# Collaborative IDE - Final Report


Group 28:

Alexandru Nica
Mihai Popa
Stefan Ciontea
James Grant
Adam Davies

Coordinator:

Anandha Gopalan


January 11, 2016

# Contents

# 1 Executive Summary

## 1.1 Overview

ColDE is a collaborative development environment that allows users to write and edit code in real-time, designed with the specific target of optimising the student-teacher interaction. It ensures that students have an enjoyable and efficient learning experience, while working on projects of various scale, from very simple ones for beginners, to fully functional projects.

## 1.2 The Problem

It is often difficult to efficiently organise a programming class. Even if students have to complete the same challenge, they always have different work speeds and some of them need more help than others for progressing. Teachers can easily be overwhelmed by multiple students needing assistance at the same time, while spotting mistakes in their code by leaning over their shoulder is not the most comfortable experience. Currently, a few online collaborative IDEs exist. However, none of them is designed specifically for teachers and students, most of them have a high pricing, and are focused on people collaborating as work colleagues. Therefore, programming classes would benefit from having a lightweight online platform, to replace their existing local environments.

## 1.3 Solution

Our cloud based platform, ColDE, allows teachers to set up classroom environments, where they can distribute exercises and tasks to students, monitor their progress, and even help them by adding helpful comments in their code. Students can work individually, on their own project, or together, contributing to the same code which can be edited concurrently. Being more than just an editor, our platform supports running code written in:

- Python; Python is one of the most popular choices to be learnt by students as their first programming language. ColDE supports working with multiple Python files, and comes with an extensive variety of libraries that can be used.
- JavaScript, CSS and HTML; Therefore, complete web pages can be created and visualised inside ColDE, as working on projects with graphical output is always more enjoyable for students.

# 2 Introduction

## 2.1 Background

TuringLab [1] is an organisation comprising of four Imperial College graduates who teach children to code using real world projects based on the national curriculum. They organise and administer two 2 hour laboratory sessions every Saturday for children aged 8-13.

Children are taught using visual programming languages, such as Scratch [2]. By doing this, they can treat segments of code as building blocks to create programs in a visual way that is easy to understand. They then progress onto Python, a much more widely used text-based programming language with real-world application, yet relatively simplistic in syntax and structure, as well as onto web development using JavaScript, CSS and HTML.

## 2.2 Motivation

What started out as a project meant to aid Turing Lab, during their sessions, turned out could have a more positive impact on people trying to learn how to code around the world. We came to this conclusion after considering the importance coding has come to have nowadays, which will increase further in the future as well. We thought that making a product which will provide support in the education of a new generation of programmers would be a great achievement.

As in any other engineering area, the tools you use while developing software are highly determinant for your efficiency and comfort. In addition to these two aspects, the general opinion of teachers is that the motivation of people to start learning programming highly depends on how comfortable they feel with writing their first tens of lines of code. We believe that by supplying a tool that will aid new programmers with their work, they will find the experience of programming to be far more enjoyable and rewarding. By lowering the initial difficulty, which dissuades many users from furthering their programming skills, the chance of a person continuing to do programming is significantly higher.

## 2.3 Objectives

We set out to tackle the problem of improving the teacher-student interaction by firstly considering the challenges that TuringLab were facing during their weekly sessions.

The product that we would provide had to cope well with the fact that the teachers would always be in a smaller number compared to the students and thus provide an efficient way for the teachers to divide their attention among the class, without neglecting anyone.

Another issue was that standard IDEs only support one person working at a time, locally. Our client needed the ability to work on a project collaboratively across multiple computers, as it would give a better sense of involvement for their students. Also, teachers should be able to directly contribute towards completing the project: the teacher-student interaction can be improved by offering the teachers the ability of getting directly involved in their students work without walking up to each computer in the classroom.

By attending TuringLab sessions, we also noticed the incredible enthusiasm of students for graphical output. This suggested us to consider providing enhanced support for it a top priority. At the same time, we realised the importance to have in our product an enjoyable and user friendly UI, which would attract students towards programming.

Given our decision to make something that would improve the learning experience of anyone who wishes to learn how to code, we have focused on the ease of use and scalability of the product, thus ending up with an web application. However, this decision generated a new objective for us: the ability to run code directly in the browser, without any additional effort from the user. It would be very unpractical to develop code inside the browser, and then download an archive containing the code to test locally. In addition, our platform's stability and performance become critical, as our IDE should be responsive, and provide safety for user's files.

In our meetings with TuringLab, we gathered a set of requirements. These would change depending on our progress and the arrival of new ideas or needed change in functionality. Initially, we received 5 requirements:

- Teachers should be able to create, manipulate and manage multiple student projects simultaneously.
- Teachers should be able to initialise, manipulate and change student's code and provide relevant information to the student in regards to these actions.
- The IDE should support the use of JavaScript or Python, preferably both.
- The IDE should provide the potential for a graphical output when running code.
- The IDE should provide a form of data analysis that allows teachers to monitor and evaluate the actions and progress of students.

## 2.4 Introducing ColDE

ColDE is the product we created to meet the aforementioned objectives. It is is a cloud-based development environment for users to collaboratively write and edit code, in real-time. The platform can be used as a complete IDE by programmers with any experience, but it is primarily designed for two types of target user: students who are relatively new to

programming, and their teachers. Teachers will be able to view, edit and comment on their students work, allowing them to aid them with their programming. Students will be able to edit the code collaboratively, and work seamlessly from any machine just by logging in on a webpage, with no local installs required. This includes running the code in the browser.

**Results**

- Web application that can be run on any web server and, once running, can be accessed by anyone wishing to use our product
- Teachers can create classrooms where they group people for a particular lesson or project
- Real time collaborative editing: any project can be accessed and edited by any number of users at the same time
- Inline commenting on code: teachers can provide help to students by adding comments across their code
- Chat system: participants to a project have a private chat, where they can ask for help or discuss ideas
- Code editor: syntax highlighting, relaxing color theme, advanced features such as multiline editing
- Running of Python code inside the browser, as well as displaying web pages written using JavaScript, HTML, and CSS

These were basic requirements that covered the basic functionality required for the project. After discussion, observation and realisation, the requirements were dynamically changed to meet the specific needs for the project in certain situations.

# 3 Design and Implementation

## 3.1 Technical overview

### 3.1.1 Back-end

To build our back-end, we decided to use Python 3, a language most of us had some experience with before. Python has a very intuitive and readable syntax, and it has increasingly grown as one of the most popular server side programming languages. Several libraries for web server development written in Python exist, such as Flask [3], Django [4] or Pyramid [5]. We decided to use the Flask 0.10.1 framework, which we considered to be the best choice in our case, due to its focus on simplicity, quick setup, improved support, and, most important, its integration with the websocket protocol, via the third party library Flask-Socketio.

