



Cloud based serverless web application

**Daniel Perez Lorenzo**  
Grado de Ingenieria informatica

**Gregorio Robles Martinez**  
09-06-2022

2022 Daniel Perez Lorenzo



Reconocimiento-NoComercial-  
SinObraDerivada.

## FICHA DEL TRABAJO FINAL

<b>Título del trabajo:</b>	Cloud based serverless web application
<b>Nombre del autor:</b>	Daniel Perez Lorenzo
<b>Nombre del consultor:</b>	Gregorio Robles Martinez
<b>Fecha de entrega (mm/aaaa):</b>	06/2022
<b>Área del Trabajo Final:</b>	Desarrollo web
<b>Titulación:</b>	<i>Grado de Ingeniería informática</i>
<b>Resumen del Trabajo (máximo 250 palabras):</b>	
<p>El propósito de este proyecto es desarrollar un aplicativo web serverless y auto escalable.</p> <p>Con la actual situación de mercado y la tendencia hacia el uso de tecnologías cloud junto a las ventajas que este ofrece en términos de coste, rendimiento, escalabilidad y facilidad del despliegue de la infraestructura. El Desarrollo en la nube se ha convertido en una solución perfecta para construir casi cualquier tipo de aplicativo.</p> <p>Por las razones escritas anteriores he decidido utilizar AWS (Amazon Web Services) para construir toda la infraestructura utilizando IaC (Infrastructure as Code), configurando CI/CD y control de versión junto con una arquitectura de micro servicios, NodeJS, ReactJS y Python.</p> <p>La conclusión de este proyecto, aparte de haber alcanzado el objetivo tecnológico principal y cumpliendo el caso de uso del mismo, también puedo concluir que el uso de tecnologías cloud, para el desarrollo web, puede ser fácilmente justificado por el gran número de beneficios que este aporta en cuanto a infraestructura, coste y rendimiento.</p>	

**Abstract (in English, 250 words or less):**

The purpose of this project is to develop a self-scalable, serverless cloud web application.

With the current market trending towards cloud infrastructure and its advantages in terms of cost, performance, scalability and ease of deployment, cloud development has become a perfect solution for building almost any web application.

For that reason, I have chosen to use AWS (Amazon Web Services) for building the entire infrastructure, using IaC (Infrastructure as Code), setting up an entire development flow with CI/CD and version control together with a microservices design in mind, NodeJS, ReactJS and Python.

The conclusion of this project is that, apart from fulfilling the use case of the web application, cloud technologies can be easily justified for any type of web development for the number of benefits it brings when it comes to infrastructure, costs, and performance.

**Palabras clave (entre 4 y 8):**

AWS, Cloud, Microservices, NodeJS, ReactJS, IaC, Python

# Contents

<b>1 INTRODUCTION.....</b>	<b>1</b>
1.1 CONTEXT.....	1
1.2 PROJECT GOALS.....	1
1.3 APPROACH AND METHODOLOGY.....	2
1.3.1 <i>Tech stack</i> .....	2
1.3.1.1 Front end.....	2
1.3.1.2 Back end.....	2
1.3.1.3 Infrastructure.....	2
1.3.1.4 Test coverage.....	3
1.4 PLANNING.....	4
1.5 PROJECT RESULTS SUMMARY.....	5
1.6 FOLLOWING CHAPTERS SUMMARY.....	5
<b>2 PROJECT REQUIREMENTS.....</b>	<b>6</b>
2.1 FUNCTIONAL REQUIREMENTS.....	6
2.1.1 <i>Use cases</i> .....	7
2.2 NON-FUNCTIONAL REQUIREMENTS.....	11
<b>3 PROJECT DESIGNS.....</b>	<b>12</b>
3.1 AWS ARCHITECTURE.....	12
3.2 DATA MODELLING.....	13
3.3 FE DESIGN.....	14
3.3.1 <i>Home page</i> .....	14
3.3.2 <i>Login</i> .....	14
3.3.3 <i>My Studies tab</i> .....	15
3.3.4 <i>Create/Modify Study</i> .....	15
3.3.5 <i>Questionnaire view</i> .....	16
3.3.6 <i>Add question</i> .....	16
3.3.7 <i>Get access link</i> .....	17
3.3.8 <i>Filling in questionnaire data</i> .....	17
3.4 FE ARCHITECTURE.....	18
3.5 BE LAMBDA ARCHITECTURE.....	19
3.6 BE PYTHON ARCHITECTURE.....	20
3.7 JIRA STORIES.....	20
<b>4 DEVELOPMENT.....</b>	<b>21</b>
4.1 REPOSITORIES.....	21
4.2 AWS COGNITO CONFIGURATION.....	21
4.3 SETTING UP DYNAMODB.....	23
4.4 NODEJS LAMBDA.....	24
4.4.1 <i>Development environment</i> .....	24
4.4.1.1 <i>Installing NodeJS</i> .....	24
4.4.1.2 <i>Installing brew</i> .....	24
4.4.1.3 <i>Installing Docker</i> .....	24
4.4.1.4 <i>Installing sam-cli</i> .....	24
4.4.1.5 <i>Installing PAW</i> .....	24

4.4.1.6 Creating the first Lambda .....	25
4.4.2 Setting up CodePipeline.....	27
4.4.3 Setting up buildspec.yml .....	33
4.4.4 Development NodeJS .....	36
4.4.4.1 Architecture.....	36
4.4.4.2 Code base .....	37
4.4.4.3 Setting up API Gateway.....	41
4.4.4.4 Testing.....	44
4.5 REACT JS.....	46
4.5.1 Development environment .....	46
4.5.2 Setting up an S3 bucket for web access .....	46
4.5.3 Setting up CodePipeline.....	49
4.5.4 AWS CloudFront .....	52
4.5.5 Setting up deployment with webpack and Babel .....	54
4.5.5.1 Webpack.....	54
4.5.5.2 Babel .....	55
4.5.5.3 Deployment.....	56
4.5.6 Development.....	57
4.5.6.1 Architecture.....	57
4.5.6.2 Code .....	58
4.5.6.3 Website.....	60
4.6 PYTHON.....	66
4.6.1 Development environment .....	66
4.6.2 Setting up CodePipeline.....	66
4.7 ACCEPTANCE TESTING .....	67
4.8 DESIGN CHANGES.....	68
4.8.1 Major changes.....	68
4.8.2 Minor changes.....	68
<b>5 FINANCIAL EVALUATIONS .....</b>	<b>69</b>
<b>6 CONCLUSIONS.....</b>	<b>70</b>
6.1 LESSONS LEARNT .....	70
6.2 GOALS ACHIEVED .....	70
6.3 PLANNING.....	70
6.4 FUTURE VISION.....	71
<b>7 GLOSSARY .....</b>	<b>72</b>
7.1 AWS.....	72
7.2 CLOUD .....	72
7.3 MICROSERVICES.....	72
7.4 IAC .....	72
7.5 TECH STACK.....	72
7.6 FRAMEWORK.....	72
7.7 API.....	72
<b>8 BIBLIOGRAPHY .....</b>	<b>73</b>
<b>9 ANNEX.....</b>	<b>75</b>

## Figures

Figure 1: Gantt .....	4
Figure 2: AWS Architecture .....	12
Figure 3: Home page design .....	14
Figure 4: Login design .....	14
Figure 5: My studies tab design.....	15
Figure 6: Create/modify study design .....	15
Figure 7: Questionnaire view design .....	16
Figure 8: Add question design.....	16
Figure 9: Get access link design .....	17
Figure 10: Filling in questionnaire data design .....	17
Figure 11: FE architecture .....	18
Figure 12: Jira stories .....	20
Figure 13: Cognito configuration 1 .....	21
Figure 14: Cognito configuration 2 .....	22
Figure 15: Final Cognito configuration.....	22
Figure 16: DynamoDB table creation .....	23
Figure 17: First Lambda code.....	25
Figure 18: First Lambda CloudFormation definition.....	26
Figure 19: First Lambda testing code .....	27
Figure 20: CodePipeline configuration 1 .....	27
Figure 21: CodePipeline configuration 2 .....	28
Figure 22: CodePipeline configuration 3 .....	28
Figure 23: CodePipeline configuration 4 .....	29
Figure 24: CodePipeline configuration 5 .....	29
Figure 25: CodePipeline configuration 6 .....	30
Figure 26: CodePipeline configuration 7 .....	30
Figure 27: CodePipeline configuration 8 .....	31
Figure 28: CodePipeline configuration 9 .....	32
Figure 29: Buildspec configuration 1 .....	33
Figure 30: Buildspec build stage .....	34
Figure 31: API Gateway .....	34
Figure 32: Lambda test .....	35
Figure 33: NodeJS architecture.....	36
Figure 34: NodeJS code base .....	37
Figure 35: Code base .....	38
Figure 36: NodeJS code base use cases .....	38
Figure 37: NodeJS code base delete questionnaire .....	39
Figure 38: NodeJS code base repositories .....	39
Figure 39: NodeJS Code base use case example .....	40
Figure 40: CORS template.yml.....	41
Figure 41: API Gateway authorizer .....	42
Figure 42: CORS pre-flight.....	43
Figure 43: Disable authorization.....	43
Figure 44: NodeJS testing .....	44
Figure 45: NodeJS setting up automated testing during build .....	44
Figure 46: NodeJS CodePipeline test results .....	45
Figure 47: NodeJS PAW .....	45

Figure 48: CodePipeline configuration 1 .....	46
Figure 49: CodePipeline configuration 2 .....	47
Figure 50: CodePipeline configuration 3 .....	47
Figure 51: CodePipeline configuration 4 .....	48
Figure 52: CodePipeline configuration 5 .....	49
Figure 53: CodePipeline configuration 6 .....	50
Figure 54: CodePipeline configuration 7 .....	50
Figure 55: CodePipeline configuration 8 .....	51
Figure 56: S3 public url .....	51
Figure 57: ReactJS running on public site .....	51
Figure 58: CloudFront configuration 1 .....	52
Figure 59: CloudFront configuration 2 .....	53
Figure 60: Cloudfront public url .....	53
Figure 61: Deployment webpack Babel 1 .....	54
Figure 62: Deployment webpack Babel 2 .....	55
Figure 63: Deployment webpack Babel 3 .....	56
Figure 64: ReactJS architecture .....	57
Figure 65: ReactJS folder structure .....	58
Figure 66: Axios request .....	59
Figure 67: ReactJS home page .....	60
Figure 68: ReactJS login .....	60
Figure 69: ReactJS wrong credentials .....	61
Figure 70: ReactJS my studies tab .....	61
Figure 71: ReactJS create/modify .....	62
Figure 72: ReactJS questionnaire .....	62
Figure 73: ReactJS add question .....	63
Figure 74: ReactJS questionnaire link .....	63
Figure 75: ReactJS filling in desktop .....	64
Figure 76: ReactJS filling in mobile .....	64
Figure 77: ReactJS .csv file .....	65
Figure 78: CodePipeline .....	66
Figure 79: Python Lambda .....	66
Figure 80: Major changes .....	68



# 1 Introduction

## 1.1 Context

After some years working in a research centre leading multiple web development projects, I understood how crucial it is for such centres to have a tool for gathering data from patients.

While there are existing tools, such as SurveyMonkey, Google Forms or Lime Survey which allows their users to survey customers, this project is oriented towards providing a comfortable and easy to use tool to create surveys for scientific studies, with exporting data functionalities.

The long-term vision is to be able to offer data analysis and automatic reports. Although these will not be part of the scope of this project, the project will be a starting point for developing such services.

## 1.2 Project goals

This project has two very distinct goals. On one side, the creation of a web application tool that can be used by students and researchers to fulfil their necessities to gather data from participants. And on the other, explore the advantages of cloud development, building a cloud infrastructure with auto-scale capabilities and serverless.

These main goals can be summarized as follows:

- Provide a survey creation tool with exportable data options aimed for the educational and scientific community.
- Use scalable cloud infrastructure.

However, the target audience of this project can be quite diverse, from young researchers or students, that might be more comfortable with the usage of web applications, to more senior researchers that might not be that up to date.

For that reason, the web application must be very easy to use, and rely on simplicity. It needs to get the job done, without confusing their users.

From the technological perspective, to ensure a long-lasting life for the web application, using the latest front end and back end technologies is a must.

Secondary goals

- Easy and simple to use.
- Use latest front end and back end technologies.
- Options to export raw data in csv format.
- Test coverage.

## 1.3 Approach and methodology

The approach for this project was to use the latest web development frameworks and the latest cloud technologies offered by AWS. As one of the leading cloud providers, Amazon Web Services offers one of the best solutions in this field.

### 1.3.1 Tech stack

#### 1.3.1.1 Front end

For the front end development, I used **ReactJS** [1] a modern free and open-source front end JavaScript library that allows the developer to build user interfaces based on components. Its capabilities for re-rendering the DOM elements only when they have changed their state makes it very efficient.

**Semantic UI** [2] was used for adding styling to each component. It is a CSS framework with already in-built classes to style components.

Finally, to have backwards compatible JS code, I used **Babel** [3], and to compress the JS and CSS code, **webpack** [4].

#### 1.3.1.2 Back end

Regarding the back end, the code lives on **AWS Lambdas** [5]. AWS Lambdas are serverless, event-driven compute services that run code without provisioning or managing servers. Once one Lambda gets triggered it will spawn the code, run it and then shut down.

Although Lambdas that are recently executed stay in a 'warm' state, where the code is still loaded in case there are more consequent executions, if that is not the case it will shut down and repeat the same process once it is needed again.

My set of AWS Lambdas contains **NodeJS** [6], a JavaScript runtime environment for back end development. Because Lambdas are small functions that have a brief lifetime, NodeJS is light enough for the Lambda to start up, execute the code and die in a short period, as well as for the **Python** [7] application to handle the reports.

#### 1.3.1.3 Infrastructure

The application infrastructure is in AWS. Several services are used to achieve the most serverless [8] application possible, and to have a fully scalable architecture.

For the user's authentication, **AWS Cognito** [9] is used. This service offers user sign-up, sign-in and access control.

As an entry point for the backend APIs, **AWS API Gateway** [10] is used. This allowed me to create REST APIs [11] and connect them to Lambda functions. It was also integrated with AWS Cognito to manage the permissions of each request.

For the database, I made use of **AWS DynamoDB** [12], a NoSQL database prepared for high concurrency and connections with in-memory cache and excellent performance.

The front end of the application needed to be located somewhere; in this case, instead of using an EC2 instance which would be running 24/7 even when no users are connected to it, I used an S3 bucket with CloudFront.

**AWS S3 buckets** [13] can behave as web servers for *static* websites. But, because the back end is located behind an API Gateway, I could run the static client-side rendering ReactJS on a simple S3 bucket which accommodates as many requests as it receives. If none, the cost of the S3 bucket would be 0.

But ReactJS it is not a typical static website; everything is managed by the framework itself using one single html file as an entry point. This can be a problem with S3 because it will try to map any route /whatever to the some S3 file. To solve this problem and make it compatible with ReactJS, **AWS CloudFront** [14] was used in front of the S3 bucket to redirect all traffic to the index.html.

Finally, to be able to deploy all the previously mentioned architecture components I used 2 more services from AWS.

I used **AWS Code Pipeline** [15] for automating the deployment. I created hooks in the repositories of each service so whenever a new merge is performed to master branch, it will trigger the code pipeline. This process creates the application, runs tests or any other actions needed and finally pushes it to **AWS CloudFormation** [16], the latter will take care of the deployment to the different AWS components.

