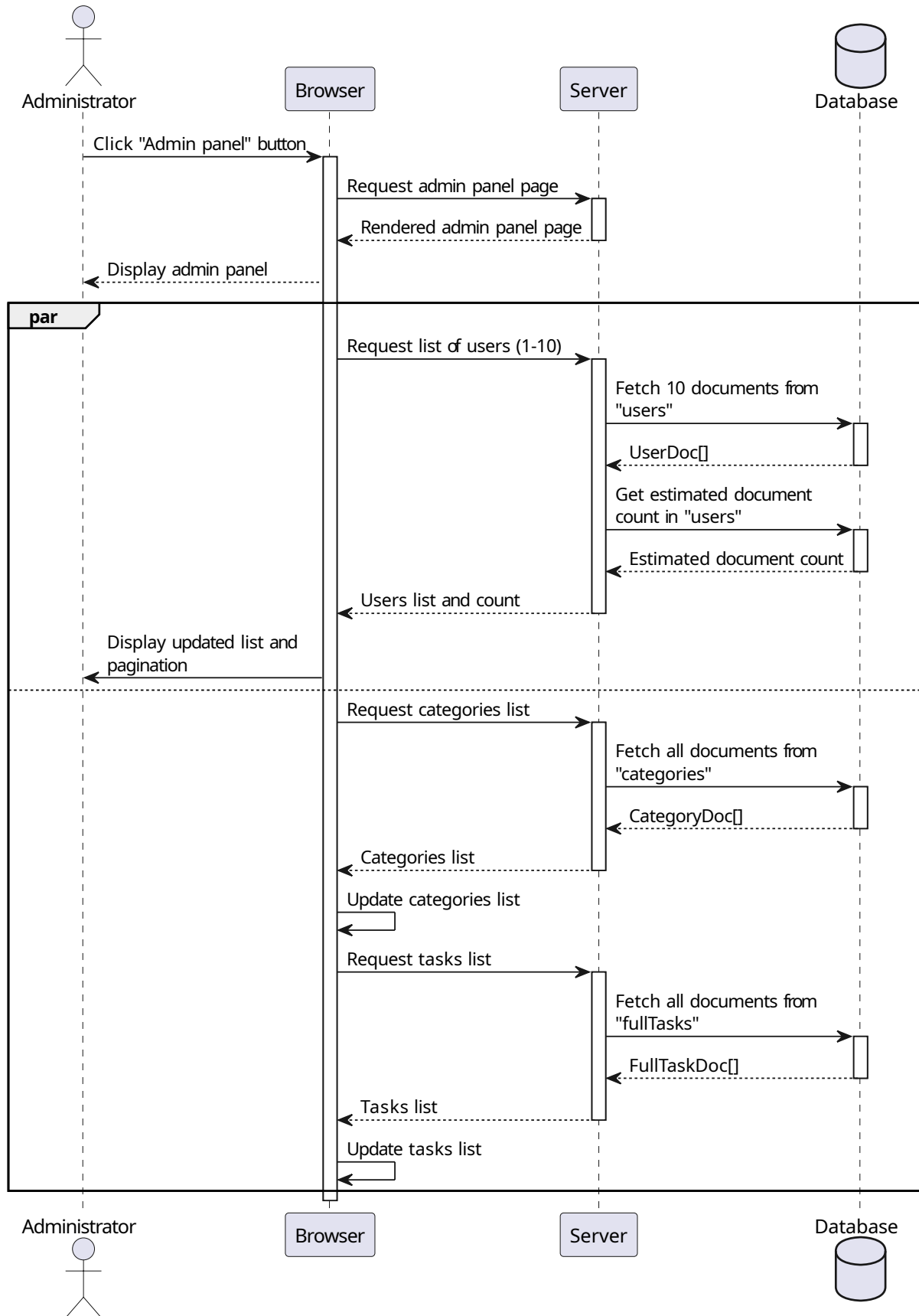
Figure 5.8: Answering task question
