Crash & Compile

Manual

Version	:	v1.4
Date	:	June 9, 2020
Managers	:	Jan Ondruch (h.ondruch@seznam.cz)
		Mark Wijkhuizen (M.Wijkhuizen@student.ru.nl)
Developers	:	Ruben Holubek (r.holubek@student.ru.nl)
		Ciske Harsema (c.Harsema@student.ru.nl)
		Leon Driessen (L3.Driessen@student.ru.nl)
		Stijn van den Beemt (stijn.van.den.beemt@student.ru.nl)
		Gerhard van der Knijff (c.vanderknijff@student.ru.nl)
		Josien Visschedijk (j.visschedijk@student.ru.nl)
		Steven Maarse (S.Maarse@student.ru.nl)

Contents

1 Introduction

Within Thalia, there are many organized events, amongst others *Crash & Compile* (hereafter called 'C&C'). This game where you can get drunk whilst playing a game is a yearly returning event. This end-to-end manual describes the whole setup for the C&C event.

2 Manual

For this event, a so-called **admin dashboard** is created. This tool gives an admin full control over the teams, scores and the game during the event. Furthermore there is a C&C **website**, which is used to place announcements, to make team registration possible, etc. Finally, a **real-time scoreboard** is developed where scores of all teams during the event are shown.

An API is deployed where all endpoints are included for completing all actions described in this manual. For deploying the application, *NestJS* (backend) and *Angular* (frontend) are used. To access all API endpoints without using the admin dashboard, software like *Postman* can be used. Not all actions described within the manual can be directly executed through the C&C tool, so it is advised to use *Postman*, the *Swagger* documentation, or any other tool to generate requests. Examples are: deleting a user, deleting a HTML file for announcements, etc.

2.0.1 Swagger documentation

The documentation regarding the API is described in Swagger. To view the Swagger documentation one first needs to run the backend (see 2.5). Then one can reach the swagger documentation through the link:

```
http://localhost:8080/api
```

Here you can find all the endpoints, a short description of the endpoint, the response formats and codes and the request format. It is also possible to generate requests by trying out an endpoint. Endpoints marked with a padlock require authentication. To use them one must first log in via the `POST /auth/login` endpoint, which returns a token. Copy this token, and click on authorize on the top of the Swagger documentation page, and paste the token. You should now be able to use authorized endpoints.

It is also possible to generate a Swagger JSON file by visiting the following link:

```
http://localhost:8080/api-json
```

2.1 General

This manual consists of three parts: a description about the flow of the competition, a guide about integrating a game into the application, and specific

information regarding the setup and deploying of the application. We shall describe both the use case of a real life game, and that of a created minigame.

2.2 Crash & Compile competition

There are three different types of users within the C&C event:

1. Admin, which is the organizer of the C&C event, and thus wants to control the whole event by means of the admin dashboard. An admin also controls the homepage.
2. Team, which is formed from players participating in the C&C event. Teams mainly use the minigame, and parts of the homepage.
3. Minigame, which is a pseudo user intended for automatic minigame integration.

For full privileges and roles see table 1.

2.2.1 Phases

The competition can be divided into 5 different phases

1. *Setup*; This is the phase before the game has started. In this phase the admins can prepare the event, for example by preparing the questions for a minigame (see Section 2.2.3). This is the default phase when starting the C&C tool.
2. *Publish Info*; This is the phase where the first announcements about the game are published (by the admins). At this points, team registration is not open.
3. *Registration*; This is the phase where team registration is open and by means of a *team name* and *password*, a team can enter the C&C event on the C&C website.
4. *Game Running*; This is the phase where the game is started and team can gain scores by means of the game.
5. *Freeze*; This is the phase where the minigame is frozen and scores cannot be updated, for instance, when an admin wants to do a (live) announcement.

When initially starting the C&C tool, one is redirected to the homepage of the C&C website, shown below.

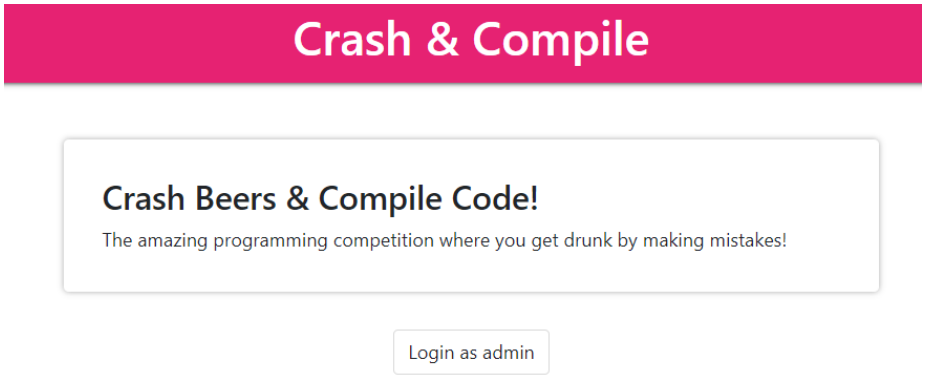

Figure 1: Homepage with phase *Setup*

This specific view has to do with the initial phase *setup*. As an admin you can log in here to access the admin dashboard. This by clicking on `Login as admin`.

2.2.2 Login

[Phase(s): Setup]

For the login, a *JWT Bearer* is used in order for a user to authenticate themselves before and during the C&C event. A token is generated for each login and bounded to a user. This is done by means of Passport JWT. For each call to the API, the token in combination with the role (see table 1) is used for authentication.

There are two types of logins: an *admin login* and a *team login*. When an admin wants to login, the following overview is seen:

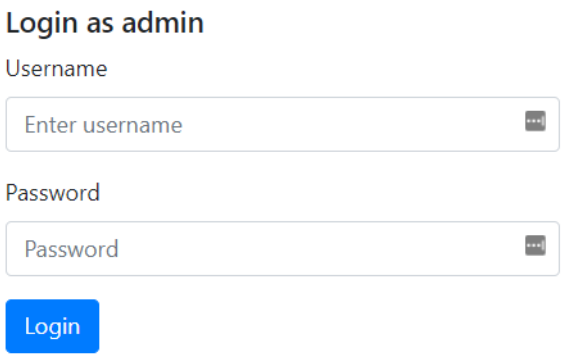

Figure 2: Admin login

For controlling the password of the admin, see section 2.4.5.

After a successful admin login, there are four different buttons shown:

- Score manager: In this section, the scores of the teams can be altered and teams can be made (in)visible.
- Phase manager: In this section, the different phases can be set and an optional timer when to freeze the event.
- Question manager: In this section, the questions and question-sets can be managed.
- Info manager: In this section, the announcements and style of the said announcements can be managed.

