University of Aberdeen Codemarker

Technical Report

~$./CODEMARKER.sh

TEAM ALPHA:

Konrad Dryja (Team Leader)
Stefan Rudvin (Deputy Team Leader)
Tautvydas Cerniauskas
Elzbieta Futkowska
Adrien Mainka

C:\> TABLE OF CONTENTS:

C:\> ABSTRACT:

The purpose of Codemarker is to assist students and professors when grading and submitting their assignments. The following technical report is an addition to the white paper to analyse the technicalities behind the project.

The system that we received in September 2017 has been originally developed using Ruby on Rails MVC framework by previous students of the University of Aberdeen. We are in touch with Dr Nir Oren, who is responsible for the current implementation of Codemarker. From this point onwards he will be referred to as the client.

We decided to move away from the present solution and rebuild it using more up-to-date technologies and frameworks. We also wanted to develop a more modern approach to the stack by using a completely API based backend. This idea was endorsed after discussing our ideas with the client, who also advised us to move away from the Rails approach. Another reason for straying away from Ruby was the teams limited knowledge of the language and the resulting learning curve of using a new framework. The new Codemarker will use a three tier architecture with Django for the backend, ReactJS for the frontend and an SQL database to store data.

C:\> USE CASE MODEL:

Codemarker is a system that will provide a platform for computing science lecturers and staff to set up assessments, and for students to make submissions to these assessments.

The actors of our system will be teachers (coordinators) and students.

In cooperation with our client we have identified the following requirements – both functional and non-functional, and ordered them by their importance:

Functional requirements:

- Coordinators should be able create new assignment and test cases
- System must provide a report on whether a submission meets its requirements
- System must handle multiple programming languages
- System must be able to run multiple session at a time
- Coordinators should get a global report with all grades for an assessment
- Coordinators can trace students who did not submit their work or who are at risk with low marks
- Reports can identify specific failed test cases
- System can process complete web applications
- Users shall be able to log in with their university ID

- Students that are enrolled in a course should be notified by email whenever an assessment is created and few days before the deadline if they have not submitted anything yet.

Non-functional requirements:

- System shall correct submissions in about 5 seconds, taking the time to start up a virtual machine into consideration
- The report shall be displayed without having to refresh the page
- System should provide an unhindered access to the website for people with disabilities.
- Application should run without any issues on Internet Explorer v6.0 and above, Chrome v48 and above, Safari v8.0 and above, Firefox v45 and above.

C:\> CANDIDATE SYSTEM ARCHITECTURE:

The purpose of our program is to receive students' code submissions, run them, and compare their output to the to the output of the sample code, given the same inputs.

Submitted code will be run on the virtual machines which will test user's program with provided inputs. The Docker software container system is used as our virtual environment to run scripts in different programming languages such as Java, Python and Ruby. It is a tool that can package an application and its dependencies in virtual container, which is able to run on any Linux server. In addition, Docker is lightweight and allows many virtual containers to run at the same time. This complete system is shown below in figure 1.