#### 1.3.1.4 Test coverage

The test coverage was done for both back end and front end. For the Lambdas in NodeJS I used **Jest** [17]. No automated test was done for Python due to the small size of the application (1 single lambda). Finally, Jest was also used for ReactJS.

# 1.4 Planning

During the planning (Figure 1) some risks were identified.

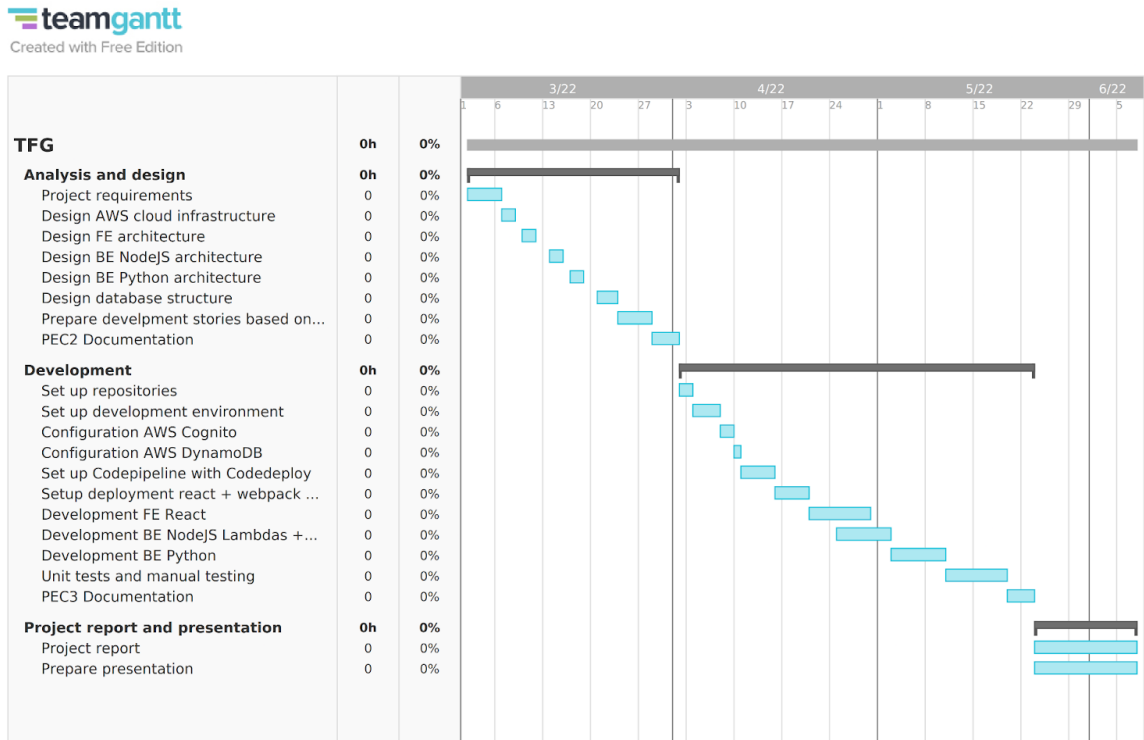


Figure 1: Gantt

Most risk were related to the configuration and setup of the entire AWS Cloud infrastructure:

- Unexpected issues with AWS deploying the applications, either due to some misconfiguration or lack of knowledge.
- Issues with pipelining ReactJS + Webpack + Babel, making an automated pipeline that executes webpack with Babel and can deliver the application to CloudFormation can be challenging to configure.

To address these issues, I had an AWS developers associate course at my disposal.

## 1.5 Project results summary

The result of the project is a set of microservices based on different technologies and frameworks, with a CI/CD and a simple, intuitive, and easy to use web interface for the users.

## 1.6 Following chapters summary

- Project requirements: Listing the functional and non-functional requirements of the project.
- Project designs: Listing the design prototypes for each page of the website.
- Development: Description of the most relevant parts of the development phase such as: Infrastructure, NodeJS, ReactJS and Python.
- Financial evaluations: small evaluation of the development and infrastructure costs.
- Conclusions: my conclusions about this project

## 2 Project requirements

### 2.1 Functional requirements

The functional requirements are split into two groups: Admins and Users. In the scope of this project the web application only contains admin users, normal users will not need to register to fill in any of the questionnaires. So, during this document any reference to 'users' would be under the premise of non-logged-in users.

- As an admin I want to be authenticated via Cognito.
- As an admin I want to create studies.
- As an admin I want to create several types of questions:
  - Dropdown question.
  - Checkbox question.
  - Input text question.
- As an admin I want to create a unique sharable link for each questionnaire.
- As an admin I want to download a .csv file with the results of any of my questionnaires.
- As a user I want to access a particular study via its unique id link.
  - Answer to all the questions.
  - Submit the responses.

More details about the functional requirements can be found in the next section.

### 2.1.1 Use cases

Overview	
Title	As an admin I want to be authenticated via Cognito.
Description	An admin must be able to authenticate in the website through Cognito
Actors	Admin
Initial status and preconditions	The admin user must exist in Cognito
Basic flow	
1: Admin click on 'login' button 2: Admin fills in the form with the username and password 3: Admin is logged in and sees the 'my studies' tab	
Alternative flow(s)	
2a: Admin fills in the wrong credentials 2a1: The system will show an error message	

Overview	
Title	As an admin I want to create studies.
Description	An admin must be able to create studies
Actors	Admin
Initial status and preconditions	The admin must be logged in
Basic flow	
1: Admin clicks on the 'my studies' tab 2: Admin clicks on the 'Create study' button 3: Admin fills in the form 4: A study is created and stored in the database	
Alternative flow(s)	

Overview	
Title	As an admin I want to create several types of questions - Dropdown
Description	An admin must be able to create dropdown questions
Actors	Admin
Initial status and preconditions	The admin must be logged in and have an existing study created
Basic flow	
1: Admin clicks on the 'my studies' tab 2: Admin clicks on the 'Questionnaire' button 3: Admin clicks on 'Add question' button 4: Admin fills in the question name and description 5: Admin selects the type 'Dropdown' 6: Admin adds a label and a value for the dropdown options 7: Admin clicks Submit 8: The question is created and stored in the database	
Alternative flow(s)	
6a: The admin makes a mistake and wants to delete the dropdown option 6a1: The admin clicks on the bullet point for that option 6a2: The icon will change to a red garbage icon 6a3: The admin clicks on the icon and the option is deleted	

Overview	
Title	As an admin I want to create several types of questions - Checkbox
Description	An admin must be able to create checkbox questions
Actors	Admin
Initial status and preconditions	The admin must be logged in and have an existing study created
Basic flow	
1: Admin clicks on the 'my studies' tab 2: Admin clicks on the 'Questionnaire' button 3: Admin clicks on 'Add question' button 4: Admin fills in the question name and description 5: Admin selects the type 'Checkbox' 6: Admin adds a label and a value for the checkbox options 7: Admin clicks Submit 8: The question is created and stored in the database	
Alternative flow(s)	
6a: The admin makes a mistake and wants to delete the checkbox option 6a1: The admin clicks on the bullet point for that option 6a2: The icon will change to a red garbage icon 6a3: The admin clicks on the icon and the option is deleted	



Overview	
Title	As an admin I want to create several types of questions - Text
Description	An admin must be able to create text questions
Actors	Admin
Initial status and preconditions	The admin must be logged in and have an existing study created
Basic flow	
1: Admin clicks on the 'my studies' tab 2: Admin clicks on the 'Questionnaire' button 3: Admin clicks on 'Add question' button 4: Admin fills in the question name and description 5: Admin selects the type 'Text' 6: Admin adds a label and a value for the text options 7: Admin clicks Submit 8: The question is created and stored in the database	
Alternative flow(s)	

Overview	
Title	As an admin I want to create a unique sharable link for each questionnaire
Description	An admin must be able to get unique sharable links for any study
Actors	Admin
Initial status and preconditions	The admin must be logged in and have an existing study created
Basic flow	
1: Admin clicks on the 'my studies' tab 2: Admin clicks on the 'Link' button 3: The system returns the new link in a popup 4: The new link is also stored in the database	
Alternative flow(s)	
3a: There is already an existing link for that study 3a1: The system will retrieve the existing link and return it	

Overview	
Title	As an admin I want to download a .csv file with the results of any of my questionnaires.
Description	An admin must be able to download a .csv file with all the answers to a study
Actors	Admin
Initial status and preconditions	The admin must be logged in, have an existing study
Basic flow	
1: Admin clicks on the 'my studies' tab 2: Admin clicks on the 'CSV' button 3: The system returns a CSV file downloaded to the admin desktop	
Alternative flow(s)	
3a: There is no data yet for that study 3a1: The csv file will be empty	

Overview	
Title	As a user I want to access a particular study via its unique id link.
Description	Any user must be able to access any study
Actors	User
Initial status and preconditions	The user must have a valid link
Basic flow	
1: User access the link in their browser 2: The system loads all the questions related with that study 3: The user answers all the questions and clicks submit 4: The system shows a thank you message page 5: The answers are stored in the database	
Alternative flow(s)	
3a: The user forgets to answer one or more questions 3a: The system will show an error message and highlight all the forgotten questions	

## 2.2 Non-functional requirements

- The system must be based on a cloud AWS architecture.
- The system must be auto scalable.
- The system performance must be good.
- The code must follow good practises:
  - ReactJS good practices.
  - Design patterns.
- The web design must be responsive for mobile devices and tablets (only the questionnaire response part, the administration will be only supported on desktop).
- There must be an automated CI/CD for each aspect of the architecture.
- The website must be easy to use without prior knowledge needed.
- There must be a clear development flow.

# 3 Project designs

## 3.1 AWS architecture

As can be seen in Figure 2, I used an S3 bucket to contain the ReactJS. The S3 bucket will receive the requests and serve the application through its own web server.

To secure the user data and have a login system in place, the ReactJS application in S3 will use Amazon Cognito to authenticate and store the users. At the same time, the API Gateways will authenticate against Cognito as well to secure the API calls made from ReactJS.

To serve the ReactJS application, I have added an Amazon CloudFront service in front of the S3 bucket. This optimizes the speed the website is delivered, and offers important redirects needed to have ReactJS working on an S3 bucket.

Next to the S3, I have the Amazon API Gateways with REST APIs for which one contains the NodeJS Lambdas for the customer facing APIs and the other one contains the Python Lambda for the csv report.

Finally, both APIs are connected to a DynamoDB database to store and retrieve all the necessary data.

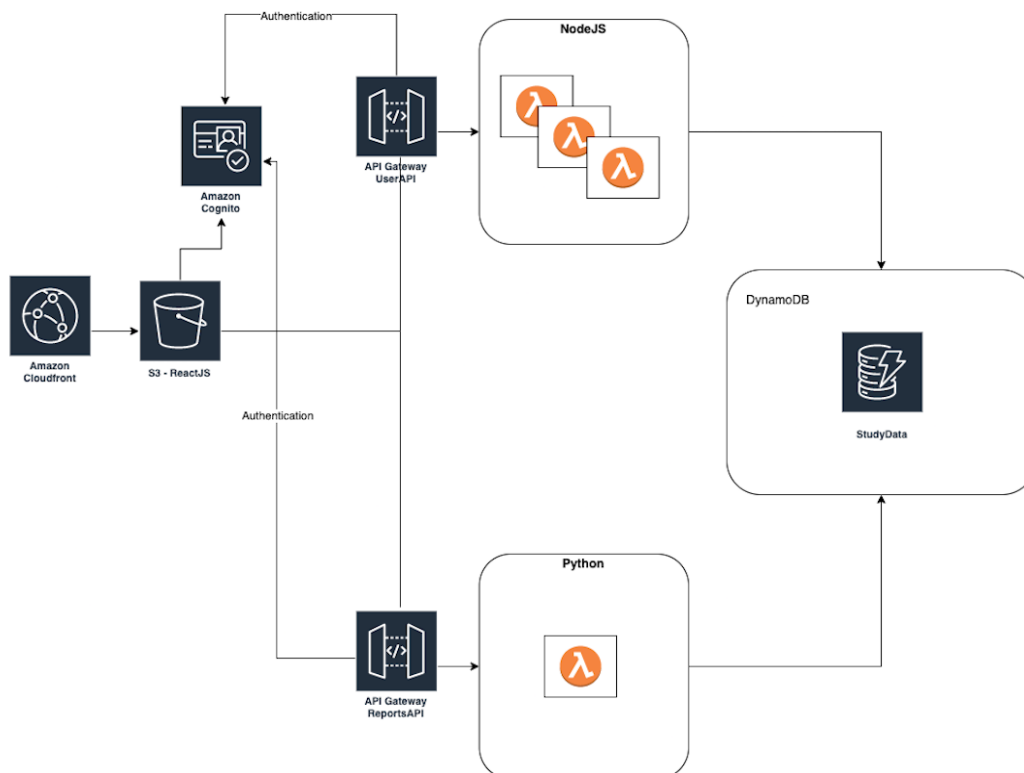


Figure 2: AWS Architecture

## 3.2 Data modelling

Amazon DynamoDB is a fully managed NoSQL database service that provides fast and predictable performance with seamless scalability. Each item in a DynamoDB table can be simple (a partition key) or composite (partition key and sort key).

Per design, I used a composite key, PK for the primary key and SK for the Sort Key.

I will describe the several types of 'Entities' and how they are represented in the DynamoDB table.

AWS recommends a single table design [18] and I use their PK and SK to query them.

Entity	PK	SK
Questionnaire	USERID#userid	QUESTIONNAIRE#questionnaireId
Question	QUESTIONNAIRE#questionnaireId	QUESTION#questionid
Answer	QUESTIONNAIRE#questionnaireId	ANSWER#answerid#QUESTION#questionid
QuestionTypes	QUESTIONTYPE	ID
Public Link	LINK#questionnaireId	userId

## 3.3 FE design

### 3.3.1 Home page

The home page (Figure 3) will contain some logo or image and the login button

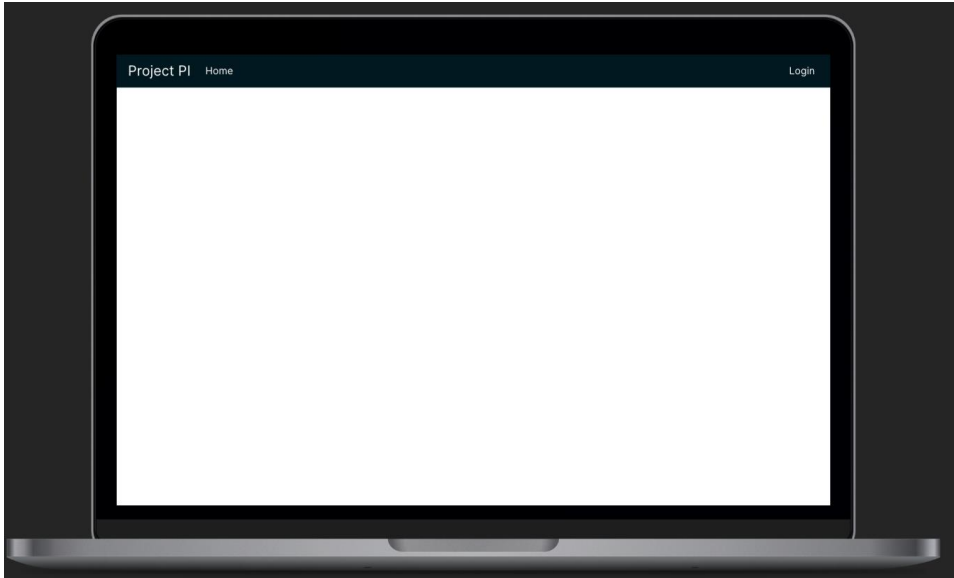


Figure 3: Home page design

### 3.3.2 Login

The login form (Figure 4) will require the users to enter their email and password. Since this will be a tool for research and medical centres, registration is not possible through the website.