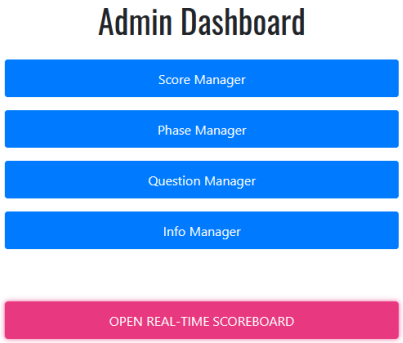

Figure 3: Admin Dashboard

An important component is the *Phase manager*, where phases can be changed, which has a direct influence on the functionality of the C&C tool. The phase manager is shown below:

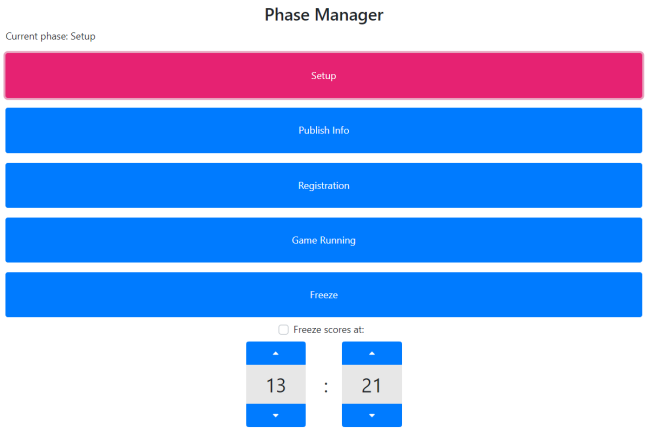

Figure 4: Phase Manager

As can be seen, the current phase is highlighted.

2.2.3 Prepare competition

[Phase(s): Setup]

The initial phase is the phase `Setup`. When preparing the competition, the admins can prepare the different questions and question-sets that can be used in the competition. They can also write the announcements that can be seen by teams before the competition. Finally, they can manage the style of the C&C tool.

Question manager

The question manager - as seen in Figure 3 - is used to add, delete or change questions or question-sets. An overview of the question manager is shown below:

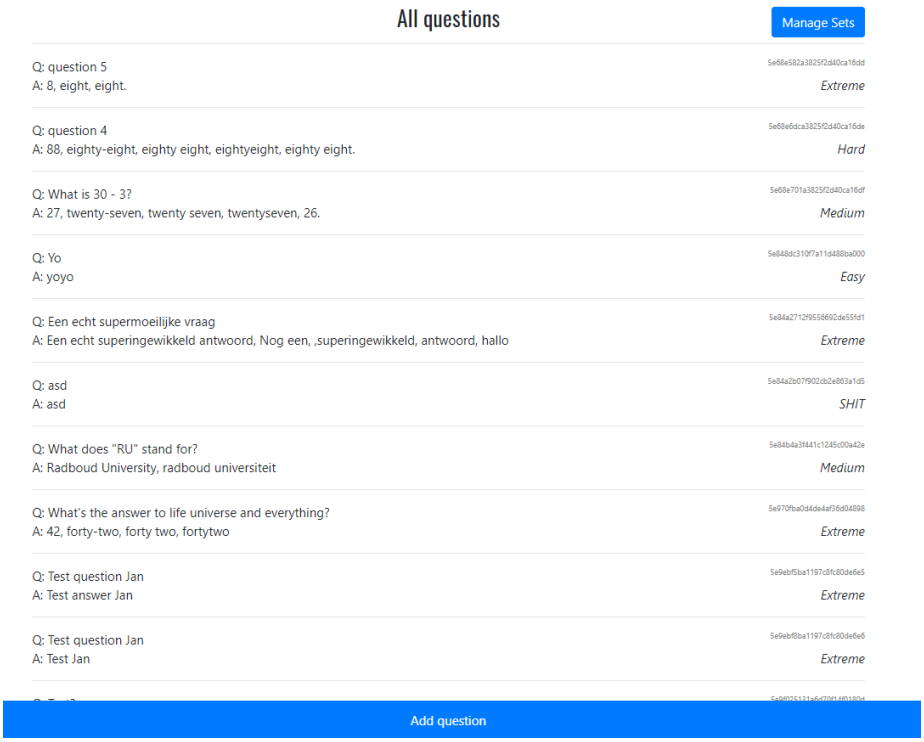

Figure 5: Question manager

As can be seen in Figure 5, all existing questions are shown.

To **add** a question, one can click on the `add question` button. This opens the following view:

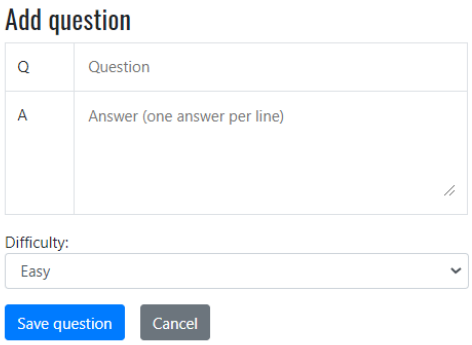

Figure 6: Add question in the question manager

As Figure 6 shows, the question and answer(s) can be inserted by means of

an input field. Finally, the difficulty level of a question (easy, medium, hard, extreme) can be set. After clicking on the `save question` button, one returns to the question overview where the new question is added.

To **edit** a question, one can click on a question in the main overview. This opens the following panel:

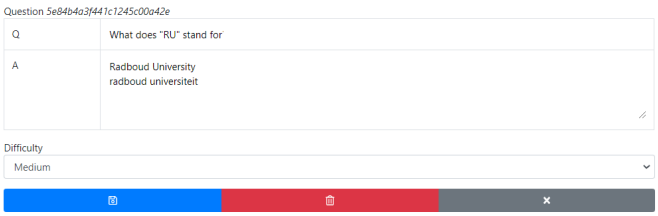

Figure 7: Panel to edit or delete a question.

In the same manner as one would insert a question, a question can be edited. After editing a question, changes are either saved by clicking on the save-button ( ) or canceled by clicking on the cancel-button ( ).

To **delete** a question, the same panel in Figure 7 can be used. In this panel one can click on the delete-button ( ) to delete the question. When questions are deleted, they are automatically removed from the question-set as well.

One can also manage the *question-sets* by clicking on `manage sets`. This opens the following view:

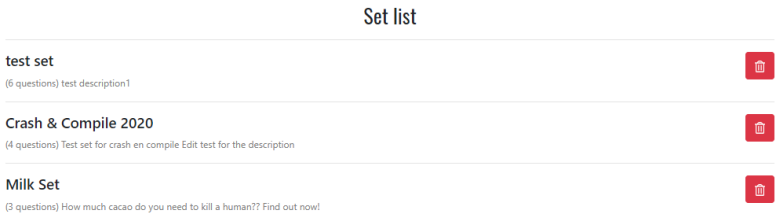

Figure 8: Question-set manager

Here, one can add, delete or change a question-set.