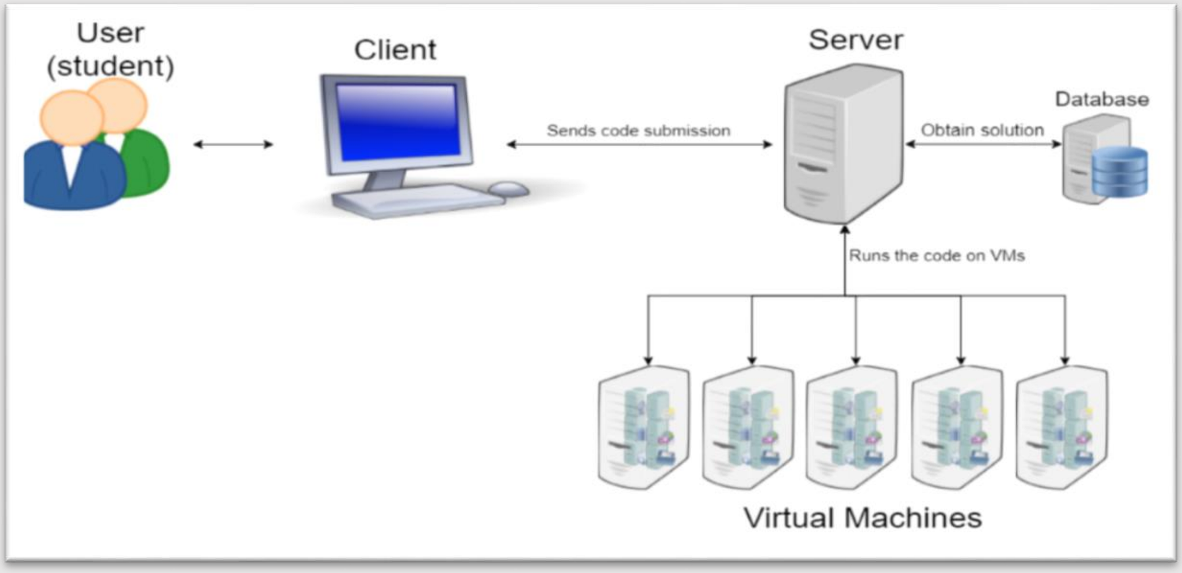


Figure 1: Simplified System Structure

C:\> REASONS FOR MOVING AWAY FROM RoR:

Due to multiple complications, we were forced to decide on deflection from Ruby on Rails to a different approach (ultimately decided on Django and ReactJS). These include inter alia:

- Outdated systems:
  - The original system is very deprecated. The Ruby on Rails version that is used in the project is 3.2.11 with the current stable version being 5.1.4 means we would probably have to refactor most of the source code to keep it relevant. The system also features the "daemons" gem which is deemed obsolete and poses a serious security risk for production use.
- Difficulty starting and running the project on local machines:
  - The project itself was originally built on a Linux server based within IT Services of University of Aberdeen. Because of that we had no way to start the project on our machines, so the only option was to analyse the systems' source code and work from there. Any further debugging attempts would fail due to incompatibilities and the closed environment of the IT Services within the University.
- Lack of experience:
  - Most of our team-members have never used Ruby on Rails to the extent that we could develop such an industry standard application. Time and resources spent on the extra training and learning would not be feasible in terms of costs.
- As advised by the client:
  - After discussing the current system in place and the department's vision for the future requirements, he advised that it would be a smarter idea to rebuild the project from square one as the needs and taught material have heavily evolved over time.

As mentioned before, we have switched our efforts to a Python and JavaScript based solutions, and the thought processes behind this decision is outlined below:

- Experienced based decision:
  - As most of us already worked with Python and JavaScript in previous projects, we decided to use them rather than learning a new framework.
- Up-to-date frameworks for future employability:
  - Django and ReactJS are two of the 5 most popular frameworks in the web development field at the moment.
  - Learning and using them provides an excellent start to future employability, as they are commonly used in the industry. While Rails remains a feasible option, its lacklustre frontend features almost forces developers to use a separate frontend framework.

C:\> THREE-TIER ARCHITECTURE:

We have implemented a three-tier architecture for our system which can be represented with the: Presentation Layer, Business Layer and Data Layer. The diagram below characterises all of them and lists their components. The Presentation and Business Layer hold the core of our project. The presentation layer holds the system architecture for the frontend which is built using ReactJS. The most important structural layer of our system is included inside the Business Layer, such as the service which processes submitted code. Having a separate frontend and backend comes with the following advantages:

- Tasks can be split between people with different skill sets to maximize efficiency
- Modularity: it is simple to change the application at different paces
- Evolution: the system can be upgraded if better technologies come out
- Working with API's: this forces us to use current industry standards

The Data Layer extends the Django framework with a MySQL database to store courses, student details, grades received, last submissions and the course coordinator's input scripts.