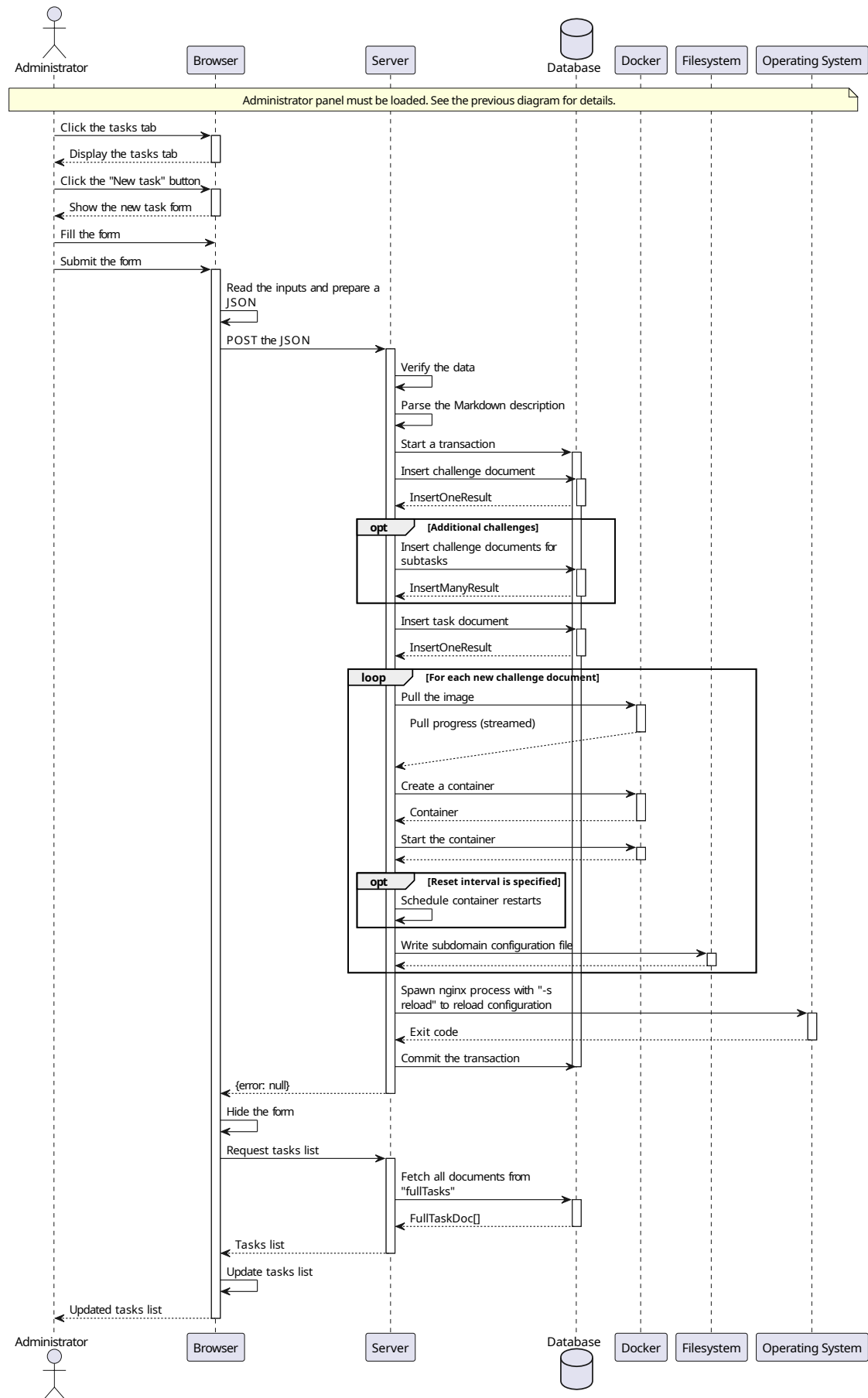
Figure 5.9: Loading administrator panel

