

Bucharest Academy of Economics Studies

Faculty of Cybernetics, Statistics and Economic Informatics

Mycelium

Browser-based IDE for interactive and proactive teaching

Coordinator

Prof. univ. dr. Catalin Boja

Candidate

Sebastian-Andrei Dochia

Bucharest 2021

**Statement regarding the content’s originality and accountability**

Everything that is presentment in this document is the result of my own efforts and it’s entirely original. The only exceptions are instances where I either refer the works of other authors. Any such material from other sources (books, articles, websites or GitHub pages) are clearly stated in the bibliographic references list.

**Table of Contents**

[**1.** **Introduction** 3](#_Toc75799690)

[**2.** **Remote teaching** 5](#_Toc75799691)

[**3.** **Used Technologies** 8](#_Toc75799692)

[**3.1** **Frontend** 8](#_Toc75799693)

[**3.2** **Backend** 10](#_Toc75799694)

[**3.3** **Database** 12](#_Toc75799695)

[**4.** **Solution Architecture** 12](#_Toc75799696)

[**4.1** **The webpage** 13](#_Toc75799697)

[**4.2** **The API** 13](#_Toc75799698)

[**4.3** **The database** 15](#_Toc75799699)

[**4.4** **Usage** 15](#_Toc75799700)

[**5.** **Solution Implementation** 15](#_Toc75799701)

[**6.** **Conclusion** 22](#_Toc75799702)

[**References** 24](#_Toc75799703)

[**Appendix** 26](#_Toc75799704)

# **Introduction**

Since the global pandemic hit the world and brought teaching in the online medium many of its shortcomings became apparent. During physical classes, it was already hard for the professor to track every student’s performance and his rate of assimilation regarding the subject. Students were usually reluctant to ask for help and the professor could only guess if they have a problem or not based on their facial expression. Now that classes moved online this problem has become even bigger. Most students don’t ever signal that they did not understand something or that they have a compiler error. The professor can’t even see their expressions so he can only assume that everything is in order when in reality he knows that there are at least a couple of students who have problems. Hitherto, there is a huge lack of feedback going between the professor and the student that leads to situations where the student fails to grasp certain concepts that will lead to him having even greater problems down the line.

In a situation like that described above the logical solution is to encourage the student to seek help when he needs it. However, this solution is unreliable so the next best solution is to enforce the student-professor feedback loop through the creative use of software solutions. That is exactly what I tried to archive with Mycelium. When it comes to coding there is one really simple way of seeing whether or not the student understood the concept that was just taught to him: Compiling and running his code. Most often than not students will get compilers errors and will simply give up without communicating their problem with the professor. With Mycelium that is not a problem as the teacher can see the last output of every student and whether or not his code has errors.

Mycelium aims to solve the professor-student feedback problem while also providing features such as secure accounts, data encryption, a secure backend with built-in resistance to most attacks, and clear separation between account types (student/professor), each with its permissions and restrictions enforced both on the front-end and the back-end. The goal of the application is to feel safe and easy to use while also streamlining the teaching process.

A great advantage of using this app is that the professor knows exactly which student has problems and the exact problem he is having. This will greatly speed up the teaching process and will increase its effectiveness as the professor always knows how the students are performing without the need for them to signal that they have a problem. Another advantage is the fact that Monaco [1] is used as the text editor which offers the same code writing experience as Visual Studio Code.

As I mentioned before this idea came as a direct result of the current global situation. When we were forced to interact using only microphones, we lost an entire dimension of the teaching process and it all became linear and non-interactive. In this current context, the app that I created will help alleviate this problem. Students will be unable to just join the zoom and pay no attention to the ongoing class as the professor will see their code outputs and is also able to view the code in its entirety. So, the students will not be able to hide the fact that they could not follow along and will be encouraged to seek help when they need it thus ensuring professor-student communication. Moreover, the application is not only useful when it comes to remote teaching. Its features could also help streamline the physical teaching process as the teacher will not have to rely on guessing whether students understood the concept or were able to implement a certain task. They will be assisted by a tool that tells them exactly who and what problem they have. Hitherto, the usability of Mycelium is not relevant only in the context of remote teaching but also for on-site teaching.

The following chapters will go in-depth on certain details of the application starting with an ample description of the problem at hand. Namely, remote teaching. After explaining where the need for such an application arose from, the technologies that have been used in the creation of the solution will be promptly presented. Once done with that aspect, the architecture of the application where the structure and use case, as well as the diagrams for the components and classes, will be explained and analyzed. Ultimately, in the implementation chapter, there will be an overall view of the application code and structure as well as a showcase of how the most important features were implemented. And finally, in the conclusion section, some final thoughts on the application will be expressed as well as some ideas for future development.

# **Remote teaching**

With the global pandemic and the introduction of the general quarantine, teaching was moved to the online medium. This proved to be a challenge both for students and professors alike. Remote teaching can take many forms, each with its own variations. The method of teaching had to be chosen to fit the subject as some subjects require more dedicated live teaching while others can be taught with minimum interaction, if necessary, resources are provided. The most common type of teaching, at least in computer science universities, is the meeting on Zoom, Microsoft Teams, Discord, and so on. During such a mating, the teacher shares his screen for all students to see while using his microphone to explain in further detail. In the ideal case, the students would also ask questions and interact with the professor. When it comes to coding classes, many shortcomings of remote teaching become more apparent. In this case, the device used to connect and teach for both professors and students begin to matter and makes a huge difference in how well the students will be able to follow the class. For example, if a student has a two-monitor setup, he will be able to watch the professor’s stream and write his code at the same time. Meanwhile, someone who only has one monitor will have to split his screen. His situation can be made even harder if the resolution (or size) of his screen is smaller. Moreover, students tend not to communicate with the teacher out of commodity, anxiety, or other such conditions. Hitherto, when it comes to teaching code in the online environment many other factors get in the way such as the student’s display size and quality, the performance of his machine, or even his willingness to follow along and write code at the same pace as the professor.

Everyone can agree that the teaching process does not go as smoothly in the online medium. It is also important to mention the fact that I will be mostly referring to coding and the process of teaching code. Even though programmers were amongst the least affected by the current pandemic [2] and the process of teaching software development didn’t have to change that much, it is still an extremely important field of study that would benefit from improvements that would make it more accessible and easier to teach. Comparing online teaching with physical teaching we can draw some simple conclusions about how to process goes and how efficient it is. Firstly, we can take a look at the interaction between students and the professor. During on-site teaching, the teacher can freely go up to a student and ask him how he is handling the subject or get instant feedback from the entire classroom by just glancing around, looking at what the students are doing, their facial expressions, and so on. Moreover, students are more inclined to ask questions when the only thing they have to do is look at the professor and speak. In contrast to this, during online teaching, students are way more likely to either not pay attention to the class or to be unable to follow and not signal this fact. Interacting with the teacher is more complicated as anxiety problems (from everyone being able to hear each other perfectly), internet problems, lack of interest, or bad microphone quality get in the way of the student-professor interaction. Thus, it requires a greater motivation to actively participate in the course. Following the professor during the class might also be a barrier that needs to be overcome by some students as they need to both watch the professor’s stream and write their own code. This is extremely hard for someone with only one monitor especially if it a small, low-resolution one like the monitor of an average laptop. So, students are likely to fall behind or not properly understand what is being taught and not signal this fact. To add to all this, we have to take into consideration the fact that the only means of interaction between the teacher and students are the webcam and the microphone. (The webcam is rarely required so I will only be referring to the microphone) If the student does not turn his microphone one to interact with the professor, this leads to the professor being in a situation where he or she has no means of getting a feel on what the students were able to understand and what was too hard to grasp and needs further explaining.

