



VILNIUS UNIVERSITY  
FACULTY OF MATHEMATICS AND INFORMATICS  
INSTITUTE OF COMPUTER SCIENCE  
INFORMATION TECHNOLOGIES STUDY PROGRAM

Problem-Based Project

# **REACTIVE**

Project authors:

Deividas Bendaravičius

Emilis Šerys

Julius Valma

Supervisor:

dr. Agnė Brilingaitė

Vilnius  
2022

# Contents

<b>1</b>	<b>Abstract</b>	<b>3</b>
<b>2</b>	<b>Introduction</b>	<b>4</b>
<b>3</b>	<b>System Architecture</b>	<b>5</b>
3.1	An Overview of the System . . . . .	5
3.2	Architectural Goals and Constraints . . . . .	5
3.3	UML Deployment Diagram . . . . .	5
3.4	ORM or no ORM? . . . . .	6
<b>4</b>	<b>Front-End</b>	<b>7</b>
4.1	Register and Login . . . . .	7
4.2	Script Build Screens . . . . .	8
4.2.1	Initial Build Screen . . . . .	8
4.2.2	Main Build Screen . . . . .	10
4.2.3	Events . . . . .	11
4.2.4	Private Routes Requests Middleware . . . . .	12
4.3	Scripts Screen . . . . .	13
<b>5</b>	<b>Back-end</b>	<b>15</b>
5.1	Endpoints . . . . .	15
5.2	Login Route . . . . .	17
5.3	Any Protected Route Authentication . . . . .	17
5.4	Token Refresh Route . . . . .	18
5.5	Password Hashing . . . . .	18
5.6	File Upload . . . . .	19
5.6.1	File Validation . . . . .	20
5.6.2	File Retrieval . . . . .	21
<b>6</b>	<b>Input Validation</b>	<b>22</b>
6.1	Front-End Input Validation . . . . .	22
6.2	Back-End Input Validation . . . . .	23
<b>7</b>	<b>Database</b>	<b>25</b>
7.1	Relational Model . . . . .	25
7.2	Database Structure Overview . . . . .	27
<b>8</b>	<b>Functional Requirements</b>	<b>28</b>
<b>9</b>	<b>Non-functional Requirements</b>	<b>29</b>
<b>10</b>	<b>Human-Computer Interactions</b>	<b>30</b>
<b>11</b>	<b>Conclusions</b>	<b>32</b>

# 1 Abstract

Tabletop exercises are a great opportunity for teams to test and hone their skills, but the exercise script generation takes a lot of time and effort and can sometimes be tedious work. Reactive is a web application powered by JavaScript library React.js in the front end and Node.js with Express.js in the back end, that achieves the generation of such scripts in a much faster and convenient way. With Reactive, users can interactively create table top exercise scripts using a very convenient drag and drop system, the scripts can be shared with other users to do collaborative work and the scripts can also be exported to a JSON format file to be used in automated systems.

Keywords: Script, React, Exercise, Screen, Component

## **2 Introduction**

Reactive is a web application, that helps its users to build a table top scenario scripts in a very fast and convenient way. Our motivation behind this project is to create a easy-to-use system for creating table top exercises, since traditional planning of these tasks is considered to be tedious work. Table top exercises are a great way to train and test employees of various scenarios that might occur during the employment, but, as mentioned, creating these scenarios and their scripts without any tools takes a lot of time and effort, and Reactive application solves exactly that. With separate accounts for all users, the Reactive system provides functionality to collaborate on script building, making the process even more efficient. The product is primarily meant to be used by corporations/organizations and educational institutes in the IT field, but can easily be customized for any field that would be able to integrate table top exercises into their system. The application is made with the philosophy of giving the user the platform to express their creativity by the use of powerful, but easy-to-understand tools, achieving faster results and giving access to all types of users, even the non-tech savvy ones!

## 3 System Architecture

### 3.1 An Overview of the System

In order to build an understanding of how the system will be implemented, an overview of the system of the REACTIVE project will be displayed and discussed in this section of the document. To display this information, various diagrams will showcase how some aspects of the application work in correlation with others.

### 3.2 Architectural Goals and Constraints

These are some of the architectural goals and constraints that need to be met for the project to be considered a success:

- REACTIVE must be compatible and run on Vilnius University resources and infrastructure.
- REACTIVE should be supported by all of the popular browsers (e.g. Chrome, Edge, Safari, Firefox, Opera).

### 3.3 UML Deployment Diagram

The UML deployment diagram models the physical deployment of software components. In the diagram, hardware components, such as web servers, are presented as nodes, with the software components that run inside the hardware components presented as artifacts. The entire system is hosted on separate virtual machines on a cloud computing platform Open Nebula, and the communication is established via HTTP requests.

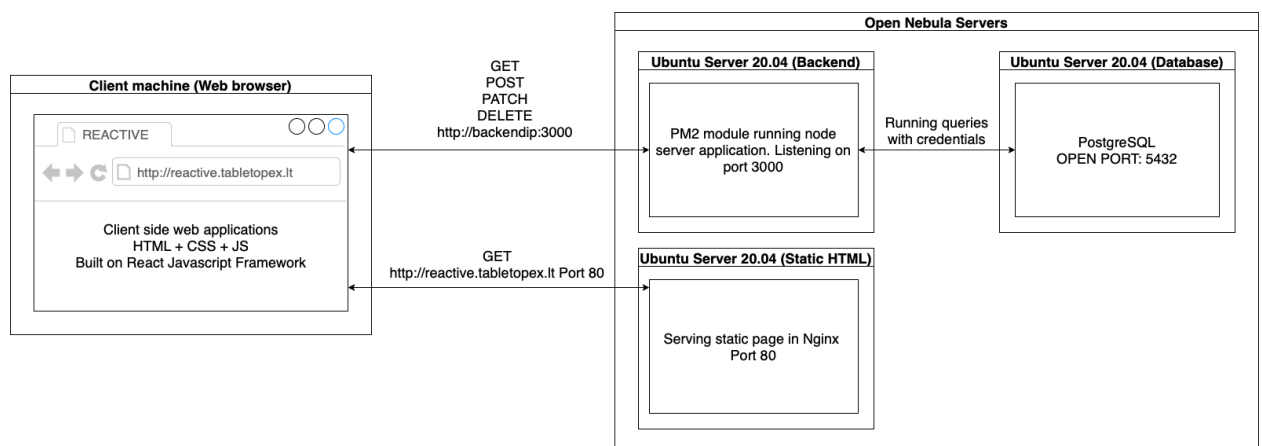


Figure 1. UML Deployment Diagram

**Ubuntu Web Server(FRONTEND)** The virtual server running Ubuntu-20 is dedicated to the front end of the system. It contains JavaScript XML or JSX code, which is used to write HTML inside JavaScript code and build reusable components in React. The code is then compiled into a singular index.html page, which is updated upon user interaction. The singular HTML page is uploaded to the Nginx web server. The web server then establishes a connection between the user's browser and itself, providing the user with the service.