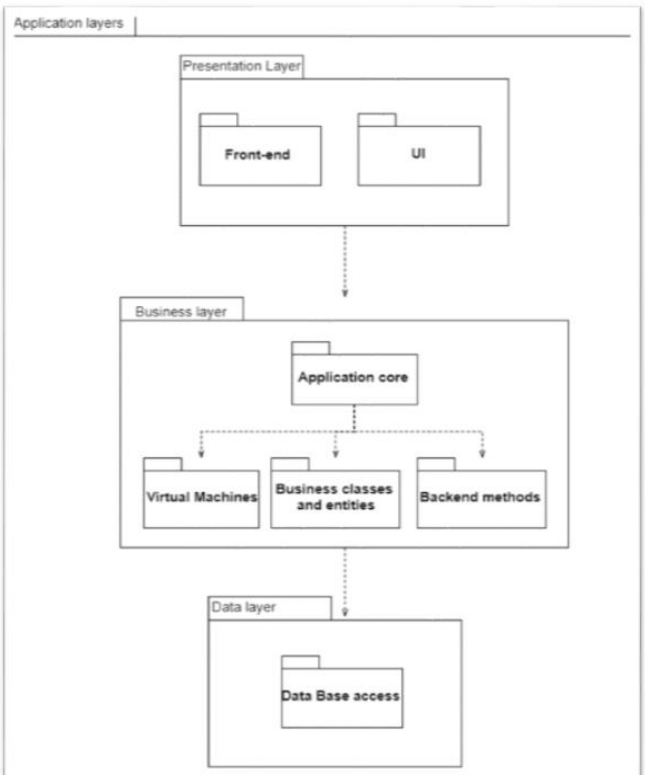


Figure 2: Three-tier Architecture

As previously mentioned, we tried to use the most up-to-date technologies currently available. ReactJS is simple to implement for small to medium sized projects, and it fits our needs perfectly. Simultaneously it uses special syntax called JSX, which lets us mix HTML code within JavaScript and produce reactive user interactions.

The backend has been implemented using Django REST Framework, which allows us to separate business logic from logic that deals with API calls. In addition, Django does not fall in a typical MVC framework - developers extended the MVC functionality and called it the "MTV" framework, which comprises of the "model", "template", and "view". The idea behind "view" is a Python call-back function describing how data is represented. The controller in the system is non-existent because it is implemented into the framework itself.

The two-tier architecture displayed below shows the server side connection to the client. The client sends a request to the server, which sends back an API response in JSON format. The user only sees the user interface, which provides functionality to upload submissions. The server analyses the submission and returns a report which is presented to the user.

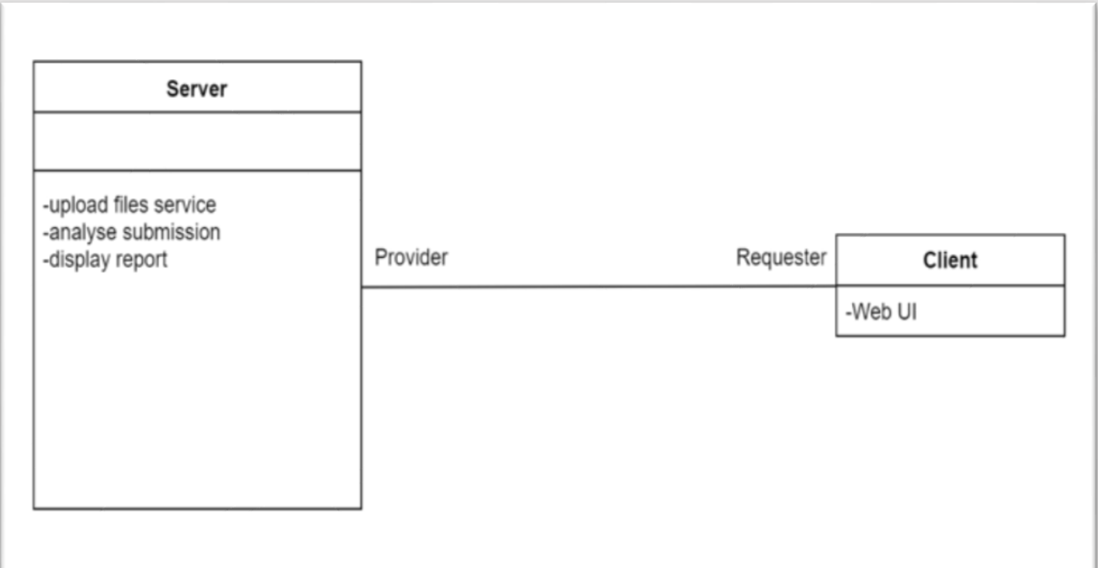


Figure 3: Two-tier Architecture

C:\> INITIAL RISK ASSESSMENT:

Codemarker can be identified with three degrees of impact: high, medium and low.