In terms of persistently storing data, we used the database abstraction layer SQLAlchemy [6], which Flask supports, facilitating our information manipulation. Flask together with its SQLAlchemy adheres to the MVC (model-view-controller) architecture, which we tried to follow in our back-end implementation. This way, models (such as *User*, *Project*, *Pad*, etc.) are declared in models.py, while views (actually named templates) are under the *templates* folder and rendered by Flask in *views.py*.

### 3.1.2 Front-end

Our front-end was developed using HTML, CSS and JavaScript. We made extensive use of JavaScript to enable functionality in our webpage, by running program code in the browser: not only for the visible UI elements (in order to improve user experience), but also for preparing and interpreting data during the communication with the server. The JQuery [7] JavaScript library was also used in certain places mostly to simplify UI implementation.

The design of our page makes extensive use of Bootstrap elements. Bootstrap is a comprehensive, very popular framework which provides ready to use web elements. The ones we used were very customisable and easy to integrate in the context of our application.

### 3.1.3 Client-server and client-client communication

In our collaborative environment, each change produced by one user needs to be sent to the server, and then propagated to a number of other participants. Therefore, small quantities of data are very often transferred between clients and the server. We concluded that having a reliable and fast way to do the transmissions is critical and decided to use the WebSocket protocol for the communication between clients and the server. We actually made use of its socket.io wrapper API, which was available as a JavaScript implementation, as well as in

the Flask server. Using the WebSocket protocol, a full duplex TCP communication channel is established between the server and the client, which remains open until one of the two participants disconnects. Both client and server can push data through the channel at any time, the connection is reliable and has a low latency. On both sides, the socket.io API is generating events when data is received, or the connection was established.
As mentioned before, when an user produces code changes, they are firstly sent to the server, which then broadcasts them to the appropriate other clients. Note that any form of communication between clients is done via the server, as they never transfer data directly. We preferred this system architecture, due to a number of advantages:

- existence of a global version of files in server, by directly applying changes received from clients. We do not have to query clients for their current file version in order to update the database, and the risk of loosing updates is minimised
- determining what connections have to be established between clients can be difficult to handle
- in peer to peer connections it is difficult to ensure that all clients receive updates in the same order (aspect which was important in our case)
- a peer to peer based architecture is more sensible to high differences in quality of connections between pairs of clients

The model we used is visualised in Fig. 1.

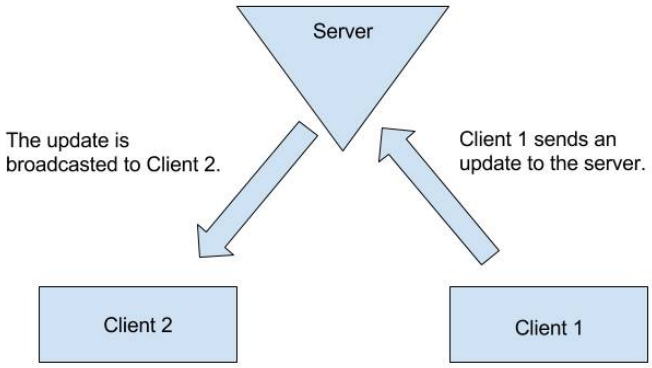


Figure 1: Client Server communication

## 3.2 Technical aspects

### 3.2.1 User Accounts

To be able to provide user specific content in our environment, and the functionality we wanted for our project we had to ensure that every user had an account. See Fig. 1 for a picture of the login page included in our project. Once logged in, the main page shows the projects the user can access, which can be owned by him or by a different user who shared them with him. An example of this can also be seen below. In the future we could add the option to use our product in guest mode, but for now we decided that every user should make use of an account.