To **add** a question-set, one has to click on `Add set` button. This opens the following view:

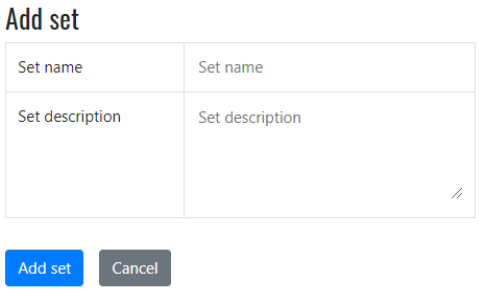

Figure 9: Add question-set.

As can be seen in Figure 9, only the *name* and *description* of the question set can be set. To add the actual questions to the corresponding set, an *edit* on a question-set has to be performed, which is described below.

To **edit** a question-set one clicks on the corresponding question-set. This opens the following view:

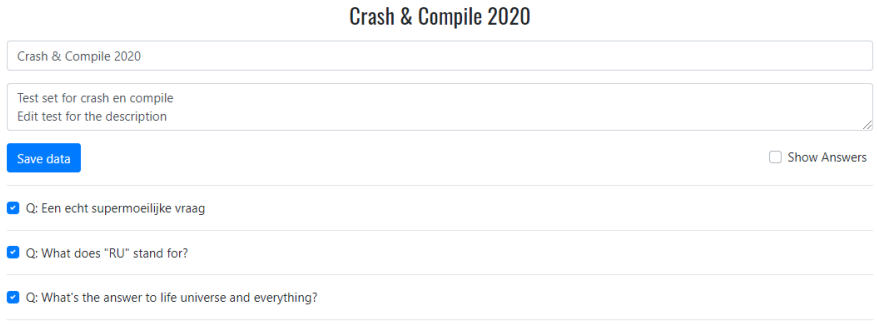

Figure 10: Edit question-set.

In this overview, several things can be edited: 1) title, 2) description, 3) included questions. By means of a checking or unchecking a question, one respectively in- or excludes question in the corresponding set. Changes to the set are saved by clicking on the `Save data` button.

To **delete** a question-set, one can click on the delete-button ( ) in the main overview of the question-sets. This *only* deletes the question set, but *not* the corresponding questions, as they may be reused in later C&C events.

Info manager

As aforementioned, within the info manager, the announcements and the style of the said announcements can be managed. By accessing the announcement manager in the admin dashboard, the following overview is given:

Figure 11: Info manager.

The **announcements** - organized in a HTML file - can be uploaded under the section `Upload HTML file`.

The **style** of the announcements - organized in a CSS file - can be uploaded under the section `Upload CSS file`.

2.2.4 Before competition: info and registration

[Phase(s): Publish Info, Registration]

After preparing the competition, an admin can publish information about the game and open team registration, by setting the phase on `Publish Info`. This updates the C&C website with the stored announcements and a panel for team registration, shown below.

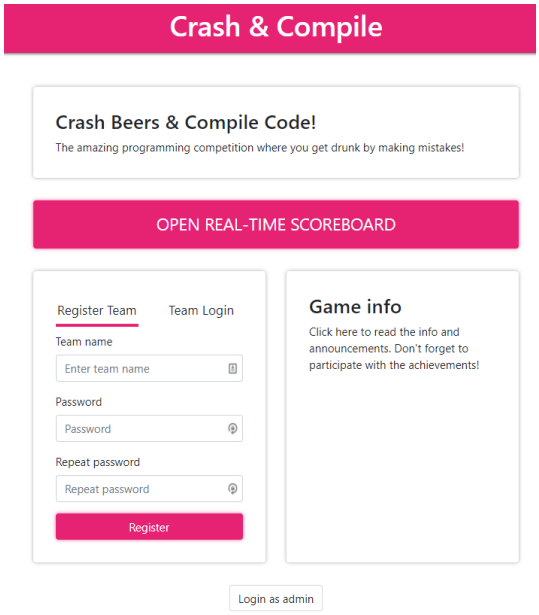

Figure 12: Homepage with page *Publish Info*.

As aforementioned, teams can enter the event by entering a *team name* and a *password*. After registration, a team can login in the `Team Login` in the panel. After a team logs in, it is redirected to the following overview where a team can find all its info:

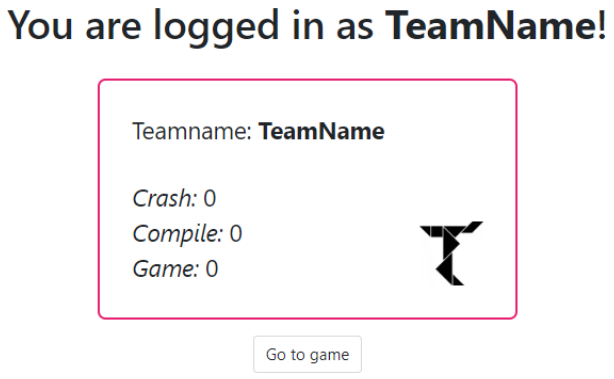

Figure 13: Team information after login.

In case of an online minigame, a team can redirect to the game by clicking on the `Go to game` button. When the game is a real-life game, this button is not

shown.

NB : Do not forget to change the right variables to redirect to the online minigame (if needed) (see section 2.4.5).

2.2.5 During competition: score- and team management

[Phase(s): Game Running, Freeze]

During the competition, after setting the phase to `Game Running`, an admin can use the *score manager* in the dashboard. The score manager has the following view:

Figure 14: Score manager under panel *Add scores*.

In this view, all scores of all teams can be shown of the three categories of the C&C event (Crash, Compile, Game).

Score management

As an admin, one has the control of the scores of a team in the score manager. As can bee seen in Figure 14, there are two panels: **add scores** and **set scores**. By default, **add scores** is the panel that is toggled when opening the score manager.

Add scores The panel "Add scores" gives an admin the opportunity to in- or decrease the score by an amount they want. Within this panel, an admin has the following control:

1. *Increase score by 1*; click on the +-button
2. *Decrease score by 1*; click on the −-button
3. *Add score*; add value to a score by means of an input field by clicking on the score

When clicking on a score, the following overview is shown:

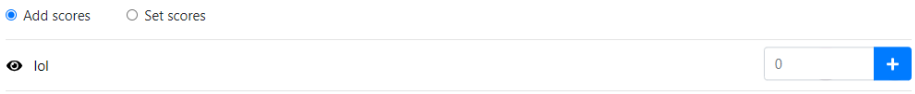

Figure 15: Panel to add scores.

Here, one can enter the amount one wants to add to the score. If one wants to decrease a certain amount, one has to enter a *negative* number (e.g. -10). One can save changes by clicking on the +-button, right next to the input field.

Set scores The panel "Set scores" gives an admin the opportunity to change the score to an amount they want. Below, an overview of the panel *Set scores* is shown:

Figure 16: Score manager under panel *Set scores*

Within this panel, an admin can change score by means of an input field by clicking on the score. In a similar manner as adding scores, score can be set by entering a score and save the changes.