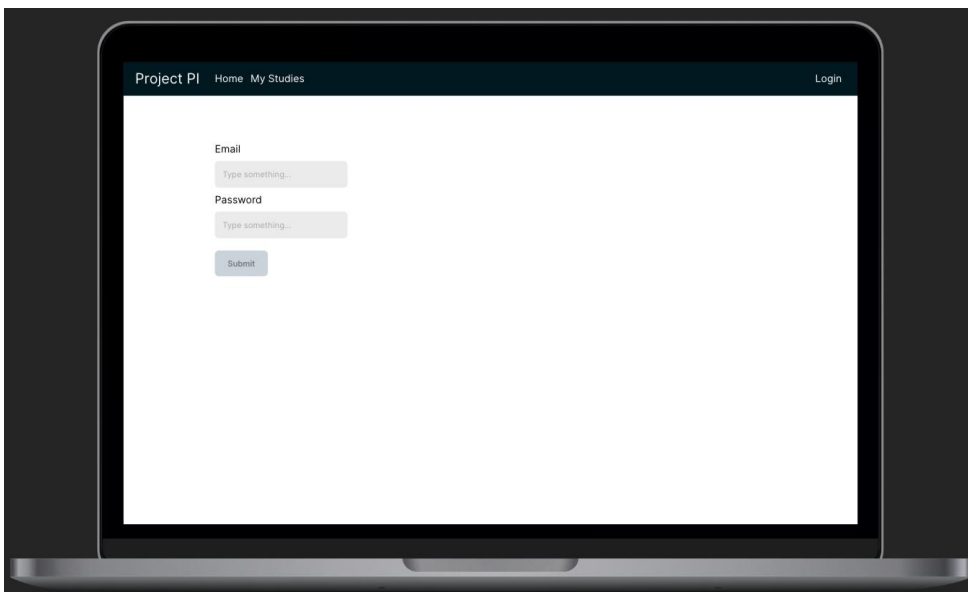


Figure 4: Login design

### 3.3.3 My Studies tab

On the 'My Studies' tab (Figure 5) there will be a list of all the created studies, with an option to access the questionnaire and create questions, to get a public access link, download the csv report or to delete it.

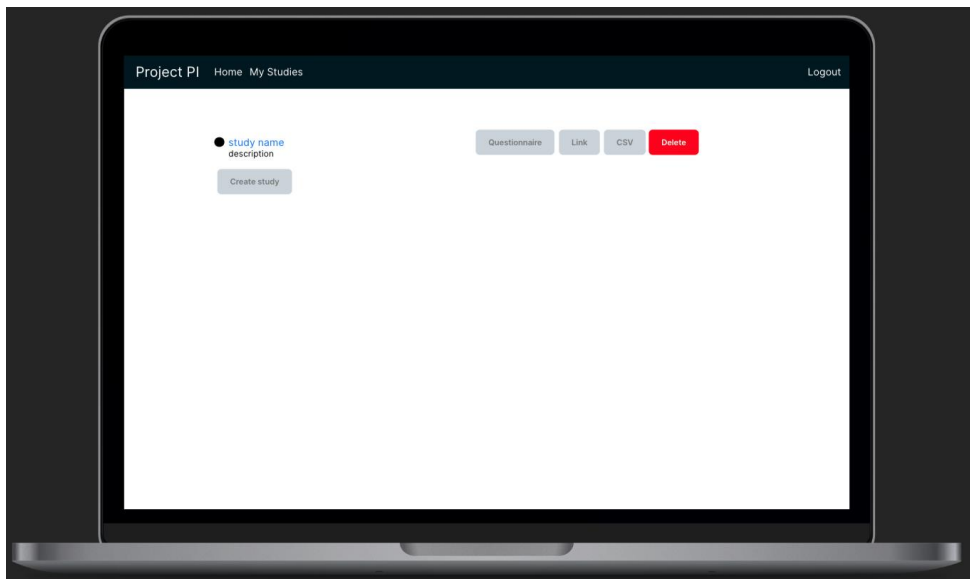


Figure 5: My studies tab design

### 3.3.4 Create/Modify Study

The creation of the study (Figure 6) is quite simple; it just requires a study name and a description.

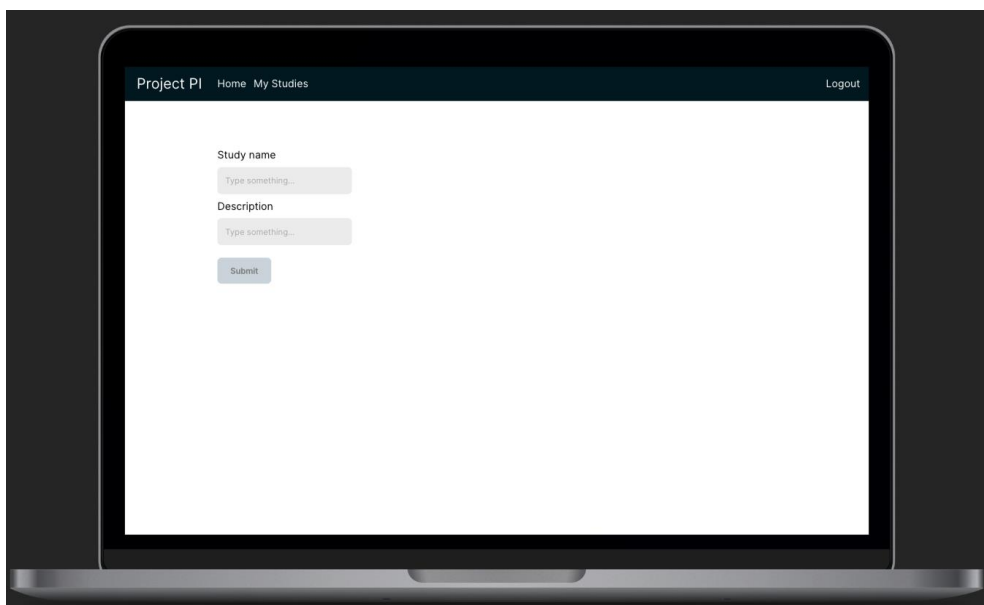


Figure 6: Create/modify study design

### 3.3.5 Questionnaire view

On the questionnaire view (Figure 7) the users will be able to see the list of all the questions that are contained inside the questionnaire. Along with the option to edit, delete and add new questions.

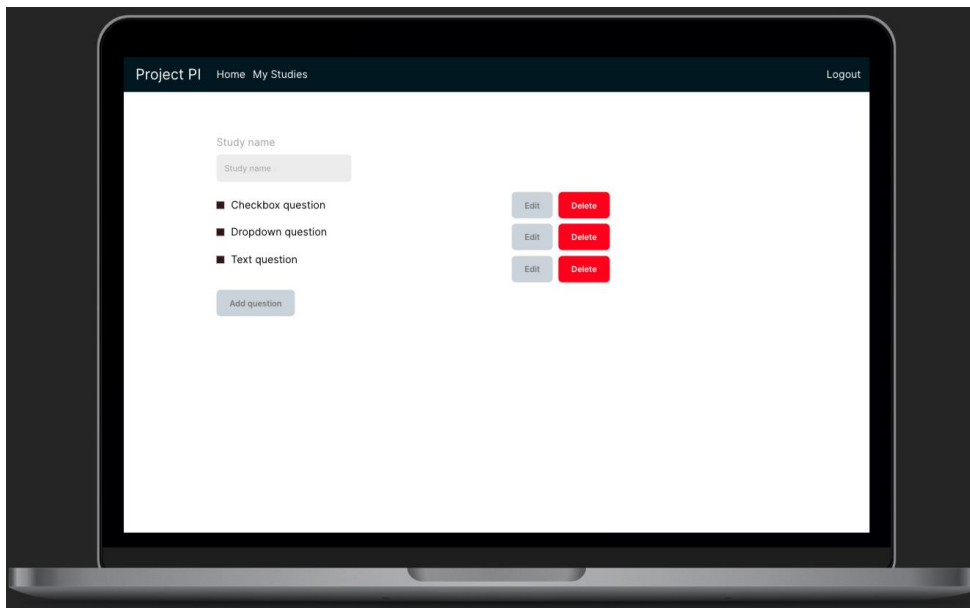


Figure 7: Questionnaire view design

### 3.3.6 Add question

There will be 3 types of questions: text, dropdown and checkbox. Each type of question will show a distinct set of options below 'Type'. For example, dropdown will allow the user to enter several types of options and their values. (Figure 8).

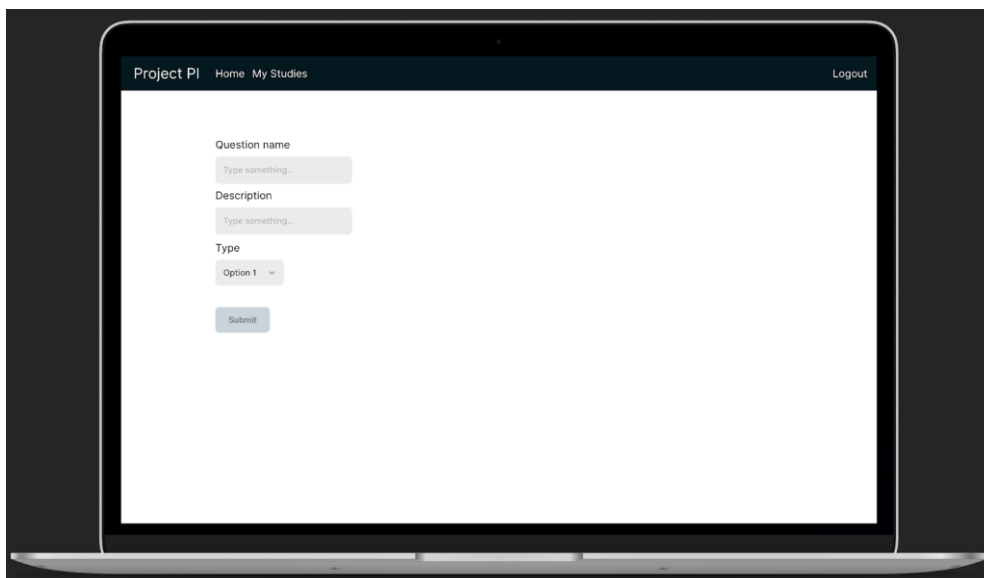


Figure 8: Add question design



### 3.3.7 Get access link

The access link button (Figure 9) will generate a unique link for that specific study which will be publicly accessible. If the link was previously created already, it will show the same one.

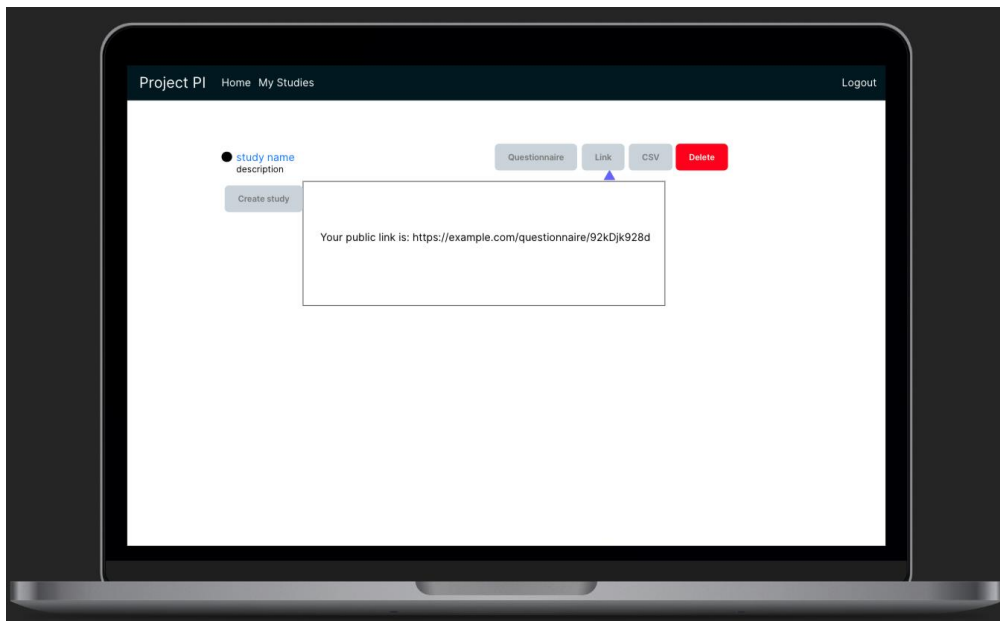


Figure 9: Get access link design

### 3.3.8 Filling in questionnaire data

All the questions will be loaded vertically (Figure 10) and there will be validation to ensure all questions are answered.

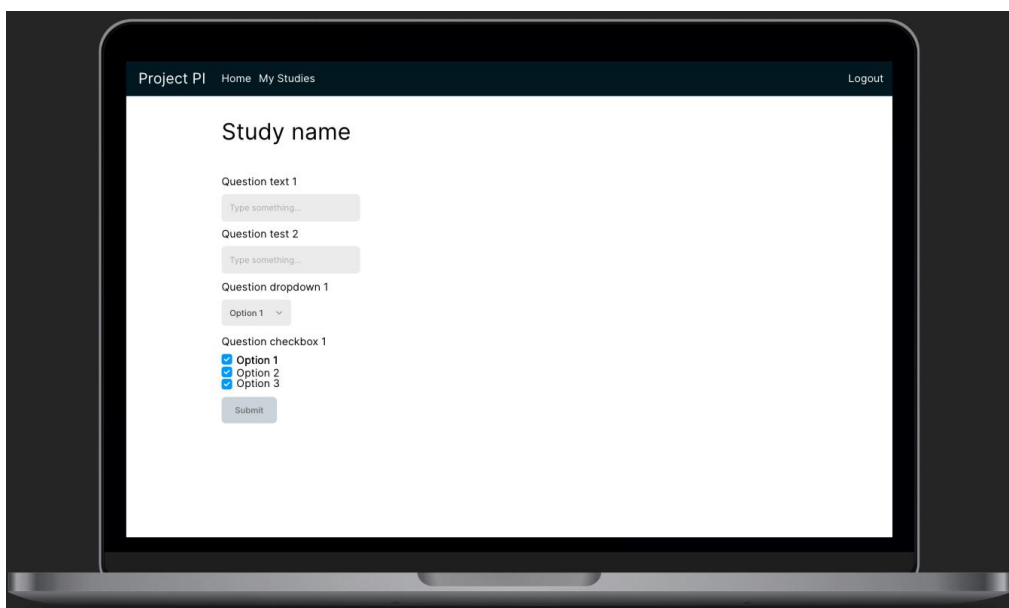


Figure 10: Filling in questionnaire data design

## 3.4 FE Architecture

React works on a component level basis (Figure 11), each component has one responsibility and it can contain child components. The set of immutable values are passed to the components rendered as properties in its HTML tags. So the child component itself cannot modify the properties but a call-back is used instead to communicate with the parent component.