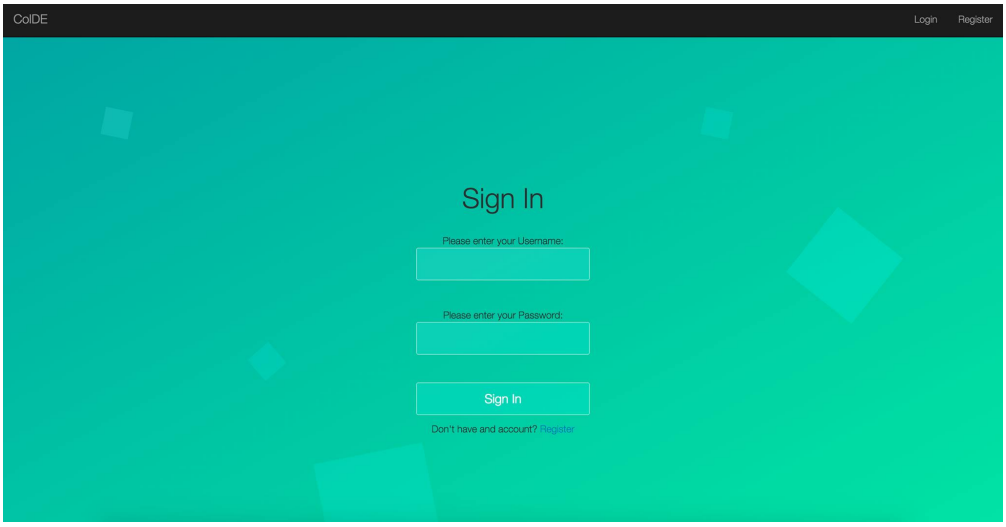


Figure 2: Login Page

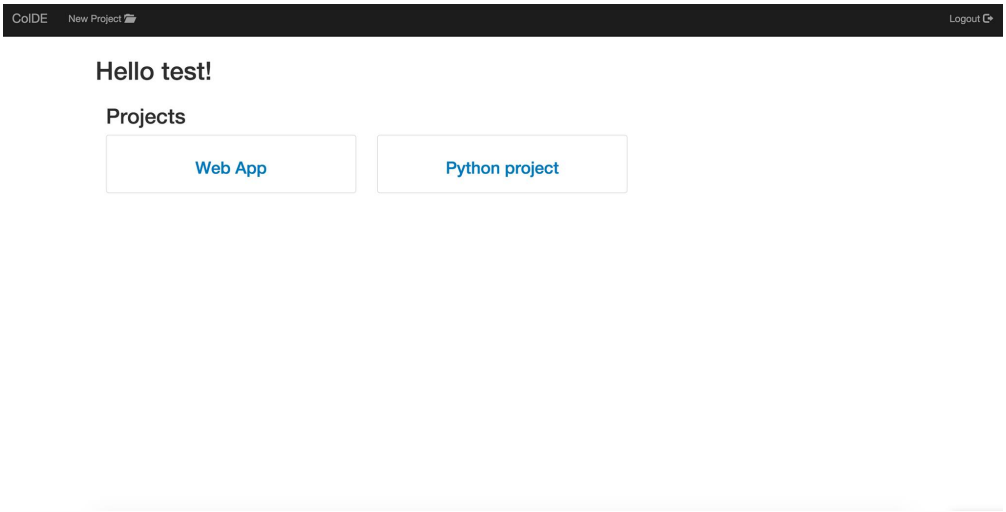


Figure 3: Home Page

### 3.2.2 The editor

During the first stage of the project, one of the important aspects we had to address was the integration of a code editor.
When we started, we were suggested to use Etherpad [8]. Etherpad is a highly customizable open source online editor which also allows collaborative editing in real-time. However, it was not really perfect for our needs, since extending it with a number of features we wanted proved to be overcomplicated. Once we started modifying the core code behind Etherpad to fit our needs, we realised that it would consume a lot of our time and would not necessarily provide the expected results. Although it is a high quality project, we found Etherpad's codebase difficult to follow, as result of many code dependencies. This, coupled with Javascript being a language we did not have lot of experience with, determined us to take a different approach. We decided that we would implement the code that deals with the synchronisation and the file management on the front-end and back-end ourselves, and we would try to find a performant solution for what we needed besides that.

We decided to use CodeMirror [9], a versatile text editor implemented in JavaScript, which can be included in the browser. It is specialized for editing code, and comes with a number of supported languages, as well as add-ons that implement more advanced editing functionality. After some research, we realised how widely spread CodeMirror is (it is used in Chrome DevTools [10], the Mozilla Firefox console [11], and many more), which made it clear that we should try it as well. We felt our choice was good, as it was easy to integrate with our project, its documentation was comprehensive, and its style was significantly better compared to the editor included in Etherpad.

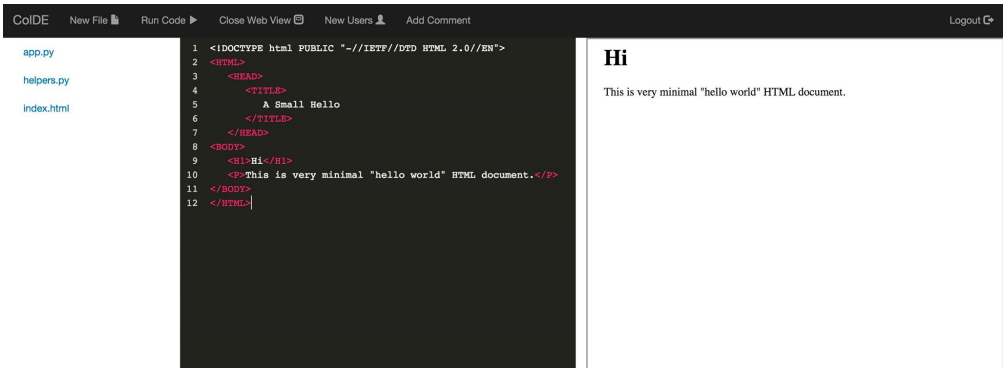


Figure 4: Editor

### 3.2.3 Designing the database

We have the following data models in our app:

- *User*. The *User* model contains attributes for user's username, and the user's password (stored after a hashing algorithm is applied). *User*s are also associated with *Project*s and *Classroom*s in two seperate many-to-many relationships i.e. users have multiple projects and projects have mutliple users. The same is true for users and classrooms.
- *Classroom*. A classroom is simply a collection of users and projects. A classroom is created and managed by an user. When an user creates a classroom, they gain a teacher role in that classroom. A teacher role has a higher level of permissions in the classroom allowing that user to manage the class's users and projects. *Classroom*s have a title attribute, a many-to-many relationship with *User* (as described above) and a one-to-many relationship with *Project*, i.e. there are multiple projects in a classroom but a project cannot be in multiple classrooms.
- *Project*. A project is the place the magic of ColDE really takes place, being the place where the text editor appears. It contains an one-to-many relationship with *Pad*.
- *Pad*. Pads are simply stores of code data. Every file in ColDE is stored as a pad in the database. It also has a filename which may or may not have an extension at the end. If it does, ColDE automatically recongises the filetype and applys the correct syntax highlighting for the file.
- *Comment*. Comments are strings of text associated with a position in a pad. Users can click on them in the client and they can read the comments. We took inspiration from Google Docs'[12] comments.

### 3.2.4 The filesystem

We aimed to achieve a tree file structure for the user to handle projects with multiple (and maybe nested) folders. Thinking of a way to store the several files of a project on the server, we came to the conclusion that it is best to have filenames as complete absolute paths from project root to the actual file. This was a straightforward solution to simulate a file system, considering the actual meaning of the paths only in the client where the tree structure is displayed.

For example, the string *static/js/app.js* would be structured on the client as two nested folders, and the *app.js* file inside the *js* folder. To ease our work of coming up with an appealing way of displaying the filesystem in our frontend, we integrated the JsTree open source [13] library.

### 3.2.5 Synchronising the code

The implementation of a reliable system of online collaboration between users was considered a priority in our project. Obviously, user changes (e.g. typing, deleting, pasting, etc.) have to be reflected with zero delay in the code editor: typing should never be blocked because of waiting to send or receive data. On the other hand, more than one user can edit the same code concurrently, so user changes often need to be merged together using different approaches. In any case, we need to ensure that, eventually, all users working on a file will reach one common version.
Note that, if the connection between clients and server was ideal (zero delay), changes from one client could be directly applied to another client without any conflict. However, since this is not the case, the delay of the transmission has to be taken into account.

When a user performs a local change, a *changeset* is created to represent it in a specific format. Note that these changesets transmitted on the route *client - server - other client* are relative to the already existing file content. When thinking of their design, our objectives were:

- compactness of the model, in order to minimise the amount of data transmitted between clients, and decrease the workload of the server
- allow elegant operations related to changesets: application, composition, merging. See the next section for more details.

In order to meet these objectives, our changeset representation has the following content:

- the length of the file the change is applied on
- the length of the file after the change will be applied

- a "char bank", string containing characters that will be inserted at differerent points throughout the existing file content
- a list of operations, in the order they are applied. An operation can be:
  - $=x$, where x is a natural number: leave the next x characters from the initial text unchanged
  - $-x$: remove the next x characters from the initial text
  - $+x$: insert x characters from the charbank in the file at the current point

For instance, applying the operations [-1, =2, +1, =1] with the charbank "x" on the initial text "abcd" will have the result "bcxd".