Figure 5.10: Adding a new task

Chapter 6

Verification and validation

The solution was tested to ensure that it meets all the requirements and provides a bug-free experience.

6.1 Testing procedure

The testing was performed entirely manually. Features were tested as they were developed. The manual incremental approach was chosen due to time considerations. The project is rather small and the result of many operations is a UI rendered from an HTML template, which would make automated testing much more time-consuming. Additionally, the interface had to be visually inspected, so feature testing could be performed as a part of this procedure. Some features such as creating and restarting challenges were tested by either running small scripts or using them via the Node.js REPL and observing results in external tools.

6.2 Requirements verification

The verification of chapter 3 requirements fulfilment.

6.2.1 Verification of functional requirements

1. **Account creation**: There is an option to register a new account. Minimum password length is verified both on the client side and on server. It is impossible to register an account with the same username. Additionally, the password has to be typed twice to prevent typos. Stored password is hashed and salted. Accounts are created with the `"user"` role. The functionality is available under `/auth/register`. **Requirement met completely.**

2. **Signing in**: Possible through a login page, which is linked in the navbar and on the

home page. The functionality is available under `/auth/login`. **Requirement met completely.**

3. **Logging out**: Users can log out by clicking the logout button, which redirects to `/auth/logout`. **Requirement met completely.**

4. **Changing password**: Password can be changed on the profile page `/profile`. The previous password as well as the new password typed twice are required. **Requirement met completely.**

5. **Listing categories**: The list of categories is available in the *Categories* dropdown and on the `/categories` page. **Requirement met completely.**

6. **Displaying category**: Category pages are provided under `/category/<name>`. **Requirement met completely.**

7. **Displaying task**: All task properties specified in the requirements are displayed on task pages `/task/<id>`. **Requirement met completely.**

8. **Solving challenge**: Logged-in users can solve tasks as described in the requirements. **Requirement met completely.**

9. **Answering quiz**: Users who have solved the task challenge can answer the quiz. After answering the correctly marked answers are displayed in green. A summary of correct answers is displayed below the quiz. **Requirement met completely.**

10. **Administrator panel**: Administrator panel is available only for administrators as `/admin` page. **Requirement met completely.**

11. **Listing users**: The list of users is available under the *Users* tab in the admin panel. The list can also be accessed as JSON as a response to `GET` request to `/admin/users` with `page` and `size` query parameters. **Requirement met completely.**

12. **Changing user permissions**: User role can be toggled by clicking a button next to the user on the users list. **Requirement met completely.**

13. **Deleting user**: An account can be removed by clicking the red bin icon next to the user on the users list. **Requirement met completely.**

14. **Creating category**: Categories can be created using a form in the *Categories* tab in the admin panel. A *Preview* button can be clicked to toggle between the description input and the preview of rendered HTML. **Requirement met completely.**

15. **Editing category**: Categories can be edited by clicking the *Edit category* button below the category description in the admin panel. The same editor is used for modifying categories as for creating them. **Requirement met completely.**

16. **Creating task**: Tasks can be added as described in the requirements in the *Tasks* tab of the admin panel. **Requirement met completely.**

17. **Starting challenges**: When a task is added, the system automatically starts related challenges. Challenges containers and related nginx configuration are also configured while starting the server. **Requirement met completely.**

6.2.2   Verification of non-functional requirements

1. **Responsiveness**: The interface has been tested on multiple screen sizes and devices as well as in-browser zoom. The tested native resolutions include: 1920×1080 (24 in - desktop), 1440×900 (19 in), 1336×768 (15.6 in - laptop) and 2400×1080 (6.67 in - smartphone). Additional screen sizes were tested using the *responsive mode* option in Firefox developer tools. No issues were found during the visual inspection although some answers may cause the viewport to scroll on small screens if they contain specific HTML. **Requirement met completely.**

2. **Accessibility**: There are no accessibility errors in the webhint scan results run from Firefox developer tools. Warning are related to contrast especially in code blocks in light mode. The accessibility results of the scan are presented on Fig. 6.1. **Requirement met completely.**

3. **Visual consistency**: Styles from the Bootstrap library were used. Additional style sheets follow the Bootstrap colours. Code highlighting follows the theme variant changes (light/dark) and in both modes an appropriate variation of the Panda Syntax theme is used. **Requirement met completely.**

4. **Page load performance**: 98 points performance score in PageSpeed Insights mobile test. Test results are presented on Fig. 6.2. **Requirement met completely.**

5. **Compatibility**: The user interface has been verified to display and work correctly on Firefox, Chrome and Tor Browser (based on Firefox ESR) on Windows, Firefox on Ubuntu and Firefox on Android. Client-side form verification errors are not displayed on Firefox for Android due to bug 1510450. Bootstrap dropdowns do not work without JavaScript, so it is impossible to log out or access the profile page with JS disabled. Changing page theme also requires JavaScript, but it is not a core functionality. **Requirement met partially.**