There are also other gaps that need feeling in order to come up with a good online teaching solution. Those who have built online programs over the years will attest that effective online learning aims to be a learning community and supports learners not just instructionally but with co-curricular engagement and other social supports. Consider how much infrastructure exists around face-to-face education that supports student success: library resources, housing, career services, health services, and so on. Face-to-face education isn't successful because lecturing is good. Lectures are one instructional aspect of an overall ecosystem specifically designed to support learners with formal, informal, and social resources. Ultimately, effective online education requires an investment in an ecosystem of learner supports, which take time to identify and build. Relative to other options, simple online content delivery can be quick and inexpensive, but confusing that with robust online education is akin to confusing lectures with the totality of residential education. [3]

In order to solve this problem (more specifically the lack of student-professor feedback going around during the class), I tried to come up with a software solution that can be useful both in remote teaching situations and physical classes. What sparked the idea in the first place was re repeating pattern I kept observing at all coding subjects. Almost always there were people complaining in private that they couldn’t follow the code or did not understand the lesson. The professor has no way of knowing this and has to keep going on with teaching as if everyone followed perfectly. So, the best way to approach this issue is to create a real-time way for the professor to see how the students are handling the subject and if they are following along or not. Furthermore, the solution must be secure and trustworthy.

To expand on the security side of things, from past events I’ve noticed a great lack of trust regarding solutions developed by the university and as personal data becomes increasingly more valuable, solutions, especially those who handle the data of students who are compelled to use the solutions should be properly secured. The students tend to be distrustful of applications that ask for too many permissions, trigger the antivirus, or seem easy to hack. Thus, I decided that the solution I develop should be secured to the best of my abilities.

As I mentioned before, in the context of the global pandemic and with classes moving into the online medium, it is extremely important to create the proper environment for students. This should include both access to resources and more accent placed on individual work, as well as better interaction between students and the professor. Encouraging students to ask questions and seek out help should be one of the main priorities of an educational institution. Thus, there is a need for a software solution that could help solve this problem through a simple and secure approach. If there is nothing to encourage and even force the student to ask questions and interact with the professor, he or she will most likely choose the comfort of sitting muted the entire session and possibly not even paying attention to what is being taught. The approach through which this positive outcome should be achieved should be as simple and non-intrusive as possible. If the student knows that he is likely to make a mistake because he did not understand the lesson and that the professor will be able to see the way he or she is performing, they will be more likely to seek help beforehand and will not end up in situations where they failed to understand a concept and fell behind while the professor has no idea of this. Ultimately, such an approach will lead to fewer cases when the professor only finds out how well the students understood the subject at the exam and will be able to follow their progress much closer.

# **Used Technologies**

The application has three major components. Those being the frontend AngularJS webpage, the backend NodeJS REST API and the MongoDB database. All those parts can work fully independent from each other and can be tested as such.

## **Frontend**

The website page was developed using the AngularJS [4] framework and was written using TypeScript [5], HTML5 and CSS3. Additionally, Bootstrap [6] has been used in a few instances for a quick-to-build, standard and pretty visual interface. For the code editor, the Monaco Editor has been used. Angular was also enhanced through the use of RxJS [7].

Angular has been chosen for a couple of reasons. Mainly for its “holistic” software development approach. It is entirely self-sufficient and works without the need for additional libraries. It is also a full-fledged JS framework unlike React which is a library for UI development. It also works for both web and mobile (if integrated with Ionic or NativeScript) and has great potential for future improvement as it is used by large companies such as Apple, Adobe or Microsoft. It is especially useful in my context as it gives access to the model, the component and the view and is optimized with change detection as opposed to the virtual DOM approach of React. With the help of Angular CLI, you can further refine the experience as it allows you to quickly perform actions from the command line. In addition to all the already built-in features, Angular supports dependency injection and can be improved with RxJS or NgRx. Data binding, component-based routing, project generation, form validation, and dependency injection – can be implemented with the means of Angular package. The only downside of using Angular in the project is the fact that Angular itself is a huge library, and learning all the concepts associated with it will take much more time than in the case of React. Angular is more complex, harder to understand, there are a lot of syntax difficulties such as the need of knowledge about another language, namely Typescript, and component management is intricate. Some complicated features are embedded into the framework core, which means that the developer cannot avoid learning and using them. Moreover, there are a lot of ways of solving a single issue. Although TypeScript closely resembles JavaScript, it also takes some time to learn. Since the framework is constantly updated, the developer needs to put some extra learning effort. [8]

Bootstrap is a front-end framework used to quickly design responsive sites. It is extremely useful in situations where you need to quickly create an interface that will work well both on mobile a desktop web browser. Bootstrap includes user interface components, layouts and JS tools along with the framework for implementation. “In the physical world, a bootstrap is a small strap or loop at the back of a leather boot that enables you to pull the entire boot on and in general usage, bootstrapping is the leveraging of a small initial effort into something larger and more significant. There is also a common expression, "pulling yourself up by your own bootstraps," meaning to leverage yourself to success from a small beginning.” [9]

TypeScript is an open-source programming language which builds upon JavaScript. The main difference between it and JavaScript is the fact that TypeScript is a strict syntactical superset of JavaScript and adds optional static typing to the language. It ultimately transcompiles to JavaScript via the TypeScript compiler or Babel. I decided to go with TypeScript because of how enforcing types on variables helps develop complex apps. It increases testability and speeds up debug times as many errors can be caught ahead of compile-time in the editor. It also has great long-term support as it is developed and maintained by Microsoft.

HTML stands for HyperText Markup Language and is the standard markup language for documents designed to be displayed in a web browser. It offers many options for structuring content in a webpage and it consists of a series of elements, which you use to enclose, or wrap, different parts of the content to make it appear a certain way, or act a certain way. The enclosing tags can make a word or image hyperlink to somewhere else, can italicize words, can make the font bigger or smaller, and so on. [10] HTML can be further enhanced by adding styling through the use of CSS or Cascading Style Sheets and scripting languages such as JavaScript.

CSS stands for Cascading Style Sheets and it is a style sheet language used for describing the presentation of a document written in a markup language such as HTML. [11] Through its many features such as selectors, declaration blocks, sources, positioning and so on, CSS offers incredible flexibility when separating and styling the presentation of content. This improves the accessibility of the webpage and enables reusability as the code can be applied to any HTML document thus creating the possibility of reusing the same CSS for many different documents.

The Monaco Editor is the code editor the powers Visual Studio Code. It is an open-source editor that is designed for high productivity. When used together with programming language services, it gives you the power of an IDE and the speed of a text editor [12] and makes it ideal to be used in a browser environment. For the project, I decided to use an open-source project named “ngx-monaco-editor” [13]. It is an implementation of the Monaco Editor as an Angular component. This allows easy and seamless integration of Monaco inside any Angular project. It can simply be added to angular as any regular dependency by using the Node Package Manager or npm.

