# Gamification project

Andreas Hanjua, Lenny Händler

October 13, 2019

# Contents

# Chapter 1

# Introduction and project overview

An important part of modern cybersecurity research is the human factor in IT systems. Few years ago many companies started trying to minimize the vulnerability of human failure as a core vulnerability for most of their security critical systems. For this purpose they educated their employees with often expensive seminars and professional cybersecurity experts.

With this project we try to research a different technique of educating employees by developing an educational cybersecurity game-infrastructure. By making the system userfriendly and playful we hope to motivate users to learn cybersecurity subjects in order to success in the competitive game environment. We think that such an approach could be more cost effective and could archieve better results then cybersecurity seminars. Also both approaches can be combined to deepen knowledge, which could be interesting for eg. cybersecurity students.

Our infrastructure consists of a central user server, which manages the registered users and their scores. As a first example application it also contains a quiz application, where users can challenge each other to compete in answering cybersecurity related questions for gaining score points.

Further possible applications could be a capture the flag like games or for example a "phishing" application, that test employees behaviour by sending phisihing mails to them. It then could track their reaction and give/take score points of them.

The handing over state coveres the userserver and a first version of the quizserver. In agreement to Oliver Schranz some features of the quizserver are not fully developed and are replaced with placeholder systems. The

quizserver should act as a first minimal working product, to research its efficiency and propose a first design draft.

# Chapter 2

# Project Structure and development

The most intuitive seperation of work is the split in back-end and front-end.

The back-end consitst out of a database and two servers. One server for tracking user accounts (refered as userserver), and one for the quiz back-end (refered as gameserver). Both Servers use the framework express and provide a REST API implemented in Node.js. We decided us for Javascript, as REST APIs are a typical usecase for it's usage. Out of the scope of the project we made an admin UI in C# using Unity3D, which we built for Android and Windows and which is also deployable on Linux and IOS. This part was the work of Andreas Hanuja.

The front-end is programmed using the react framework. This part was the work of Lenny Händler.

On both parts we helped each other with testing, finding and fixing bugs, application design and UI design. But in general we split the implementation and documentation work strictly to our scopes.

# Chapter 3

# Deployment

The front-end expects the routes of both back-end server to be on the same domain as the front-end. This can be adjusted, with a complimentary same-origin-policy, or all servers can be put on the same origin via a reverse proxy, we recommend this solution.

To deploy the back-end you first have to setup the database. for this purpose we have a dump of our database delivered in the back-end folder. Our configuration was:
- DBNAME=CYSECPROJECT
- DBUSER=root
- DBHOST=localhost
- DBPASSWORD=testpassword

If you change this configuration you also have to reconfigure the servers. After setup the database you can start the servers by executing the start.bat on windows or the start.sh. in the subfolders game/user. If you want change some configuration, edit the startfile bevore executing it. You find a list of the propertys at the end of this chapter.
If starting the servers result in errors, please double check if all node modules are installed.

| property | description |
|---|---|
| PORT | Default port of the server. |
| GAMESERVER | Full address of the gameserver. |
| DBNAME | Configured name of the database. |
| DBUSER | Configured user of the database. |
| DBHOST | Host of the database.height<br>Host it on localhost is recommended. |
| DBPASSWORD | Configured password of the database. |
| ADMINUSER | ID of the Adminuser. Check twice if valid!<br>If invalid all users have admin access. |
| TOKENDECAYTIME | Time for auth. tokens to decay.<br>As they can also get revoked a big value<br>is recommended. |

# Chapter 4

# Front-end structure

We created several components that manage the api calls for different parts of the application. They can be found in different routes, which are registered in App.js. The paths can be found in the frontend-react/src/App.js.
Some design elements that apply globally are in index.css, while some more component specific design tags are set inside of the components.

The nav bar is found in App.js, the overview of a game is shown in Game.js. Playing a round is split up into two components. One component loads the list of questions (Round.js) while another is loading the questions and sending the answers (Question.js). When a question is answered the round is informed via a callback.
The start page (Startpage.js) presents the user a button to start matchmaking and a list of all ongoing games.
The score board (Scoreboard.js) shows a sorted view of all users, with buttons to challenge them to a match.
history.js displays all finished games.

# Chapter 5

# Back-end structure

## 5.1 Files and Folders

The backend is devided in user and gameserver. Both servers have a similar structure. In the backend folder you find some documentation, which could be usefull to understand the code, but is not the official updated documentation for the project. Also there is the folder of the game and userserver. In the folder the follwing scripts are located:
- index.js: It defines the staring behaviour and which other .js files are used for paths.
- maria.js: Provides functionality to other scripts, to interact with the Maria DB.
- authentication.js: Provides functionality to other scripts, to authenticate users.
The other scripts provide paths that are used at administrative purpose. The scripts in the subfolders provide paths that are used by the frontend. All paths are documented in the next chapter, so we can scip this here.