6.3   Detected and fixed bugs

Most issues were found while writing new features and immediately fixed as a part of adding these features. Some issues, however, slipped through unnoticed and were found
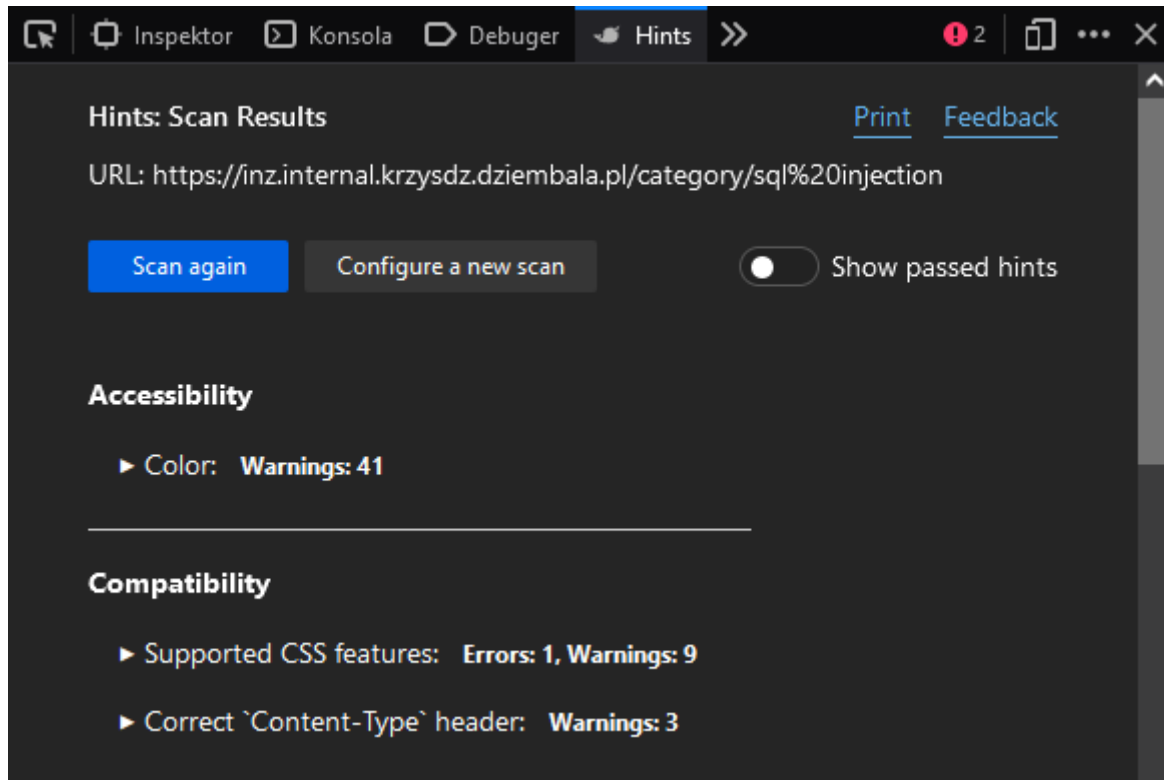
Figure 6.1: Webhint scan warnings and errors

and patched separately.

6.3.1 Containers were not started at launch if stopped

If for some reason (e.g. OS restart) challenge containers were stopped while starting the server, these would not be removed and their recreation and start would fail.

The problem was fixed in `9fd9017` by ignoring *"container already stopped"* errors when stopping a container.

6.3.2 Cookies not set in production

Cookies were not sent in responses if `NODE_ENV="production"` environmental variable was set. The problem happened, because Express does not send cookies marked as `secure` if the request was not made with HTTPS. While nginx terminates the external connections with HTTPS, it uses plain unencrypted HTTP to communicate with the server which recognizes it as an insecure protocol.