RxJS or Reactive Extensions Library for JavaScript is a library for composing asynchronous and event-based programs by using observable sequences. It provides one core type, the Observable, satellite types (Observer, Schedulers, Subjects) and operators inspired by Array#extras (map, filter, reduce, every, etc) to allow handling asynchronous events as collections. [14]

## **Backend**

The REST API has been developed using NodeJS [15] and additional modules or “middleware”. Those middleware components provide functionality from security to colored strings in the console for improved readability. For security purposeless bdcrypt.js, cookie-parser, cors, express-mongo-sanitize, helmet, hpp and xss-clean were used. Additional middleware such as detenv, node-jsonwebtoken, mongoose, morgan, nodemailer, nodemon, slugify and colors were used either to make the development process easier or to provide ease of use for the user. It was created as the core of the application and any frontend application can communicate with it and use it in a simple and streamlined manner as long as it knows the rules. The entire code is simple and robust and satisfies all principles of clean code in order to be easy to understand by anyone willing to use it. It can be further expanded and features can be turned on or off by simply commenting a single line of code. The added functionalities of the third-party middleware and the reason for their use are the following:

* It uses the dotenv package in order to access and use a set of environment variables that are constant and can be easily changed. Such as the database connection link or settings for the security middleware.
* For database communication the mongoose package was used as it is simple to use, expertly powerful, feature rich and well maintained. Even though the database is nonrelational, mongoose provides great querying ability as well as clean table schema creation and management.
* For logging purposes morgan was used in conjunction with colors.js.
* As changing something in the node app requires a restart of the server, I used nodemon solely for the development environment. This package will automatically restart the server when it detects changes to the files.
* As logging in is an important process because it serves as the user’s gateway into the app, it was designed to be as simple and secure as possible. The bcrypt.js package was used in order to hash and encrypt the password.
* node-jsonwebtoken was used to generate web tokens that will be sent to the client and stored as cookies. They are also the key to the application as almost all routes are protected and require authorization. That is why any request has to have the required headers with the java web token bearer token to serve as the authentication key in order to determine if it belongs to an actual user and figure out the user’s role (student/professor).
* For password resetting (in case of a forgotten password), nodemailer was used for sending email with the link containing the token for password reset.
* Security is an extremely important part of the backend. Many middleware solutions were used for the single purpose of eliminating risks of attacks such as XSS and DDOS. Firstly, cors was introduced for adding security headers for the CORS protocol.
* express-mongo-sanitize was used to sanitize the incoming data and prevent malicious code from being executed on the server. It acts as a barrier against sql/nosql injection.
* Http secure headers were added with helmet.
* Against HTTP parameter pollution, the hpp package was added.
* The express-rate-limit package is used to limit the number of allowed requests from a single source, good for protection against DDOS attacks.
* xxs-clean was used to prevent cross site scripting attacks.

Representational State Transfer or REST is a software architectural style that uses a subset of HTTP. It is commonly used to create interactive applications that use Web services. A Web service that follows these guidelines is called RESTful. Such a Web service must provide its Web resources in a textual representation and allow them to be read and modified with a stateless protocol and a predefined set of operations. This approach allows interoperability between the computer systems on the Internet that provide these services. REST is an alternative to, for example, SOAP as a way to access a Web service. [16]

Node.js is an open-source, cross-platform, back-end JavaScript runtime environment that runs on the V8 engine and executes JavaScript code outside a web browser. Node.js lets developers use JavaScript to write command-line tools and for server-side scripting—running scripts server-side to produce dynamic web page content before the page is sent to the user's web browser. Consequently, Node.js represents a "JavaScript everywhere" paradigm, unifying web application development around a single programming language, rather than different languages for server-side and client-side scripts. [17]

Express is a minimal and flexible Node.js web application framework that provides a robust set of features for web and mobile applications. Express also provides a thin layer of fundamental web application features. [18]

## **Database**

The database of the application has been implemented using MongoDB [19]. The main advantage of MongoDB is that it is a document database so data is stored in JSON-like documents. This offers a more natural way of thinking about data for a programmer as opposed to the traditional row/column model. MongoDB also boasts a powerful query language which is themselves JSON so is easily composable, unlike the concatenation of strings required to dynamically generate SQL queries. Moreover, non-relational databases are way faster when it comes to accessing data thus fits better for the case of this application.

MongoDB also has a big advantage in the fact that it is based in the cloud. MongoDB Atlas is the multi-cloud database service for MongoDB available on AWS, Google Cloud, and Azure. Best-in-class automation and built-in proven practices provide continuous availability, elastic scalability, and support with regulatory compliance. For visualizing the contents of the database MongoDB Compass [20] was used during the entire development process. It is an application that allows the user to search, visualize, and work with your data through an intuitive GUI and manipulate your data with a powerful visual editing tool.

# **Solution Architecture**

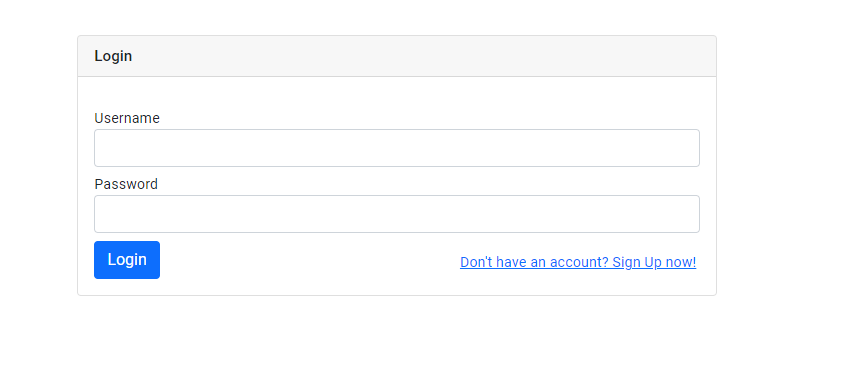
The application as a whole is made of two big components. One is the webpage and the other is the API. Another third smaller component is integrated with the API, the database. The two components are designed to work together but can also be used independently of each other. The role of the API is to handle the user accounts, application data and execute the code. Each of its component satisfies one of these roles while also providing security and control through the use of authentication mand middleware. The role of the webpage is to provide an easy-to-use interface for the user so that he can take advantage of all the features of the API. As the application was developed with the backend first approach, the frontend was designed in close relation with the API and makes it easy for the user to interact with the application.

## **Webpage**

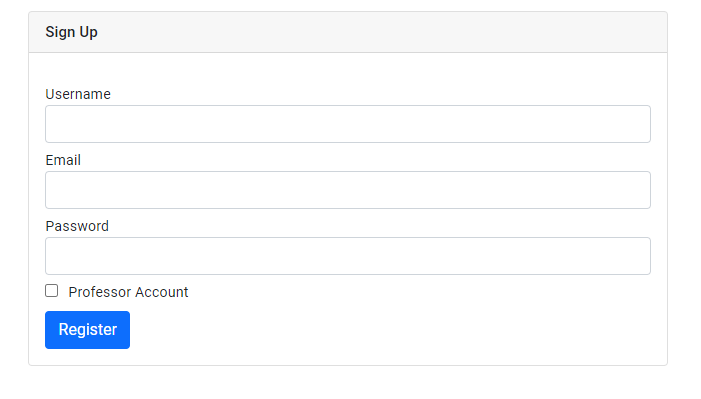
It is made out of three main components, all connected to the base one, the “app” component. Those three are: The log-in/sign-up page, the dashboard and the working environment.