## 5.2 "Objects" and terminology

In this section we talk about which "objects" our backend works with. Usually this objects are stored in the corresponding database tables. The backend takes them, apply a received request and encoded them in dictionarys to send them to the frontend, which interprets and display them.

### 5.2.1 Userserver

The userserver works only with users. The authentication system is still missing an authentication method, because we agreed to let this open for later implementation. After authenticating with this method the user will receive a 180Bit cryptographic secure random token. With all later requests the user send this token in a header and the server will verify the token to authenticate and identify the user. The important properties of a user are explained in Section 5.4.

### 5.2.2 Gameserver

The gameserver usually work with games. A game is created by a user and an other user can join the game. The matchmaking is also open for later implementation and can be implemented in the game.js file. When playing the game rounds will be created. To create a round the creator of the round (alternating, beginning with user2) has to choose one of 3 random categories. After 6 rounds are played the game will be finished and closed. One round consists out of 3 random questions and each question can have a variable amount of answers. The answers can individually be wrong or correct. Also multiple and all answers could be correct.
Its important that for each category there exists at least 3 questions and that there are at least 3 categories. Else the server will crash when trying to play, as we consider this not as a program error, but an error of the administrator. All important properties of the objects are explained in Section 5.4.

## 5.3 Paths

Information about the properties is given in Section 5.4.
xyz
:= replace xyz with parameter. body: name, score := Encode body in json with the properties name, score.

### 5.3.1 Userserver

Manage users:

| type | path | body | description |
|---|---|---|---|
| GET | /api/users | | Array of all users |
| GET | /api/users/?sortBy=id | | Array of all users, primary sorted by id |
| GET | /api/users/?sortBy=name | | Array of all users, primary sorted by usernames |
| GET | /api/users/?sortBy=score | | Array of all users, primary sorted by score |
| GET | /api/users/?sortBy=level | | Array of all users, primary sorted by level |
| GET | /api/users/[ID] | | Get a specific user |
| POST | /api/users | name, score, level | Create an user |
| PUT | /api/users/[ID] | name, score, level | Edit an user |
| DELETE | /api/users/[ID] | | Delete an user |

Authentication:

| type | path | body | description |
|---|---|---|---|
| POST | /api/authentication | userID | Generates and shows the authentication token of a user. Resets the decay time. (placeholder, as all users can perform this) |
| DELETE | /api/authentication/[ID] | | Revokes the authentication token of a user. |
| GET | /api/authentication/[token] | | Returns the user of a valid token (for testing). |

Others:

| type | path | body | description |
|---|---|---|---|
| GET | /api/ping | | Returns "pong" or "pong admin" if an admintoken was used. |

### 5.3.2 Gameserver

Manage questions:

| type | path | body | description |
|---|---|---|---|
| GET | /api/questions | | Array of all questions |
| GET | /api/questions/ ?containAnswers=true | | Array of all questions containing coresponding answers |
| GET | /api/questions/[ID] | | Get a specific question |
| GET | /api/questions/[ID] ?forRound=[RoundID] | | Get a specific question, use extra parameter for backend to manage timeouts. |
| GET | /api/questions/[ID] ?containAnswers=true | | Get a specific question with it's answers |
| POST | /api/questions | text, categoryID, requiredLevel, score, answerTime | Create a question |
| PUT | /api/questions/[ID] | text, categoryID, requiredLevel, score, answerTime | Edit a question |
| DELETE | /api/questions/[ID] | | Delete a question |

Manage answers:

| type | path | body | description |
|---|---|---|---|
| GET | /api/answers | | Array of all answers |
| GET | /api/answers/[ID] | | Get a specific answers |
| POST | /api/answers | text, questionID, isCorrect | Create a new answer |
| PUT | /api/answers/[ID] | text, questionID, isCorrect | Edit an answer |
| DELETE | /api/answers/[ID] | | Delete an answer |

Manage categories:

| type | path | body | description |
|---|---|---|---|
| GET | /api/categories | | Array of all categories |
| GET | /api/categories/[ID] | | Get a specific category |
| POST | /api/categories | name, requiredLevel | Create a new category |
| PUT | /api/categories/[ID] | name, requiredLevel | Edit a category |
| DELETE | /api/categories/[ID] | | Delete a category |

Manage games:

| type | path | body | description |
|---|---|---|---|
| GET | /api/games | | Array of all games |
| GET | /api/games/[ID] | | Get a specific game |
| GET | /api/games/[ID] ?containsFullHistory=true | | Get a specific game and all rounds with all details for the game |
| GET | /api/games/current | | Get all opened games for the current user |
| GET | /api/games/history | | Get all closed games for the current user |
| POST | /api/games/current | | automatches it with matchmaking and creates new game if necessary |
| POST | /api/games/current ?matchWith=[otherP.ID] | | matches with specified player, creates new game if necessary |
| DELETE | /api/games | | Deletes all rounds and games. USE ONLY FOR TESTING! |