This problem was fixed in `94ca9c4` by setting appropriate headers in the proxy

```
proxy_set_header X-Forwarded-Proto $scheme;
proxy_set_header X-Forwarded-For $remote_addr;
```
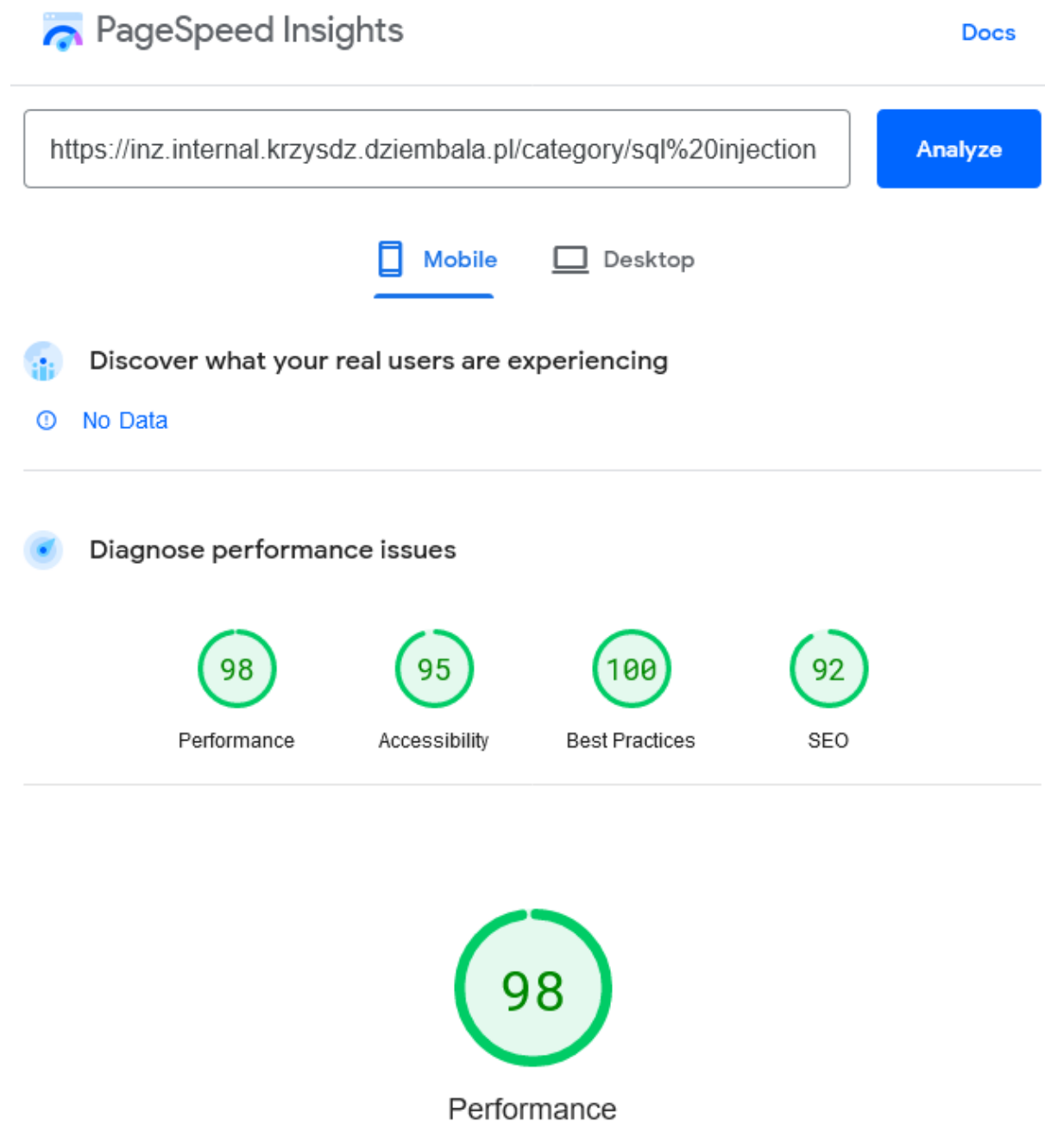
and telling Express to trust the proxy

Figure 6.2: PageSpeed Insights mobile test results

```
// In production the only way is through nginx, which sets
↪  X-Forwarded-For to $remote_addr
if (process.env.NODE_ENV === "production") app.set("trust proxy", true);
```

With this change Express recognizes HTTPS connections to the proxy as secure and sends cookies in responses.

6.3.3 Unsolved question on task page was escaped

Task questions should support HTML content and be inserted into template without escaping. In commit `b5ae1b1` support for HTML in questions and answers was added, but this particular place has been overlooked.

The problem was fixed in `91128c8` with a single line patch:

```
- <h4><%= locals.task.question %></h4>
+ <h4><%- locals.task.question %></h4>
```

6.3.4 Flag not set as an environmental variable on start

If `flagInEnv` was `true`, the flag would be exposed as an environmental variable when restarting the container, but not after creating it for the first time or restarting the server.

In this instance the problem was fixed in `99a0035` by adding the missing `Env` option to the `createContainer` call in `startChallengeContainer`.

6.3.5 Flag not set as an environmental variable for the main challenge

The `flagInEnv` option was not saved for the main challenge. The problem was discovered when testing the fix for 6.3.4.

The problem was fixed in `f934efb` by passing the missing option.

6.3.6 Task creation fails without additional challenges

If there were no additional challenges for the incorrect answers, task creation would fail, because `insertMany()` throws an error if an empty array is passed. There was an empty array check, but it had been implemented incorrectly.