**The log-in/sign-up page**

Is the page a non-authenticated user is taken upon trying to access the app. Here the user can log-in or sign-up if he doesn’t already have an account. It has a simple design with only the necessary buttons present. Even though it is not the main page of the app, it is the place where the user is taken to if the user tries to access any other route, he or she does not have access to. There are two different components used for this part of the app and the user is redirected to the login one by default but can go to the signup one by pressing a button.



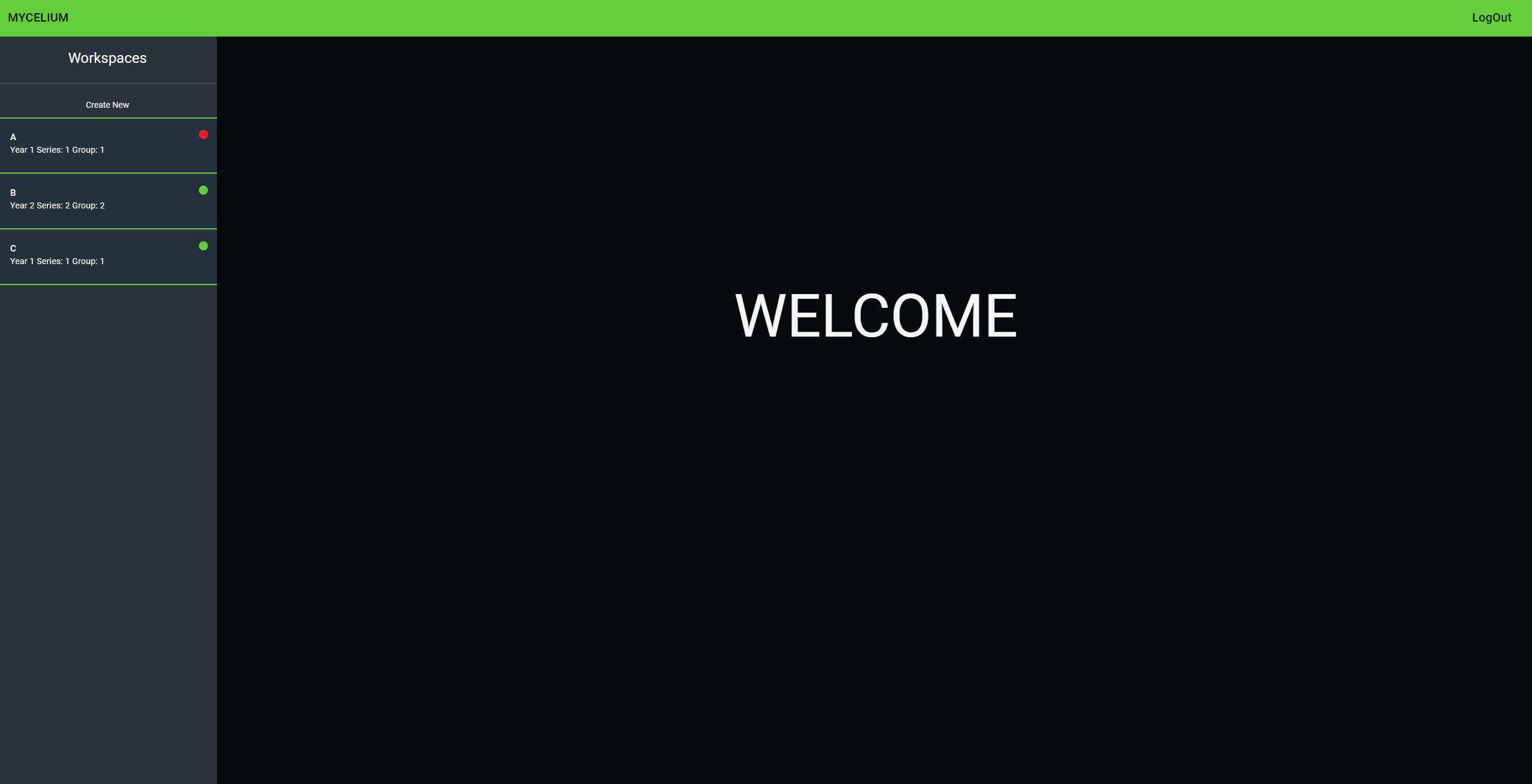
**Figure 4.1.1** – Login page



**Figure 4.1.2** – Sign Up page

**The dashboard**

This is the component where the user is able to visualize all the classes he is was added to. The professor has a different interface. Apart from being able to see the classes, he is also able to add users to a class and create new ones and also starts the class so students can enter the working environment of that class.