**Ubuntu Web Server(BACKEND)** The backend server running the Ubuntu-20 is used as a middleware between the web server and then database. It uses Node.js to run a non-blocking, event-driven server and acts as an API server. Using the HTTP requests received from the front end, the API sends out queries to the database requesting for information and returning it to the front end.

**Ubuntu Web Server(DATABASE)** The virtual server running Ubuntu-20 is dedicated to the deployment of the database. The relational database PostgreSQL is used to store various data about users, projects and events. In the UML diagram, a simplified version of the database is displayed, the more in-depth model can be found in the Database Overview section.

### 3.4 ORM or no ORM?

To begin with we need to understand what an ORM is. ORM - short for Object Relational Mapping is a technique that maps software objects to database tables. So if translate to simpler word - it allows developers to create classes, initiate objects and automatically transfer all of this to the database. The schemas and even relations are formed automatically, many methods are predefined and available for use. Also, ORM's most of the times support multiple types of databases and switching between them is really easy.

Knowing this information, we decided not to use an ORM. First of all, we did not intend to make our back end object oriented. Reactive system is database driven and the back end it self is a simple crud with some additional functions. OOP tends to make many of the objects tightly connected, therefore making any major changes to the structure becomes a huge hassle and a long process. In the reactive system, each endpoint is independent, with around 10 reusable methods, that provide specific usage, for example authentication. Most of those methods are used as middleware. In the conclusions, each endpoint is pretty light and as a result fast. Back end structure is simple to understand and well readable. Also, this type of back end is super easy to scale, as new endpoints just require new controllers and maybe a couple of utils.

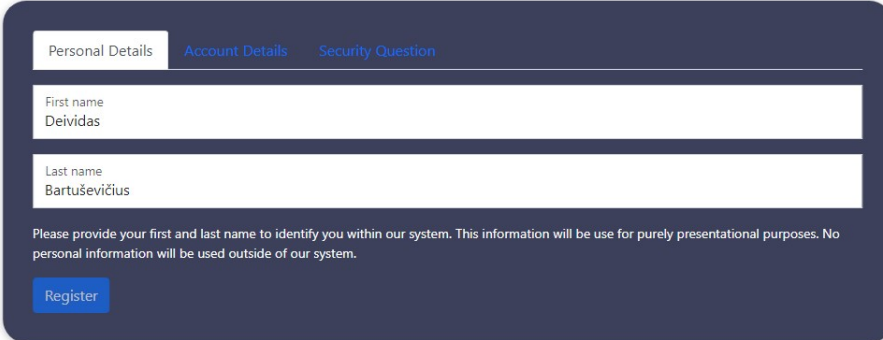
In conclusion, Reactive is using node-postgres node library, which makes connecting to database simple and enables creation of pools for executing queries. Writing queries is usual for our team members and is super fast and well translatable.

## 4 Front-End

### 4.1 Register and Login

The Reactive Table Top exercise script creating platform utilizes a user authentication system which can be seen in the figure 2. A new user will firstly have to create an account in order to start creating projects. The register process is simple and requires only the essential data, the input tabs as separated into 3 parts: personal details, account details and the security question. As mentioned, and as can be seen in 2, during the registration process the user has to create an answer for the secret question. The secret question is used to recover the user's password if it is ever lost. The inputted data is sent out to the Reactive database for safe storage. Bcrypt is used for password encryption.

Welcome to **Reactive**, Deividas!



Personal Details Account Details Security Question

First name  
Deividas

Last name  
Bartuševičius

Please provide your first and last name to identify you within our system. This information will be used for purely presentational purposes. No personal information will be used outside of our system.

Register

Figure 2. Register screen

After creating an account, the user is able to log in to the system. Once the user enters their details a request to fetch the data from the database is sent and if the credentials are found, the user logs in.

Welcome back!

Email address  
sss@gmail.com

Your email information will not be shared.

Enter password  
.....

Your password must be 6-20 characters long, may contain letters and numbers, and must not contain spaces, special characters, or emojis.

☐ Remember me

Login

Figure 3. Login screen

## 4.2 Script Build Screens

The main and most important component of the Reactive web application is the “Build screen”. In this screen, the users of the application use the user-friendly front end system to create table top exercise scripts.

The build screen is divided in to two separate but related parts which share information required for each: the initial build screen and the main build screen.

### 4.2.1 Initial Build Screen

After users of the system route to “Create a script” they are firstly met with the screen which prompts them to enter the initial data of their table top exercise script. The users enter the title of script, the teams that will be participating in the script and also use a slider to select the time frame of the whole exercise.



Reactive

Home Create a script My scripts About Profile Log Out

### Setup screen

Please fill out these initial settings before proceeding to the script builder

Enter the exercise title:

My TableTop Exercise

In minutes, select the duration of the exercise using the slider.

240 minutes

Participating groups

Group's name Add a new group

**Developers**  
No members yet, click to add!

**Managers**  
No members yet, click to add!

**Executives**  
No members yet, click to add!

Continue

Figure 4. Initial build screen

Once the groups that will be participating in the Table Top Exercise script are added, the creator of the project also needs to add members (people) to each of these groups. To achieve this we provide an interactive modal for the users.

### Add members to the *Developers* team!

Name Add a person to the group

Tom Lilly Gerald

Discard Save Changes

Figure 5. Adding members to groups

All of the data entered by the user in the initial build screen is saved and stored using React contexts and accessed using hooks. After the user hits the button “Continue”, it launches the function which creates a new project in the context. The data saved in the project is then sent directly to the Reactive database.

Once the data is successfully passed and stored, users can start creating their table top exercise in the main build screen.

### 4.2.2 Main Build Screen

The data passed from the initial build screen is used to compile the structure of the main build screen. This screen is compiled using 4 main components: departments, timeline, events and the build windows components. The build screen algorithm calculates how many possible event containers should be shown and in what time frame.

```
for (let i = 0; i < props.numberOfDepartments; i++) {  
  events[i] = [];  
  for (let j = 0; j < props.numberOfElements; j++) {  
    events[i].push({  
      ui_id: j,  
      group_id: project.groups[i].group_id,  
      event_time: j * 10,  
    });  
  }  
}
```

Figure 6. Build screen structure compile algorithm

One of the algorithms used to compile the build screen is shown in figure 5. As can be seen, the main build screen component receives data from initial build screen component via props and uses this information in loops to compile the correct size and length of the table top exercise script.

Entered data example:

- Title: My TableTop exercise
- Length of the exercise: 240 minutes
- Teams:
  - Developers: Tom, Lilly, Gerald
  - Managers: Jessica, Harvey, Michael
  - Executives: Louis, Rob, Andreas



Figure 7. Main build screen

In figure 6, with regards to the data passed, three rows in the build window section have been compiled by the user interface and the number of events in a row directly related to the length of the script itself.

#### 4.2.3 Events

The whole table top exercise is build using different events that can be placed in different time frames. The system provides the users with the most common premade events that they can use to build their script. To make the user interface more engaging a React Drag and Drop library has been implemented in to the project allowing users to select and drag the premade events in to the build window.