The following diagram presents the impact that the failure of any component will cause to the system. The letters A or B are classified as major problems, and their repair time could take from days to weeks. They must be seriously addressed at all times. Lower degree risks must be addressed as well, but not immediately. For example, were a new Python version come out, the team may consider upgrading to it in due time.

Impact

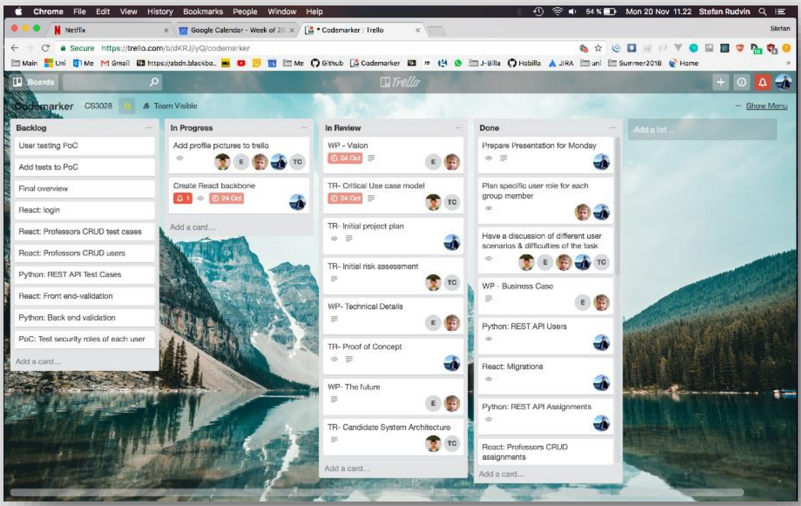
High-level	B	A	A
Medium-level	C	B	A
Low-level	C	C	B
	Low	Medium	High

Likelihood

- High-level:
  1. Security of the Data:
     - As we are using student information such as students' ID, email addresses, and lecturer information, the data has to be protected from malicious attacks. A failure in security would resolve in a massive loss of student and professor data. Furthermore, this could result in a number of law suits against the company and a general loss of credibility.
  2. Security of the System:
     - The only people allowed to login to the system are students, lecturers and administrators. The Codemarker must be able to hold against most of the OWASP web application security categories.
  3. Privacy
     - All the data of the system has to be private and secure. This is accomplished by signed documentation with the universities that staff members that are managing the courses. The information is only accessible by the parties that have rights to use it.
- Medium-level:
  1. High Number of Users
     - Handling more than 30 simultaneous submissions at the same time puts a lot of stress on Docker, which creates a virtual machine for each submission. This problem can be solved two different ways:
       - Increase number of virtual machines possible by allocating more resources to the backend server
       - Add a short waiting time before the assignment can be run
  2. Data Server Loss
     - Another issue that could increase user dissatisfaction if Codemarker loses the connection the data server. This problem could prevent students from submitting their assignments, which could result in them failing the course. This can be avoided by signing a Service Level Agreement (SLA) with the server provider, which ensures that connection issues are fixed as soon as possible.
- Low-level:
  1. New Versions of Languages Supported:
     - If code with a newer version of programming language is submitted, it will display an error that code could not run on the system. Because of this, users will be notified on which versions of programming languages they can use.
  2. Update on the Tools in Use:
     - Upgrading of the current tools in use can problems with compatibility. Since this can affect the continuous development Codemarker, the necessity of future updates would be highly considered before they are taken into production.

C:\> PROJECT PLAN:

When started with the project we had to rethink our approach to planning. Efficiency and speed were the key aspects as the term is pretty short and task was lengthy. Upon further discussion, we decided to utilise an Agile approach to ensure a fast development pace. A SCRUM board has been set up through Trello, so we can check the status of the given part of the project in real time. We have also maintained weekly meetings with strict note-taking to avoid losing vital information.


Chrome File Edit View History Bookmarks People Window Help
Mon 20 Nov 11.22 Stefan Rudvin
Netflix
Google Calendar - Week of 20
• Codemarker | Trello
https://trello.com/b/dKRJjlyQ/codemarker
Boards
Trello
Codemarker CS3028 Team Visible
Show Menu
Add a list...