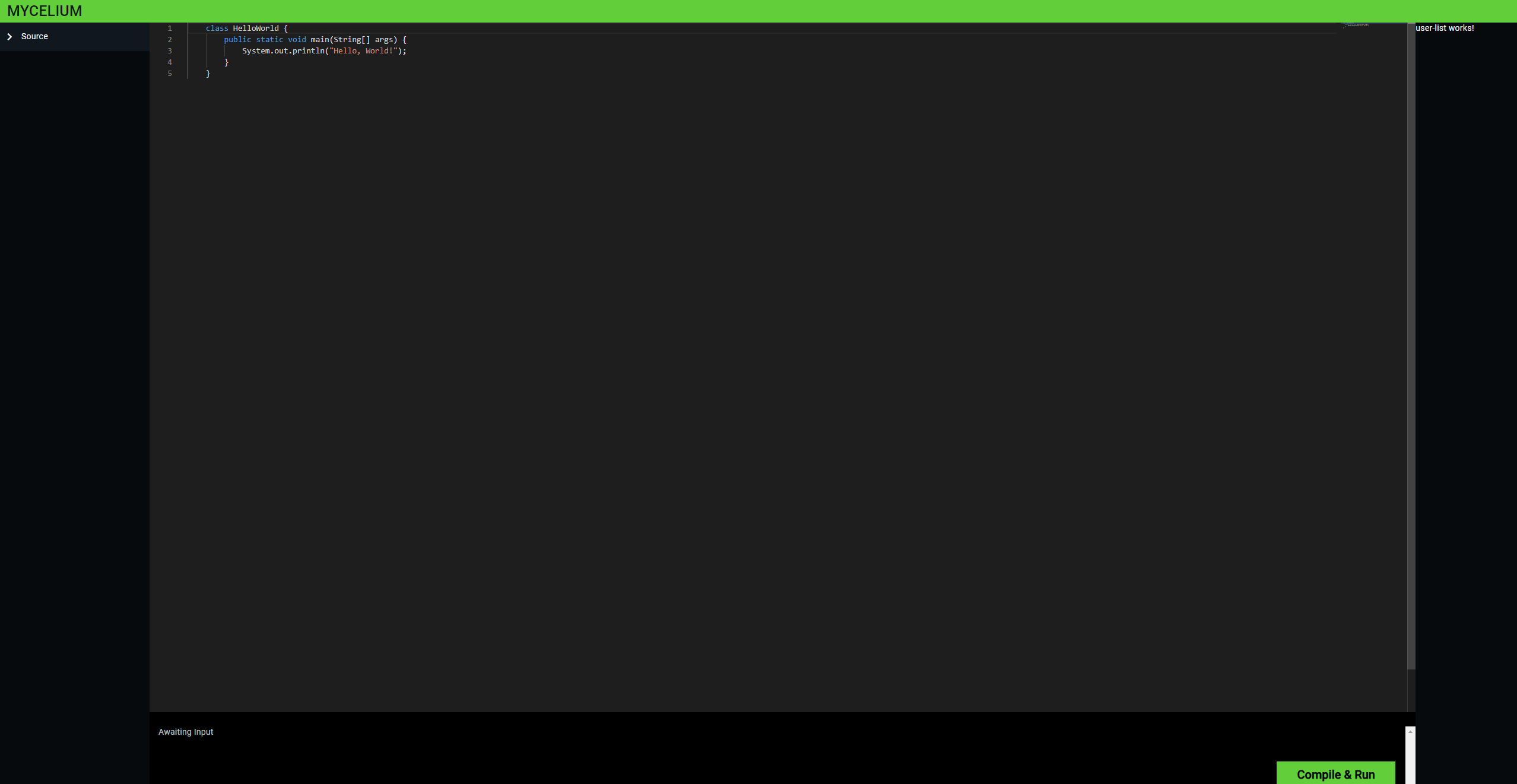


**Figure 4.1.3** – Dashboard welcome page

The dashboard is comprised of several components. First is the navbar from where the user can log out or return to the home screen depending on the button he clicks. Another component is the list of classes. Here all the classes the user is part of are displayed. Each entry in the list is another component generated based on a list retrieved from the API. Those smaller components are used to display some brief data about the class they represent. In the list of classes component, there is also a button displayed only for a professor which will load the component with the class creation. The third and last main component of the dashboard is the statistics component where data about the selected class from the list component is displayed. If the professor clicked the add class button on this component the class adding component is loaded. Moreover, this is the component from where, when a class is selected, the user can join a launched class and the professor can launch the class, add participants or close the class. See [Appendix 4.1] for a detailed component diagram for the dashboard. The dashboard talks with the backend to get all of its data as well as to know what features to display, enable or disable.

**The working environment**

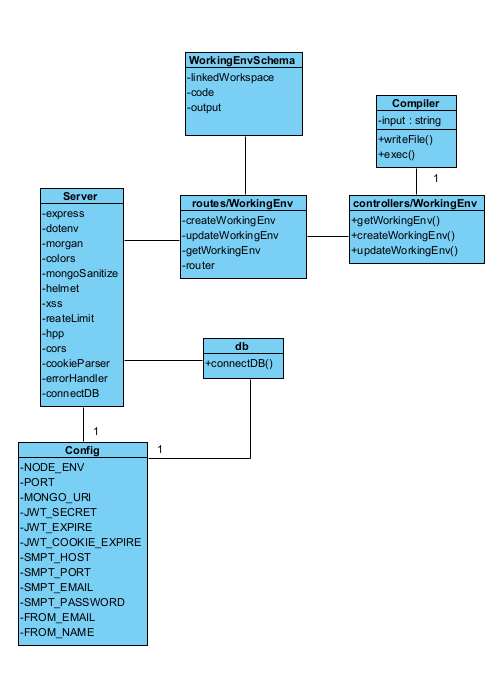
The working environment has four components. The first and the largest one is the Monaco Editor. This is where the user writes the code that is later sent to the server to be compiled and ran. The other component is the file tree where the code files can be created and selected. The last component is the list of connected that is built on the same structure as the list of classes in the dashboard by having one component that is displayed multiple times with different data. All those components are connected to the base component for the working environment which is the one the user is routed to when launching the working environment.



**Figure 4.1.4** – Working Environment

## **API**

The API has one main component, the server. Inside the server are integrated all other components, controllers, middleware. The API routes are integrated through the controllers. There is one module class for each module from the frontend in order to maintain everything clean and easy to understand. Each controller has an associated routes module where the methods of each related API call are implemented. The routes use the models for managing objects and interacting with the database. The compiler class handles all the compilation tasks and returns the results that will be sent as a response. The structure is based on a modular approach and is created by having the server as the center component. Then, there are the routes components which hold all the API end points as well as set the routs to protected and sets the authorized roles. Inside the routes class, the methods from the controllers are also assigned to a route. The controllers class holds all the methods that are being used for the API/database operations. Through this modular approach, more routes classes for different purposes can be easily added to introduce new functionalities without modifying the currently existing ones.



**Figure 4.2.1** – API component diagram

**Backend functionality guide:**

Workspaces:

GET - /api/v1/workspaces – return all workspaces

GET - /api/v1/workspaces/:id – return a workspace with the specified id

POST - /api/v1/workspaces – create a new workspace – should provide the following information: name: string(unique), description: string, year: number, series: string, group: string, owner: string

PUT - /api/v1/workspaces/:id – update a workspace (to add a participant use this route and specify the email as “members”, this will add the email to the current list of members)

DEL - /api/v1/workspaces/:id – delete the workspace with the specified id

WorkingEnvs:

POST - /api/v1/working-env – create a new working environment

PUT - /api/v1/working-env/:id - update a working environment (use “isStarted” for toggling its state) – should provide following information: name: string(unique), description: string, year: number, series: string, group: string, owner: string, isStarted: boolean

GET- /api/v1/working-env/:id - get a working environment with the specified id

Authentication and User Management:

POST - /api/v1/auth/register – register a new user - should provide following information: name: string, email: string, role: string (should either be “student” of “professor” or “admin”), password: string

POST - /api/v1/auth/login – logs the user in and generates his token – should provide the following information: email: string, password: string

POST - /api/v1/auth/forgotpassword – will send an email to the email address of the request containing the password reset link - should provide the following information: email: string

PUT - /api/v1/auth/resetpassword/:token – will look for the token and if it is valid, it will change the password - should provide the following information: password: string

PUT - /api/v1/user/updatedetails – updates the details of the currently logged in user

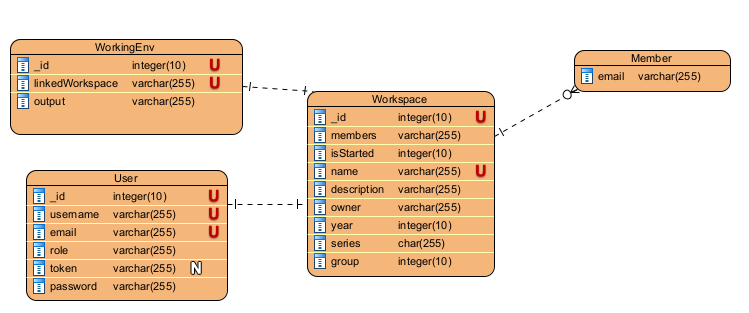
PUT - /api/v1/user/updatepassword – updates the password of the currently logged in user

GET - /api/v1/auth/me – gets the details of the currently logged in user

GET - /api/v1/auth/logout – logs the current user out

## **Database**

The database is not relational and the tables are only linked by the backend logic by using lists of ID’s. This is used for creating ownership. The structure of the database is pretty simple and its main use is for the user accounts. Each user account has one or more Workspaces linked to it (if the user is a professor, otherwise he cannot have associated workspaces) while each workspace has multiple Members and an associated Working Env. The members of a workspace hold a reference to user profiles so the backend knows which users should be able to access which workspace. Moreover, each working environment holds the id of its linked workspace so it can easily be retrieved on load. The database is non-relational so all tables are independent JSON structures linked only through code and special queries.