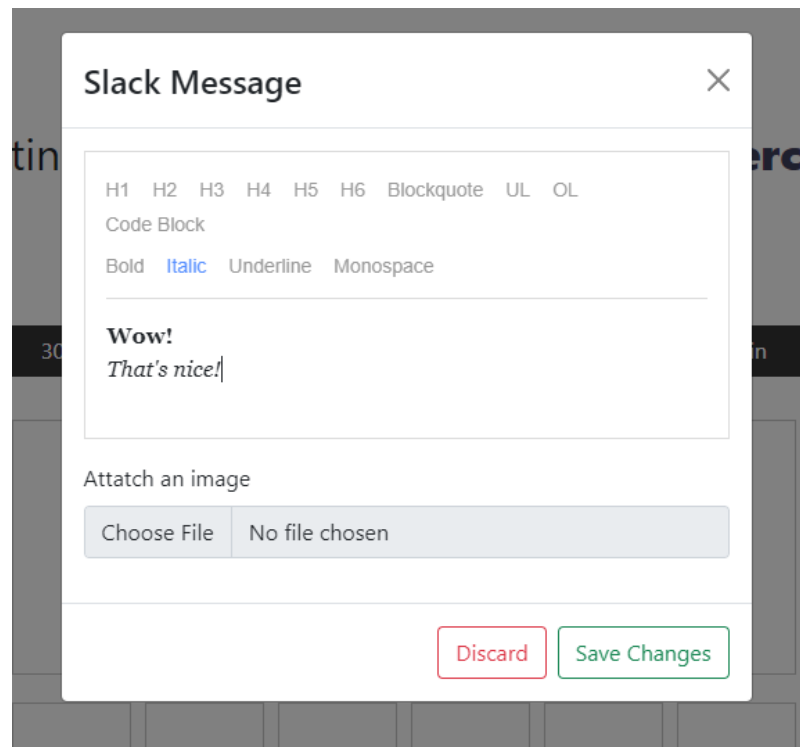


Figure 8. Event edit modal

The events themselves are fully customizable and the user is not limited to the premade data. Users can change the title of the event, enter custom rich text and upload images. All of the information, including the rich text, entered by the client is once again saved in the Reactive database.

#### 4.2.4 Private Routes Requests Middleware

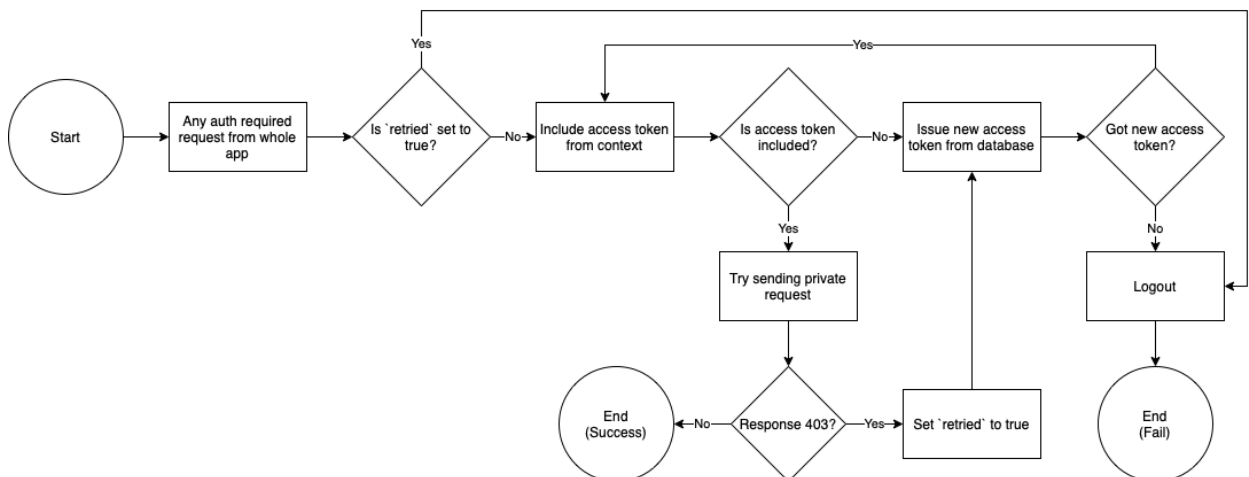


Figure 9. Private routes requests middleware flowchart

In the frontend we have created a middleware that processes every request made to protected routes in the database. It catches every request response and waits for a 403 status. If a 403 was returned from backend it can mean a couple of things: either access token expired, or there is no access token in frontend (as access tokens are only stored in memory, which means on every system

refresh, tokens are lost), or user is not logged in. In case of catching 403 response, middleware issues a new access token from the back end. If the response was still 403, user is logged out in front end. If the response is 200 and new token is present in response, new token is added to memory and original request is retried.

### 4.3 Scripts Screen

In this page of the Reactive system users can find their previously made Table Top exercise scripts. All of the scrips are fetched from the remote databases and compiled to a list where the most important information like title, status and data can be seen. Moreover, the creator can enter their scripts and edit them as many times as necessary. All of the scripts can also be exported for download in JSON format.

The screenshot shows the 'Reactive' system interface. The top navigation bar includes links for Home, Create a script, Scripts, About, Profile, and Log Out. The main content area is divided into two sections: 'My scripts' and 'Shared scripts'.