Backlog
- User testing PoC
- Add tests to PoC
- Final overview
- React: login
- React: Professors CRUD test cases
- React: Professors CRUD users
- Python: REST API Test Cases
- React: Front end-validation
- Python: Back end validation
- PoC: Test security roles of each user

In Progress
- Add profile pictures to trello
- Create React backbone (24 Oct)

In Review
- WP - Vision (24 Oct)
- TR- Critical Use case model (24 Oct)
- TR- Initial project plan
- TR- Initial risk assessment
- WP- Technical Details
- TR- Proof of Concept
- WP- The future
- TR- Candidate System Architecture

Done
- Prepare Presentation for Monday
- Plan specific user role for each group member
- Have a discussion of different user scenarios & difficulties of the task
- WP - Business Case
- Python: REST API Users
- React: Migrations
- Python: REST API Assignments
- React: Professors CRUD assignments

Figure 4: Trello Kanban board

Below are the details outlining achievements, milestones and the progress of every meeting throughout the first term, which were produced thanks to the weekly minutes documents:

Inception:

Week 1:

- Get to know with each other
- Brainstorm some project ideas
- Allocate Konrad Dryja as project manager and Stefan Rudvin as deputy manager
- Create a Slack discussion board to communicate and exchange ideas

Week 2:

- Meeting with Dr Bruce Scharlau - our guide
- Evaluate the feasibility and the business logic of gathered ideas
- Decide to partake Codemarker project – internal University of Aberdeen system, with course coordinator’s approval

Week 3:

- Meeting with Dr Nir Oren, our client, original Codemarker's maintainer
- Discuss the current software
- Discuss his requirement, points of improvement
- Conclude that we might need to rewrite the code, as the current one is old and not very usable
- Consider migration to Docker
- Evaluate the functional and non-functional requirements under the form of user stories

Elaboration:

Week 4:

- Discuss the strengths, weaknesses and ability of each group member
- Determine use of a relational database
- Discern and assign the different tasks such as sections of the white paper or technical report to team members

Week 5:

- Define the programming paradigm as object programming, focus on reusability, stability and security
- Rewrite the requirement as use cases
- Decide to rewrite the system using Django and ReactJS

Week 6:

- Decide that the software will comply with the architectural patterns as the three-layer architecture and the client-server architecture
- Start to develop the first draft of the prototype

Week 7:

- Precisely refined the architectural pattern
- Decide to have the software running on a Docker
- Define the design criteria such as high cohesion and low coupling

Week 8:

- The prototype can display courses, assessments
- The submission module is functional

Week 9:

- The prototype can handle multiple submissions for an assessment
- After a submission, it produces a report and displays it to the user
- It displays the past submissions and time taken to run, mark, along with status and result

Week 10:

- Finish technical report and white paper and send it to the guide for criticism
- Define testing strategies
- Prototype allows creation of courses and assessments
- Prototype creates a new Docker instance on submission upload and sends text to it

C:\> PROOF OF CONCEPT:

Developing the initial Proof of Concept software was a welcome challenge of learning two new frameworks and assessing the technical viability of the product. The project source code is saved in a collaborative GitHub repository, and the project is hosted on http://notey.me/ (URL **must not** contain the 'www' prefix). Functionality can be tested with the "CS101 Learn to code!" course and the "Sum Two Numbers" assessment (http://notey.me/assessments/2). The python file '**sum2nums.py'** is included in the assessment root, which can be submitted to the assessment with a passing result.

What has been completed so far?

The following core requirements have been completed in full working order with backend and frontend implementations:

- User can see all available courses and their descriptions
- User can see all assessments linked to a specific course
- User can view an assessment with its end date, start time, description and additional help fields
- User can see all previous submissions linked to an assessment
- User can upload multiple submissions into an assessment
- Backend assesses code against a 'Sum two numbers' test and produces report based on code correctness and time taken
- After uploading file, user is presented with report in a modal
- User can view previous submission reports which display the time taken to run the code, the mark, status and result
- User can view and add courses via administrator panel
- User can add new assessments to a course
- User-friendly UI with the Bulma framework
- All data is retrieved from a live MySQL database
- Project is hosted on two DigitalOcean servers for the frontend and backend
- Domain http://notey.me/ was created with Codemarker directed to it
- On submission upload, the server creates a new Docker instance and passes text to it