**Figure 4.2.2** –Database diagram

## **Usage**

The user interacts only to the frontend part which talks to the backend which, in turn, talks to the database. The user creates an account then logs in using the sign up and the log in page which are the first and only ting a non-logged in user is will see. After logging in, the user will be able to see all classes he of she was added to. A professor has to add him to an existing class and when the class is started, the student can join the ongoing working environment. This will launch the working environment page. There he or she can write code and run it. The professor can see all connected users and the output they had on the code they last ran as well as if they have any error or if their code ran with no problems. This is done in the members list component. The professor can also end the class. An example of the user interactions and the flow of the application can be seen in the diagram found at [Appendix 4.2].

# **Solution Implementation**

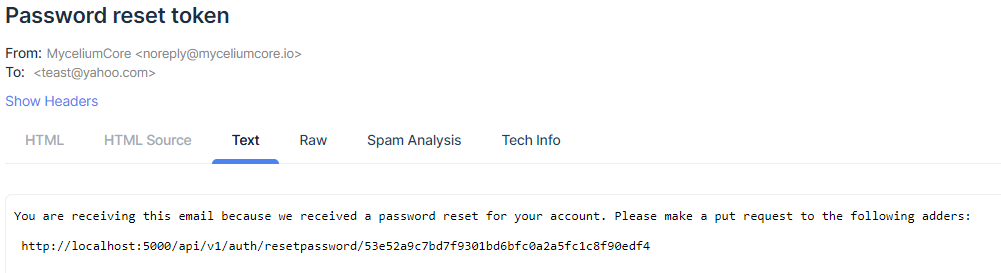
The solution is first and foremost a way to organize and streamline programming class. So, there were certain things that had to be implemented in order to satisfy the baseline conditions. There needs to be a way for managing the classes, which student is in which class and what classes they can and cannot join. Apart from this there was also the need for a working environment where students can write and run code. In this environment the professor should be able to see how the students are progressing and if they have problems or not.

First and foremost, the application needed an account system. This account system also needed to be secure and have full functionality. Thus, the API has implemented routes for account, creation, logging in, changing the password, resetting the password and getting the data of the currently logged in user. This is achieved through the use of a “BEARER token”. The token creation and management are done by the backend through the use of a custom auth middleware. The middleware exports two methods: **protect** and **authorize**. The **protect** method checks the headers for any authorization content and saves the token. If the token is not found an error response (using the **ErrorResponse** custom middleware which is used to return different error types and messages in the response of a request) is issued and the method returns. On the other hand, if a token is fount, it is decoded using the jsonwebtoken middleware using the **JWT\_SECRET** global variable stored in the **config.env** file. After the token is decoded, a request to the MongoDB database is sent looking for the user with the Id retrieved from the token. If the user is not found, the function returns and a “Not authorized to access this route” error with a code of 401 is issued and sent in the response exactly the same as it is for an inexistent token. However, if the user is found, **next()** is simply called and the request will go through. The other method exported by the auth middleware, namely **authorize**, is used to check the user role to see if they should have permission to do a certain operation. The roles allowed to access certain routes are assigned in the routes file by giving the method with the allowed roles as parameters as a parameter in the router.

**router.route('/').post(protect, authorize('professor', 'admin'), createWorkingEnv);**

In case the user forgot his or her password there is a route implemented for password reset. The user first makes a request with the email associated with an account. Then nodemailer will be used inside the **sendEmail** custom module. Here a transport will be created containing some text and the token used for resetting the password. When the first request with the email is sent, inside the database, on the entry of the user that was found, two proprieties are added: the password reset token and the expiration time and date of said token. After the user receives the email with the token (which is in the form of a link) he has to make a request to that address and the content should be the new password. Once the request gets to the backend, the database is searched for a user with a matching password token that is not expired. If one is found, the password is changed and the properties that were added are deleted.

For development only, mailtrap.io [21] (which is a website for catching emails and viewing them instead of sending them to their intended destination) was used as the email addresses used for testing are not actual email addresses and it also made the testing process a lot faster and simpler.



**Figure 5.1.** Example of a password reset token email

Security was a big concern when working with accounts and passwords. So, all passwords are hashed and then saved and the password is never stored inside the browser cookie or in the browser storage. The JSON web token is also signed and expires after a certain number of days.

But stolen passwords are only one of the problems that could arise from the lack of security. The combination of NodeJS and MongoDB has some vulnerabilities that could lead to the entire integrity of the application to be compromised. The biggest one is the way MongoDB fetches data based on the received parameters. Through a technique much simpler than SQL Injection this can be achieved. Considering the next line of code, the user can authenticate from anywhere in the app:

**SELECT \* FROM users WHERE username = '$username' AND password = '$password'**

For MongoDB and injection primer can be used and the same result of an SQL Injection can be achieved by writing this int the request:

**{**

**"username": {"$gt": ""},**

**"password": {"$gt": ""}**

**}**

Going even further, In ExpressJS we can still achieve the bypass effect without using JSON at all but simple query strings. For example, we can submit a request like the one illustrated bellow. Use this link to open the request in Rest:

**POST http://target/ HTTP/1.1**

**Content-Type: application/x-www-form-urlencoded**

**username[$gt]=&password[$gt]=**

In order to avoid those exploits the middleware express-mongo-sanitize has been integrated in the API and is used for any type of incoming request.

The compiler is another integral part of the backend. The Compiler class with the **compileAndRun** method is responsible for handling incoming code. It returns a promise containing the result of the received code. It works by firstly writing the code to a temporal “.java” file and then using a child process created through the use of the **child\_process** package, which is a package which allows the creation of a new command prompt process, to run the “**javac**” cmd command and then run the “**java**” cmd command on the file created by “**javac**”. The result is then resolved and the result is put in the response and sent back to the client.

**exec(`cd tmp & javac tmp.java & java HelloWorld`, (error, stdout, stderr) => {**

**if (error) {**

**console.log(`error: ${error.message}`);**

**resolve(error);**

**}**

**if (stderr) {**

**console.log(`stderr: ${stderr}`);**

**resolve(stderr);**

**}**

**resolve(stdout);**

**});**

The professor is able to launch a class or add new participants. Both operations are done through the update workspace route for simplicity. Thus, this route has to be able to handle all those respective operations. First, the workspace is looked for in the database. If it is found, the security checks run. After that the update operation is done. The difference lies in the fact that, when adding new users, we are adding elements to an array while for launching the class a Boolean is flipped. To chose which operation is done, we check the request body for a ‘members’ parameter. If none is found then a find by id and update method is used with the id and the value as the parameters. However, if the request body contains ‘**members**’, then the find by id and update method has the id parameters as well as a mongoose operation.

**{ $addToSet: {members: req.body.members} }**

The frontend was implemented as a way to visually test the functionality of the backend. Even so, it has lot of functionality of its own even though it constantly communicates with the API in order to know what the user should see and what he or she should not see. Displaying a button to a student that for example will activate a certain feature that only a professor should be able to control will not break the application as every route on the backend is protected so the request that is sent will not go through and will return an Error Response. Still, it would be a bad practice to have all this so the currently connected user data is stored in the local storage so the app can check what is the user’s role in order to display relevant information. The user’s authorization token is stored as a cookie and is used when any request is sent to the API. The header service is a angular class which has to be instantiated in any component that wants to communicate with the server as that is where the proper headers are defined. The service uses **ngx-cookie-service** in order to do operations with the browser cookies.