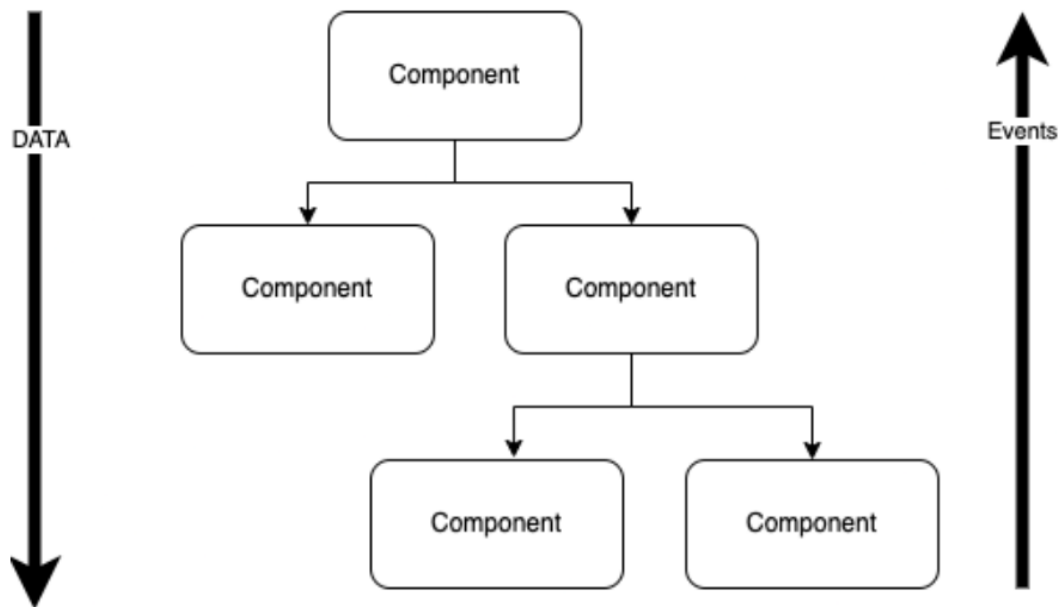


Figure 11: FE architecture

React JS does not define a way to structure your components files. There are some common ways listed on the ReactJS website, but it is up to the developer.

For this reason, I went with the following structure:

- components
  - This folder contains all the individual atomic components, such as the AWS Cognito login, questions (create, edit questions... etc.) and studies related components.
- pages
  - This folder contains the components related to the web pages itself; for example, the login page.
- services
  - Any service that talks with the BE will be stored here
- tests
  - Unit testing

## 3.5 BE Lambdas architecture

For the Lambdas I used NodeJS in this case because I used CloudFormation to deploy them all, they all live in the same repository, with each folder containing its set of CRUD [19] operations for each API.

There is also a middleware folder, that contains common middleware used in all the lambdas for handling CORS [20].

- Lambdas
  - answer
  - public
  - question
  - questionnaire
  - types
- middlewares
  - cors
- repositories
- tests
- use\_cases
- template.yml
- buildspec.yml

**Answer** folder that contains a single Lambda operation that can be accessed without login to post the answers to a questionnaire.

**Public** folder that contains Lambdas associated with retrieving questionnaires and questions for the shared link, this also does not require authentication.

**Question** folder that contains necessary CRUD operations and an operation to get all the questions related to a questionnaire.

**Questionnaire** folder that contains basic CRUD operations for questionnaires.

**Type** folder that contains a simple get operation to get all the distinct types of questions (text, dropdown, checkbox, etc...).

**Middlewares** folder that contains the middleware responsible to return proper response headers for cors available and used by all the lambdas.

**Repositories** folder that contains a repository per entity with its database operations.

**Use\_cases** folder that contains the use cases with the business logic.

Finally, cloudformation.yml and buildspec.yml are the files in charge of building the CI/CD.

## 3.6 BE Python architecture

For the Lambda responsible for generating the CSV [21], I used Python. Due to the simplicity of the task, this is one single Lambda with all the logic needed inside.

The repository was also set up as the previous ones, with code pipeline and automatic deployments, so it can be scaled up with more lambdas and more diverse types of reports.

## 3.7 Jira stories

As can be seen in Figure 12, the basic stories for the development and the documentation phases of the project are written. During the following phases, each story was refined before getting picked up, and split into several smaller stories if necessary.

Issue ID	Issue Title	Category	Key
TFG-24	Set up repositories	Development	TFG-24
TFG-7	Set up development environment	Development	TFG-7
TFG-19	Configuration AWS Cognito	Development	TFG-19
TFG-21	Configuration AWS DynamoDB	Development	TFG-21
TFG-22	Set up Codepipeline with CodeDeploy	Development	TFG-22
TFG-27	Setup deployment react + webpack + babel	Development	TFG-27
TFG-26	Development FE React	Development	TFG-26
TFG-23	Development BE NodeJS Lambdas + API Gateway	Development	TFG-23
TFG-25	Development BE Python	Development	TFG-25
TFG-28	Unit tests and manual testing	Development	TFG-28
TFG-12	Project report	Documentation and pr...	TFG-12
TFG-13	Prepare presentation	Documentation and pr...	TFG-13

**Figure 12: Jira stories**

Once there were enough stories for a week's worth of work, I created a sprint so that I could work on them and repeat the process until the project was finished.

I used a 'light' version of Scrum. Because I am the only one developing, I just used some strategies from the framework to make my life easier and more efficient.

## 4 Development

### 4.1 Repositories

For this project I decided to have three different repositories in Github:

- 1- Back end repository, **project-pi-be-less**: for the back end serverless application, meaning NodeJS Lambdas and its CloudFormation files.
- 2- Back end repository, **project-pi-be-reports**: for the back end Lambda Python application and its CloudFormation files.
- 3- Front end repository **project-pi-fe**: for the front end React application and its CloudFormation files.

### 4.2 AWS Cognito configuration

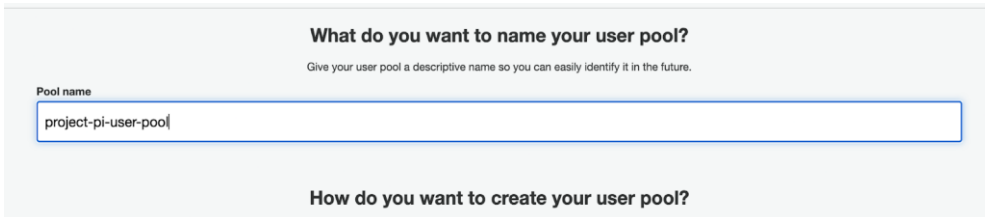
The entire project revolves around a microservice architecture, for which the main challenge to be addressed is the authentication and authorization.

To guarantee the security of the data, each microservice needs to be able to secure its requests and responses. However, because its services are independent and do not contain any user data, we need to have a way to sign and verify the requests.

Cognito offers this functionality; once a user is logged in, it returns an authentication token which can be validated later on by any other service that has access to it. This makes it possible to sign and send the request with the token from ReactJS to any of the API Gateways, and then API Gateway will check for the validity of the token.

In this step, I configured the AWS Cognito so it could be used by the rest of the services.

First, I had to create a user pool in AWS (Figure 13) to provide sign-up and sign-in options for the web application users.



The screenshot shows a configuration page for an AWS Cognito user pool. The main heading is "What do you want to name your user pool?". Below this, there is a sub-heading "Give your user pool a descriptive name so you can easily identify it in the future." and a text input field labeled "Pool name" containing the text "project-pi-user-pool". At the bottom of the visible section, there is another heading "How do you want to create your user pool?".

Figure 13: Cognito configuration 1

When it came to the configuration, I went with the default settings (Figure 14). The required attribute is the email with a strong password policy.

Pool name project-pi-user-pool

Required attributes email

Alias attributes Choose alias attributes...

Username attributes Choose username attributes...

Enable case insensitivity? Yes

Custom attributes Choose custom attributes...

Minimum password length 8

Password policy uppercase letters, lowercase letters, special characters, numbers

User sign ups allowed? Users can sign themselves up

FROM email address Default

Email Delivery through Amazon SES Yes

MFA Enable MFA...

Verifications Email

Tags Choose tags for your user pool

App clients Add app client...

Triggers Add triggers...

Create pool

Figure 14: Cognito configuration 2

Once the pool was created, I got a pool id (Figure 15). This pool id was used later in the component responsible for the authentication in ReactJS.

Your user pool was created successfully.

Pool Id eu-west-1\_H34HhJcPm

Pool ARN arn:aws:cognito-idp:eu-west-1:258885766453:userpool/eu-west-1\_H34HhJcPm

Figure 15: Final Cognito configuration

## 4.3 Setting up DynamoDB

DynamoDB can also be specified through CloudFormation, but there is one issue with that: the possibility of having the stack updated and the DynamoDB table being deleted and recreated.

For that reason, I created the table manually through the AWS web console (Figure 16).

The screenshot shows the AWS DynamoDB 'Create table' console page. The breadcrumb navigation at the top reads 'DynamoDB > Tables > Create table'. The main heading is 'Create table'. Below this is a 'Table details' section with an 'Info' link. A descriptive text states: 'DynamoDB is a schemaless database that requires only a table name and a primary key when you create the table.' The 'Table name' field is labeled 'StudyData' and has a note: 'This will be used to identify your table.' Below the field is a note: 'Between 3 and 255 characters, containing only letters, numbers, underscores (\_), hyphens (-), and periods (.).' The 'Partition key' section is labeled 'pk' and has a note: 'The partition key is part of the table's primary key. It is a hash value that is used to retrieve items from your table and allocate data across hosts for scalability and availability.' The field contains 'pk' and the type is 'String'. Below the field is a note: '1 to 255 characters and case sensitive.' The 'Sort key - optional' section is labeled 'sk' and has a note: 'You can use a sort key as the second part of a table's primary key. The sort key allows you to sort or search among all items sharing the same partition key.' The field contains 'sk' and the type is 'String'. Below the field is a note: '1 to 255 characters and case sensitive.'

Figure 16: DynamoDB table creation

## 4.4 NodeJS Lambda

### 4.4.1 Development environment

Lambdas are run in AWS infrastructure, but there is a command line available called sam-cli for developing and testing Lambdas locally

Sam-cli requires brew and docker. I installed them first.

#### 4.4.1.1 Installing NodeJS

The installation of NodeJS can be done through <https://nodejs.org/>.

#### 4.4.1.2 Installing brew

```
/bin/bash -c "$(curl -fsSL  
https://raw.githubusercontent.com/Homebrew/install/master/install.sh)"
```

#### 4.4.1.3 Installing Docker

In my case I already had Docker installed in my machine, but it can be downloaded from <https://www.docker.com/>

#### 4.4.1.4 Installing sam-cli

```
brew tap aws/tap  
brew install aws-sam-cli
```

#### 4.4.1.5 Installing PAW

PAW is a full-featured HTTP client that lets me test and describe the APIs calls I built. It is an especially useful tool for testing the API endpoints and to create documentation about them.

It can be installed through <https://paw.cloud/>



#### 4.4.1.6 Creating the first Lambda

AWS Lambdas are triggered by AWS when an event occurs, those events can range from database events to manual triggers to, in this case, an API call.

AWS will always look for a *handler* function in our lambdas to trigger, and it will always pass 2 objects: **event** and **context**.

The object event contains the data that is passed to the function upon execution, such as http method, headers, query params... etc.

The object context contains methods and properties that provide information about the invocation, function, and execution environment.

So it was time to create a basic HelloWorld Lambda (Figure 17) to correctly test the development environment, and to have a starting point to set up the infrastructure of the API Gateway and their Lambdas.

```
const AWS = require("aws-sdk");

const docClient = new AWS.DynamoDB.DocumentClient({apiVersion: '2012-08-10'});
AWS.config.update({region: 'eu-west-1'});

const handlerFunction = async (event, context) => {
  return {
    statusCode: 200,
    body: "HelloWorld"
  }
}

exports.handler = async (event, context) => {
  return await handlerFunction(event, context);
};
```

Figure 17: First Lambda code

Sam-cli checks for a .yml file where the CloudFormation is contained to know how to run a specific Lambda.

A CloudFormation file is needed to set up a CI/CD in CloudFormation, it is part of the IaC [22] (Infrastructure as code). AWS uses these templates as blueprints for building the AWS resources.

In the following example (Figure 18) I am building a serverless API, with one Lambda attached to the /helloworld endpoint.

This Lambda will use nodejs12 runtime, with a default memory size of 128mb and a max timeout of 30 seconds.

```
AWSTemplateFormatVersion: '2010-09-09'
Transform: 'AWS::Serverless-2016-10-31'
Description: Project pi be less
Resources:
  MyApi:
    Type: AWS::Serverless::Api
    Properties:
      Name: UserAPI
      StageName: Prod
      Description: API for FE user requests
      Cors:
        AllowMethods: "'POST, GET, DELETE, OPTIONS'"
        AllowHeaders: "'*'"
        AllowOrigin: "'*'"

##### Hello world lambda #####
HelloWorldLambda:
  Type: 'AWS::Serverless::Function'
  Properties:
    Handler: lambdas/helloWorld.handler
    Runtime: nodejs12.x
    CodeUri: ./
    Description: 'Hello world lambda'
    MemorySize: 128
    Timeout: 30
    Events:
      DeleteQuestionLambdaEvent:
        Type: Api
        Properties:
          Path: /helloWorld
          Method: get
          RestApiId: !Ref MyApi
    Environment:
      Variables:
        REGION: "eu-west-1"
```

Figure 18: First Lambda CloudFormation definition

Once I made sure it worked locally (Figure 19) by running the `sam-cli` command, it was time to set up the deployment so it would be created in AWS.

```
% sam local invoke HelloWorldLambda
Invoking lambdas/helloWorld.handler (nodejs12.x)
Skip pulling image and use local one: public.ecr.aws/sam/emulation-nodejs12.x:rapid-1.43.0-x86_64.

Mounting /Users/daniel.perez/dani/project-pi-be-less as /var/task:ro,delegated inside runtime container
START RequestId: 5b4b6db0-a7e8-480b-850b-94878c44fa85 Version: $LATEST
END RequestId: 5b4b6db0-a7e8-480b-850b-94878c44fa85
REPORT RequestId: 5b4b6db0-a7e8-480b-850b-94878c44fa85 Init Duration: 0.29 ms Duration: 480.23 ms Billed Duration: 481 ms Memory Size: 128 MB Max Memory Used: 128 MB
{"statusCode":200,"body":"HelloWorld"}

```

**Figure 19: First Lambda testing code**

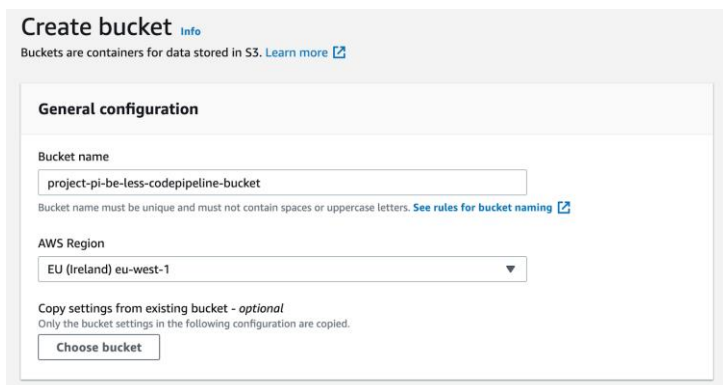
## 4.4.2 Setting up CodePipeline

This project uses CodePipeline to create a CI/CD for all the repositories I previously mentioned.