**My scripts**

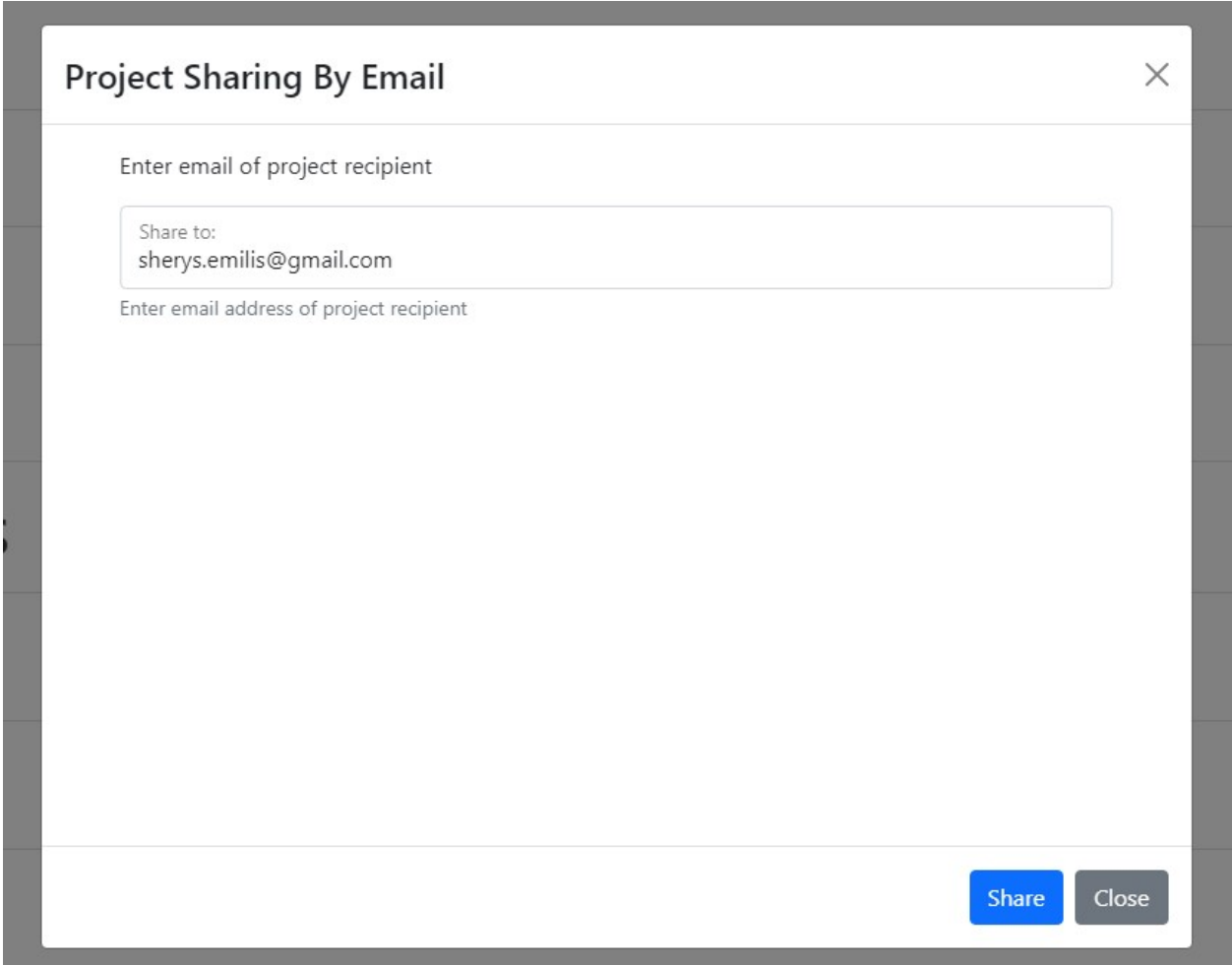
1. <b>123123</b> Exercise length: 200 min	2022-06-09 Options ▾
2. <b>123</b> Exercise length: 170 min	2022-06-09 Options ▾
3. <b>Yep</b> Exercise length: 190 min	2022-06-01 Options ▾

**Shared scripts**

3.1. <b>Example 1</b> Exercise length: 220 min Shared by: sss@gmail.com	2022-05-25 Options ▾
3.2. <b>Example script 2</b> Exercise length: 350 min Shared by: sss@gmail.com	2022-05-25 Options ▾
3.3. <b>123123</b> Exercise length: 220 min Shared by: aaa@gmail.com	2022-06-09 Options ▾

Figure 10. Scripts screen

Moreover, as can be already seen in figure 11, Reactive provides users with a very convenient functionality to share exercise scripts between users. A user can enter the email of another user and give access to said user to see, edit or export the script.

A screenshot of a web application dialog box titled "Project Sharing By Email". The dialog has a close button (X) in the top right corner. Inside, there is a label "Enter email of project recipient" above a text input field. The input field contains the text "Share to:" followed by "sherys.emilis@gmail.com". Below the input field, there is another label "Enter email address of project recipient" which is currently empty. At the bottom right of the dialog, there are two buttons: a blue "Share" button and a grey "Close" button.

Project Sharing By Email

Enter email of project recipient

Share to:  
sherys.emilis@gmail.com

Enter email address of project recipient

Share Close

Figure 11. Share scripts functionality

## **5 Back-end**

### **5.1 Endpoints**

This part of the subsection of the server-side of the system is dedicated to some of the available endpoints. Each of the endpoint will be explored and explained in detail. Some of the endpoints may require the user to have an access token in order to be able to use them.

Access or refresh tokens are used as a session management tool. Access tokens last 15 minutes and refresh tokens are valid for 24 hours before they are removed from database.

ENDPOINT	BODY	DESCRIPTION
/api/projects/get_projects GET		Returns a list of projects by a specific user.
/api/events/get_compiled_project GET		Returns a fully compiled project in JSON format.
/api/users/login_user POST	<i>{ "email": "", "password": "" }</i>	On successful login validation, returns an access and a refresh token.
/api/users/register_user POST	<i>{ "f_name": "", "l_name": "", "email": "", "password": "", "confirm_password": "", "security_question_id": "", "security_answer": "" }</i>	On successful registration process, adds a new user to the database and returns a user object, with added id.
/api/events/file_upload POST	a file of the event	Enables file uploading to a certain event identified by it's event_id.
/api/users/forgotPassword POST	{ "email": "" }	Returns a security question id from the database for the given email address. This id is used to find the security question of the user, which the user has to answer to reset the password.
/api/users/confirmAnswer POST	{ "email": "", "password": "", "answer": "" }	Given the correct answer to the security question, enables the user to change the old password to a new one.
/api/events/update_event PATCH	{ "event_id": "", "property_key": "", "property_value": "" }	Updates any of the event parameters with the given value.
/api/events/add_event POST	{ "project_id": 75, "event_type": 2, "event_time": 35, "event_text": "some text", "event_groups": [107, 106] }	Adds an event to the groups of the project.
/api/projects/get_projects GET		Get list of user projects.
/api/projects/new_project POST	{ "project": { "project_title": "", "project_status": "", "groups": [ { "group_title": "", "group_color": "", "group_members": [ { "name": "", "email": "" } ] }, ] } }	Creates a new project.



## 5.2 Login Route

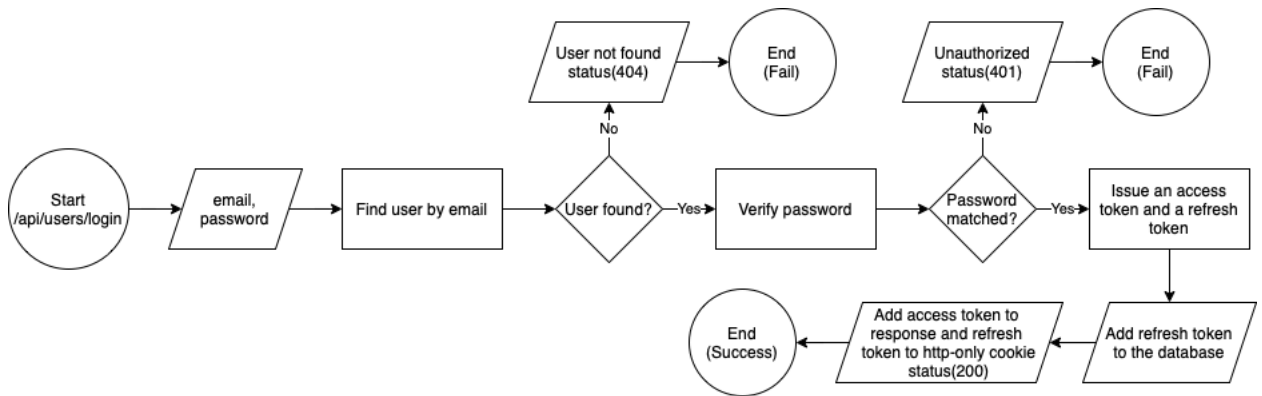


Figure 12. Login route flowchart

Backend endpoint `/users/login` is used to authenticate users in the system. This route is not protected, which means it will work even without providing access token. Flow of the login process starts with inputs sensitisation and verification. After, backend search for user in database by email provided in request. If user is found, the password is then verified and if it matched, a new access token and refresh token will be issued. Tokens are issued with JWT RFC 7519 standard. While access token is added to response object, the refresh token is attached to http-only cookie and is inserted into `refresh_tokens` table in database. Also it is important to mention, that any issued access token is valid for 15 minutes only and a refresh token is valid for 24 hours, or until user logs out (then the refresh token is removed from database).