The problem was fixed in `f934efb` by verifying the truthiness of array length instead of the array itself:

```
-        const subChallengeResult = answerChallengeDocs
+        const subChallengeResult = answerChallengeDocs.length
            ? await challengesCollection.insertMany(answerChallengeDocs, {
```

```
        session,
    })
```

6.3.7   HSTS header set three times per response

The `Strict-Transport-Security` header was sent 3 times with each response. The problem was detected by one of external tools while checking compatibility, accessibility and HTTPS configuration.

The problem was fixed in `eec703c` by setting the `hsts` option of the Helmet middleware to `false` and removing an `add_header` directive from the main configuration, since the shared `ssl_common.conf` configuration already contains one.

6.3.8   Accounts with `U+0000` character in username cannot be manipulated by administrators

Username is used as a part of URI path of account management endpoints. This part of the path is always encoded using `encodeURIComponent()` and can be safely sent to the server. Unfortunately, nginx since version 0.8.38 responds with status code 400 if the URI contains a NULL character, even if it is properly encoded. According to Neal Poole the nginx author decided that 'zero byte in URI should not appear in any encoding, operating system, etc., and just makes more problems than helps' and removed support for it in changeset 3527 [12].

The problem was worked around in `0cc6258` by forbidding the creation of accounts with the null character in username.

Chapter 7

Conclusions

7.1 Results

The project at least partially meets all the requirements listed in chapter 3. The only requirement not met completely is compatibility, but it should not cause problems for the overwhelming majority of users.

7.2 Future development ideas

Although the project is usable and demonstrates a general idea there is still an area for improvements. This section lists some propositions for future project development.

7.2.1 More content

The most valuable part of the system for regular users is the content - description of vulnerabilities and tasks. The currently offered examples just scratch the surface of web application security. Expanding the topics, adding more diverse tasks and introducing new categories certainly would enrich the user experience. Broadening the repertoire could make the service useful to a wider audience as well as present the vulnerabilities in more details.

Increasing the number of tasks in a single category with different difficulty levels creates a risk of a less legible and usable category pages and could discourage users less experienced with those categories. To avoid that problem the tasks could be tagged with a difficulty level and ordered by it, so everyone will be able to choose what best matches their abilities and ambitions.

7.2.2   Better task management

Current task management is restricted to task creation, which is rather complicated and involves filling a complex form properly in one try. This aspect can be greatly improved by introducing the following changes:

- allow editing existing tasks - ideally the edits would be applied separately to each part (name, description, question, challenge, etc.) of the task to reduce the complexity of changes and make the implementation simpler; this functionality could be used to fix mistakes in the text or update challenges if the image is fixed to a specific version,

- task drafts - tasks hidden from users, which may not have all the required fields filled,

- health checks and container status - monitoring the status of a challenge container and periodical checks of the application inside could help diagnose issues and prevent users accidentally or purposefully taking the tasks down; could be paired with an alert system to administrators and automated restarts,

- forced manual restart - in case a task is down or vandalized and the automatic reset is not going to happen soon, the administrators should be able to force a challenge container reset,

- time to the next restart - when the challenge will be recreated from the initial state automatically, this could be also shown on the task page to users.

7.2.3   User overview for administrators

To help with debugging possible issues user actions such as flag and answer submission could be logged, even if the flag is invalid. An overview of such events could be then checked by an administrator to verify the user findings, detect misconfigured challenges or hint the user if e.g. they found a bait flag left by another user.

7.2.4   Progress tracking

The users can see their progress for a single task when they open it. A progress bar or a colour change could be added to task tiles on category pages to indicate whether a task has been solved and to what level (challenge solved, task answered, all challenges solved).

7.2.5 Increasing user engagement

User engagement could be improved by introducing a public scoreboard and therefore an element of competitiveness. However, publishing own results should not be mandatory not to unnecessarily put pressure on privacy-focused or less confident users.

Bibliography

[1]   Razvan Beuran, Dat Tang, Cuong Pham, Ken ichi Chinen, Yasuo Tan and Yoichi Shinoda. 'Integrated framework for hands-on cybersecurity training: CyTrONE'. In: *Computers & Security* 78 (2018), pp. 43–59. ISSN: 0167-4048. DOI: `https://doi.org/10.1016/j.cose.2018.06.001`. URL: `https://www.sciencedirect.com/science/article/pii/S0167404818306527`.

[2]   Peter Chapman, Jonathan Burket and David Brumley. 'PicoCTF: A Game-Based Computer Security Competition for High School Students'. In: *2014 USENIX Summit on Gaming, Games, and Gamification in Security Education (3GSE 14)*. San Diego, CA: USENIX Association, 2014. URL: `https://www.usenix.org/conference/3gse14/summit-program/presentation/chapman`.