Manage rounds:

| type | path | body | description |
|---|---|---|---|
| GET | /api/rounds/[ID] | | Get a specific round |
| GET | /api/rounds/ ?forGame=[GameID] | | Get an open round for a specified open game. If all rounds are closed and the correct user requests (alternating) a selection of categories is sent back |
| PUT | /api/rounds/[roundID] | answerID | Sends an answer to a round. If the user has timeouted the question you can send -1 |
| PUT | /api/rounds/[gameID] | categoryID | Sends a category to the game and creates a new round in the new category |

Others:

| type | path | body | description |
|---|---|---|---|
| DELETE | /api/authentication/token] | | Deletes a token from the gameserver cache. It will query it from the userserver again, when requested |
| GET | /api/ping | | Answers with pong |

All paths have a specific authentication rule. In general the token, which corresponds to the user, that is corresponding to the userID, which is chosen as admin user, has all privileges. All other paths are restricted to only users, that has to call the path to play the game. If something fails a respond with an error code will contain a small error description.
Only exception is the path /api/authentication which everybody can call without a valid token and which will tell you the token of a selected user. This has to be changed in authenticationAPI.js as soon as a kind of authentication is implemented. Currently it is necessary that everybody can call this, to avoid a chicken egg problem, considering the admin token.

## 5.4 Admin interface

The admin interface is out of scope of the agreed scope. It is developed in Unity3D, as I (Andreas Hanuja) am very incorporated in Unity. Also for the admin interface a built solution, which you have to install, is more practicable then for the fron-tend user solution.

### 5.4.1 Setup and generall usage

On Windows you can just launch the included .exe file in the MBC_Build folder. On Android you can install and launch the provided .apk file, if you have unknown sources enabled on your phone.
After staring the application the first time, you have to configure the credentials. Click on configure credentials and enter the admin token and the full server addresses, including ports. For local network eg. "http://192.168.0.42:3000". Then click "Save changes" to apply your settings. The indicators at the bottom left will show if a connection is possible.
For the general navigation you can always press on the MBC logo to return to he main menu. Only if the needed server connection is provided you can enter the administrate menus. If you enter these menus you have a scrollview in the blue area of the screen. You can scroll onto it if you swipe your screen without touching a button or if you hold and drag the small scrollbar at the right.

### 5.4.2 Important rules

- You can not undo delete actions, so pay attention what you are deleting.
- If you delete something usually related data is not deleted. This means on the one hand, that if you delete a category by accident, you do not loose all questions and answers of this category, they are only not usable currently. On the other hand the data is still present in the database, so if you want keep your database clean always delete related data before delete data.
- There have to be at least 3 categories - Each category always have to have at least 3 questions for each existing min. level! - Each question always have to have at least 1 answer. - Usernames should not contain [, ], ", ,

### 5.4.3 Usage and functionality

For administrating users press "administrate users" in the main menu. You can scroll and inspect users in this submenu. The UI uses the back-end path GET api/users here. You can edit a specific user by clicking the edit icon, or you can add a new user with the button at the bottom of the scrollview. The username has not to be unique, but you should keep them unique to provide a good user experience. Also you should not change the name of a user, without informing him.
The score are the current points of the user.
The level is an integer which represents the skilllevel of the user. A user will always just get questions and categories with a skilllevel $\leq$ his own level.

By clicking on administrate games, you get an overview over all played and active games. The UI uses GET api/games to get all games including the names of all players. Then it will query additional information for each individual game, which could consume much bandwidth and could overload the server in the future, when many many games are played.
The truncate button will delete all games and rounds forever, so just use for testing and with care.

By clicking on administrate questions, you get an overview over all categories and questions. With the small scrollbutton, besides the category, you can change the current displayed category. On the last page you also get additional options, like adding a new category. Use an edit button to edit an element.
A category consists out of a name and a level.
A question consists out of a question text, an integer, which indicates how much points the question is worth. A level which affects who can play this question. A player can get this question if his playerLevel is $\leq$ the questionlevel and playerLevel $\leq$ categoryLevel. Also a question contains the time in seconds, which the user have to answer the question.
The answers are attached to specific questions and can be either correct or wrong. They also contain a text which can be changed.

### 5.4.4 Further work

In the admin UI currently the possibility to cleanup the database is missing. If you eg. delete a question without deleting its answers before, they stay in the database without getting used. You could edit them manually with the back-end paths to recover them, but a button which finds unused data and deletes it would be a cool idea. You can simply add it to the main menu, by instantiating the button prefab into the main Menu.
Implementing the logic to find and remove the unused data will be not so easy, look in chapter 5.3.2 for details which paths can be used.

As mentioned above the authentication and the matchmaking are missing in the back-end.
The authentication can has to be integrated in our authentication system, to control if a user gets his token. The matchmaking system has to be connected to the game creation in game.js.

An other thing that is open, is the exact algorithm for determine how

many points a user get for winning a game. Our currently very simple algorithm can be changed in the function doScoring of rounds.js in the gameserver.