## 5.3 Any Protected Route Authentication

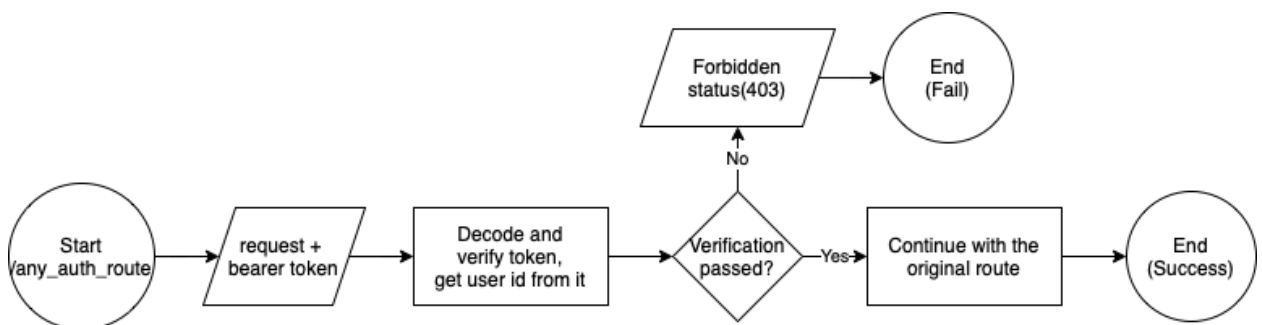


Figure 13. Protected route auth flowchart

Every request that is sent to any protected route in the backend is routed through authentication middle-ware before any other processes. The middle-ware gets access token from request authentication header section, decodes it and authenticates it. Also this middle-ware gets `user_id` from token and adds it to the variables for further processes of a given route.

## 5.4 Token Refresh Route

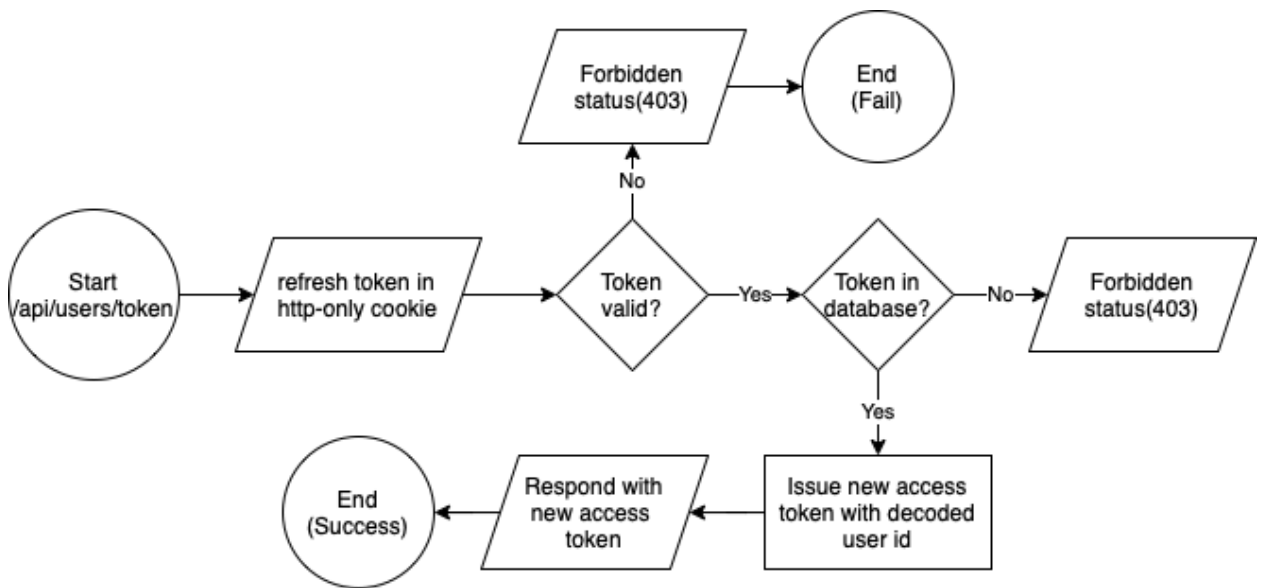


Figure 14. Token refresh route flowchart

Frontend application often sends requests to protected routes in the backend with attached access token. Access token only lasts for 15 minutes, and when it expires, front end gets a 403 response. When that happens, frontend makes a request to `/api/users/token` and includes cookies with the refresh token. When receiving that request, backend goes through new access token issuing.

First, backend checks if refresh token is not expired yet. After, it checks if this refresh token is in database. If it is, decoding of refresh token is performed and new access token is issued and attached to response.

## 5.5 Password Hashing

To ensure security, passwords are stored in the database in a hashed format. In case of a data breach or other ways of obtaining data from the database, the passwords would not be accessible to the hackers. The REACTIVE project hashes user's passwords in the backend, using the bcrypt.

Bcrypt is a password-hashing function based on the blowfish cipher. The function not only incorporates a salt, which is a random string that makes the hash unpredictable, bcrypt is an adaptive function: it is slow by design and the iteration count can be increased to make it even slower, so it remains resistant to dictionary attacks even with the ever increasing computing power. In addition to splitting the system between multiple servers, hashing passwords completely ensures the user's security.

In the Figure 15 an example of a password that has been hashed can be seen. Each part of the hashed password is separated by \$ signs in such format `$(algorithm)$(cost)$(salt)[hash]`. Each colored section is further explained below.

- **2a:** The hash algorithm identifier. In this case it identifies that the password is hashed using bcrypt.

\$2a\$10\$N9qo8uLOickgx2ZORZoMyeIjZAgcfl7p92ldGxad68LJZdL17lhWy

Alg Cost Salt Hash

Figure 15. Hashed password example

- **10**: The cost factor. This parameter is used to make ensure even further that the passwords are not decryptable. As this number grows, the amount of work the necessary to compute the hash increases exponentially.
- **N9qo8uLOickgx2ZMRZoMye**: 16-byte (128-bit) salt, base64 encoded to 22 characters
- **IjZAgcfl7p92ldGxad68LJZdL17lhWy**: 24-byte (192-bit) hash, base64 encoded to 31 characters

After the password has been hashed, the password with the rest of the data is sent to the remote database server using an HTTP request (see section 3.3 to better understand the communication between servers).

## 5.6 File Upload

Reactive application enables users to upload files, attaching them to specific events in the project. Files up to 3 megabytes in size are allowed, as to prevent from using reactive as a cloud storage service. Also only several data types are allowed. Jpeg, png, txt and sh file types are the only valid file types for upload, to prevent any malicious uploads.

