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 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

- introduction into the problem domain
- settling of the problem in the domain
- objective of the thesis
- scope of the thesis
- short description of chapters
- clear description of contribution of the thesis's author – in case of more authors table with enumeration of contribution of authors

Chapter 2

[Problem analysis]

- problem analysis
- state of the art, problem statement
- literature research (all sources in the thesis have to be referenced [3, 2, 4, 5])
- description of existing solutions (also scientific ones, if the problem is scientifically researched), algorithms, location of the thesis in the scientific domain

Mathematical formulae

$$y = \frac{\partial x}{\partial t} \tag{2.1}$$

and single math symbols x and y are typeset in the mathematical mode.

Chapter 3

Requirements and tools

- functional and nonfunctional requirements
- use cases (UML diagrams)
- description of tools
- methodology of design and implementation

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 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 [1]. 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

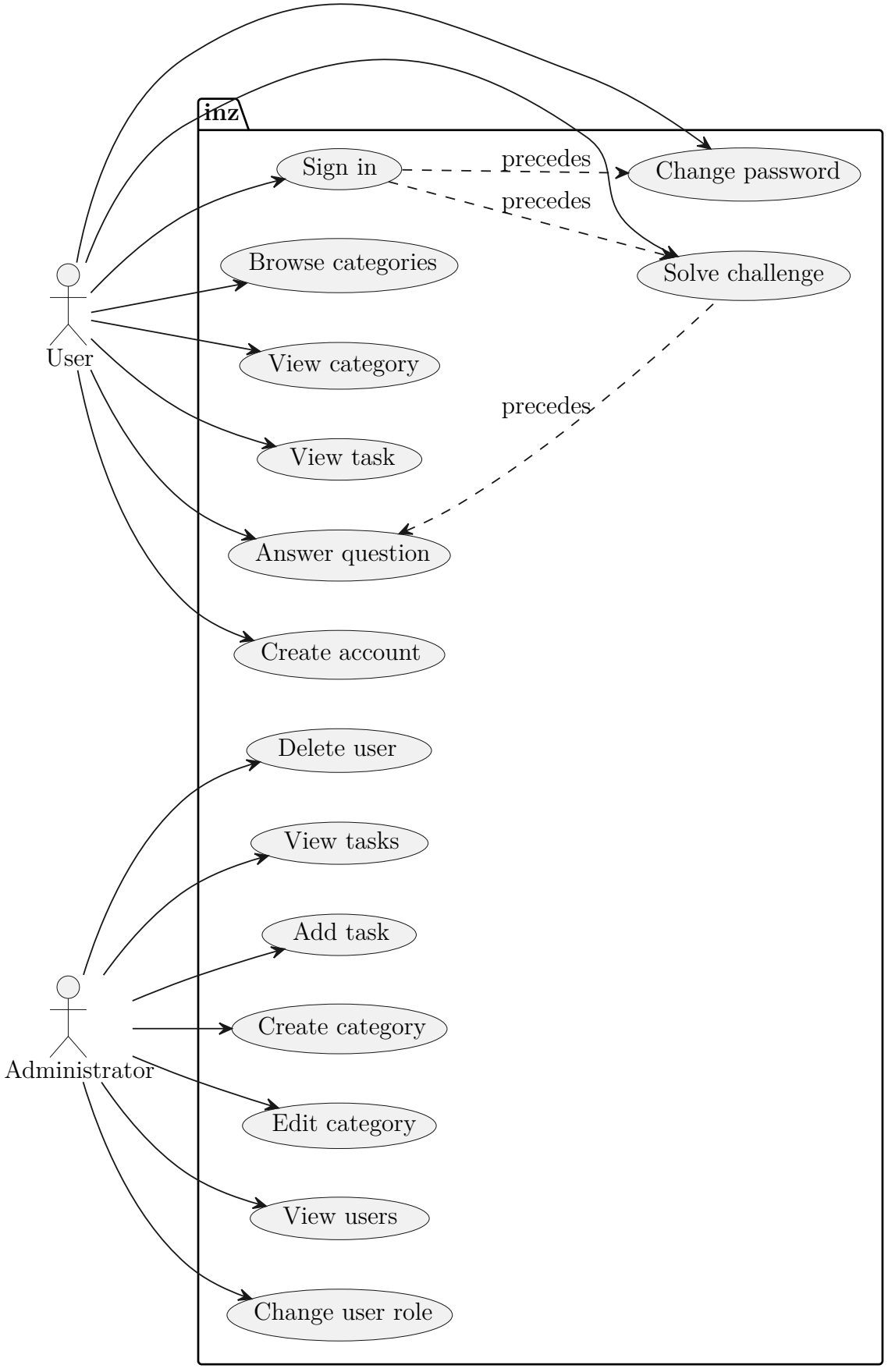

Figure 3.1: Use case diagram

mongosh were used. These tools allow database management and, except for mongosh, provide helpful user interfaces.

Docker

Docker is a platform for managing containerised software. Challenges are running as Docker containers and managed using Docker Engine API. For debugging, development 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 synchronised 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 Plugin, 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 plugin the tool could be used as a default formatter in the editor. A plugin for ESLint allowed marking improperly formatted code as lint error.