***Client state***
At any moment in time, a client editor pad maintains its state in the form of three changesets:

- csA is the latest composition of changeset applications, as received from the server (either changesets generated by other clients, or changesets generated by the current client for which acknowledgement was received from the server)
- csX is the composition of all changesets the current client has submitted to the server, but has not yet received acknowledgement from the server
- csY is the composition of all changesets the current client has produced locally, but has not yet submitted to server

During execution, each client has to be involved in the following events:

- local typing by user: a new changeset is created and composed with csY
- once every 500 ms, try to submit csY to server. This can happen only if csX is the identity changeset (no changeset has been submitted but not acknowledged)
- receive acknowledgement from server for the submitted csX. In this case, compose csA with csX and set csX to identity
- receive a changeset newCs generated by another client: update csA, csX and csY to reflect the change. Note that newCs has to be first transformed to be relative to the content displayed in the editor, content the server does not know about if csY is not identity

***Server state***
The server keeps track of the current global content of the editor, as well as a list of revisions received from clients.

On the connection of a new client, he is replied with the global state of the editor.

When the server receives a new changeset from a client, it may be the case that the sender client did not know about a number of revisions the server has in the list. In this case, the received changeset is adapted to be relative to the last server revision, by making it step by step relative to revisions more recent than the one the client change is based on.

We briefly presented how code synchronising is done for one editor. Obviously, in our implementation we have to manage multiple distinct contents. However, the ideas remain the same, and they are the foundation of a robust data synchronization algorithm which meets our needs.

### 3.2.6 Inline comments

Inline comments are another feature we considered essential to the purpose of our application. Bookmarks can be set in CodeMirror at a position in an editor, and its local movement is then handled automatically. This was very helpful, but we still needed to synchronize them across multiple clients, so we realised that the very same possible synchronization problems enumerated in the previous section also applied here. Therefore, we aimed to find a possibility to extend the already implemented algorithm to support comments as well.

The approach we used is a probabilistic one. Periodically, the server generates a new prefix and a new suffix of variable length. Both are broadcasted to all clients, and, in order to add a new comment to the current editor, the client has to include the comment hash between the prefix and the suffix, and insert the string obtained this way inside the code. For example, if the current prefix and suffix are "a?!", respectively ".<>", and the generated hash for the new comment is "abxa", the client performs an instertion for the string "a?!abxa.<>".

Although we realise this solution is not perfect, the proability to have collisions is very low, so in practice it should not be a problem. We decided to use this approach in order to keep changesets manipulation code clear and focused on characters only, as extra complexity would have been added otherwise.

### 3.2.7 The chat and notification system

Given the fact that the purpose of our project is improving the communication between students and teacher, we realised that having a chat system is a must. By using it, users and teachers can efficiently communicate privately with each other, or within the group discussion, and can request the teacher's assistance.

The chat system is built using socket.io and jQuery. When a user begins to type in the chatbox, it sends a websocket message to the other connected clients which then updates on their end with a message, "X is typing...". When the user hits Enter or clicks the send key,

the message is sent to the other clients immediately. We wanted these two features to add to the real time element of our application, as communication between users should be quick and easy.

### 3.2.8 Running the code

One of the challenging parts of the development environment was adding support for running code in the editor. We had to decide between running the code on the client side or on the server side. There are benefits and drawbacks for both approaches, but since we were focused on supporting Python and web development, with graphical output being a priority, we opted to run the code on the client side. Whereas JavaScript/HTML/CSS code is run in the browser directly, we had to consider how to do the same thing for Python code. There were two main competitors in terms of the technologies to use to this purpose: Skulpt [14] and Pypy.js [15]. Even though both of them enable running Python in the browser, they take different approaches:

- Pypy.js is a very large library. It is a 12MB JavaScript file that contains an entire virtual machine, and hence this gets the closest to Python implementation completeness. It is also the faster solution of the two, but it is slower to load(since it is so big) and also the first time a user runes a code there is a loading time of around one minute.
- Skulpt on the other hand, has evolved into a teaching tool over time. It compiles the Python code into a state machine that emulates the CPython interpreter very closely, being a handwritten implementation of it in Javascript. It is the only one that supports asynchronous execution, for instance $while(true) : printhi$ without locking up the browser. It also has a lot of out of the box features that are helpful to us, for instance support for the turtle library.

We decided that in our case Skulpt would fit the requirements better, due to its asynchronous execution and graphical output support. Since we design an IDE for young students who are just learning to code, we considered this choice to have a better impact.

### 3.2.9 Importing files

Since the files in our cloud are reduced to strings in the database, importing files has become a more difficult task than initially anticipated. We could not use the built in imports of the functions we created, so instead, we added a preprocess function that does the imports in place in the current file before it is run.

For Javascript and CSS it does a search through the file to find any src tags. If any are found, the preprocess function then looks through the files available in the project. If there

is a file found with the same name as what comes after src =, then that file is copied into the file that is being run with the corresponding tags, for instance a Javascript file is put between <script>tags in the HTML file.

For python imports, the same preprocess function will look for imports, and for every file that is imported, if a file with the corresponding name is found, its contents are enclosed within a class and instantiated in the current file. A list of the files already imported is populated as this goes on, so we wouldnt import the same file multiple times. The preprocess function is then recursively called on each imported file, so its imports will also be imported. This, however, brought up a new problem. Since the files that are being run are different from the files visible in the editor, error messages needed to be tweaked such that the line numbers match the actual lines in the visible editor. This was done by remembering how many lines were added when running a file, and then subtracting that amount from the error message line number.

### 3.2.10 Testing

Testing the Javascript code on the front end, we chose QUnit [16], since it is a powerful, easy-to-use JavaScript unit testing framework. It is capable of testing any generic JavaScript code, and it is very easy to include in a project and use. For the tests themselves it uses assertions, so for each javascript method that requires testing, you call it with some arguments and assert what the result should be. Running the tests is also very easy, you just go to a local webpage, which in turn will run all the tests and display which tests failed and why. We used this feature extensively within our development process: A feature was not considered complete until tests for that feature were implemented as well.

### 3.2.11 Deployment

At first we tried to deploy our app on to Heroku [17]. Heroku is a cloud Platform-as-a-Service which attempts to handle all of the building and maintaining of infrastructure, typically associated with developing and launching an app. However, when we tried to deploy to Heroku, multiple issues arose. The app itself would work fine, but the websockets refused to go beyond the handshake part of the protocol. Without having ease of access to Heroku's backend we decided to switch to our own deployment solution hosted on Digital Ocean [18]. Luckily for us Docker [19] (and Dokku) exist. Docker offers an additial level of abstraction of operating system-level virtualization on Linux. Dokku is a project which esstinally allows you to own your own personal Heroku built on top of Docker. We installed Dokku on to our Digital Ocean host and suddenly we were able to push our application (via git) to our server. After setting up the postgres database (which dokku also made easy with its built in postgres plugin) the application worked out of the box.

# 4 Project Management

## 4.1 Planning

This project is one of the largest group works that we have done and found that we needed to approach the project in a way that made efficient use of our time and efforts. We did not want there to be a situation where members would not know who was doing what and what needed to be done. We needed to use a software engineering development method.