CodePipeline uses 2 `.yml` files: one for the building phase where I specified what to install and which commands to run during the building phase, as well as post-build scripts, for example for running tests. And finally, one last `yml` file where I described the resources I needed to be created in AWS during the deployment phase (Figure 18).

The first step was to create the CodePipeline in the AWS console.

CodePipeline uses S3 buckets to build the application so I had to create an S3 bucket first (Figure 20). As AWS Region I chose EU (Ireland).



**Create bucket** [Info](#)  
Buckets are containers for data stored in S3. [Learn more](#)

**General configuration**

Bucket name  
  
Bucket name must be unique and must not contain spaces or uppercase letters. [See rules for bucket naming](#)

AWS Region

Copy settings from existing bucket - *optional*  
Only the bucket settings in the following configuration are copied.

**Figure 20: CodePipeline configuration 1**

Then I could create a new pipeline (Figure 21). A service role is required for each pipeline. The service role contains policies with permissions to perform actions in resources, I allowed the pipeline to create its own service role.

**Choose pipeline settings** Info

**Pipeline settings**

**Pipeline name**  
Enter the pipeline name. You cannot edit the pipeline name after it is created.  
project-pi-be-less-pipeline  
No more than 100 characters

**Service role**

**New service role**  
Create a service role in your account

**Existing service role**  
Choose an existing service role from your account

**Role name**  
project-pi-be-less-pipeline  
Type your service role name

**Allow AWS CodePipeline to create a service role so it can be used with this new pipeline**

**Figure 21: CodePipeline configuration 2**

On the following step I had to choose which source provider I wanted. Because my code is stored in GitHub, I choose it as a source provider (Figure 22). A new connection to GitHub is required, authorizing AWS to connect to our Github accounts.

Once the connection was created, I was able to select the repository that contained the code and selected which branch would trigger the pipeline every time a new change gets merged into.

**Source**

**Source provider**  
This is where you stored your input artifacts for your pipeline. Choose the provider and then provide the connection details.  
GitHub (Version 2)

**New GitHub version 2 (app-based) action**  
To add a GitHub version 2 action in CodePipeline, you create a connection, which uses GitHub Apps to access your repository. Use the options below to choose an existing connection or create a new one. [Learn more](#)

**Connection**  
Choose an existing connection that you have already configured, or create a new one and then return to this task.  
arn:aws:codestar-connections:eu-west-1:258885766453:connection/45b7d5: X or **Connect to GitHub**

**Ready to connect**  
Your GitHub connection is ready for use.

**Repository name**  
Choose a repository in your GitHub account.  
shake93/project-pi-be-less  
<account>/<repository-name>

**Branch name**  
Choose a branch of the repository.  
master

**Change detection options**

**Start the pipeline on source code change**  
Automatically starts your pipeline when a change occurs in the source code. If turned off, your pipeline only runs if you start it manually or on a schedule.

**Output artifact format**  
Choose the output artifact format.

**CodePipeline default**  
AWS CodePipeline uses the default zip format for artifacts in the pipeline. Does not include git metadata about the repository.

**Full clone**  
AWS CodePipeline passes metadata about the repository that allows subsequent actions to do a full git clone. Only supported for AWS CodeBuild actions.

**Figure 22: CodePipeline configuration 3**

For the next step, I had to create a build project (Figure 23). This is the step in CodePipeline where we will do the build of our application so it can be deployed.

### Create build project

**Project configuration**

Project name

project-pi-be-less-build

A project name must be 2 to 255 characters. It can include the letters A-Z and a-z, the numbers 0-9, and the special characters - and \_.

**Figure 23: CodePipeline configuration 4**

To build the NodeJS lambdas I choose to run it in a standard AWS Linux environment (Figure 24).

### Environment

Environment image

Managed image  
Use an image managed by AWS CodeBuild

Custom image  
Specify a Docker image

Operating system

Amazon Linux 2

**i** The programming language runtimes are now included in the standard image of Ubuntu 18.04, which is recommended for new CodeBuild projects created in the console. See [Docker Images Provided by CodeBuild for details](#).

Runtime(s)

Standard

Image

aws/codebuild/amazonlinux2-x86\_64-standard:3.0

Image version

Always use the latest image for this runtime version

Environment type

Linux

Privileged

**Figure 24: CodePipeline configuration 5**

Next, in the buildspec section I used the default configuration (Figure 25). By default, CodePipeline will look for a 'buildspec.yml' file to run the build commands.

**Buildspec**

Build specifications

Use a buildspec file  
Store build commands in a YAML-formatted buildspec file

Insert build commands  
Store build commands as build project configuration

**Buildspec name - optional**  
By default, CodeBuild looks for a file named buildspec.yml in the source code root directory. If your buildspec file uses a different name or location, enter its path from the source root here (for example, buildspec-two.yml or configuration/buildspec.yml).

Figure 25: CodePipeline configuration 6

At this point I could continue to the deployment stage (Figure 26).

**Add build stage** [Info](#)

**Build - optional**

**Build provider**  
This is the tool of your build project. Provide build artifact details like operating system, build spec file, and output file names.

AWS CodeBuild

**Region**

Europe (Ireland)

**Project name**  
Choose a build project that you have already created in the AWS CodeBuild console. Or create a build project in the AWS CodeBuild console and then return to this task.

project-pi-be-less-build or [Create project](#)

Successfully created project-pi-be-less-build in CodeBuild.

**Environment variables - optional**  
Choose the key, value, and type for your CodeBuild environment variables. In the value field, you can reference variables generated by CodePipeline. [Learn more](#)

[Add environment variable](#)

**Build type**

Single build  
Triggers a single build.

Batch build  
Triggers multiple builds as a single execution.

Cancel Previous Skip build stage Next

Figure 26: CodePipeline configuration 7

Next, it was time to set up the deployment. As a deployment provider I chose CloudFormation (Figure 27) and the option to create or update the stack so any latest changes will trigger an infrastructure update.

For the BuildArtifact file name, I used outputtemplate.yml which will be the output .yml file from the build phase.

Next, some required capabilities (IAM and AUTO\_EXPAND) for the deployment phase, and a CloudformationServiceRole with permissions for deployment.

**Deploy - optional**

**Deploy provider**  
Choose how you deploy to instances. Choose the provider, and then provide the configuration details for that provider.

AWS CloudFormation

**Region**

Europe (Ireland)

**Action mode**  
When you update an existing stack, the update is permanent. When you use a change set, the result provides a diff of the updated stack and the original stack before you choose to execute the change.

Create or update a stack

**Stack name**  
If you are updating an existing stack, choose the stack name.

project-pi-be-less-stack

**Template**  
Specify the template you uploaded to your source location.

Artifact name	File name	Template file path
BuildArtifact	outputtemplate.yml	BuildArtifact::outputter

**Template configuration - optional**  
Specify the configuration file you uploaded to your source location.

Use configuration file

Artifact name	File name	Template configuration file path

**Capabilities - optional**  
Specify whether you want to allow AWS CloudFormation to create IAM resources on your behalf.

CAPABILITY\_IAM CAPABILITY\_AUTO\_EXPAND

**Role name**

arn:aws:iam::258885766453:role/CloudformationServiceRole

Figure 27: CodePipeline configuration 8

Before I could move to the next phase, it was important to give the right permissions (Figure 28) to the CodeBuild role I created in the first step so it could have access to the S3 bucket I created. Attaching the AmazonS3FullAccess policy was enough.

[IAM](#) > [Roles](#) > [project-pi-be-less-service-role](#)

## project-pi-be-less-service-role

### Summary

Creation date

April 02, 2022, 11:47 (UTC+02:00)

Last activity

None

**Permissions**

Trust relationships

Tags

Access Advisor

Revoke sessions

### Permissions policies (2)

You can attach up to 10 managed policies.

Policy name [↗](#)

[CodeBuildBasePolicy-project-pi-be-less-build-eu-west-1](#)

[AmazonS3FullAccess](#)

**Figure 28: CodePipeline configuration 9**

By this point I had CodePipeline configured, but I still had to set up the buildspec file.



### 4.4.3 Setting up buildspec.yml

Next step was to prepare the buildspec.yml (Figure 29) file so it could build the application.

Here I specified the NodeJS version, and the commands needed to run during the build phase.

- Npm install: to install all the dependencies needed.
- AWS CloudFormation package: this creates the output .yml file to be used during the deployment.

This step will make sure that the application can be build. For NodeJS this means it can install the npm packages correctly.

During the setup of the development environment, I had created a template.yml file with the HelloWorld Lambda example, that is the CloudFormation file that will be used during the buildspec to generate the outputtemplate.yml that will be used by CloudFormation to create the resources.

```
version: 0.2
phases:
  install:
    runtime-versions:
      nodejs: 12
    commands:
      - npm install
      - aws cloudformation package --template-file template.yml --s3-bucket project-pi-be-less-codepipeline-bucket --output-template-file outputtemplate.yml
artifacts:
  type: zip
  files:
    - template.yml
    - outputtemplate.yml
```

Figure 29: Buildspec configuration 1

Once I pushed these changes to master, the pipeline triggered automatically, built and deployed the stack (Figure 30).

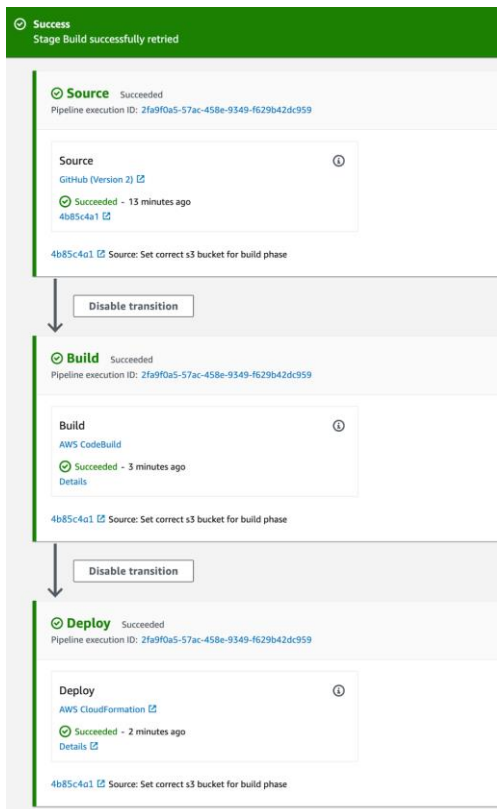


Figure 30: Buildspec build stage

At that point I could see the newly created API with its Lambda in API Gateway (Figure 31).

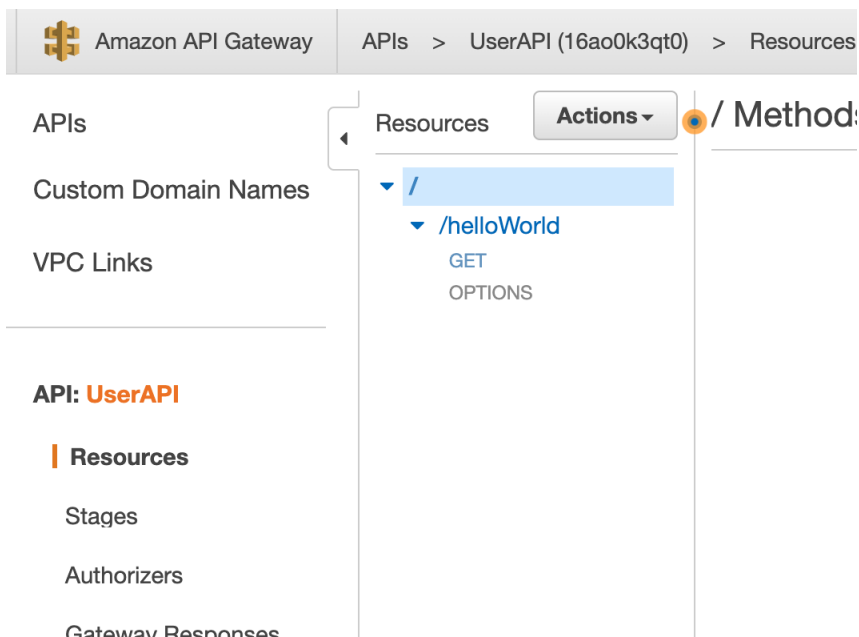


Figure 31: API Gateway

and could test the response from the /helloWorld endpoint (Figure 32).

Request: /helloWorld

Status: 200

Latency: 157 ms

Response Body

```
HelloWorld
```

Response Headers

```
{ "X-Amzn-Trace-Id" : "Root=1-62482442-15e659925c328027753a132b;Sampled=0" }
```

Loas

**Figure 32: Lambda test**

Finally, I was ready to start the back end development.

## 4.4.4 Development NodeJS

### 4.4.4.1 Architecture

The architecture of the application (Figure 33) is as follows:

Each Lambda is grouped by its entity: **question**, **public**, **questionnaire**, **types** and **answer**.

**Public** and **answer** are publicly accessible, so non-logged-in users can answer questionnaires, the rest are protected via Cognito.

Each set of lambdas uses 'use cases' to perform actions, each use case contains only the business logic, and all the database operations are in **repositories**.