Sh files are allowed, because in case of table top exercises sh files may be used as malicious files when sent in email or as slack message. However, files are stored without file extensions, just as plain text files, so they are not malicious for the server it self. File upload process can be observed below (Figure 16).

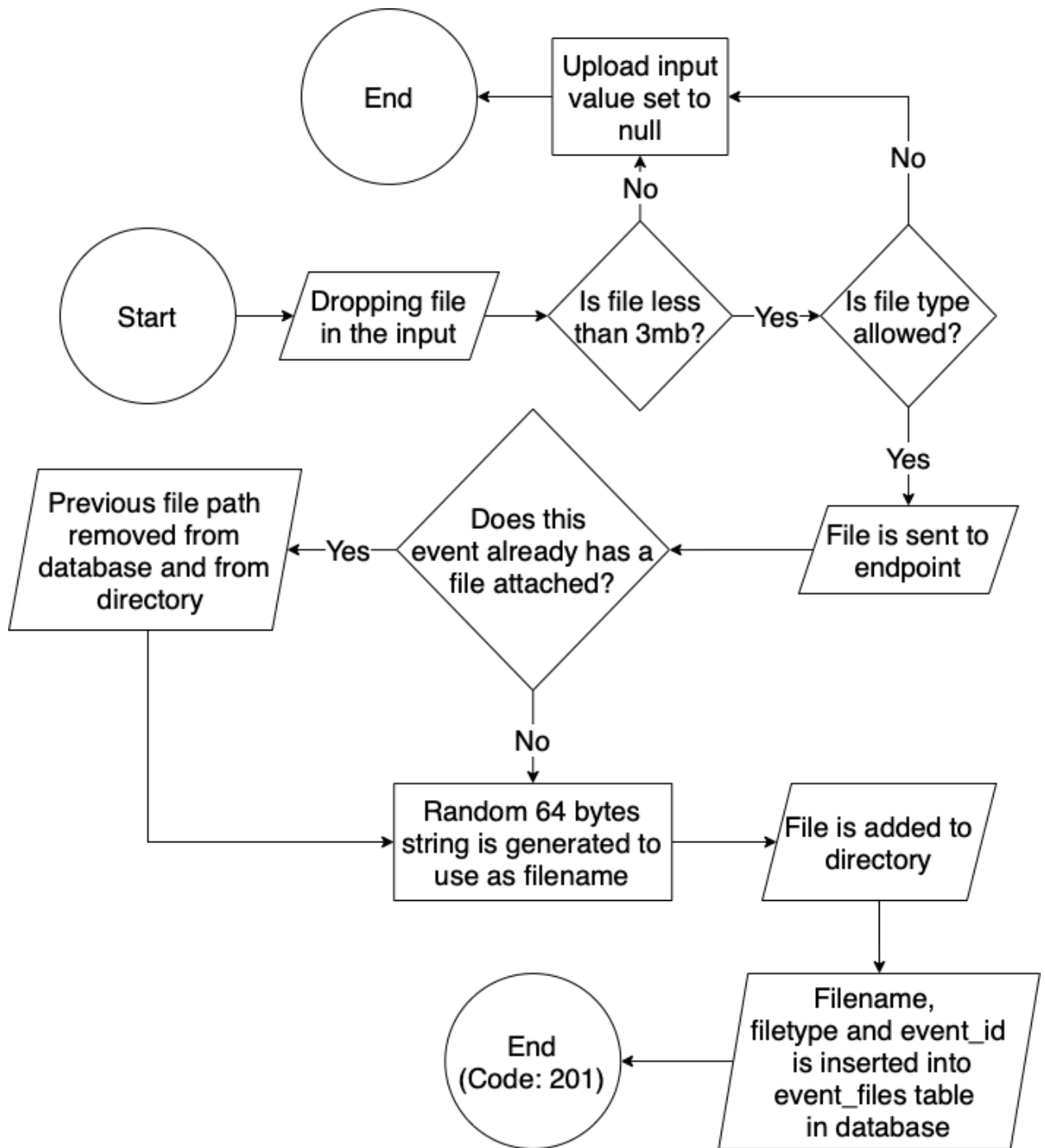


Figure 16. File upload flowchart

### 5.6.1 File Validation

First check that is executed in the front end is for file size. No files larger than 3 mb are allowed to be uploaded. This is executed by checking size property of input. If the file size is larger than 3 mb the input value is set to null and after clicking 'save changes' no file is sent to the back end. Also, the error explaining the reason for failed upload is shown in the front end.

Second check is validating datatype. It is not by just checking file extension, but by validating file type property provided from input element. If file type is not allowed, the input value is set to null and no file will be sent to back end on save of event. Also, the error explaining the reason for failed upload is shown in the front end.

### **5.6.2 File Retrieval**

When event with uploaded file is opened in the front end, the user is presented with a link of the uploaded file. When the link is pressed the file is downloaded. On the other hand, when exporting the project, file download URL is available in the project JSON file. When no file is yet uploaded to this event, file upload input is presented.

## 6 Input Validation

Input validation is performed to ensure only properly formed data is entering the workflow in an information system, preventing malformed data from persisting in the database and triggering malfunction of various downstream components.

The Reactive system also incorporates user input validation both in the front end of the application and in the back end. In this section, only user text input will be talked about, since the previous section of the report already covered file upload validation.

### 6.1 Front-End Input Validation

Firstly, every text input provided by the user is checked in the front end of the application. This is accomplished by reading the user's input and using the Bootstrap library to automatically validate the input by the set specifications.

Welcome to **Reactive** , !

Personal Details

Account Details

Security Question

Email

This is a required field

Password

.....

The minimum length of a password is 7

Repeat Password

....

The passwords do not match!

Please provide a combination of email address and a password to be able to login to the system in the future. A password must be between 7 to 16 characters and may include special symbols. No information will be shared and the passwords are hashed before saved in the database.

Register

Figure 17. Email and password validation

As can be seen in figure 17, all of the fields in form are required to be filled. The user's first and last name do not have any specific rules, the email has to contain an "@" symbol, the password must be at least 7 characters long but not more than 16 and the security question answer must also be defined. The system also checks if the "password" and "repeat password" fields are the exact same. This is done so that the user does not create a password using random characters that might be easily forgotten.

```

<Form.Control
  type="password"
  id="password"
  placeholder="Password"
  {...register("password", {
    required: "This is a required field",

    minLength: {
      value: 7,
      message: "The minimum length of a password is 7",
    },
    maxLength: {
      value: 16,
      message: "The maximum length of a password is 16",
    },
  })}
/>;

```

Figure 18. Password length validation

```

<Form.Control
  type="password"
  id="password2"
  placeholder="Repeat Password"
  {...register("confirm_password", {
    required: "This is a required field",
    validate: (value) =>
      watch("password") === value || "The passwords do not match!",
  })}
/>;

```

Figure 19. Password match validation

Figures 18 and 19 showcase the code from the application. In the first of said figures it can be seen that "Form.Control" component is used from Bootstrap. The register function has a few different instructions to check the validity of the user's input, in this case: required, minimum length and maximum length. If the requirements for the input are not met, the user is presented with the defined messages.