Problems we faced

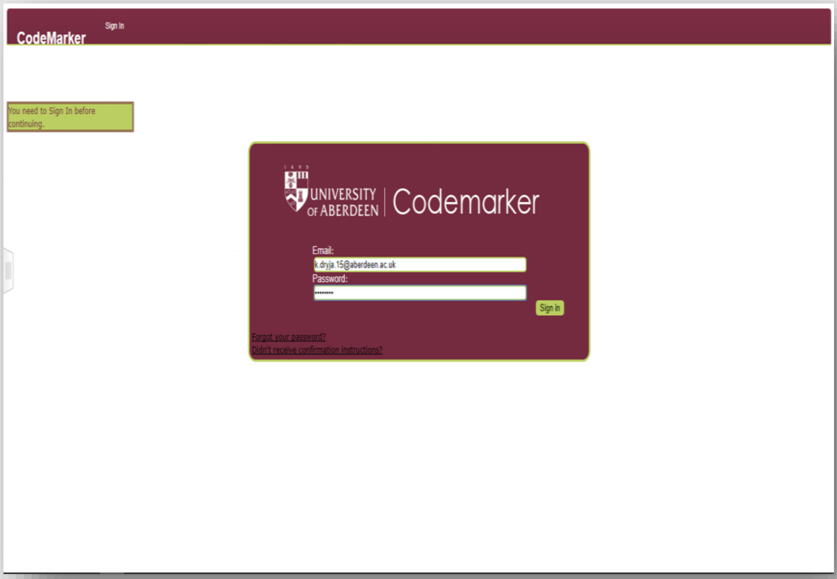
As we used two frameworks we had barely used before, we were bound to be greeted with a slight learning curve. The combination of using Django and React is also uncommon, which led to a low availability of documentation.

The first of our issues were package manager related – initially we used a local pip installation, but then realised that our code did not execute in a similar manner on all of our machines. After some research we decided to use virtualenv, a popular python virtual environment solution which wraps all of the required packages into a local repository. This way our code runs seamlessly across all python platforms.

Our other issues were related to the communication between the frontend and the backend. On the backend, we used the Django rest framework, which supplied us with excellent boilerplate methods, but some unwanted conventions for a Proof of Concept.

While our backend API calls worked great through a direct http request, our frontend could not access them at all. Through some debugging we found out that the calls were blocked by the rest framework CORS (Cross-origin resource sharing) protection. After disabling it in settings.py, the API calls did not return a code 500, but a code 403 instead. Puzzled by the response, we realized that the middleware of the rest framework was preventing access to the Django views. This was once again solved by adding a flag to settings.py.

Another issue was uploading files to the backend. While GET and POST requests worked excellently, submitting a form with a file was returning an error code. We found that the form object was at fault, as we forgot to add the CSRF (Cross-Site Request Forgery) tag to it. This was a new lesson for us, since we were used to using frameworks that added these tags automatically.


CodeMarker
Sign In
UNIVERSITY OF ABERDEEN | Codemarker
Sign in
Forgot your password?
Didn't receive confirmation instructions?

You need to Sign In before continuing.

Email: k.dryja.15@aberdeen.ac.uk
Password: ••••••••

Figure 5: Original Project

Lessons learned

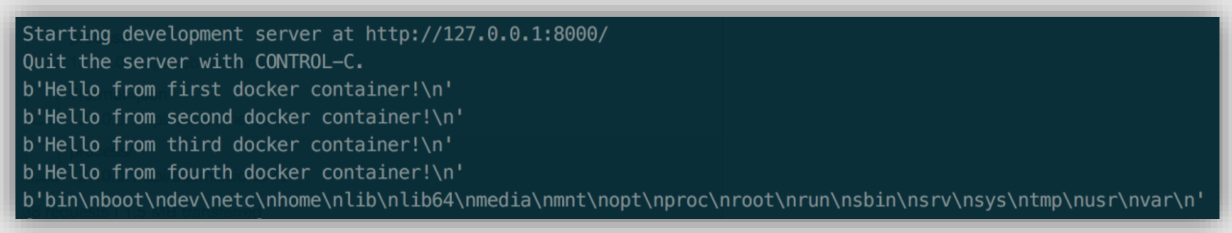
Using a modern JavaScript framework such as React was a great learning experience. We learned about tools such as NPM and WebPack, as well as packages such as Axios and React router.