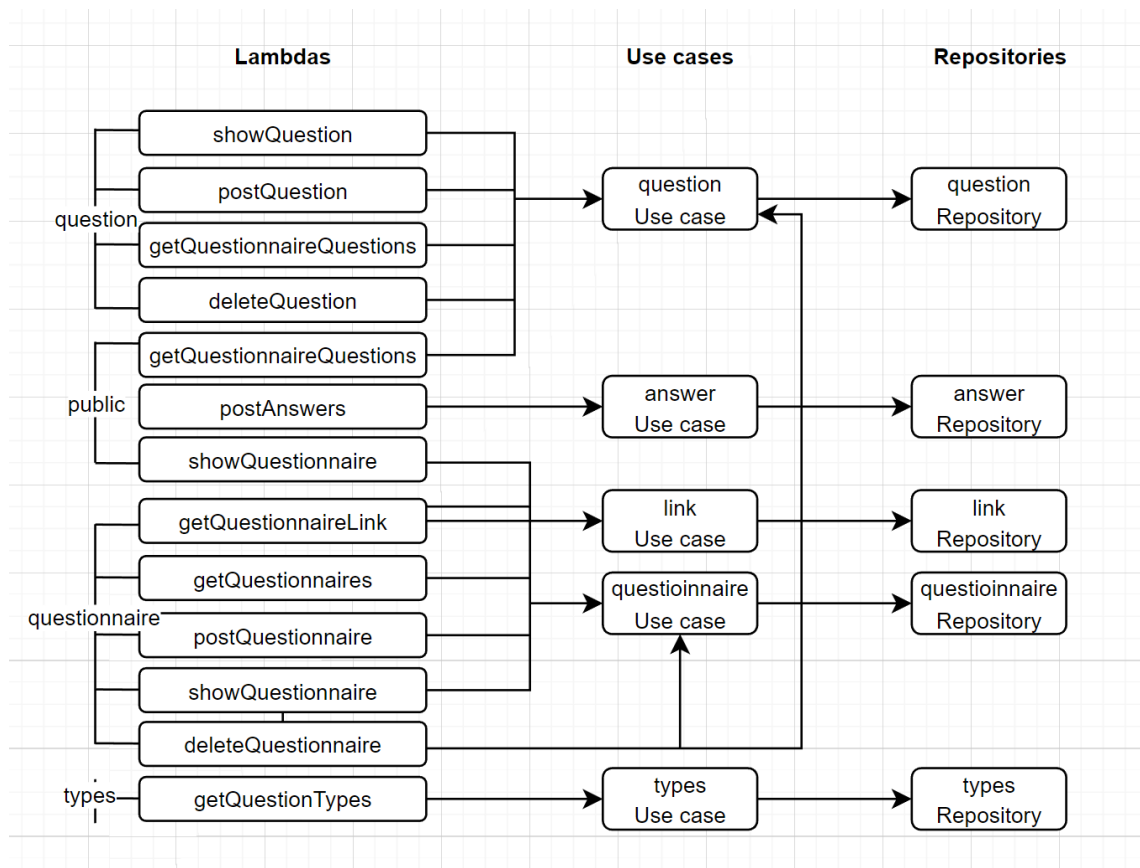


Figure 33: NodeJS architecture

#### 4.4.4.2 Code base

For each entity there is a folder with all the Lambdas needed for the CRUD operations (Figure 34). These Lambdas are responsible for getting the necessary information from the request (ids and user data), then calling the use case that it is needed and finally returning the data.

```

  ✓ lambdas
    ✓ answer
      JS postAnswers.js
    ✓ public
      JS getQuestionnaireQuestions.js
      JS showQuestionnaire.js
    ✓ question
      JS deleteQuestion.js
      JS getQuestionnaireQuestions.js
      JS postQuestion.js
      JS showQuestion.js
    ✓ questionnaire
      JS deleteQuestionnaire.js
      JS getQuestionnaireLink.js
      JS getQuestionnaires.js
      JS postQuestionnaire.js
      JS showQuestionnaire.js
    ✓ types
      JS getQuestionTypes.js

```

Figure 34: NodeJS code base

Example of postAnswers Lambda (Figure 35).

```
const cors = require('../../middlewares/cors');
const uuid = require('uuid');
const { saveAnswers } = require('../../use_cases/answer')

const handlerFunction = async (event, context) => {
  const questionnaireId = event.pathParameters.id;
  const attemptId       = uuid.v4();
  const parsedBody      = JSON.parse(event.body);

  try {
    const data = await saveAnswers(questionnaireId, parsedBody, attemptId);
    return { statusCode: 200, body: JSON.stringify(data) };
  } catch (error) {
    return {
      statusCode: 400,
      body: `Could not query: ${error.stack}`
    };
  }
};

exports.handler = async (event, context) => {
  return cors(await handlerFunction(event, context));
};
```

Figure 35: Code base

As can be seen in the previous image (Figure 35), Lambdas do not contain business logic; that is the responsibility of the use cases (Figure 36).

```
▼ use_cases
  JS answer.js
  JS cognito.js
  JS link.js
  JS question.js
  JS questionnaire.js
  JS questionTypes.js
```

Figure 36: NodeJS code base use cases

A clear example is the deleteQuestionnaire action (Figure 37) from the questionnaire use case. It takes care of deleting the questionnaire, using its repository and calling the questions use case to delete all questions.

```
const deleteQuestionnaire = async (questionnaireId, userId) => {
  try {
    await questionnaireRepository.del(questionnaireId, userId);
    return await questionUseCase.deleteAllQuestions(questionnaireId);
  } catch (error) {
    throw error;
  }
}
```

Figure 37: NodeJS code base delete questionnaire

This takes me to the repositories folder (Figure 38). The repositories contain all the operations to the database for each entity. Repositories should only be called from use cases, as I did in this application.

```
▼ repositories
  JS answer.js
  JS link.js
  JS question.js
  JS questionnaire.js
  JS type.js
```

Figure 38: NodeJS code base repositories

Example of a repository (Figure 39) for creating an answer.

```
const AWS = require("aws-sdk");
const docClient = new AWS.DynamoDB.DocumentClient({ apiVersion: '2012-08-10' });
AWS.config.update({ region: 'eu-west-1' });

const create = async(questionnaireId, questionId, value, attemptId) => {
  const params = {
    TableName: 'StudyData',
    Item: {
      pk: 'QUESTIONNAIRE#'+questionnaireId,
      sk: 'ANSWER#'+attemptId+'#'+questionId,
      questionId: questionId,
      value: value
    },
    ReturnValues: 'ALL_OLD',
  };

  try {
    return await docClient.put(params).promise();
  } catch (error) {
    throw error;
  }
}

module.exports = { create }
```

Figure 39: NodeJS Code base use case example



### 4.4.4.3 Setting up API Gateway

I have shown in previous steps how I set up the `template.yml` that describes the AWS resources. During the development I added all the extra Lambdas following the same pattern as the HelloWorld Lambda.

But there were a few issues to take care of after that. First, configuring CORS in API Gateway and enabling the authorization of each request against Cognito.

#### 4.4.4.3.1 CORS

To configure CORS in API Gateway I just had to add a CORS configuration parameter in the `template.yml` file (Figure 40). The allowed methods are POST, GET, DELETE and OPTIONS. Allowing all headers and for the origin, I allowed any for the purpose of developing locally against the public API.

In normal circumstances, this would only happen in testing environments, and production would only allow the ReactJS site as origin.

```
AWSTemplateFormatVersion: '2010-09-09'
Transform: 'AWS::Serverless-2016-10-31'
Description: Project pi be lesss
Resources:
  MyApi:
    Type: AWS::Serverless::Api
    Properties:
      Name: UserAPI
      Mode: overwrite
      StageName: Prod
      Description: API for FE user requests here
    Auth:
      DefaultAuthorizer: MyCognitoAuthorizer
      Authorizers:
        MyCognitoAuthorizer:
          UserPoolArn: arn:aws:cognito-idp:eu-west-1:258885766453:userpool/eu-west-1_UFKL1H1wi
      AddDefaultAuthorizerToCorsPreflight: false
    Cors:
      AllowMethods: "'POST, GET, DELETE, OPTIONS'"
      AllowHeaders: "'*'"
      AllowOrigin: "'*'"
```

Figure 40: CORS `template.yml`

#### 4.4.4.3.2 Cognito authorization

I wanted to have most of the requests to be verified against Cognito. All those requests that comes from a logged-in user will require authentication. However, users do not need to be logged in to fill in the answers of a study; for these non-logged-in users, no authorization is required.

Once again, some configuration was needed in the template.yml file.

First, I had to enable the user pool as an Authorizer (Figure 41).

```
AWSTemplateFormatVersion: '2010-09-09'
Transform: 'AWS::Serverless-2016-10-31'
Description: Project pi be lesss
Resources:
  MyApi:
    Type: AWS::Serverless::Api
    Properties:
      Name: UserAPI
      Mode: overwrite
      StageName: Prod
      Description: API for FF user requests here
      Auth:
        DefaultAuthorizer: MyCognitoAuthorizer
        Authorizers:
          MyCognitoAuthorizer:
            UserPoolArn: arn:aws:cognito-idp:eu-west-1:258885766453:userpool/eu-west-1_UFKL1H1wi
      AddDefaultAuthorizerToCorsPreflight: false
    Cors:
      AllowMethods: "'POST, GET, DELETE, OPTIONS'"
      AllowHeaders: "'*'"
      AllowOrigin: "'*'"
```

Figure 41: API Gateway authorizer

This would enable the authorization in ALL endpoints and actions (POST, GET, DELETE, OPTIONS).

The main issue is to have the authorization also enabled for OPTIONS. Pre-flights CORS OPTIONS calls are not performed with authorization headers so I added a small configuration line (Figure 42) to disable it for those.

```
AWS::Serverless::Api
Type: AWS::Serverless::Api
Properties:
  Name: UserAPI
  Mode: overwrite
  StageName: Prod
  Description: API for FE user requests here
  Auth:
    DefaultAuthorizer: MyCognitoAuthorizer
    Authorizers:
      MyCognitoAuthorizer:
        UserPoolArn: arn:aws:cognito-idp:eu-west-1:258885766453:userpool/eu-west-1_UFKL1H1wi
        AddDefaultAuthorizerToCorsPreflight: false
  Cors:
    AllowMethods: "'POST, GET, DELETE, OPTIONS'"
    AllowHeaders: "'*'"
    AllowOrigin: "'*'"
```

Figure 42: CORS pre-flight

Finally, I had to disable the authorization for the public endpoints (Figure 43).

```
postAnswersLambda:
  Type: 'AWS::Serverless::Function'
  Properties:
    Handler: lambdas/answer/postAnswers.handler
    Runtime: nodejs12.x
    CodeUri: ./
    Description: 'Lambda function'
    MemorySize: 128
    Timeout: 30
    Role: 'arn:aws:iam::258885766453:role/LambdaDynamoDBRole'
    Events:
      postAnswersLambdaEvent:
        Type: Api
        Properties:
          Path: /public/questionnaire/{id}/answers
          Auth:
            Authorizer: NONE
          Method: post
          RestApiId: !Ref MyApi
        Environment:
          Variables:
            REGION: "eu-west-1"
```

Figure 43: Disable authorization

#### 4.4.4.4 Testing

Finally, I created tests for all Lambdas and uses cases (Figure 44). The tests are under the 'tests' folder and they can be run locally.

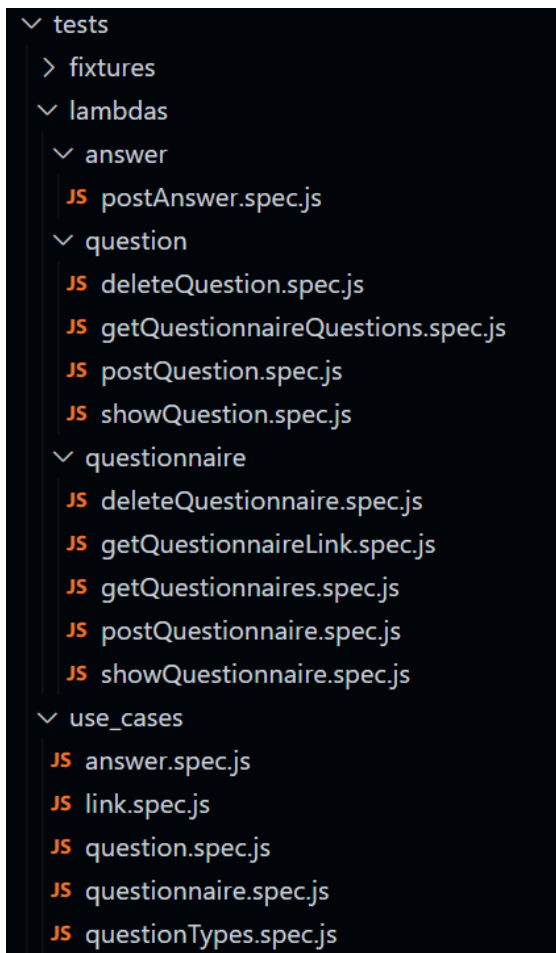


Figure 44: NodeJS testing

The most important requirement for tests is to run them every time something gets pushed to master branch in GitHub and before it gets deployed (Figure 45). For that I added an extra step in CodePipeline, in the buildspec.yml file I added a post\_build step that will run the npm test.

If any test fails during the building phase it will stop and not deploy.

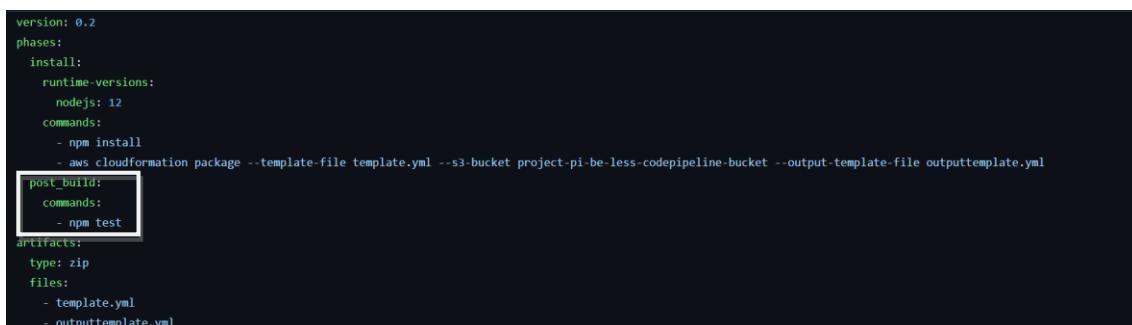


Figure 45: NodeJS setting up automated testing during build

This is how it looks during the build phase:

```
> project-pi-be-less@1.0.0 test /codebuild/output/src009975283/src
> jest

PASS tests/use_cases/questionnaire.spec.js
PASS tests/use_cases/question.spec.js
PASS tests/lambda/questionnaire/deleteQuestionnaire.spec.js
PASS tests/lambda/questionnaire/showQuestionnaire.spec.js
PASS tests/lambda/questionnaire/postQuestionnaire.spec.js
PASS tests/lambda/questionnaire/getQuestionnaires.spec.js
PASS tests/lambda/questionnaire/getQuestionnaireLink.spec.js
PASS tests/use_cases/link.spec.js
PASS tests/use_cases/answer.spec.js
PASS tests/lambda/question/getQuestionnaireQuestions.spec.js
PASS tests/lambda/question/postQuestion.spec.js
PASS tests/lambda/question/showQuestion.spec.js
PASS tests/lambda/question/deleteQuestion.spec.js
PASS tests/use_cases/questionTypes.spec.js
PASS tests/lambda/answer/postAnswer.spec.js

Test Suites: 15 passed, 15 total
Tests: 22 passed, 22 total
Snapshots: 0 total
Time: 2.672 s
Ran all test suites.

[Container] 2022/05/19 12:50:24 Phase complete: POST_BUILD State: SUCCEEDED
```

Figure 46: NodeJS CodePipeline test results

To be able to test these operations in the AWS environment I used “PAW” (Figure 47) and API Client, which can also be used as documentation for the API.