Every 3 seconds, these scores will be updated.

Team management

As an admin, one has the following control of the teams:

1. *Hide team*; click on the icon, where a panel will open (see Figure 17). Here one can hide a team ( )
2. *Make team visible*; click on the icon in the panel

Figure 17: Panel to hide a team.

Hiding a team can be explained as a soft-delete, where the team will be removed from the real-time scoreboard. Hiding a team will be visualized by greying out a whole row.

Real-time scoreboard

On the real-time scoreboard, all scores from the teams can be shown during the C&C event.

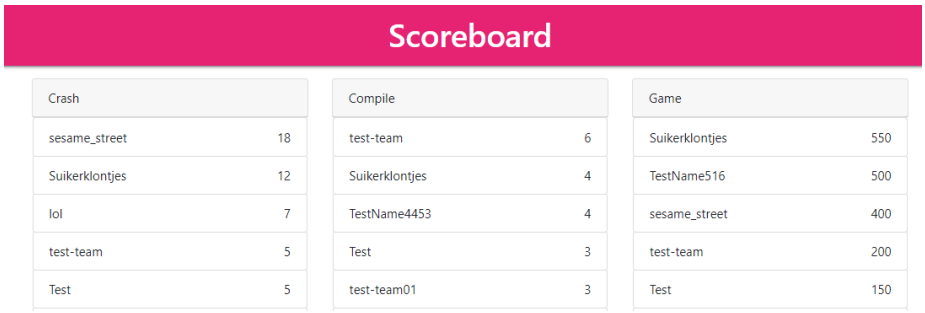

Figure 18: Real-time scoreboard.

Every 3 seconds, these scores will be updated.

Also, a timer is shown in case one is set. This timer counts down, after which the scores will be frozen and cannot be updated by the minigame or the admin.

Freeze scores

During the competition, an admin might want to freeze the scores for any reason. One can do this in two ways: 1) using a timer, 2) manually by changing the phase.

In the phase manager (see Figure 4) one can change the timer, where an end time is set, on which all scores (including the minigame) will be frozen. One can also do it manually by pressing the `Freeze` button.

When scores are frozen, this will be displayed in real-time scoreboard by a sort of alert-message. Furthermore, the real-time scoreboard will contain a count down to this moment, so teams know how much time they have left. This view is shown below.

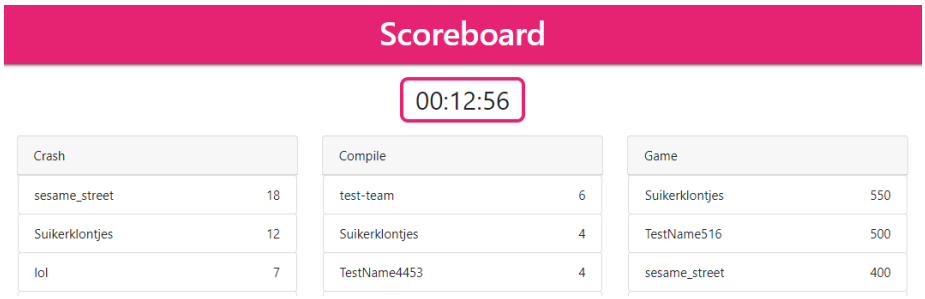

Figure 19: Score board with timer.

2.2.6 After competition: cleanup

[Phase(s): Freeze/Setup]

After the C&C event, the cleanup mainly exists of cleaning up the database. Assuming the question-sets have to be kept, only the teams are to be removed. This can be done with the following endpoint: `DELETE /team/all`, where all teams are deleted from both the *Team* table as well the *User* table. There is no functionality in the admin dashboard for this, so a tool like *Postman* or the *Swagger* documentation has to be used in order to clean up the table.

2.3 Game integration

There are two types of games that can be integrated into the event; a real life game and a minigame. In a real life game, the scores of the teams are added by an admin in this online environment. However, there is also the possibility to make a minigame in which the teams can earn (and lose) points automatically, for instance when they answered a question (in)correctly.

2.3.1 Real life game

In the case of a real life game, the admin dashboard provides all the needed functionality. The admins can select any team and add or subtract points manually (see section 2.2.5).

2.3.2 Minigame

In the case of the minigame, the maker has to 'hook' the minigame onto the backend by using the endpoints in the API. The main endpoints used by the endgame are:

- Get user information based on JWT Bearer token: `GET user/whoami`
- Check the answer of a team: `POST /question/checkanswer`
- Update the score of a team: `PATCH /team/:scoreCategory/:id`
- Get all question-sets, with ID `GET /questionset`
- Get all questions from a question-set `GET /questionset:id/export`
- Change the phase of the game: `POST phase/:newphase`

See the Swagger Documentation (section 2.0.1) for full description of the endpoints and their corresponding parameters.

Redirecting

As aforementioned, teams can redirect to the game after they logged in on the homepage. In this redirect, the *JWT Bearer token* is passed to the minigame.

Authentication

As the *JWT Bearer* of each team is known by the minigame, the minigame can get all information of the user by the endpoint `GET /user/whoami`. This results in - amongst others - the ID of the team. This ID can be used to update team scores, check answers and all other privileges that a minigame-user has. The entire login process of a team in the context of a minigame can be illustrated via the sequence diagram shown in Figure 20.