**private autorizationHeader = new HttpHeaders(**

**'Content-Type': 'application/json',**

**'Authorization': `Bearer ${this.cookieService.get('user')}`**

**});**

The application was structured in a way that relies on services to provide reusable methods and state handling. A service like the working-env service has the job of handling the code that is written in a certain file as well as deal with file changing and creation. It uses a behavior subject of type string in order to store the string containing the code. When the code in the file is changed, the **setCode()** method is called which calls **next()** on the code with the inputted value while for getting the code, the method that handles this will return the value of the code but as an observable so, in case the value of the code changes, the change will propagate all through the application.

For compiling and running the code the **onCompile()** method is used which formats the code as a JSON and then makes a put request to the server. The request will contain in its response the result of the compilation.

**this.http.put(`${environment.apiUrl}/working-env/${this.id}`, JSON.parse(`{"code": "${btoa(this.code)}"}`), { headers: this.headerService.getHeaders() }).pipe(map((result) => result['data']), tap(data => console.log(data))).subscribe(response => this.output = response);**

The **HttpClient** built-in feature is used to send the put request. This request receives as parameters the url composed of the base API url stored in the environmental variables, the specific path and the id of the working environment the user is currently connected to; the code itself parsed as JSON and encoded in base-64; the headers from the **headerService**. Then the result is piped for processing. Inside the **pipe()** method the operators from rxjs are used. First the result is mapped is mapped by only selecting the ‘data’ field from the response, then the data is printed to the console inside the **tap()** method which does not manipulate the data in any way. Finally, the **subscribe()** method is used to set the output variable equal to the data that was extracted from the response.

Another job of this component is to get the id of the working environment the user has connected to or, in case this is the first time someone connected to it, to create the working environment inside the database. The routes are already implemented on the backend so all there needs to be done is check when and if there should be a new working environment created.

**this.http.get(`${environment.apiUrl}/working-env/${this.linkedWorkspace}`, { headers: this.headerService.getHeaders() }).pipe(map((result) => result['data']), tap((data) => console.log(data))).subscribe(result => {**

**try {**

**this.id = result[0].\_id**

**} catch (error) {**

**this.http.post(`${environment.apiUrl}/working-env`, JSON.parse(`{"linkedWorkspace": "${this.linkedWorkspace}"}`), { headers: this.headerService.getHeaders() }).pipe(map((result) => result['data']), tap((data) => console.log(data))).subscribe();**

**location.reload();**

**}**

**});**

First a get http request is made towards the url containing the liked workspace, the id of which is in the route snapshot fragment as it is also the id of the class the user joined so we have a form of ownership here. Then the result is piped and the rxjs operators are used. In **map()**, the ‘data’ content is mapped. Inside the **tap()** method, first the data is logged to console and then **subscribe()** is used to get the data and be able to run operations on it. A try/catch block is used to determine if any id exists. If the id exists inside the result, then it will be assigned to the local id variable. However, if an error occurs, a new http request is made. This time is a post request and it sends the linked workspace as contents. This request creates the working environment in the database. Finally, the page is reloaded to clear the state and fill it with the correct variables once they are all on the database.

The code writing process is one of the most important parts of an IDE. So, a good text editor had to be used to provide users with a great experience. Monaco Editor was chosen for this reason, or more exactly, ngx-monaco-editor, which is a free open-source software that consists of a rewrite of the Monaco editor in typescript as an Angular component. The **MonacoEditorModule** is included in the working-env component and has the following settings:

**editorOptions = { theme: 'vs-dark', language: 'java' };**

And it is initialized in the **onInitEditor()** method by setting the local variable **editorRef** equal to the editor. The settings and the **ngModel** as well as the style of the component are assigned in the html:

**<ngx-monaco-editor [options]="editorOptions" [(ngModel)]="code" style="width: 100%; height: 92%;" (onInit)="onInitEditor($event)"></ngx-monaco-editor>**

Right now, the application only supports the Java language. However, more languages can be added in the future as Monaco has built-in support for the most popular languages and modules such as IntelliSense [22] for other languages can be easily found on GitHub and implemented in the application. Support for such languages can also be easily added to the backend by writing such middleware and adding them to the method that handles the code request.

# **Conclusion**

To sum it all up, the solution has great potential in streamlining the process of teaching. It is built in such a way that it is extremely modular and scalable. In the current state, the application is fully usable and all basic functionality is implemented. However, many quality-of-life improvements can be made and many more features can be added on top of the existing solution for a completer and more useful package. Some aspects of the application can also be further refined and new technologies such as soket.io can be added. For example, the teaching session can be implemented using sockets in order to improve real-time response and data accuracy. Implementing such technology will allow for new features such as real-time code streaming. The backend also needs more polishing as the compiler has a huge vulnerability. Right now, the code is received and compiled on the local machine that runs the server. So, malicious code can be executed and there will be nothing to stop it from affecting the local system. To fix this issue, a layer of virtualization should be added. Also, support for more programming languages can be added in the future. Moreover, more bug testing is needed especially in real use case scenarios.

Apart from the features mentioned above, there is a huge side of the application left untouched. That being the gathering and usage of data from the students. This data can be used to either show a comprehensive “picture” of the student’s or group’s progress or to predict how the students will do at the examination beforehand by using machine learning and predictive analytics. This could help in various ways from telling a student that unless he or she studies harder the chances of passing the exam are pretty low to even tell the likelihood of a student cheating on the exam (if the student had really low performance and really low chances of passing the exam and he ends up passing anyway with a high grade it could be because he or she cheated). This entire feature would require tapping a completely new filed and the development of an entirely different API that would handle all those tasks.