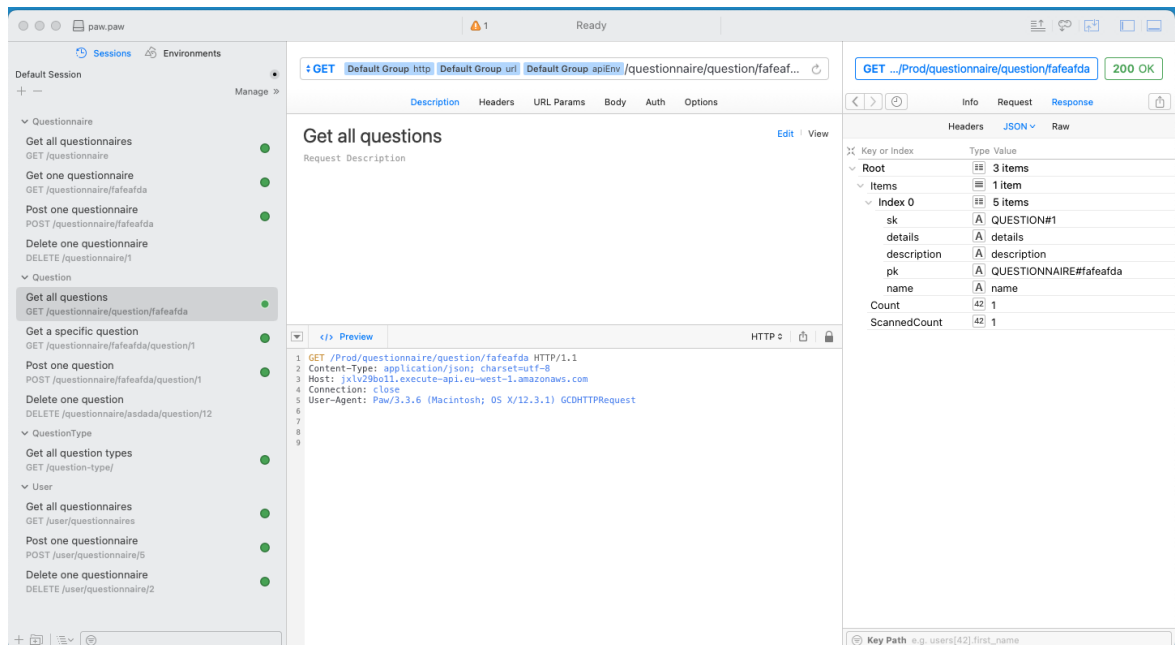


Figure 47: NodeJS PAW

## 4.5 React JS

### 4.5.1 Development environment

React can be installed through npx a package runner from npm (Node Packet Manager). Because I already had npm installed from the previous steps, I just had to run `npm i -g npx`

and then create the react app using npx with:

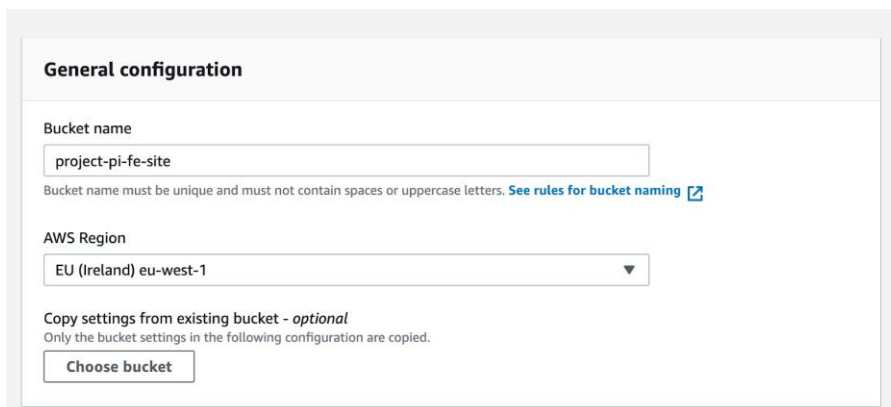
```
npx create-react-app project-pi-fe
```

ReactJS offers a development environment when running the `npm start` command.

### 4.5.2 Setting up an S3 bucket for web access

As mentioned in the AWS architecture, the ReactJS application is served from an S3 Bucket. This bucket needed to have a specific configuration to allow public access and to have the option to serve a static website enabled.

First, I created an S3 bucket that contained the static React site (Figure 48).



The image shows a screenshot of the AWS S3 bucket configuration page, specifically the 'General configuration' section. The 'Bucket name' field is filled with 'project-pi-fe-site'. Below it, a note states: 'Bucket name must be unique and must not contain spaces or uppercase letters. See rules for bucket naming'. The 'AWS Region' dropdown menu is set to 'EU (Ireland) eu-west-1'. At the bottom, there is a section for 'Copy settings from existing bucket - optional' with a 'Choose bucket' button.

**Figure 48: CodePipeline configuration 1**

I allowed public access (Figure 49).

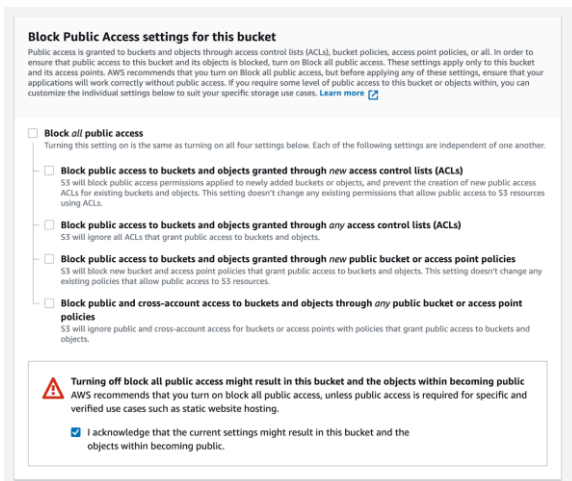


Figure 49: CodePipeline configuration 2

Once the bucket was created, I went to the properties of the bucket where at the bottom there was an option to enable the bucket for hosting static websites (Figure 50).

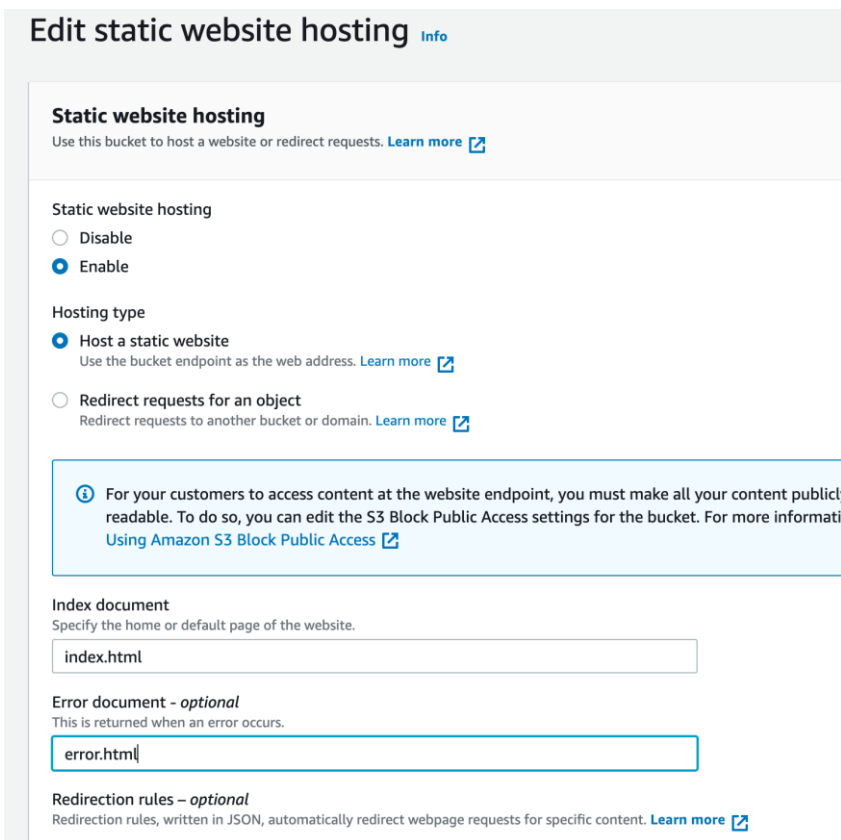


Figure 50: CodePipeline configuration 3

It was also important to attach an appropriate bucket policy to allow get actions on the bucket publicly (Figure 51).

**Bucket policy**  
The bucket policy, written in JSON, provides access to the objects stored in the bucket. Bucket policies don't apply to

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "PublicReadGetObject",
      "Effect": "Allow",
      "Principal": "*",
      "Action": "s3:GetObject",
      "Resource": "arn:aws:s3:::project-pi-fe-site/*"
    }
  ]
}
```

**Figure 51: CodePipeline configuration 4**



## 4.5.3 Setting up CodePipeline

To deploy the ReactJS application to the publicly accessible S3 bucket I had to create a new pipeline (Figure 52), the main steps are also explained in the NodeJS CodePipeline and differ only slightly, so I will only show those different steps.

**Source**

**Source provider**  
This is where you stored your input artifacts for your pipeline. Choose the provider and then provide the connection details.

GitHub (Version 2)

**New GitHub version 2 (app-based) action**  
To add a GitHub version 2 action in CodePipeline, you create a connection, which uses GitHub Apps to access your repository. Use the options below to choose an existing connection or create a new one. [Learn more](#)

**Connection**  
Choose an existing connection that you have already configured, or create a new one and then return to this task.

arn:aws:codestar-connections:eu-west-1:258885766453:connection/45b7d5! X or [Connect to GitHub](#)

**Ready to connect**  
Your GitHub connection is ready for use.

**Repository name**  
Choose a repository in your GitHub account.

shake93/project-pi-fe X  
<account>/<repository-name>

**Branch name**  
Choose a branch of the repository.

master X

**Change detection options**

**Start the pipeline on source code change**  
Automatically starts your pipeline when a change occurs in the source code. If turned off, your pipeline only runs if you start it manually or on a schedule.

**Output artifact format**  
Choose the output artifact format.

**CodePipeline default**  
AWS CodePipeline uses the default zip format for artifacts in the pipeline. Does not include git metadata about the repository.

**Full clone**  
AWS CodePipeline passes metadata about the repository that allows subsequent actions to do a full git clone. Only supported for AWS CodeBuild actions.

Cancel Previous **Next**

Figure 52: CodePipeline configuration 5

The main difference was the deployment stage step where I choose the S3 bucket as the deployment provider (Figure 53).

**Add deploy stage** [Info](#)

**Deploy - optional**

Deploy provider  
Choose how you deploy to instances. Choose the provider, and then provide the configuration details for that provider.

Amazon S3

Region  
Europe (Ireland)

Bucket  
project-pi-fe-site

Deployment path - optional

Extract file before deploy  
The deployed artifact will be unzipped before deployment.

▶ Additional configuration

Cancel Previous Skip deploy stage Next

Figure 53: CodePipeline configuration 6

At this point, the pipeline ran for the first time but failed during the build phase. I later added the `buildspec.yml` file (Figure 54) and pushed it to master so it triggered again.

```
version: 0.2

phases:
  pre_build:
    commands:
      - npm install
  build:
    commands:
      - npm run build
  post_build:
    commands:
      - npm test

artifacts:
  files:
    - '**/*'
  discard-paths: no
  base-directory: public
```

Figure 54: CodePipeline configuration 7

The deployment was then successful (Figure 55).

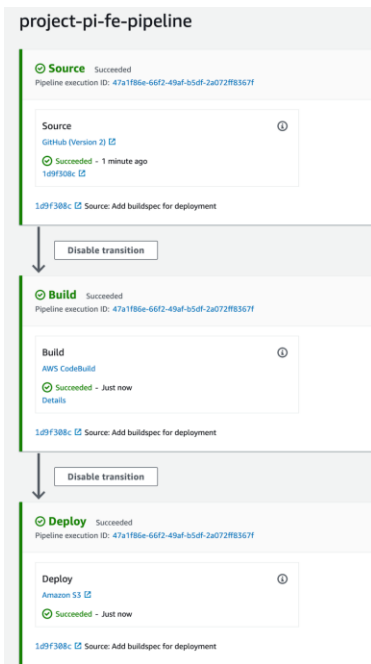


Figure 55: CodePipeline configuration 8

Once the application was deployed, I needed to make sure the ReactJS website was accessible. I went to the S3 bucket and below the S3 properties I could get the static URL (Figure 56).

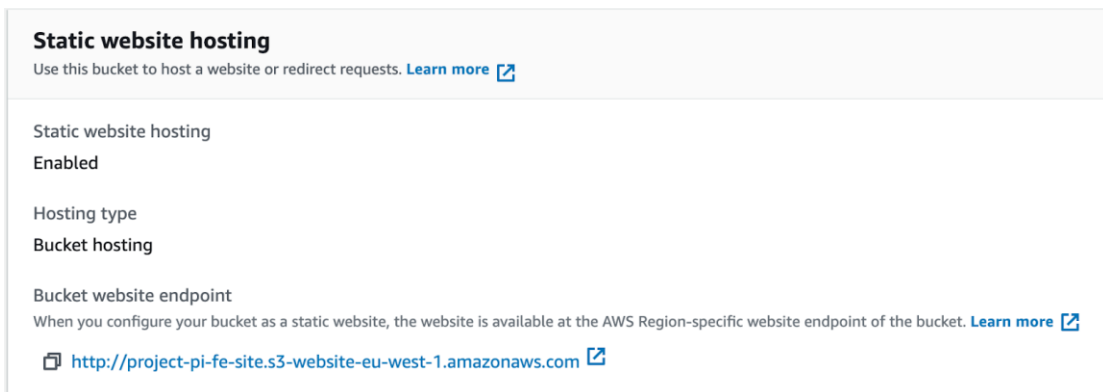


Figure 56: S3 public url

The website was publicly accessible (Figure 57).

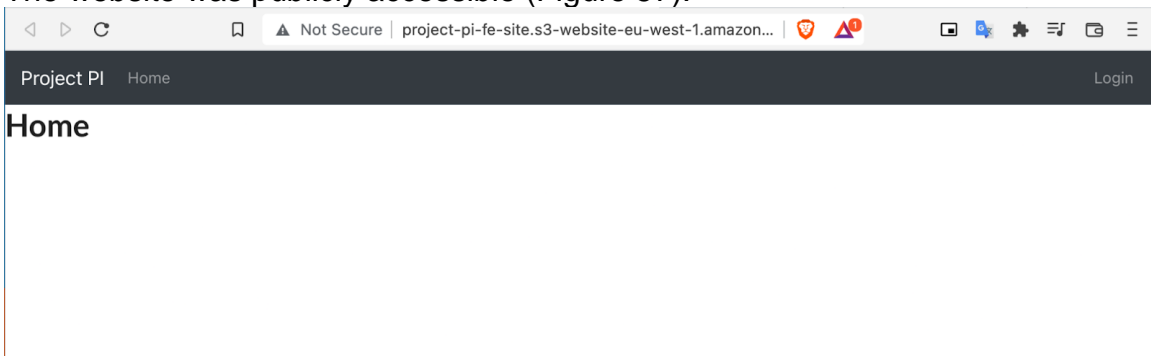


Figure 57: ReactJS running on public site

## 4.5.4 AWS CloudFront

At that point I had the website already running on the S3 bucket, it was time to add CloudFront with the default settings (Figure 58).

**Create distribution**

**Origin**

**Origin domain**  
Choose an AWS origin, or enter your origin's domain name.

**Origin path - optional** | [Info](#)  
Enter a URL path to append to the origin domain name for origin requests.

**Name**  
Enter a name for this origin.

**S3 bucket access** | [Info](#)  
Use a CloudFront origin access identity (OAI) to access the S3 bucket.

Don't use OAI (bucket must allow public access)  
 Yes use OAI (bucket can restrict access to only CloudFront)

**Add custom header - optional**  
CloudFront includes this header in all requests that it sends to your origin.

**Enable Origin Shield** | [Info](#)  
Origin Shield is an additional caching layer that can help reduce the load on your origin and help protect its availability.

No  
 Yes

▶ **Additional settings**

**Figure 58: CloudFront configuration 1**

One last crucial step was to configure some redirects because the S3 bucket together with react routing has a small issue:

When accessing any other URL different than the home page, I would get a 404 error.

S3 static web hosting serves static HTML files. React serves only one HTML file `index.html`, and React Router oversees serving the content based on what we query in the URL. So, technically the 404 ERROR CODE is correct since the only HTML file is `index.html`.

The solution was simple, I just had to add some redirects in CloudFront (Figure 59), to redirect any 404, 403 and 502 errors to the index.html, so ReactJS can take care of it.

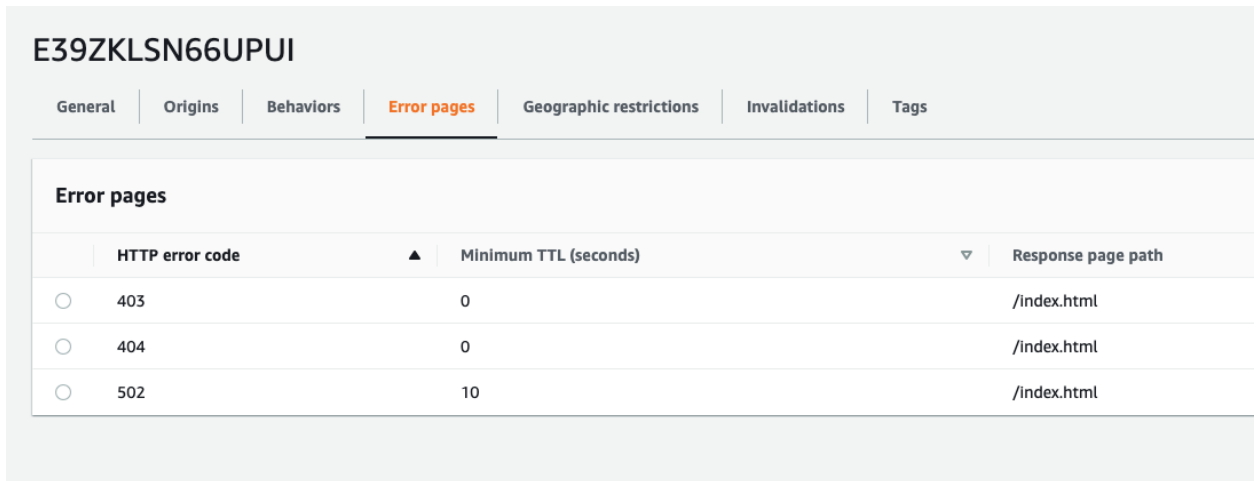


Figure 59: CloudFront configuration 2

And get the CloudFront public site url (Figure 60).

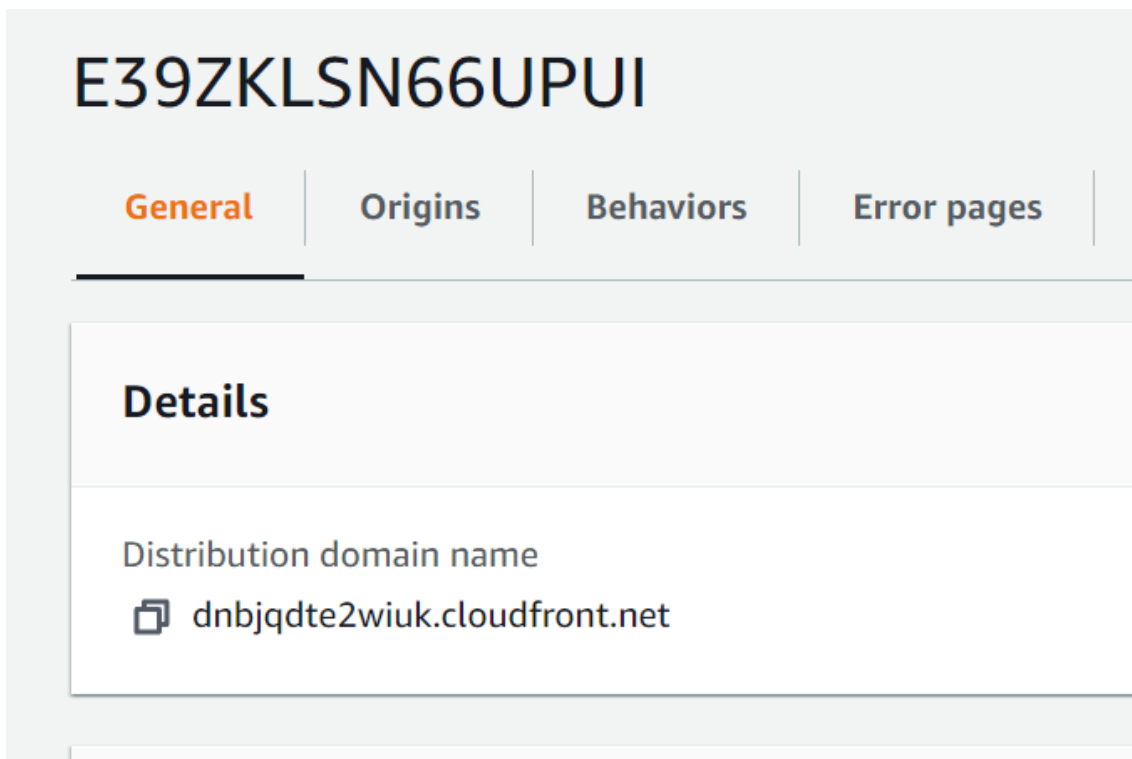


Figure 60: Cloudfront public url

## 4.5.5 Setting up deployment with webpack and Babel

Now that the ReactJS application could be deployed to the S3 bucket and be able to access it publicly through CloudFront, it was time to set up the compression of the JS and CSS with webpack and to convert the JS code into a backwards compatible one using Babel.

### 4.5.5.1 Webpack

For this step, I had to create a webpack.config.js file with the appropriate configuration for a React app along with a development and production mode (Figure 61).

The entry of the application is the index.js file from React, and it needs to output into the public folder the bundle.js with all the compressed js code.