### 4.1.1 Choosing our development method

We started looking into agile development methods and first looked at Extreme Programming, but we considered some of its aspects to be too rigid for our project requirements. Even if iterative planning could have been applied to our development progress, we wanted to keep the freedom of adding and assigning tasks dynamically, while we are making progress. We found having a fixed development plan every few weeks too difficult to predict, and we thought that this would rather apply to projects within larger codebases, than to our open ended project started from scratch. Also, we thought pair programming would be worth opting to only when we feel the need to be supported while writing sensitive code segments, rather than enforcing it as a general rule. We are going to follow the small and often releases advice of Extreme Programming by merging feature branches into master as soon as they are ready and delivering prototypes to TuringLab often, but this rule is also encouraged by Kanban.

Then we researched Scrum, but in this case we felt that the existence of a product owner and scrum master would restrict the communication between some members of the group and TuringLab, and would cause an overall inefficiency in the project. We also decided that since Scrum is merely a project management methodology, and not specifically a software development methodology, we could fall into the trap of performing Scrum well, but producing a product that fails to meet expectations. We wanted the model we choose to be more focused on the task at hand, while minimising the chances of actually impeding us because of our unfamiliarity with it.

We then looked into Kanban and realised that this was the development method best suited for us and our project. We decided to use this method for a number of reasons:

- With our variety of skills and experience we knew that flexiblity was key to the success of the project, allowing people to pick and choose what parts of the project they want to do.
- We have a very 'close to home' customer who we can meet with regularly. With these opportunities, we find it best to try and minimise the cycle time, the time from when

we start implementing a feature to the time we finish that feature. The minimising of cycle time feature of Kanban synegises with our need to minimise cycle time so we can show features to our customer as they are produced. This frequent feedback would allow us to maintain the quality and purpose of the project.

- With our frequent meetings with our customer, we found the idea of efficiency through focus a good way to go, such that we can complete a small number of features quickly and frequently so we could gather consistent feedback from our customer. By working on a smaller number of features we could minimise inefficieny that comes with multi-tasking, as there are fewer things to be done at one time. As our customer was just around the corner, we could recieve quick and easy confirmation if any issues come up.
- With the use of a variety of tools, we can statistically analyse metrics, such as the cycle time, to continually improve the efficiency of our work. By visualising these metrics, we can easily spot and bottlenecks in the progress of the work and adapt our process of developing a feature to account for these.

Because of this, we chose to use the Kanban development method. To do this, we found a number of tools and applications to help us.

### 4.1.2 Tools used

**Trello [20]** As all our group members were only able to meet a few times a week, we needed a way to let everyone find out what the others were doing at a glance. Trello is a simple yet powerful tool which allowed us to do this. It is a project management web application using the Kanban paradigm consisting of boards with lists of cards representing the different tasks in a project. We used this to categorise tasks into a backlog, the next upcoming tasks, tasks currently being worked on and finished tasks. The tool also allowed us to assign each task to a person or a group of people.

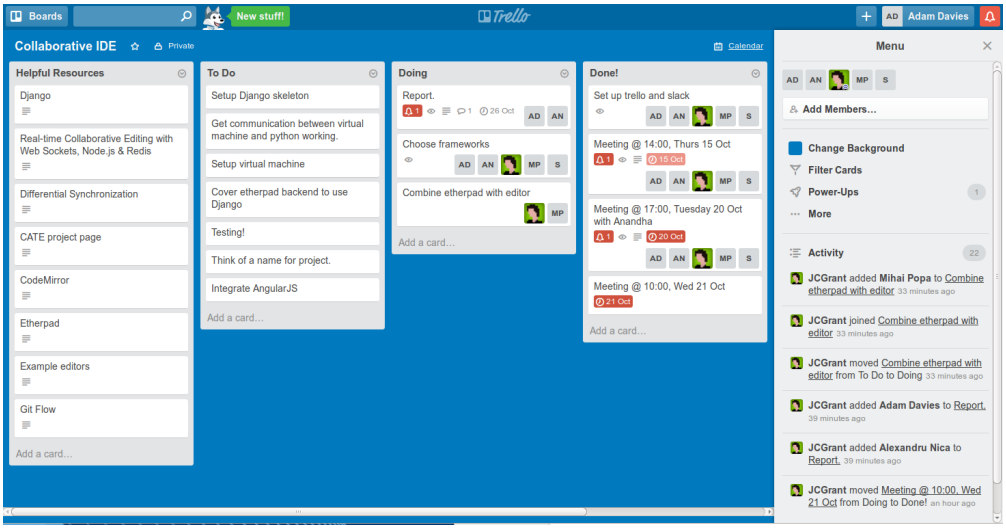


Figure 5: Trello Board

**Slack** [**21**] To further improve communication between team members, we set up a Slack team and integrated it with our Trello board and our Gitlab repository. Whenever someone makes a change to the trello or the codebase on gitlab we get a notification through slack informing everyone of the change. We also use Slack to store all of our Google Docs and other files of interest. We also used Skype [22] over the holidays, as we were unable to meet in person, which proved to be next best thing.


Figure 6: Skype calls

**GitLab [23]** For version control we decided to use GitLab, since all the members of our team have a good understanding of it. Version Control is very important when working in a group, allowing everybody to work in theyre own way, from the places and at the times that each individual is most comfortable with.

## 4.2 Group Organisation

We quickly discovered that the skills and experience each team member has is wide and varied. Trello allowed us to split the different aspects of the project into their own sections, in order to assign the work appropriately to the people most experienced in the respective area, and even allowing members to pick for themselves what task they want to do next.

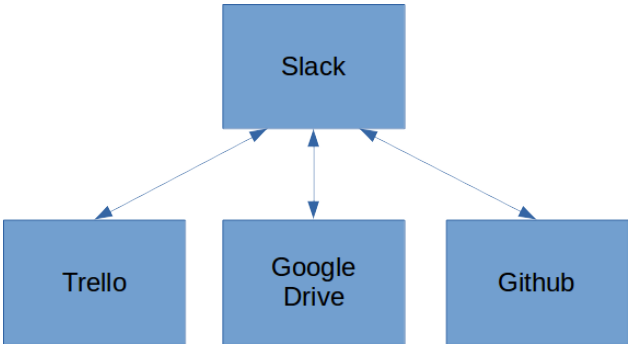


Figure 7: Project Tools

Our coding meetings were taking place a few times a week and were mostly about getting the technical side of our product ready, but we made sure to have regular weekly meetings which were focused on improving our work-flow, setting priorities and also discussing the steps that we would take until the next meeting. During the meetings, one of us, usually taking turns, would take notes on a Google document, which would later be attached to the slack documents archive. These notes ended up being very useful, helping us keep track of ideas and decisions. The ideas in these documents eventually got their own Trello column, so that we could start implementing them. All these steps made our process quite efficient after a few weeks, so we were releasing new code to our master branch on a weekly basis, sometimes even faster.