[3]   Stylianos Karagiannis and Emmanouil Magkos. 'Adapting CTF challenges into virtual cybersecurity learning environments'. In: *Information & Computer Security* 29.1 (2021), pp. 105–132. ISSN: 2056-4961. DOI: `10.1108/ICS-04-2019-0050`.

[4]   Stylianos Karagiannis, Elpidoforos Maragkos-Belmpas and Emmanouil Magkos. 'An Analysis and Evaluation of Open Source Capture the Flag Platforms as Cybersecurity e-Learning Tools'. In: *Information Security Education. Information Security in Action*. Ed. by Lynette Drevin, Suné Von Solms and Marianthi Theocharidou. Cham: Springer International Publishing, 2020, pp. 61–77. ISBN: 978-3-030-59291-2. DOI: `10.1007/978-3-030-59291-2_5`.

[5]   Simon Kemp. *Digital 2023: Global Overview Report*. 26th Jan. 2023. URL: `https://datareportal.com/reports/digital-2023-global-overview-report` (visited on 13/02/2023).

[6]   Bogdan Ksiezopolski, Katarzyna Mazur, Marek Miskiewicz and Damian Rusinek. 'Teaching a Hands-On CTF-Based Web Application Security Course'. In: *Electronics* 11.21 (2022). ISSN: 2079-9292. DOI: `10.3390/electronics11213517`. URL: `https://www.mdpi.com/2079-9292/11/21/3517`.

[7]   Michał Leszczyński. *Techniczne aspekty organizacji zawodów i ćwiczeń CTF*. 9th July 2018. URL: `https://cert.pl/posts/2018/07/techniczne-aspekty-organizacji-zawodow-i-cwiczen-ctf/` (visited on 16/02/2023).

[8] Ajay Nagarajan, Jan M. Allbeck, Arun Sood and Terry L. Janssen. 'Exploring game design for cybersecurity training'. In: *2012 IEEE International Conference on Cyber Technology in Automation, Control, and Intelligent Systems (CYBER)*. 2012, pp. 256–262. DOI: `10.1109/CYBER.2012.6392562`.

[9] Ahmad Nassri. *So long, and thanks for all the packages!* 14th Apr. 2020. URL: `https://blog.npmjs.org/post/615388323067854848/so-long-and-thanks-for-all-the-packages` (visited on 19/01/2023).

[10] Node.js. *Don't Block the Event Loop (or the Worker Pool)*. URL: `https://nodejs.org/en/docs/guides/dont-block-the-event-loop/` (visited on 27/01/2023).

[11] Node.js. *The Node.js Event Loop, Timers, and process.nextTick()*. URL: `https://nodejs.org/en/docs/guides/event-loop-timers-and-nexttick/` (visited on 27/01/2023).

[12] Neal Poole. *Possible Arbitrary Code Execution with Null Bytes, PHP, and Old Versions of nginx*. 24th Aug. 2011. URL: `https://nealpoole.com/blog/2011/08/possible-arbitrary-code-execution-with-null-bytes-php-and-old-versions-of-nginx/` (visited on 10/02/2023).

[13] Michał Sajdak. 'Wstęp'. In: Michał Bentkowski, Artur Czyż, Rafał 'bl4de' Janicki, Jarosław Kamiński, Adrian 'vizzdoom' Michalczyk, Mateusz Niezabitowski, Marcin Piosek, Michał Sajdak, Grzegorz Trawiński and Bohdan Widła. *Bezpieczeństwo aplikacji webowych*. Ed. by Michał Sajdak. Securitum Szkolenia, 2019, pp. 35–39. ISBN: 978-83-954853-0-5.

Appendices

Index of abbreviations and symbols

API	Application Programming Interface
BSON	Binary JSON
CLI	Command-line Interface
CSS	Cascading Style Sheet
CTF	Capture The Flag
DNS	Domain Name System
DoS	Denial-of-service
HSTS	HTTP Strict Transport Security
HTML	HyperText Markup Language
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
JS	JavaScript
JSON	JavaScript Object Notation
OS	Operating System
RCE	Remote Code Execution
REPL	Read-eval-print loop
REST	Representational State Transfer
SQL	Structured Query Language
TLS	Transport Layer Security
UI	User Interface