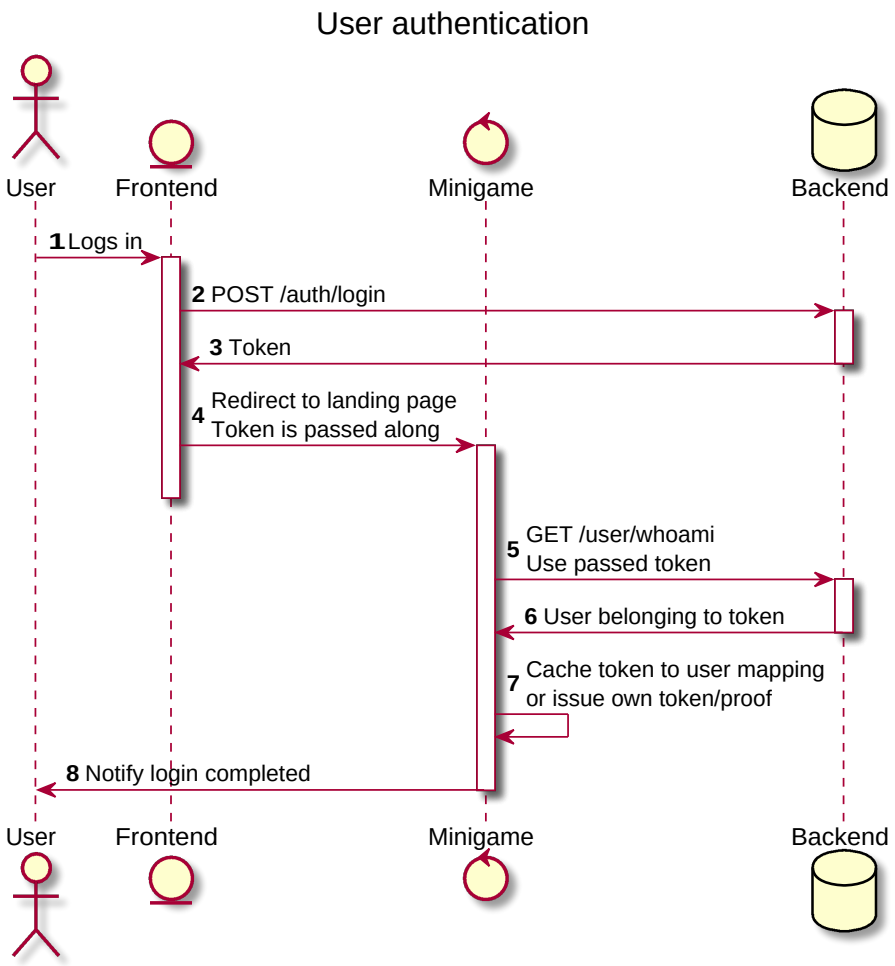

Figure 20: Sequence diagram of team authentication

1. The process starts when a user from a team attempts to log in on the frontend.
2. The frontend will then send a login request to the backend.
3. The backend, assuming the login is correct, will respond with a token that serves as a proof of identity.
4. The frontend receives the token, and redirects to the minigame landing page, sending the token along.
5. Upon receiving the token the minigame can send a `whoami` request to the backend, using the received token.

6. The backend will inspect the token, and if valid will respond with some basic information about the associated user, such as their role, username, and identifier. Remember that the user identifier is identical to the team identifier in case the user role is that of a team.

7. Now that the minigame knows for sure who belongs to the token, by authority of the backend, it can decide to cache this information. Alternatively it could also decide to issue its own token or proof back to the user now that it has been authenticated, for later use in the minigame. We leave this decision up to the implementers of the minigame.

8. Finally, the user is notified that the login process has completed successfully.

API Calls

For privileged calls to the API, the minigame has to include their token. This means that the minigame user has to login in the same manner as any other user in order to obtain the token, by the endpoint `POST /auth/login`. This returns the token for the minigame user to use in order to make calls to the API. The setup and startup process of a minigame can be illustrated via the sequence diagram shown in Figure 21.

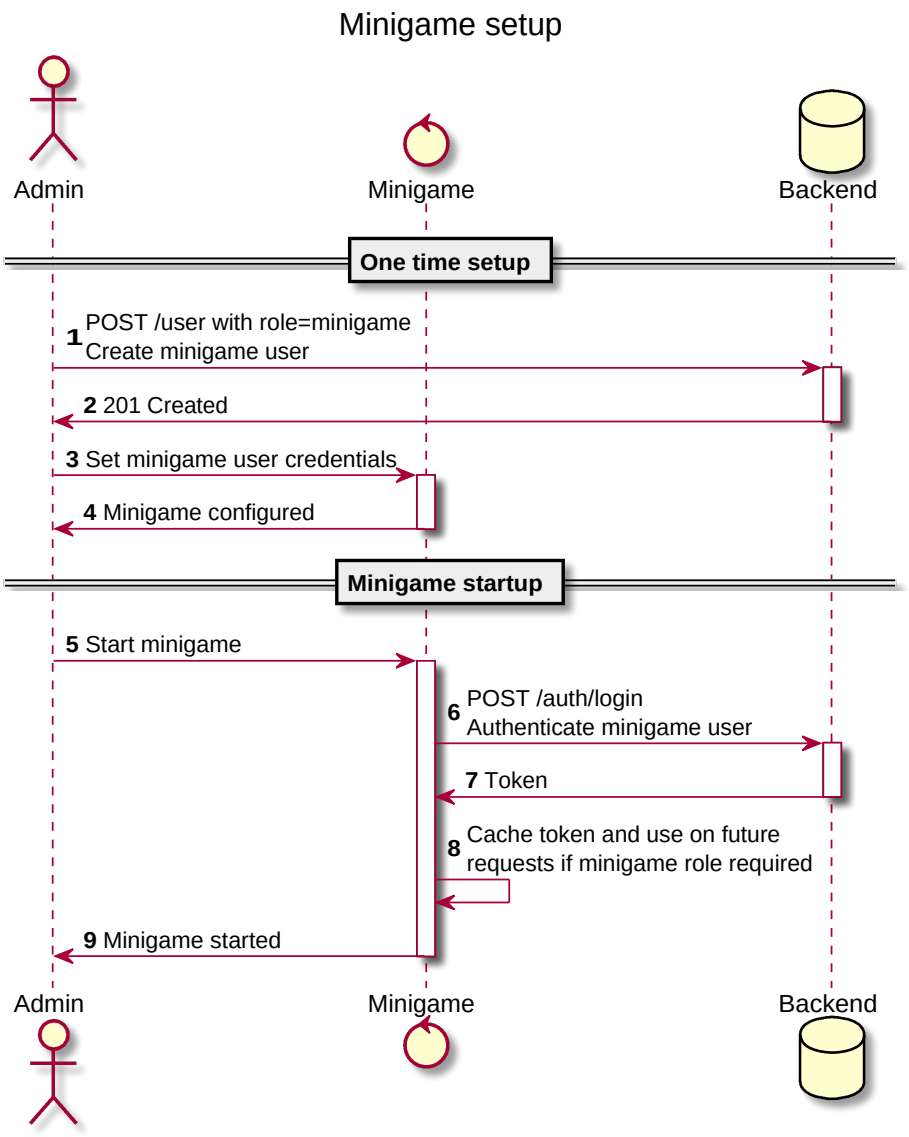

Figure 21: Sequence diagram of minigame setup

This process consists of two parts, namely one part that only has to be performed once (One time setup), and another part that has to be performed each time the minigame application is started (Minigame startup). Each step is described below:

1. An admin must begin by creating a user in the backend with the `minigame` role. This is the user that the minigame application will use to perform

privileged operations in the backend. There is currently no way to do this via the frontend, so we advice using a tool such as *Postman* or using the *Swagger* API documentation.

2. After the user has been created, the backend will respond accordingly.
3. The admin or developer must now configure the minigame application such that it has access to the minigame user credentials.
4. After the minigame has been configured, the one time setup part has now been completed.
5. Now the second part begins each time the admin, or a developer, starts the minigame application.
6. The minigame application must first send a login request to the backend.
7. The backend will, assuming the credentials were valid, respond with a token.
8. As this token is required for the minigame to perform any privileged operations, such as modifying the score of a team, it must be cached and attached as a bearer on those requests.
9. The minigame is now all set to begin, which completes the minigame startup part.