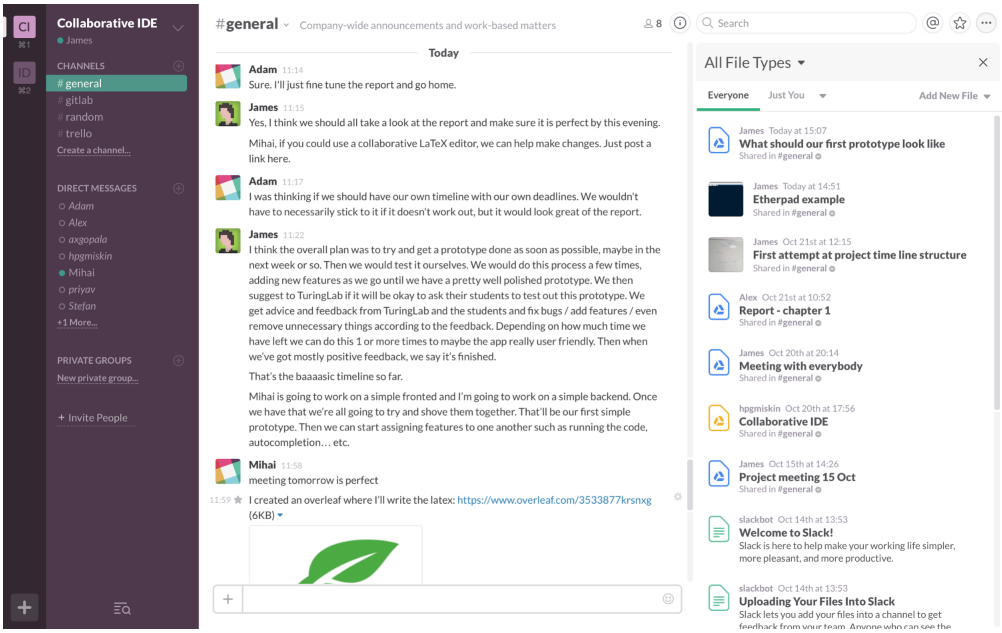


Figure 8: Slack

## 4.3 Task Allocation

This system, with people choosing the tasks that they wanted, even though it seems unorganised at first, ended up working very well. All the members constantly had something to do. We thought, at first, that there will be some more desirable tasks that everybody would want to take up, and other tasks, that will just be left until we reached a point when there was nothing else left to do. Surprisingly, though, this was not the case: The diversity of the background of the team members ended up ensuring that everybody had different interests and people were not fighting over tasks. Here is a brief description of the main tasks that each member of the team took up:

Mihai focused a lot on the backend implementation for this project. Given his knowledge and experience with algorithms, he was the perfect person to implement the synchronisation algorithm for all the pads, and the custom changeset designed specifically for this specific case of synchronisation. He also implemented the floating comments feature, since that required altering the synchronisation algorithm that he wrote.

Alexandru worked mostly on running the code and the afferent issues that came with this: displaying results in the web-page instead of the actual console of the browser, displaying the web-page being developed in the web-view when it is opened. He also implemented the preprocess function that takes care of the imports, and wrote most of the tests in the front-end test suite.

Stefan, having some experience in front end development, took up the challenge of making the interface user friendly, and implementing neat features, such as the ability to open and close the web-view, the front end of the file system, with a file tree structure and contextual menus. He also designed the log-in page, the popups for creating new projects and files.

James was the one with the most web-app development experience, so he was the one who implemented the initial set up of the project, implemented the database, took care of the deployment, and kept us on track in terms of the Agile development guidelines and the tools that we used (Slack, Trello, etc.). He was also the one who implemented most of the helper scripts that became very useful through the development process.

Adam spent most of his time on documentation, analysis and maintaining project quality. He made a variety of contributions, such as implementing the initial structure and design of the log-in page.

# 5 Evaluation

## 5.1 Quality

We maintained the quality of our project by making use of an automated testing suite to test the functionality of individual features in the program. When additional features are added to the project, a set of appropriate unit tests are created with them. Before code is committed to the git branch, all unit tests are run by the commiter to ensure that the changes did not break any previously working functionality. This is the standard procedure to confirm that a feature is correctly implemented independently of the rest of the program.

We took the approach of: the passing of unit tests doesn't confirm correct implementation, the tests are used to identify easy-to-fix problems if the unit tests fail. Even if some tests have passed, the respective feature may not necessarily be correctly implemented as a fragile feature may unexpectedly fail when a new, completely independent, feature is changed or implemented.

## 5.2 GitLab usage

We are assuring the quality of our project through the use of five different types of branches: Master, Development, Feature, Release and Hotfix.

- The master branch is the face of our project. It holds the most recent, fully working version of the project, as well as the history of these versions. This is the only version that we will make available to our customer and no changes can be pushed directly to the master branch, any changes are implemented by merging the master branch with another branch, the release branch.
- The development branch is the backbone of our project. This is the active branch that runs parallel to the master branch. The development branch is used as a common-ground between different features within the project and is where members merge together the work they have been doing. The development branch will always have what is on the master branch, but can potentially contain some unreleased features.
- Feature branches are the only kind of branch where there can be multiple of them in use at the same time. A feature branch is created by a member when they take up an individual task from the Trello board and is made by branching off development. The purpose of this new branch is to implement the individual feature specified in the relative task and to create any unit tests required to certify the functionality of this feature. Once the feature has been implemented and the unit tests for the feature pass, then it can be merged into the development branch. There is never any merging a feature with the master branch.

- The release branch is used when we want to create a new version in the master branch with new features that have been added to the development branch. The release branch is created from branching off development and then all unit tests from every feature are run. We then observe the results and resolve any conflicts between features by implementing a series of bug fixes.

  When it has been confirmed that the release branch is correct, then it is merged to both master and development as the most recent version. This is done separately to the development branch so no additional features are added to the release when it is resolving conflicts, although new feature can still be added to the development branch and will be merged with the release when it has finished.

- The hotfix branch is for emergency use if the version on the master branch is not correctly implemented, as an error may manage to slip past the checks done in the release branch. A hotfix is branched directly off the master branch and implements any necessary action that will get the version on the master branch working again. When done, it merges to both the master branch and the development branch.