While we have plenty of experience with other JavaScript frameworks, learning about React's object reactiveness and passing of props was a look into another way to handle such things. For example, React uses the 'this.state' object to handle all reactive elements of the page. These can be accessed and modified with the 'SetState()' method rather than directly like in other JavaScript frameworks.

Regarding DRY principles, we implemented our own 'configurable Table' component which is reused across the system. We also reused other code such as the navigation bar across all pages.

With the backend, Django projects require more setup and knowledge about 'apps' versus 'projects' structure. Separating every concern of the system into an app can be a confusing paradigm for newcomers. However, this is a much more desirable model for reusing code.

We learned many things about setting up a development server, such as installing packages, setting up MySQL and redirecting IP addresses to domain names. Docker was also a new experience, as well as the Python-Docker package that handles it. When uploading a submission, the backend creates four Docker instances and sends text to them, which they echo back to the service. This output is shown below in figure 6.



```
Starting development server at http://127.0.0.1:8000/
Quit the server with CONTROL-C.
b'Hello from first docker container!\n'
b'Hello from second docker container!\n'
b'Hello from third docker container!\n'
b'Hello from fourth docker container!\n'
b'bin\nboot\ndev\netc\nhome\nlib\nlib64\nmedia\nmnt\nopt\nproc\nroot\nrun\nsbin\nsrv\nsys\ntmp\nusr\nvar\n'
```

Figure 6: Output of Docker Instances

Things that worked as expected

Working with Ajax requests using the Axios package was a soothing experience. While it is an external dependency, Axios decouples API calls from React and makes the system more modular. Documentation was excellent and implementing features such as form uploads was a breeze.

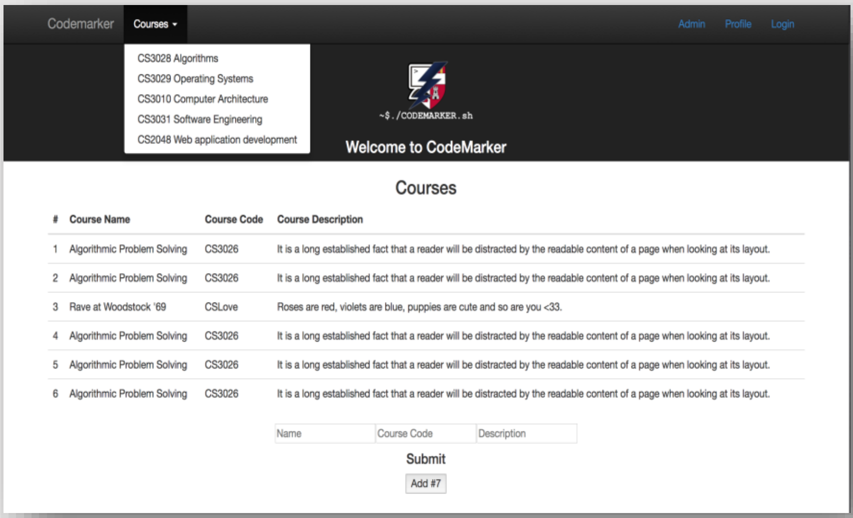
Using the rest framework in Django also worked as expected. Creating new resourceful endpoints simply required a new URL route, view and serializer.

Viability of the system

Overall, our product satisfies client requirements well. Codemarker will implement all functionality that the current solution does, as well as extra features such as Docker integration and the benefits of a single page application.

The Proof of Concept handles core complex logic such as running an uploaded file on the server, sending sample inputs to it and producing a report based on its output. Docker is also integrated in the system. Although in an early stage, it proves that running multiple virtual containers and interacting with them simultaneously is completely feasible.

Last of all, the Proof of Concept is well documented, follows current industry standards and is built in an extensible manner. The code follows the Python PEP8 and JavaScript JSLint standards, as well as clean code (such as DRY) guidelines. This provides the team with a solid cornerstone to begin developing the final product.



Welcome to CodeMarker

Courses

#	Course Name	Course Code	Course Description
1	Algorithmic Problem Solving	CS3026	It is a long established fact that a reader will be distracted by the readable content of a page when looking at its layout.
2	Algorithmic Problem Solving	CS3026	It is a long established fact that a reader will be distracted by the readable content of a page when looking at its layout.
3	Rave at Woodstock '69	CSLove	Roses are red, violets are blue, puppies are cute and so are you <33.
4	Algorithmic Problem Solving	CS3026	It is a long established fact that a reader will be distracted by the readable content of a page when looking at its layout.
5	Algorithmic Problem Solving	CS3026	It is a long established fact that a reader will be distracted by the readable content of a page when looking at its layout.
6	Algorithmic Problem Solving	CS3026	It is a long established fact that a reader will be distracted by the readable content of a page when looking at its layout.