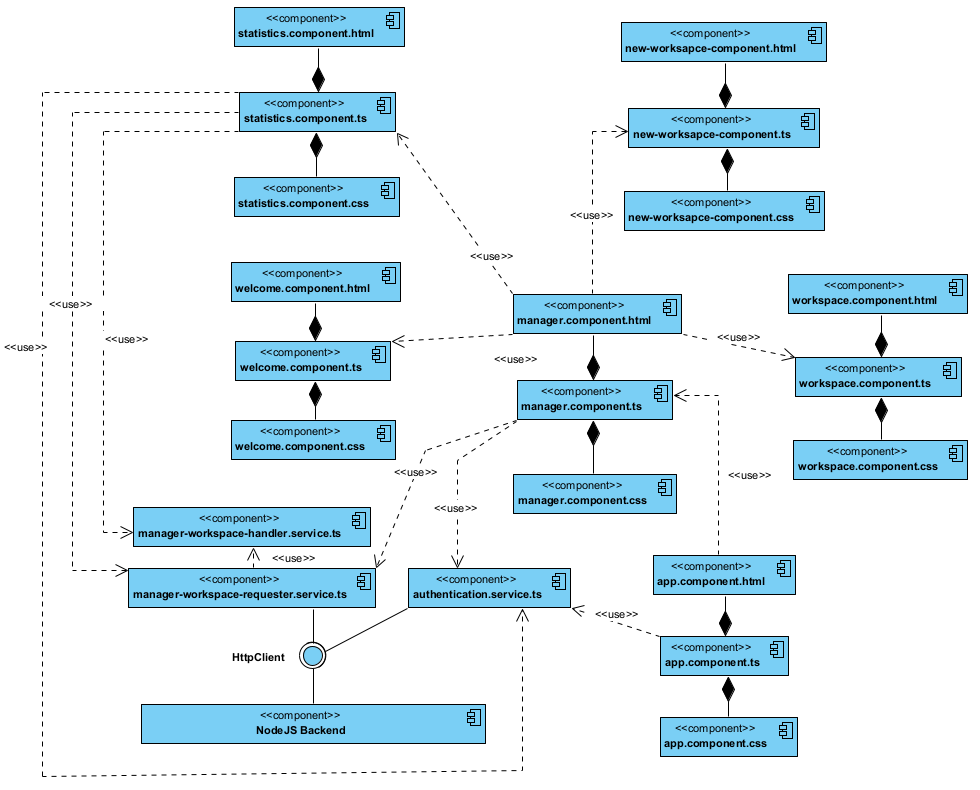
In its current state, the application provides an interesting way of connecting the student and the professor on a different level. There is no application designed in this way and with those functionalities even though something similar is definitely needed to boost the performance of the education system. Another strong point of the application is its scalability and modularity. The backend is especially designed in a way that is easy to understand and use by anyone with comments detailing what is written and how the methods should be used.

On the other hand, the solution lacks many features that any proper IDE should have such as debugging, the ability to save and manage multiple projects, autogenerated files modularity for the user, and many smaller quality of life features. The application is also not tested with a high number of concurrent users and does not work without an internet connection. Some features such as the management of study groups/classes are extremely rudimental and are only there to prove the functionality of other features that depend on them. Another drawback is the competition for this application. Even though there is no solution clearly aimed at academic use, all IDEs currently on the market are maintained by big companies and have entire teams with huge funding working on them. Hitherto, it is almost impossible for a similar application that just went into development to win over well-established ones unless it has extremely useful features and a great team behind it.

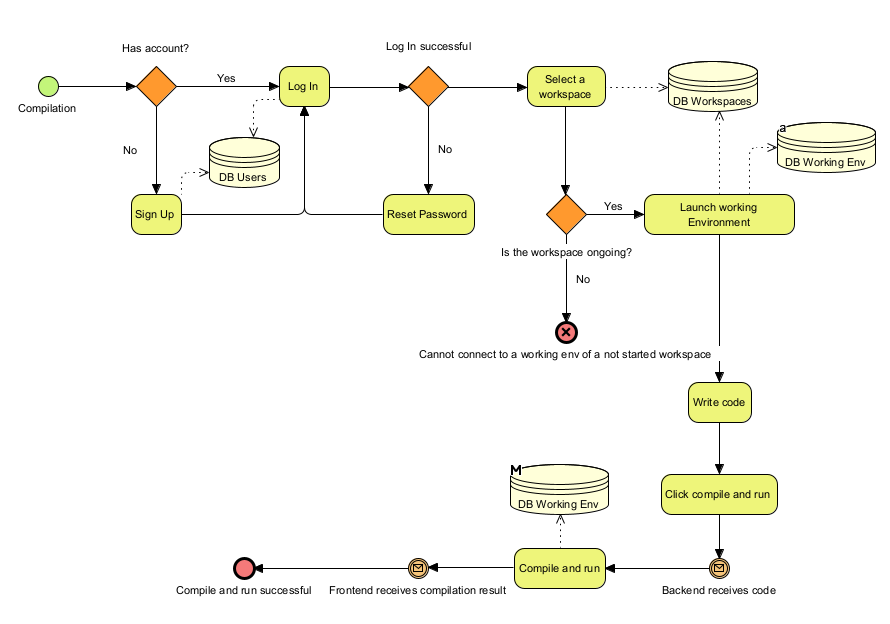
# **References**

|  |  |  |  |
| --- | --- | --- | --- |
| [1] |  |  | Microsoft Corporation, [Online]. Available: https://microsoft.github.io/monaco-editor/. |
| [2] |  |  | K. Terrell, "job-losses-during-covid," [Online]. Available: https://www.aarp.org/work/job-search/info-2020/job-losses-during-covid.html. |
| [3] |  |  | S. M. B. L. T. T. a. A. B. Charles Hodges, "The Difference Between Emergency Remote Teaching and Online Learning," [Online]. Available: https://er.educause.edu/articles/2020/3/the-difference-between-emergency-remote-teaching-and-online-learning. |
| [4] |  |  | Google LLC, "angular," [Online]. Available: https://github.com/angular/angular. |
| [5] |  |  | Microsoft Corporation, "typescriptlang," [Online]. Available: https://www.typescriptlang.org/. |
| [6] |  |  | J. T. Mark Otto, "bootstrap," [Online]. Available: https://github.com/twbs/bootstrap. |
| [7] |  |  | O. K. A. S. N. J. Ben Lesh, "rxjs," [Online]. Available: https://github.com/ReactiveX/rxjs. |
| [8] |  |  | O. Romanyuk, "freeCodeCamp," [Online]. Available: https://www.freecodecamp.org/news/angular-vs-react-what-to-choose-for-your-app-2/. |
| [9] |  |  | TechTarget, "techtarget," [Online]. Available: https://whatis.techtarget.com/definition/bootstrap. |
| [10] |  |  | MDN Web Docs, "developer.mozilla.org," [Online]. Available: https://developer.mozilla.org/en-US/docs/Learn/Getting\_started\_with\_the\_web/HTML\_basics. |
| [11] |  |  | Wikimedia Foundation, "CSS," [Online]. Available: https://en.wikipedia.org/wiki/CSS. |
| [12] |  |  | Microsoft Corporation, "visual studio docs," [Online]. Available: https://code.visualstudio.com/docs/editor/editingevolved. |
| [13] |  |  | D. W. A. I. g. Z. F. M. S. L. Z. Atul Kumar, "ngx-monaco-editor," [Online]. Available: https://github.com/atularen/ngx-monaco-editor. |
| [14] |  |  | R. Group, "rxjs.dev," [Online]. Available: https://rxjs.dev/guide/overview. |
| [15] |  |  | OpenJS Foundation, "nodejs," [Online]. Available: https://nodejs.org/en/. |
| [16] |  |  | Wikimedia Foundantion, "Representational state transfer," [Online]. Available: https://en.wikipedia.org/wiki/Representational\_state\_transfer. |
| [17] |  |  | W. Foundation, ""Node.js"," [Online]. Available: https://en.wikipedia.org/wiki/Node.js. |
| [18] |  |  | "express," [Online]. Available: https://expressjs.com/. |
| [19] |  |  | MongoDB Inc., Navionics, "MongoDB," [Online]. Available: https://www.mongodb.com/. |
| [20] |  |  | MongoDB Inc., Navionics, "MongoDB Compass," [Online]. Available: https://www.mongodb.com/products/compass. |
| [21] |  |  | MODX Docs, "mailtrap.io," [Online]. Available: https://docs.modx.com/3.x/en/building-sites/sending-mail/mailtrap. |

# **Appendix**



[Appendix 4.1] Component Diagram for the Dashboard component.



[Appendix 4.2] Usage flow diagram