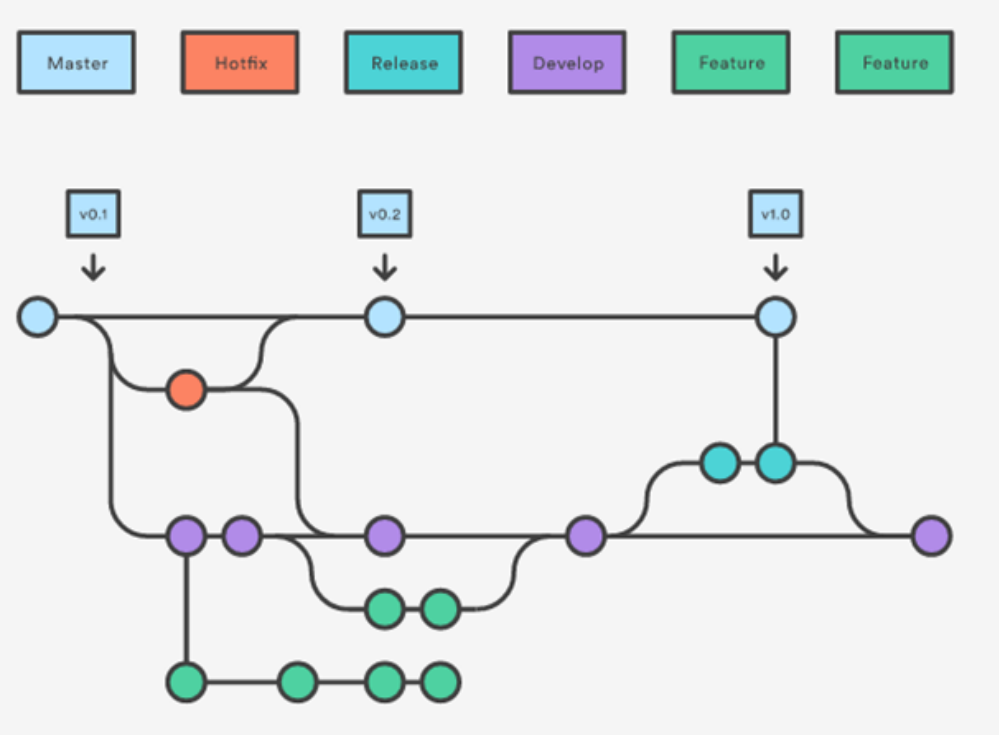


Figure 9: GitLab workflow [24]

## 5.3 Deployment pipeline

We use automated unit testing whenever we push a change to a branch and take special note of changes in the results of the automated tests between different versions. This way, we can easily identify if the change we made caused any tests that previously failed to now pass and if the change caused a previously passing test to now fail.

We make use of continuous integration by making the tasks on Trello, and consequently the feature branches, small in size and quick to implement. This way, in a full working day, an individual member may be able to create a handful of different features which will be consistently merged with the development branch. This keeps each member relatively up-to-date with what each other member is doing allowing easier communication and understanding within the agile environment.

Our use of a deployment pipeline allows us to have confidence in presenting our project to our customer, as individual features would be tested multiple times in different environments: tested individually to ensure the features functionality and tested with the rest of the project to ensure its compatibility with the rest of the project.

We evaluate our working practices whenever we notice something being done inefficiently or there is some miscommunication between group members. An example of this would be when it was noticed that some tasks on Trello were too ambiguous, so we redefined how we would place tasks by splitting the To Do column into To Do and an Ideas column. Some generic tasks, such as create backend would be either placed in the Ideas column or split into more specific tasks.

## 5.4 Feedback

We hold regular meetings with our customer, TuringLab, where we demonstrate and provide them access to our most up-to-date version of the project on the master branch. We then receive feedback and comments in regard to new features and overall project ideas. Taking notes in the meetings, we then update our Trello to-do board, splitting up the task of analysis across multiple members.

Another important piece of feedback that was really helpful for us were the Software Engineering face to face consulatations. We held two of those meeting, both having a great impact on our organisation and workflow. They helped us focus on the important tasks, rather then starting a lot of additional features and then not knowing how to proceed.

We have two different kinds of users, teachers and students. The teachers provide feedback to us in the meetings with the aim of producing a project that will make teaching easier and more effective. The feedback we get from the teachers has the advantage of being direct and precise, however the teachers don't necessarily know how to achieve their goals

and can only give us an approximate indication of the solution. We receive feedback from the students indirectly from analysis of the result of using our IDE rather than the previous implementation. Testing involving the students has the advantage of the quantity of different perspectives on the project which may reveal any previously unidentified bugs in the program.

We make sure that we are implementing the most important requirements of the project and building the right features by talking to our customer on a regular basis, even outside of our fortnightly meetings, and getting to understand any immediate concerns with the progress of certain features in the project.

For example, the IDE was initially described as for use with Python and JavaScript but, after talking with TuringLab more and attending one of their teaching sessions, we found that the students mostly use Scratch and Python. We also found that the students mostly favour graphically-based output rather than console-based output. From this, we came to the conclusion that we should prioritise Python and graphical-based output when it comes to choosing what to implement next, as it would be better to introduce the students to our IDE with familiar functionality to what they have already been using.

## 5.5 Comparing Requirements with Implementation

We can measure the success of our project my comparing our initial requirements that were provided by TuringLab with what we have produced. Recall the initial requirements:

- Web application that can be run on any web server and, once running, can be accessed by anyone wishing to use our product.

  ColDE has been successfully been deployed and tested while running on DigitalOcean. We were able to log in to it and use all its features, even from different parts of the world (while on holiday), so we consider this requirement as met.

- Teachers can create classrooms where they group people for a particular lesson or project.

  We implemented a home page where individual projects can be created and managed, as seen in section 3.2.1. A set of these projects can be made to simulate a classroom, a group of people working on a similar and initially identical project that each individually vary depending on each student's work.

- Real time collaborative editing: any project can be accessed and edited by any number of users at the same time

  We have created a collaborative editor, as detailed in section 3.2.2 , which performs the function of allowing multiple users to access and edit the same project at the same time.

- Inline commenting on code: teachers can provide help to students by adding comments across their code

  Inline comments were implemented, as detailed in section 3.2.6 , such that teachers can create messages specific to an individual section os a student's project.

- Chat system: participants to a project have a private chat, where they can ask for help or discuss ideas

  A chat and notification system was implemented, as detailed in section 3.2.7 .

- Code editor: syntax highlighting, relaxing color theme, advanced features such as multiline editing

  We implemented these user-friendly features, as detailed in section 3.2.2 , and a few other similar utility features, such as auto-completion.

- Running of Python code inside the browser, as well as displaying web pages written using JavaScript, HTML, and CSS

  The editor allows for running Python within the brower, as detailed in secion 3.2.2 .

From the comparison between our initial objectives and what was achieved, we could measure the project as a success.