## 6.2 Back-End Input Validation

The back end of the application checks mostly the same validation points that are being checked in the front end. This is done to avoid any potential usage of HTTP Proxys to send dummy data into the database.

```

router.post(
  "/register_user",
  [
    check("f_name").not().isEmpty(),
    check("l_name").not().isEmpty(),
    check("email").isEmail(),
    check("password").isLength({ min: 7 }),
    check("security_question_id").not().isEmpty(),
    check("security_answer").not().isEmpty(),
  ],
  usersController.registerUser
);

```

Figure 20. Registration data validation in the back end

As seen in figure 20, once all of the registration data from the user is entered in the front end it is being sent to the "/register\_user" endpoint in the back end of the system. Then the inputs are once again checked and if the data is declared valid, the data is stored to the database and the user's account is created.



## 7 Database

Reactive takes advantage of a database management system to store various data and information: user profile data, various project and event data, refresh tokens, etc. The particular database management system used is **PostgreSQL**.

### 7.1 Relational Model

As mentioned, data is stored in a remote database server using PostgreSQL. This section explains the way information is stored and how different tables of data depend on each other.

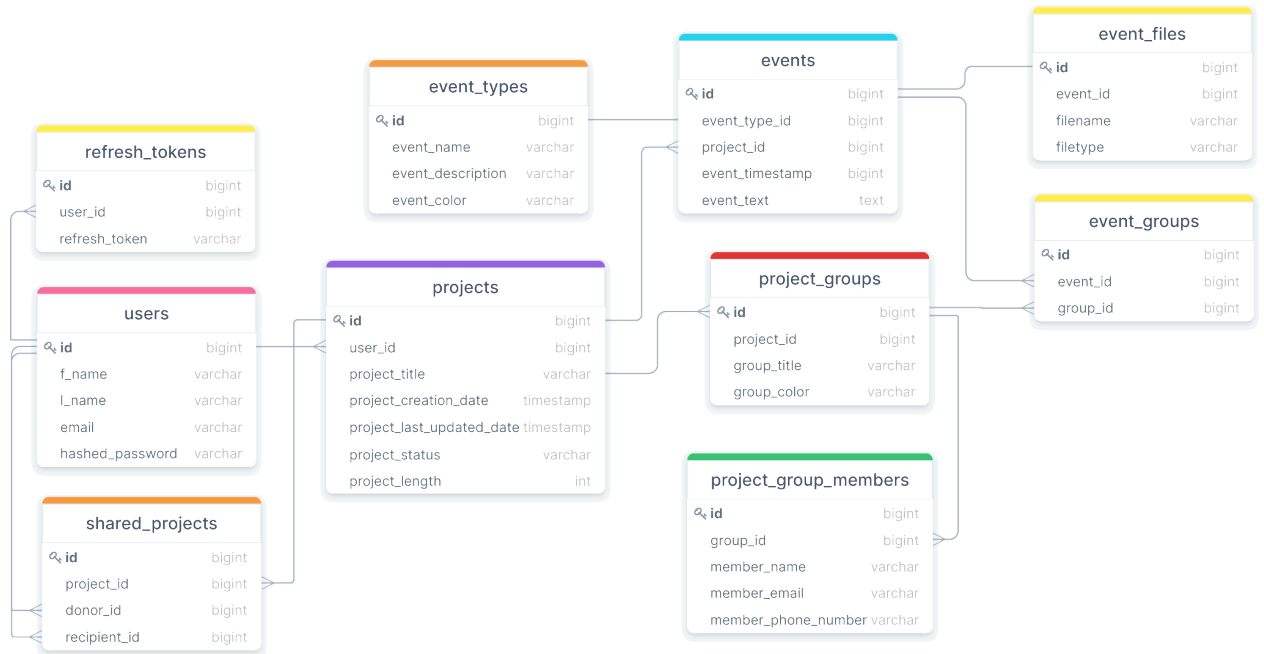


Figure 21. Relational model of the REACTIVE project

**refresh\_tokens - table containing information about tokens for authentication**

id - data entry identifier

user\_id - user identifier

refresh\_token - token for authentication

**users - table containing information about users**

id - user identifier

f\_name - user's first name

l\_name - user's last name

email - user's email

hashed\_password - user's password hashed with bcrypt

**event\_types - table containing information about singular events**

id - event identifier

event\_name - event name

event\_description - event description

event\_color - color code of the event in the UI

**projects - table containing information about the projects (scripts) of users**

id - identifier of the project

user\_id - identified of the user

project\_title - title of the project (script)

project\_creation\_date - project (script) creation date

project\_last\_update\_date - date of when the project (script) was last updated

project\_status - status of the project (new, edited)

project\_lenght - the lenght (time) of the exersice script

**events - table containing information about where particular events are used in scripts**

id - event identifier

event\_type\_id - event type identifier

project\_id - project's where the event is used identifier

event\_timestamp - timestamp of a particular event in a particular project (script)

event\_text - rich text of a particular event

**project\_groups - table containing information about groups (departments) in the project (script)**

id - group identifier

project\_id - project identifier

group\_title - team name

group\_color - team's color code in the UI

**project\_group\_members - table containing a data of members of particular groups (departments)**

id - member identifier

group\_id - group identifier

member\_name - name of the member

member\_email - email of the member

member\_phone\_number - phone number of the member

**event\_files - table containing data on uploaded files**

id - file identifier

event\_id - event identifier to which a particular file is attached to

filename - name of the file

filetype - type of the file

**event\_groups - table containing information on which events are assigned to which groups**

id - table entry identifier

event\_id - event identifier

group\_id - group identifier

## 7.2 Database Structure Overview

In the diagram above Reactive project database structure is represented. It is hosted on PostgreSQL database management system. There are 3 main parts to the system: user data, project data and events data.

User first name, last name, email and password are stored in Users table. Id is serial and identifies each user. More interesting table is refresh\_tokens table, which stores currently logged in users refresh token. These are stored in the database until either refresh tokens expire, or user logs out. Id is a serial and identifies refresh\_token / user\_id pair.

Main table in this section of the database is Projects table. It stores general data about any created project: id - which identifies the project, user\_id which identifies creator of the project, project\_title, project\_creation\_date which stores timestamp of when the project was added to the database, project\_last\_update stores timestamp of when any aspect of project was last updated, project\_status - stores if the project just started, or already in progress, or done. Project\_length - stores the duration of project tabletop exercise in minute. Next to projects, table project groups is placed. It stores data about groups that take part in this project tabletop exercise. Generic data like groups name, id and project id is available and also group colour, which stores hex of color that will be used in frontend for particular group. Project\_group\_members stores additional data about groups. Particularly members of each groups. Many fields can be stored: member name, member email and member\_phone\_number. These are mainly defined for projects future, when it will be connected to exercise execution system, so it can actually send emails, spam sms texts and various phone calls.