Assumptions

There are a few assumptions regarding tasks that wil have to be performed and implemented by the developers of the minigame:

- The minigame handles further authentication of teams (e.g. caching the ID of the team or asking the ID for every call)
- The minigame has its own question management (e.g. unlocking a question, retrieving right question set); This by means of `GET /questionset:id/export`
- The minigame has its own score management, by increasing the score of the teams themselves; This by means of `POST question/checkanswer` and `PATCH /team/:scorecategory/:id`
- The minigame has its own phase management, by freezing the game (e.g. in case the timer ends or by manually freeze the game); This by means of the endpoint `POST phase/:newphase`

These assumptions stem from the desire to keep the backend and API as generic as possible. Moving these tasks into the backend would likely restrict the possibilities of minigames, forcing them to implement the desired behavior themselves on top of the more generic endpoints anyway, or cluttering the backend with lots of minigame specific endpoints. We would only recommend moving functionality into the backend if it is commonly shared by typical minigames.

2.4 Setup

This section describes the setup of the C&C tool, for example creating the MongoDB database, preparing questions, etc. Note that these steps are described in order of execution.

2.4.1 Architecture

The C&C tool exists of a backend and frontend, both of which can be ran in individual docker containers. The root folder of the backend and frontend can be found at `Crash-and-Compile-[Backend/Frontend]-2020`. A visual overview of the architecture is shown below in Figure 22.

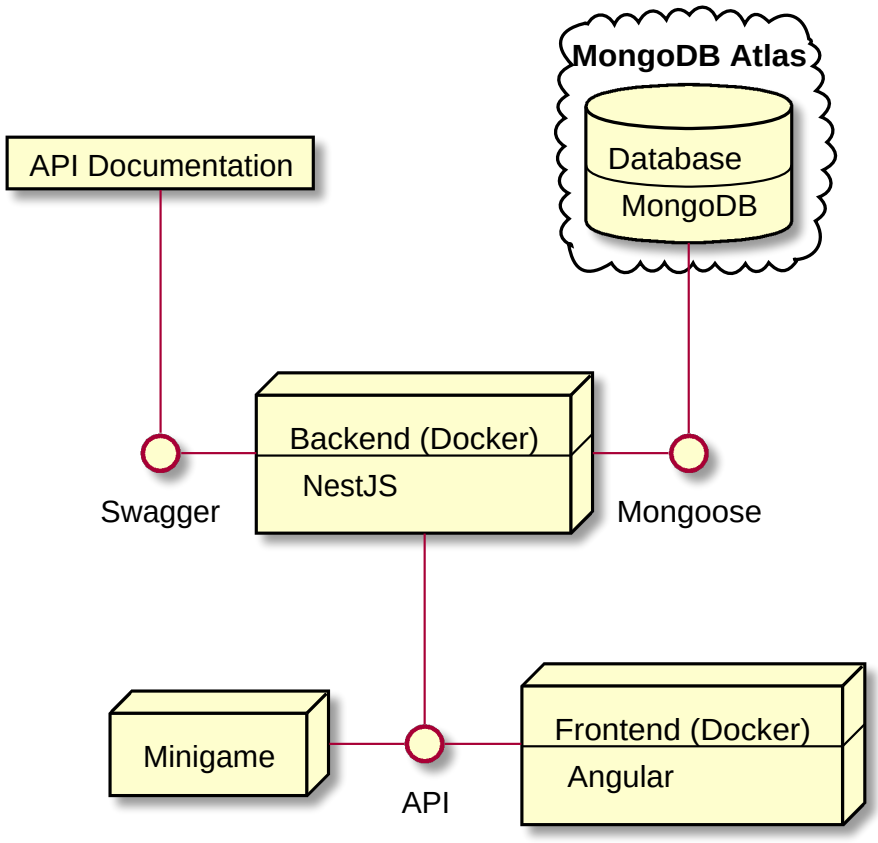

Figure 22: Overview of the application architecture

The backend is built on top of the NestJS framework using Typescript, and uses Jest as the test framework. It provides API documentation via Swagger, and

is connected to a MongoDB database via Mongoose. The database is currently hosted in the cloud service MongoDB Atlas, but it could also be deployed locally, e.g. via a Docker container. The backend provides a typical CRUD/REST API to integrate it with other components.

The frontend is built using Angular and Typescript, and uses Jasmine as the test framework. It is connected to the backend via the API.

Minigames are also connected to the backend via the API, and can thus be developed using a vast range of technologies, languages, and frameworks.

2.4.2 Setting up the MongoDB database

This application makes use of the online cloud database service MongoDB. MongoDB is a documented oriented database, where up to 512 MB of data can be stored for free. As for this event, the data is pure text, it takes only about a couple of hundreds KB storing all data that is needed for the C&C event. A MongoDB account should be made by an admin in order to create a cluster (database). This cluster in turn contains *collections* (tables). By means of a connection string, one can connect to the cluster (see section 2.4.5). A video where this process is explained among the basics of NestJS and MongoDB, which was very useful, can be found here.

2.4.3 Schemas

Below, the structure of the 'tables' (strictly, MongoDB does not make use of structured tables) that should be used for the C&C tool are shown. Within the database there are four instances: 1) user, 2) team, 3) question and 4) question-set.

! Important ! : All these instances and corresponding fields should be reconstructed in the newly made cluster, in order to make the C&C work properly.

The abstract "ERD" (note that MongoDB is not a relational database) of the database is shown below in Figure 23.

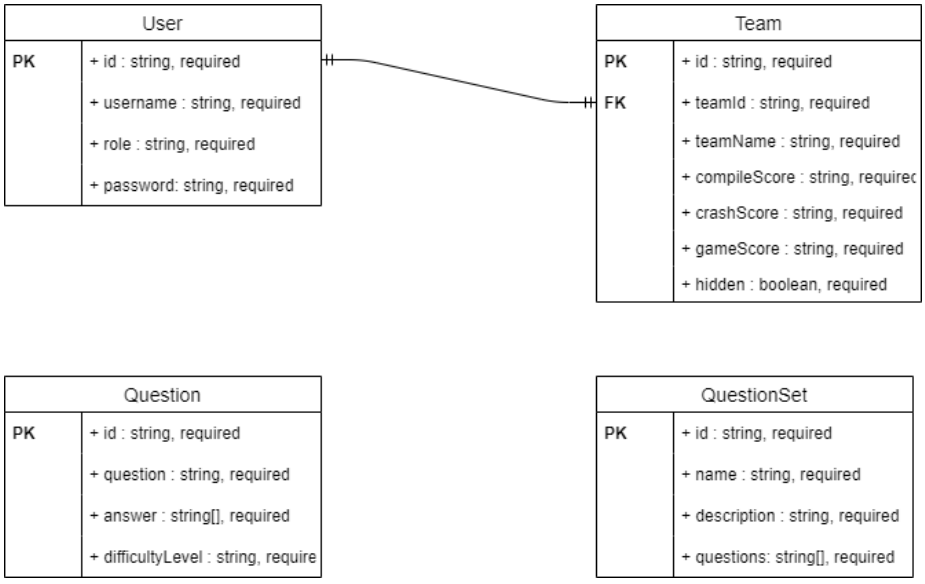

Figure 23: "EDR" of the MongoDB Database.