Figure 7: First draft

The Future

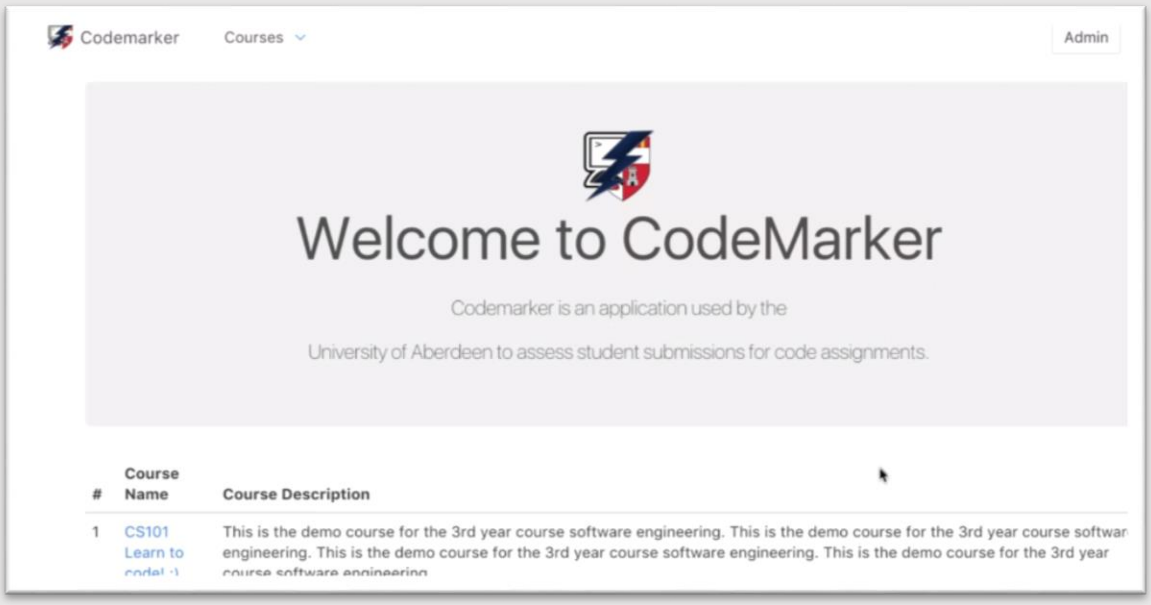
Regarding refactoring, we will implement our own vanilla JavaScript service provider with an api.js route file to completely decouple API calls from React. We will also use the react event system to better handle the opening of modals and user feedback. In the backend, we will possibly move some components into their own apps and create multiple route files instead of putting all routes into one file. We may also refactor some of the models to better fit our future marking scheme.

Technical features for final product:

- Link user model to frontend via oAuth
- Implement logged in user token to all API calls
- Implement further administrator functionality
- Implement middleware for endpoint protection
- Implement policies for resource protection
- Implement Frontend validation to all forms
- Allow professor to upload sample code that assessment is based on

- Implement support for multiple programming languages
- Implement creation of random test cases that sample code and submitted code will be run against
- Implement full marking service that creates a report based on submission performance
- Run submissions on Docker instances, pass input to them and extract output
- Implement Docker integration with full-stack web applications
- Add test coverage for endpoints, resources, security
- Add frontend minimum viability tests

While our proof of concept software is limited in terms of functionality, we believe it proves it is a viable system that can be taken into production. With all of the above features it will be a competitive product that satisfies all current needs of the customer.



Welcome to CodeMarker

Codemarker is an application used by the

University of Aberdeen to assess student submissions for code assignments.

#	Course Name	Course Description
1	CS101 Learn to code! :)	This is the demo course for the 3rd year course software engineering. This is the demo course for the 3rd year course softwar engineering. This is the demo course for the 3rd year course software engineering. This is the demo course for the 3rd year course software engineering

Figure 8: Current state of the prototype