Events table stores generic data about event. Event\_type\_id represents which type of event it is (spam sms, work email, slack message, etc.). Project id is used to tie event to project. Event timestamp represents in which time of given project event should be executed (intervals of 10 minutes). Event text stores text input, which is assigned in frontend. This field is text data type and stores raw rich text converted to json. Event groups table is used to represent which groups are assigned to which events. It is connected to project groups and to events tables to form event\_id/group\_id pairs.

Event types is table that stores premade events, that are present in the tools section in the frontend buildscreen. Those are predefined by hand for now, but admin side creation of premade events is in the pipeline.

## 8 Functional Requirements

The functional requirements are a way for the user to communicate with the developers. Therefore, the team has decided on using user story template to represent functional requirements.

- As a User, I expect to be able to run the application on any of the popular browsers.
- As a User, I want to create an account to have a layer of security with scripts and data.
- As a User, I want to have an account so I would not lose any of the data in case of device failure.
- As a User, I expect to be able to create custom events.
- As a User, I expect to be able to download the scripts once I have finished them.
- As a User, I expect to be able to edit previous scripts.
- As a User, I expect to be able to import scripts from other users and edit them.
- As a User, I expect to be able to exit the editor after saving the progress and return to finish it at a later date.
- As a User, I expect to be able to save all my scripts remotely on the website.
- As a User, I expect to receive JSON output from my built scripts.

## 9 Non-functional Requirements

**SECURITY** All of the system components are separated from each other to ensure maximum security in case of a cyber attack. The communication between the components is done through HTTP requests to ensure reliable data encryption. Sensitive user information is transformed into hashed form before being sent out and stored in the database.

**COMPATIBILITY** The system's services can be accessed through any of the following web browsers: Google Chrome, Microsoft Edge, Firefox, Safari, Opera and Opera GX. As long as the device can run previously mentioned web browsers by meeting their requirements, the workflow of using REACTIVE's services will be smooth. To ensure the best experience, a latest version is recommended but not mandatory.

**USABILITY** Most users, that are familiar with Drag-And-Drop principal in other websites, will have no issues using REACTIVE, since the idea of the system is to be as user-friendly as possible to an average internet user, whilst pushing the power of the tools to the fullest. In addition, a dark themed version of the website might be support and proper accessibility requirements will be met to ensure the accessibility to all users without damaging their health.

**SCALABILITY** To ensure that the system can handle the projected increase in user traffic, data volumes, etc., various tests are done to determine the application's limits and its causes. Since the application is mostly based on the API, testing parameters are testing the amount of requests at once the system can manage, how many users can be logged in at once and other ways of pushing the API and web server to the limit.

**PERFORMANCE** The application contains one page, which dynamically changes based on the user's input, but REACT is well optimized to handle this. No calculations are done in the client-side of the service, therefore, the user will not experience any performance decrease, providing the user smooth experience in using the website. Changing between parts of the pages takes less than a second.

**ARCHITECTURE** For the relational database management system - PostgreSQL will be used. For the service that is the medium for the connection and communication between the client and database server - Node.js with Express.js will be used. The client-side will be written using React's JSX code which is then compiled into HTML, CSS, SCSS and JavaScript.

## 10 Human-Computer Interactions

Human-computer interaction (in short HCI) is a multidisciplinary field of study focusing on the design of computer technology and informational technologies, in particular, the interaction between humans (users of the system) and computers (the system). In order to create products that are intuitive the HCI study field combines User Interface (UI) and User Experience (UX).

Heuristic evaluation is a process where experts use rules of thumb to measure the usability and experience of user interfaces. Evaluators use established heuristics (the REACTIVE project uses Nielsen-Molich's heuristics) and reveal insights that can help design teams enhance product usability from early in development.

### Ten Nielsen's heuristics

The ten Nielsen's heuristics are provided below with additional examples of how the Reactive system satisfies them.

- 1. Visibility of system status** - the system should always keep users informed about the current state and what is going on. In the React system this is done through appropriate visual effects happening within reasonable time of user interaction. The effect is achieved through various custom REACT components such as Popups, Modals, Alerts and so on.
- 2. Match between system and the real world** - the system should follow simple and common users' language, with phrases and concepts that are familiar to the user instead of system-oriented and complicated terms. In addition to simple language the Reactive system uses common examples found on other websites (such as Drag and Drop functionality in script builder) to fulfill this criteria.
- 3. User control and freedom** - the system should provide the user with appropriate means of abandoning an action by a press of a clearly marked "emergency exit" button in case of choosing a system function by mistake. Every modal in the system or a popup has an exit button at the header of the component.
- 4. Consistency and standards** - the system should follow set standards in regards to styling, wording, phrasing and actions. A consistent color palette should be also kept to give the user the feeling of familiarity and easier understanding. Throughout the Reactive system, the color palette is preserved and the same components used.
- 5. Error prevention** - the system is carefully crafted with error prevention before the error occurs. In case of an error occurring, the user is presented with an error message clearly stating what went wrong. Various checks within the system's code and backend are present to prevent the user of experiencing failure in the system.
- 6. Recognition rather than recall** - the system minimizes the user's memory load by making important objects or actions visible throughout the usage of the system. The Reactive system does not rely on user's memory at all, in fact relevant data is always displayed on the screen.

- 7. Flexibility and efficiency of use** - the system should have accelerators that speed up the interactions between the system and the user. Tools such as drag n drop, keyboard shortcuts and other tools are used to speed up the experience of a user.
- 8. Aesthetic and minimalist design** - the system does not include irrelevant information in any of the components, since it competes with the relevant units of information and disrupts the efficient and simple workflow of the user.
- 9. Help users recognize, diagnose, and recover from errors** - as mentioned before, the system contains error messages that express in plain and simple language the root and cause of the problem, and a solution to it. For example, if the user tries registering with email address that is already taken, an error message will display that information.
- 10. Help and documentation** - the system can easily be used without a need for documentation, however in some cases, the help of documentation may be needed. This will help the user to complete an action without needing for additional human support. For now, the Reactive system does not have a documentation, but if continued, the project will require documenting some features for the ease-of-use.

## 11 Conclusions

In conclusion, Reactive is web application that is created to eliminate the tedious, traditional process of creating in-house table top exercise scripts. Reactive is free to use and mainly aimed for every corporations/organizations in the IT field, but can easily be customized for any field that would be able to integrate table to exercises into their system. The system uses various JavaScript libraries to achieve the most important goal - convenience. Keeping that in mind, the system has been implemented with a drag and drop system to create table top exercise scripts in the most easy and interactive way. Moreover, users can do collaborative work when creating their perfect exercise script - this helps make the work much more efficient. Users can create, share and export as many table top scripts as they like and access them easily from the Reactive database. Scripts can also be edited and are never locked from the user, creating the possibility to create script templates to be used in many scenarios. The minimalistic design of the application also helps users not to get lost in the details and gets straight to the point. All in all, the application is made with the philosophy of giving the user the platform to express their creativity by the use of powerful, but easy-to-understand tools, achieving faster results and being usable by all types of users, not just the tech-savvy ones!

Future developments of Reactive are to create the functionality for users to import scripts from a previously exported JSON format file, implement an email based password recovery system and adapt the user interface to support mobile device users.