Creation of an instance is done by means of a JSON object with all required fields.

User

This table is used for authentication, where the hashed password (including salting) is stored, in case of creation of a new user. Furthermore, roles are attached to a user, such that certain users have more privileges than others.

Team

This table is mainly used as score management for the teams, where the scores of each team are stored. Furthermore there is a *hidden* Boolean included, which is used to soft-delete/hide a team on the scoreboard (see section 2.2.5).

In case of team registration, a team is stored in both the *user* table, and the *team* table. When creating a team, it is first inserted in the *user* table, where the user ID is used in the *team* table. Hence, there is a is a 1:1 relation between both tables. In here `user.id` == `team.teamId`

Question

This table is used to manage the questions used in the C&C event. The question, answer(s) and difficulty level of each question are stored in this table.

QuestionSet

This table contains sets of question, in order to bundle questions for each C&C event. As MongoDB is not a relational database, a question-set is build out of IDs of questions by means of a list (*questions*).

2.4.4 Roles

As is seen within the *User* schema, roles are attached to a user. There are three different roles within the database: 1) admin, 2) team, 3) minigame.

An admin is an organizer of the C&C event and thus has the most privileges. A team has least privileges as it does not have any influence on the C&C event itself. The minigame has mainly privileges to retrieve information about questions and teams and to update scores of teams. An endpoint that does not require any role, is called *public*.

These roles have a direct influence on which actions one can perform and thus, which endpoints in the backend can be reached. In table 1 an overview of the privileges for each role are shown. Each privilege that is *public* is also a privilege to all other roles. Note that in case a role is attached to an endpoint, a *JWT Bearer token* is required for authentication (see 2.2.2).

Role	Privilege category	Privileges
Public	*Login*	Can login
	Team management	Can get the categories of the C&C event
		Can get information when a team is updated
		Can get information of (a) team(s)
		Can create a team
	Score management	Can get scores of (a) team(s)
	Phase management	Can get all phases to control the C&C event
		Can get the current phase of the C&C event
		Can get the value of a timer to freeze the C&C event
	Announcement management	Can get the (uploaded) HTML file for announcements
	Style management	Can get the (uploaded) CSS file for style changes
Admin	*User management*	Can get information of (a) user(s) based on an ID
		Get information of a user, based on the JWT Bearer token
		Can create a new user
		Can delete a user
		Can update a user
	Team management	Can update scores of a team
		Can soft-delete/hide a team from the scoreboard
		Can delete (a) team(s)
	Phase management	Can change the phase to any phase to control the C&C event
		Can set a timer to freeze the C&C event
	Question management	Get information on question(s) and question-set(s)
		Can create a new question(-set)
		Can update a question(-set)
		Can delete a question(-set)
		Can check inserted answer of a team to be correct
	Announcement management	Can upload a HTML file for announcements
		Can delete a HTML file for announcements
	Style management	Can upload a CSS file for style changes
		Can delete a CSS file for style changes

Team	*Team management*	Can get all its own info including current scores
Minigame	*User management*	Can get information of a user based on the JWT bearer token
	Score management	Can update the score of a team
	Question management	Can get information about question(s) or question-set(s)
		Can check inserted answer of a team to be correct
	Phase management	Can change the phase to any phase to control the game
		Can set a timer to freeze the C&C event

Table 1: Privileges of all roles.

All the endpoints to execute all these privileges are included in the API.

2.4.5 Set (environment) variables

For both the backend and the frontend, certain (environment) variables need to be adjusted to deploy the C&C tool the right way. These variables thus have to be changed *before* deploying the C&C tool. The C&C tool can be deployed inside and outside Docker (see section 2.5, which implies differences in setting the variables.

Backend

As aforementioned, data of the C&C tool is stored within a MongoDB cluster. Thus the main variables for the backend are the connection string to this cluster, as well as admin login data. The backend is configured via `.env` files, which have `KEY=VALUE` lines, or via environment variables with name `KEY` and value `VALUE`. A template for the configuration file can be found in the root folder of the backend and is called *development.env*.

Docker deployment

A file called *production.env* must be placed in the same folder as the *docker-compose.yml* file. Docker will read the contents of that file, and create the respective environment variables in the backend docker container upon startup. During the building of the backend Docker image the *production.env* and *development.env* files are **NOT** copied over to the container, as configuring via a Docker `.env` file is the preferred method.

Non Docker deployment

During deployment, the C&C tool checks whether *development.env* exists in the same directory and if so, uses it. Otherwise, it check whether *production.env* is

available, and if so, uses it. If **both** files are available, the variables from *development.env* are overwritten by the variables from *production.env*. Environment variables always overwrite keys set in either file.

NB : if both files are available only the *existing* keys in *production.env* are overwritten in *development.env*. Keys that are only existing in *development.env* remain the same.

Within the template, the following environment variables can be set:

- `CNC_PORT`; port to run the C&C backend on
- `CNC_MONGO_URI`; connection string to the MongoDB cluster
- `CNC_JWT_KEY`; string, buffer, or object containing the secret for HMAC algorithms for authentication (see 2.2.2).
- `CNC_JWT_EXPIRES_IN`; string to control token expiration duration. If omitted tokens will never expire. Time units can be specified in the string, e.g. `4h` for four hours. See this page for a full list of the supported formats (optional)
- `CNC_ADMIN_USERNAME`; username to create (or update) an admin user during deployment (optional)
- `CNC_ADMIN_PASSWORD`; password to create (or update) an admin user during deployment (optional)

Connection string

To find the connection string for the MongoDB cluster, one can click on the `connect` button in the cluster in the Cloud. Note that all placeholders should be replaced. A connection string has the following format:

```
mongodb://[username:password@]host1[:port1][,...hostN
    [:portN]][/[defaultauthdb][?options]]
```

One can also choose to install MongoDB locally, where the connection string should be replaced in the same way as using MongoDB in the Cloud.

Admin username and password

For setting up the C&C tool, a new admin user can be created by means of setting an admin username and admin password in the (environment) variables. An admin user will then be inserted (or updated) in the *User* table (see section 2.4.3) upon initializing the backend. Thus, after deploying, an admin user is available with the desired username and password. Only if both variables (username and password) are set to non-empty values will the backend attempt to create or update the admin account, thus by omitting the variables this behavior can be disabled once an admin account has been created.