# 6 Conclusion

## 6.1 Team Impressions

With only a few of us having previous web development experience, this project turned out to be a great achievement for us and it enabled us to learn a lot in the web development area. We gained a lot of knowledge about the MVC structure; server side development using Python and Flask; front end development and design using HTML, CSS, JavaScript; and realtime server-client communication using Websockets. Since nobody had any prior experience with collaborative editing, or running python code in the browser, we were not sure that we will be able to finish this project at all. Fortunately, with the help of the weekly meetings with TuringLab and our supervisor, we were pointed in the right direction and were able to get a lot of research done in the early stages, which helped us overcome our lack of experience. We tried to distribute the tasks such that everybody would have some experience in what they are doing, focus on a specific range of features, and also learn something new.

This project turned out to be a very open-ended one. Using agile practices and development techniques, we were able to implement our task in a timely manner. As a group, we learnt how to manage a project and work together, even though we all had different timetables and were not able to meet with the full group in person very often. Using the techniques presented in the Software Engineering class, we were able to agree on practices that proved to be useful through the entire length of the project. We are extremely pleased to have managed to deliver a quality product on time, and we believe that this is a really good demonstration of how great teaching programming can be made in the near future.

## 6.2 Personal Impressions

For the purpose of this conclusion, we decided that each member of our team would write a short impression about this project:

**Stefan** What I found most interesting about this project was using different open source projects to create something new, and most importantly something that has potential to grow. I also found that the tools we used to make sure we were on the right track were quite effective and will most definitely use them for further projects.

**Alexandru** I feel like this project was a great opportunity for me, allowing me to learn about web development in a hands-on environment, with some great team-mates to help me along the way. Besides web development in general, I also understood how important the use of proper programming techniques, both in actual code and in the team management.

With this said though, I also believe that the workload could have been better distributed between all the team members.

**Mihai** I can safely say that this has been one of the most challenging projects I have ever been part of. The scale of it is much bigger than what I was used to before, and this taught me how essential development techniques are, and the fact that communication is the most important element within a team. I also got to learn a lot about different web development methods and techniques.

**James** This project has definitely been challenging, yet rewarding. I feel I like learnt a lot about websockets and real time web application development, and I shall definitely be applying these skills in future personal projects of mine. I found the use of Trello and Slack to be invaluable resources when it came to keeping our thoughts organised as a whole. Communication was key in this project and I feel like our group achieved a high level of comradery whilst working together. There were times where the problems at hand were to difficult to tackle individually, so members would team up and aid one another until a solution was found.

I had never used Agile Software Development in my workflow before, so implementing the the Kanban methodology was interesting, to say the least. I would say that it positively boosted our productivity, and managed to aid in the process of delegating work to team members.

**Adam** I found this project to be one of the most challenging to date. I was, at first, not familiar with programming in JavaScript but am glad that this was an additional skill I have got out of the project. I found working using a development method was a great experience and found the communication and organisation of the team was greatly benefited by this. After first being introduced to Trello and Slack, I did not realise how much these particular resources were used in real-world web development and appreciate the experience in using these tools.

## 6.3 Reflection

### 6.3.1 Current issues

Running python code in the browser is a difficult thing to do. The main challenge about this, with the frameworks in their current state, is that importing files requires them to be compiled to JavaScript in advance. The only workaround is a hack to concatenate the files together.Also, debugging Python code that has been compiled to JavaScript can be tricky, especially when the problem may simply be that the feature you want to use has not been

added to Skulpt yet. Fixing these issues is one of the main things that we would improve given more time. A better way to import python files, and a more roboust way to run python code, while maintaining the advantages of Skulpt, such as a graphical interface.

### 6.3.2 Additional Features

On Trello, we have a column for cool ideas. These are features that were not part of the original requirements, but that we felt that would add to the quality and ease of use of our product. Some of them we managed to implement, some we did not. Given more time, we would like to add tablet support, and a notification system, in order to support the scenario when the teacher is walking around the classroom with a tabled, and when a student needs help, he presses a button within ColDE, the tablet starts beeping and the teacher can see which student needs help and with what. He then is able to decide whether he needs to walk over to the student, or he can just do a quick fix straight from the table.

Another idea of a great feature that could be added in the future, would be to add support for more languages, and also to change the colour of each user's cursour, such that it is easyer to change which user is doing what.

# 7 Appendix

## 7.1 Acknowledgements

We used the following third-party resources in our project:

- Bootstrap
  MIT License (https://github.com/twbs/bootstrap/blob/master/LICENSE)
- CodeMirror
  MIT License (https://github.com/codemirror/CodeMirror/blob/master/LICENSE)
- jsTree
  MIT License (https://github.com/vakata/jstree/blob/master/LICENSE-MIT)
- socket.io
  MIT License (https://github.com/socketio/socket.io/blob/master/LICENSE)
- jQuery
  MIT License (https://tldrlegal.com/license/mit-license)
- Skulpt
  MIT License (https://github.com/skulpt/skulpt/blob/master/LICENSE)

- QUnit
  MIT License (https://github.com/jquery/qunit/blob/master/LICENSE.txt)
- Docker
  Apache 2.0 License (https://github.com/docker/docker/blob/master/LICENSE)

## 7.2 Licenses

### 7.2.1 MIT License

We can use content under a MIT License, even for commercial purposes, provided we include a copy of the License and the copyright notice in the form $Copyright(c)[year][copyrightholders]$, which is available in the links provided above.

### 7.2.2 Apache 2.0 License

Similarly, we can use the content under an Apache 2.0 License provided we include a copy of the License and the copyright notice, as detailed above.

# References


[1] TuringLab
http://www.turinglab.co.uk/

[2] Scratch
https://scratch.mit.edu/

[3] Flask
http://flask.pocoo.org/

[4] Django
https://www.djangoproject.com/

[5] Pyramid
http://www.pylonsproject.org/

[6] SQLAlchemy
http://www.sqlalchemy.org/

[7] JQuery
https://jquery.com/

[8] Etherpad
http://etherpad.org/

[9] CodeMirror
https://codemirror.net/

[10] Chrome DevTools
https://developer.chrome.com/

[11] Mozilla Firefox Web Console
https://developer.mozilla.org/

[12] Google Docs
https://www.google.co.uk/docs/about/

[13] jsTree
https://www.jstree.com/

[14] Skulpt
http://www.skulpt.org/

[15] Pypy.js
http://pypyjs.org/

[16] QUnit
https://qunitjs.com/

[17] Heroku
https://www.heroku.com/

[18] Digital Ocean
https://www.digitalocean.com/

[19] Docker
https://www.docker.com/

[20] Trello
https://trello.com/

[21] Slack
https://slack.com/

[22] Skype
http://www.skype.com/

[23] GitLab
https://about.gitlab.com/

[24] Atlassian
https://www.atlassian.com/git/tutorials/comparing-workflows