URL Uniform Resource Locator

 VM Virtual Machine

VPN Virtual Private Network

Listings

```
1   # mongod.conf
2
3   # for documentation of all options, see:
4   #    http://docs.mongodb.org/manual/reference/configuration-options/
5
6   # Where and how to store data.
7   storage:
8     dbPath: /var/lib/mongodb
9
10  # where to write logging data.
11  systemLog:
12    destination: file
13    logAppend: true
14    path: /var/log/mongodb/mongod.log
15
16  # network interfaces
17  net:
18    port: 27017
19    bindIp: 127.0.0.1
20
21  # how the process runs
22  processManagement:
23    timeZoneInfo: /usr/share/zoneinfo
24
25  replication:
26    replSetName: rs0
```

Figure 1: Example `/etc/mongod.conf` configuration file. Based on the default config for Debian and Ubuntu

```
1   # /etc/systemd/system/inz-nginx.service
2
3   [Unit]
4   Description=nginx proxy for inz
5   Documentation=https://nginx.org/en/docs/
6   After=network-online.target remote-fs.target nss-lookup.target
7   Wants=network-online.target
8
9   [Service]
10  Type=forking
11  PIDFile=/run/nginx.pid
12  WorkingDirectory=/inz
13  ExecStart=/usr/sbin/nginx -p /inz/nginx -c /inz/nginx/conf/nginx.conf
14  ExecReload=/bin/sh -c "/bin/kill -s HUP $(/bin/cat /run/nginx.pid)"
15  ExecStop=/bin/sh -c "/bin/kill -s TERM $(/bin/cat /run/nginx.pid)"
16
17  [Install]
18  WantedBy=multi-user.target
```

Figure 2: Example unit file for the nginx proxy. Project root is assumed to be `/inz`.

```
1   # /etc/systemd/system/inz.service
2
3   [Unit]
4   Description=inz server
5   After=network-online.target remote-fs.target nss-lookup.target
    ↪  mongod.service inz-nginx.service
6   Wants=network-online.target
7   Requires=mongod.service inz-nginx.service
8
9   [Service]
10  Type=simple
11  WorkingDirectory=/inz
12  Environment="NODE_ENV=production"
13  # Remember to change the secret!
14  Environment="SECRET=SECRET"
15  ExecStart=/usr/bin/node /inz/index.js
16  ExecStop=/bin/kill -s TERM $MAINPID
17
18  [Install]
19  WantedBy=multi-user.target
```

Figure 3: Example unit file for the main server. Project root is assumed to be `/inz`.

```
1  import { MongoClient } from "mongodb";
2  import { DB_NAME, DB_URL } from "./config.js";
3
4  const USERNAME = "administrator_username";
5
6  const client = new MongoClient(DB_URL, { directConnection: true });
7  await client
8          .db(DB_NAME)
9          .collection("users")
10         .updateOne({ _id: USERNAME }, { $set: { role: "user" } });
11 await client.close();
```

Figure 4: A simple Node.js script, which makes the user `administrator_username` an administrator.

```
1   [
2       // add challenge as "challenge" array of 1 document
3       { $lookup: {
4               from: "challenges",
5               localField: "challengeId",
6               foreignField: "_id",
7               as: "challenge",
8       }},
9       // add challenges from answers as "answerChallenges" array
10      { $lookup: {
11              from: "challenges",
12              localField: "answers.challengeId",
13              foreignField: "_id",
14              as: "answerChallenges",
15      }},
16      // select and modify returned fields
17      { $project: {
18              name: 1,
19              categoryId: 1,
20              descriptionMd: 1,
21              descriptionRendered: 1,
22              question: 1,
23              hints: 1,
24              visible: 1,
25              // Replace the "challenge" array with the only document
                ↪   inside it
26              challenge: { $first: "$challenge" },
27              // Add a challenge field (subdocument) to each answer with
                ↪   challengeId
28              answers: {
29                  $map: {
30                      input: "$answers",
31                      as: "a",
32                      in: {
33                          $mergeObjects: [
34                              "$$a",
35                              { challenge: {
36                                      $first: {
37                                          $filter: {
38                                              input: "$answerChallenges",
39                                              cond: { $eq: ["$$this._id",
                                                  ↪   "$$a.challengeId"] },
40                                              limit: 1,
41  }}}}]}}}}]}}}}}]
```

Figure 5: Aggregation pipeline for the `fullTasks` collection.

List of additional files in electronic submission

Additional files uploaded to the system include:

- source code of the application,

- source for the presented categories, tasks and challenges,

- a video file showing how the system developed for thesis is used.

List of Figures