Frontend

For the frontend, there are certain important variables/aspects that have to be set:

- `onlineGame`; Boolean indicating whether the C&C makes use of an online minigame or a real-life game
- `gameLinkBaseURL`; base URL to the minigame
- `baseURL`; The base URL to connect to the backend API
- `domain`; Domain on which the server is hosted
- Interval variables (see below); variables to change refresh intervals
- Layout C&C tool;

The first four can be changed in the environment file for production in `src/environments/environment.prod.ts`.

Online minigame versus real-life game

The first two variables are indicating whether an *online* game or a *real-life* minigame is taking place. The `baseURL` and the `domain` variables indicate the URL of where the application is hosted. For developing and testing purposes, the variables in `environment.ts`. If you add the option `-prod=true` to the `ng build` or `ng serve` command, the variables in `environment.prod.ts` will be used. This should be done if the competition is hosted on a server.

Interval variables

Furthermore, certain interval variables can be set, for instance the interval for refreshing scores on the real-life scoreboard. All these intervals are declared as mutable variables. Below, the file names with corresponding interval variables are shown, so one can access them easily to change them as they want. Each interval is set in *ms*.

Component & File	Variable	Description
<Phase Manager> phase-manager-component.ts	fetchPhaseAndTimerInterval	Set interval to get the current phase and current timer
<Score Manager> score-manager-component.ts	lastEditInterval	Set interval to get the last team-change (hide, delete, insert)
	fetchPhaseInterval	Set interval to get the current phase
<Score Manager> score-panel-component.ts	updateScoresInterval	Set interval to update scores form a category
<Scoreboard> score-column-component.ts	fetchScoresInterval	Set interval to get scores from a category
<Scoreboard> scoreboard-component.ts	lastEditInterval	Set interval to get the last team-change (hide, delete, insert)
<Scoreboard> timer.component.ts	updateInterval	Set interval to update the timer on the scoreboard
<Team Page> team-page.component.ts	fetchProfileInterval	Set interval to get the current profile information of a team
<HomePage> homepage.component.ts	trackPhaseInterval	Set interval to detect phase changes

Table 2: Interval variables an their location

Layout C&C tool

The complete layout of the C&C tool is managed by a CSS file. This file to comply with the C&C tool is called `styles.css`. Here, the templates for the homepage, score manager and real-life scoreboard can be changed. In this file, most components are commented out, but as mentioned in the template: uncomment the attribute that one wish to change. This template can be found in the root folder of the frontend.

2.5 Deployment

The C&C tool is deployed within Docker and the addition tool Docker Compose. Compose is a tool to create a multi-container docker environment. The C&C tool exists of two separate containers for both the frontend and backend. These two containers are then merged with a *docker compose* file, so that it can be started and stopped as one single application. We provide a *docker-compose.yml* file in the backend repository. Note that port mappings from the host machine to the Docker containers are also configured in this file.

To build both containers, the following commands have to be executed in any order, in the root folder of respectively the frontend and backend:

```
$ sudo docker build -t cnc-frontend:latest .

$ sudo docker build -t cnc-backend:latest .
```

For the sake of convenience we have included two Makefiles, one per repository, that perform these commands. This means that instead of executing the above commands, one could also execute `sudo make` instead.

To compose both containers, the following command has to be executed in the folder containing the *docker-compose.yml* and *production.env* files:

```
$ sudo docker-compose up --detach
```

Full control

If one wants to have full control, one can run the containers individually. To start the containers of both the frontend and backend for the first time the following commands have to be executed in any order:

```
$ sudo docker run --publish 4200:80 --detach --name
    frontend cnc-frontend:latest
$ sudo docker run --publish 8080:8080 --detach --name
    backend cnc-backend:latest
```

Note that we specify the port mapping from host machine to docker container in the above commands. Both containers can be started again after having executed the above commands once, by executing the following commands in any order:

```
$ sudo docker start frontend
$ sudo docker start backend
```

As of right now, both frontend and backend are deployed locally, on the following ports:

- frontend: http://localhost:4200/
- backend: http://localhost:8080/

2.5.1 Manual deployment

When one does not want to use Docker, the backend and the frontend can be started individually.

Backend

Firstly, the backend should be running which can be accomplished by executing the following command inside the root of the backend in this order:

```
$ npm install
$ npm run start
```

NB: The command `npm install` is (probably) only needed for the first setup and after importing new packages.

Frontend

Afterwards, the frontend an be deployed by executing the following command, assuming one is in the root folder of the frontend:

```
$ ng serve -o -prod=true
```

NB: If one gets an error regarding `package.json`, execute the following command:

```
$ npm install --save-dev @angular-devkit/build-
    angular
```

2.6 Teardown

After the competition, in case of using *composed* Docker deployment, the docker containers can be stopped by executing the following command in the directory container the compose file:

```
$ sudo docker-compose down
```

In case of running the Docker containers of both frontend and backend separately, the following commands has to be executed to stop these individually:

```
$ sudo docker stop backend
$ sudo docker stop frontend
```

2.7 Testing

We made tests for both the frontend and the backend to test the functionality and the endpoints of the code. These tests can also be extended when developing. Moreover, we stress-tested the project such that we got an idea what the system could handle.

Frontend testing

The testing of the front-end was done using the Jasmine/Karma suite that is integrated into Angular. Here each component has its own 'spec' typescript-file, which contains the testing specifications, in the same directory as the corresponding component it contains the specs for. One can test the 166 currently implemented specs by running 'ng test' from the command line in any of the front-end directories.

Backend testing

NestJS could work with almost all testing frameworks, but the default was Jest and that was also the testing framework that we used in the backend. We made unit tests and e2e tests for almost all components and these tests can be

found in the *.spec.ts files and the *.e2e-spec.ts files, where * is replaced with the corresponding component. Unit tests were used to test the services, which contain most of the business logic. The e2e tests were used to test the integration of the controller (which receives requests, calls into the service, then returns the result), service, and database. The small, individual tests are written in these files and can be extended by manually adding more tests to them. If a new component was created in development, it is recommended to write at least an e2e test for it and, if necessary and useful, a unit test. The tests use a mock environment, by virtue of an in-memory MongoDB server, such that it doesn't conflict with the real database and that very specific situations could be tested as well. More information about testing in NestJS can be found here.

Note that we ran into an issue on one of our Linux machines regarding the in-memory MongoDB server. It would appear as if all tests were skipped. This turned out to be due to a missing dependency on the `libcurl3` package, causing the in-memory MongoDB server to crash upon startup, and silently skipping all tests. Manually installing this package, e.g. via `apt-get install libcurl3`, resolved the issue.

Stress-testing

To stress-test the system, we deployed a server on an i4 CPU, such that we could test if the system could handle all the requests. We tested it with 7 users at the same time and we did several actions, e.g. we added scores with huge numbers and sent a lot of requests to the same endpoint. At some point we used a script that generated 5000 concurrent curl requests in the span of several seconds, and used this on several endpoints (both frontend and backend). The system could handle it with no problems, so the system could handle 7 users that tried to break the system, so we are almost certain that it will also work properly in a real Crash & Compile event.