Chapter 4

External specification

- hardware and software requirements
- installation procedure
- activation procedure
- types of users
- user manual
- system administration
- security issues
- example of usage
- working scenarios (with screenshots or output files)

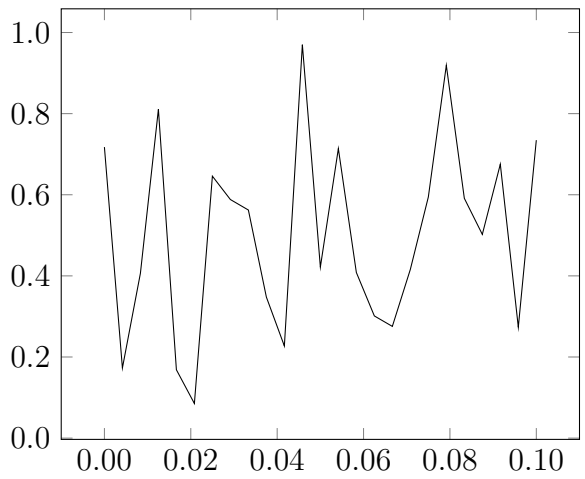

Figure 4.1: Figure caption (below the figure).

Chapter 5

Internal specification

- concept of the system
- system architecture
- description of data structures (and data bases)
- components, modules, libraries, resume of important classes (if used)
- resume of important algorithms (if used)
- details of implementation of selected parts
- applied design patterns
- UML diagrams

Use special environments for inline code, eg `int a;` (package `minted`) . Longer parts of code put in the figure environment, eg. code in Fig. 5.1 . Very long listings–move to an appendix.

```
class test : public basic
{
   public:
     test (int a);
     friend std::ostream operator<<(std::ostream & s,
                                    const test & t);
   protected:
     int _a;

};
```

Figure 5.1: Pseudocode in `minted`.

Chapter 6

Verification and validation

- testing paradigm (eg V model)
- test cases, testing scope (full / partial)
- detected and fixed bugs
- results of experiments (optional)

Table 6.1: A caption of a table is **above** it.

	method						
			alg. 3			alg. 4, $\gamma = 2$	
ζ	alg. 1	alg. 2	$\alpha = 1.5$	$\alpha = 2$	$\alpha = 3$	$\beta = 0.1$	$\beta = -0.1$
0	8.3250	1.45305	7.5791	14.8517	20.0028	1.16396	1.1365
5	0.6111	2.27126	6.9952	13.8560	18.6064	1.18659	1.1630
10	11.6126	2.69218	6.2520	12.5202	16.8278	1.23180	1.2045
15	0.5665	2.95046	5.7753	11.4588	15.4837	1.25131	1.2614
20	15.8728	3.07225	5.3071	10.3935	13.8738	1.25307	1.2217
25	0.9791	3.19034	5.4575	9.9533	13.0721	1.27104	1.2640
30	2.0228	3.27474	5.7461	9.7164	12.2637	1.33404	1.3209
35	13.4210	3.36086	6.6735	10.0442	12.0270	1.35385	1.3059
40	13.2226	3.36420	7.7248	10.4495	12.0379	1.34919	1.2768
45	12.8445	3.47436	8.5539	10.8552	12.2773	1.42303	1.4362
50	12.9245	3.58228	9.2702	11.2183	12.3990	1.40922	1.3724

Chapter 7

Conclusions

- achieved results with regard to objectives of the thesis and requirements
- path of further development (eg functional extension ...)
- encountered difficulties and problems

Bibliography

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

[2] Name Surname and Name Surname. *Title of a book*. Hong Kong: Publisher, 2017. ISBN: 83-204-3229-9-434.

[3] Name Surname and Name Surname. 'Title of an article in a journal'. In: *Journal Title* 157.8 (2016), pp. 1092–1113.

[4] Name Surname, Name Surname and N. Surname. 'Title of a conference article'. In: *Conference title*. 2006, pp. 5346–5349.

[5] Name Surname, Name Surname and N. Surname. *Title of a web page*. 2021. URL: `http://somewhere/on/the/internet.html` (visited on 30/09/2021).

Appendices

Index of abbreviations and symbols

CTF	Capture The Flag
UI	User Interface
SQL	Structured Query Language
CSS	Cascading Style Sheet
HTML	HyperText Markup Language
JS	JavaScript

Listings

(Put long listings here.)

```
1 if (_nClusters < 1)
2         throw std::string ("unknown number of clusters");
3 if (_nIterations < 1 and _epsilon < 0)
4         throw std::string ("You should set a maximal number of iteration
  ↪  or minimal difference -- epsilon.");
5 if (_nIterations > 0 and _epsilon > 0)
6         throw std::string ("Both number of iterations and minimal epsilon
  ↪  set -- you should set either number of iterations or minimal
  ↪  epsilon.");
```

List of additional files in electronic submission (if applicable)

Additional files uploaded to the system include:

- source code of the application,
- test data,
- a video file showing how software or hardware developed for thesis is used,
- etc.

List of Figures

List of Tables