```
module.exports = function(_env, argv) {
  const isProduction = argv.mode === "production";
  const isDevelopment = !isProduction;

  return {
    devtool: isDevelopment && "cheap-module-source-map",
    entry: "./src/index.js",
    output: {
      path: path.join(__dirname, "public"),
      filename: "bundle.js",
      publicPath: "/"
    },
  },
}
```

Figure 61: Deployment webpack Babel 1

There are more options configured in this file, such as CSS and node\_modules folder compression and optimizations for production.

### 4.5.5.2 Babel

For Babel I just had to add some presets (Figure 62) for React and some additional required plugins.

```
module.exports = {
  presets: [
    [
      "@babel/preset-env",
      {
        modules: false
      }
    ],
    "@babel/preset-react"
  ],
  plugins: [
    "@babel/plugin-transform-runtime",
    "@babel/plugin-syntax-dynamic-import",
    "@babel/plugin-proposal-class-properties"
  ],
  env: {
    production: {
      only: ["src"],
      plugins: [
        [
          "transform-react-remove-prop-types",
          {
            removeImport: true
          }
        ],
        "@babel/plugin-transform-react-inline-elements",
        "@babel/plugin-transform-react-constant-elements"
      ]
    }
  }
};
```

Figure 62: Deployment webpack Babel 2

### 4.5.5.3 Deployment

Finally, for the deployment I had to build webpack during the build phase in CodePipeline so it could produce the bundle.js and deploy. The buildspect.yml file (Figure 54) was already running `npm run build` during the build phase. That command is specified in the package.json from npm in the root of my project, and by default it runs the build command from npm.

To execute the webpack build I just had to adjust the command in the package.json (Figure 63).

```
{
  "name": "webpack-react",
  "version": "1.0.0",
  "main": "index.js",
  "author": "Michael Pontus <m.pontus@gmail.com>",
  "license": "MIT",
  "scripts": {
    "build": "webpack --mode production",
    "start": "webpack-dev-server --mode development",
    "test": "jest"
  },
}
```

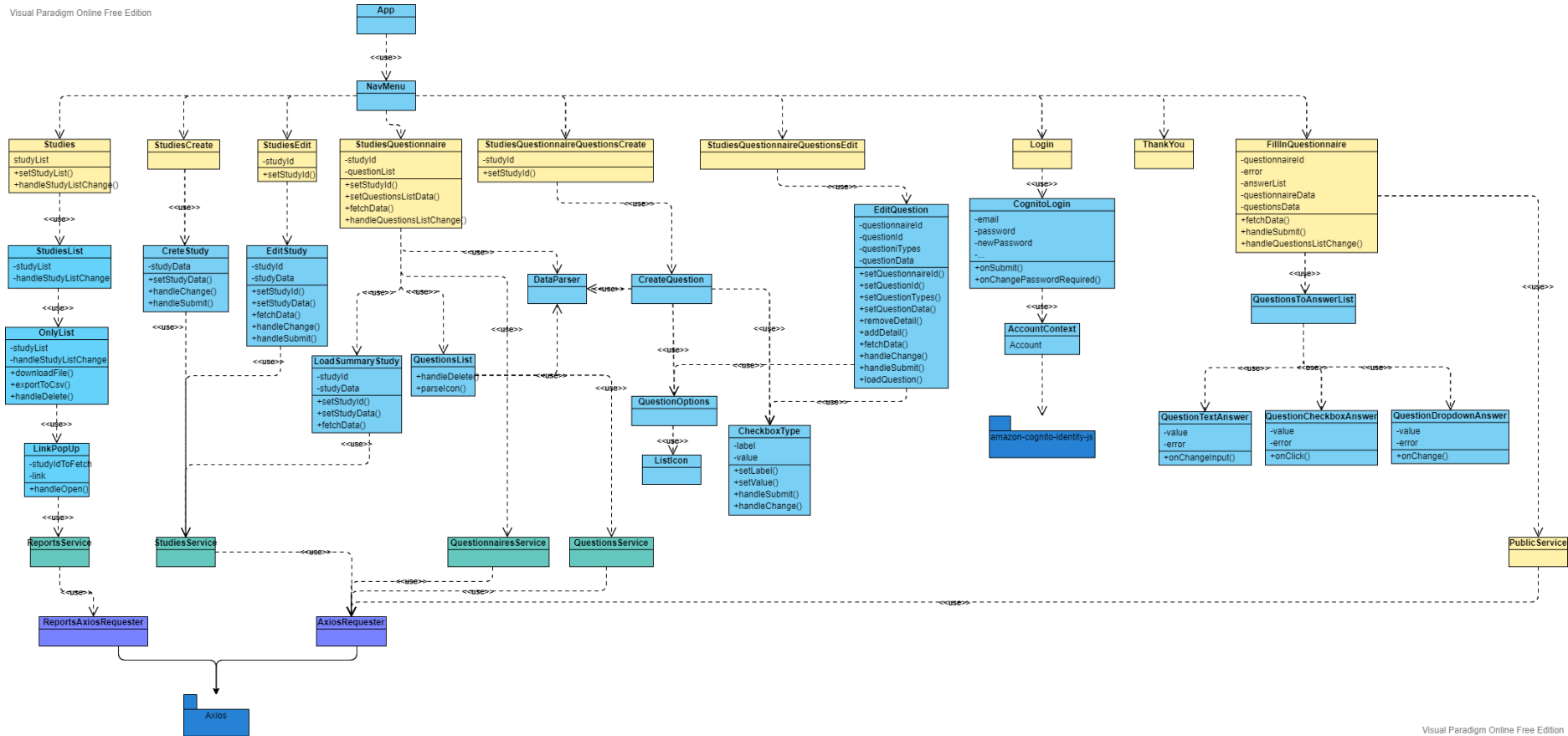
Figure 63: Deployment webpack Babel 3



# 4.5.6 Development

## 4.5.6.1 Architecture

Visual Paradigm Online Free Edition



Visual Paradigm Online Free Edition

Figure 64: ReactJS architecture

As can be seen in the previous image (Figure 64), the yellow classes are the 'pages', these are the React components that are responsible for managing each page site, they will use the necessary components (blue) to construct the page.

Each component can call other child components and, at the same time, call services (green).

Services are responsible for fetching data from the REST APIs. Each service uses a pre-configured Axios client.

This structure can also be found in the code under the src folder (Figure 65).

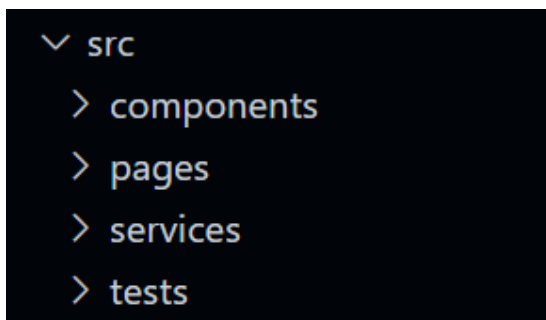


Figure 65: ReactJS folder structure

#### 4.5.6.2 Code

This section contains important aspects to mention about the code:

##### 4.5.6.2.1 Cognito authentication

To authenticate the ReactJS user with Cognito I used an npm package from AWS called 'amazon-cognito-identity-js' which takes care of the authentication part. I still had to implement all the actions such as log in, log out, get session data and token and the action for when a new password is required.

*The details of this implementation can be found in the Accounts.js file*

##### 4.5.6.2.2 Maintaining logged-in user

To maintain the logged-in user information across the entire application, I used the 'Context' provider from React. Context provides a way to share values such as the logged-in user data between components without having to explicitly pass a prop through every level of the tree.

*The details of this implementation can be found in the Accounts.js file*

#### 4.5.6.2.3 Sending authentication token to APIs

Every request is being sent with the authentication token in the Axios request from Cognito to the APIs (Figure 66).

```
export default axios.create({
  baseURL: 'https://jxlv29bo11.execute-api.eu-west-1.amazonaws.com/Prod',
  headers: {
    'Authorization': 'Bearer ' + token,
    'Accept': 'application/json',
    'Content-Type': 'application/json'
  }
});
```

Figure 66: Axios request

## 4.5.6.3 Website

### 4.5.6.3.1 Home page

The home page contains a small logo and an extra login button to make it more appealing (Figure 67). Every page of the website also uses the same background image.

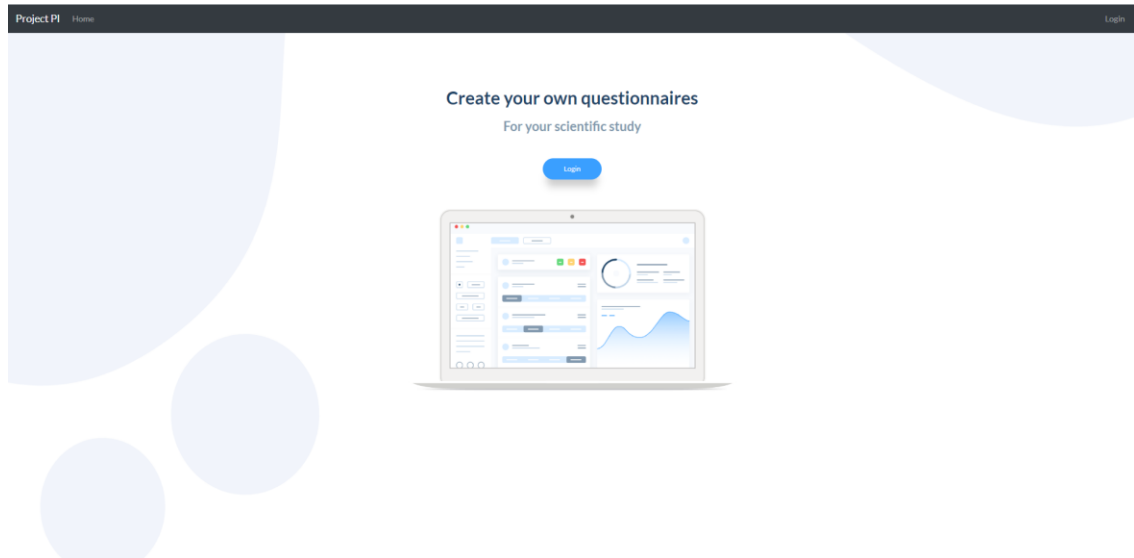


Figure 67: ReactJS home page

### 4.5.6.3.2 Login

The login form is very straight forward, comprising an email field and a password field (Figure 68).

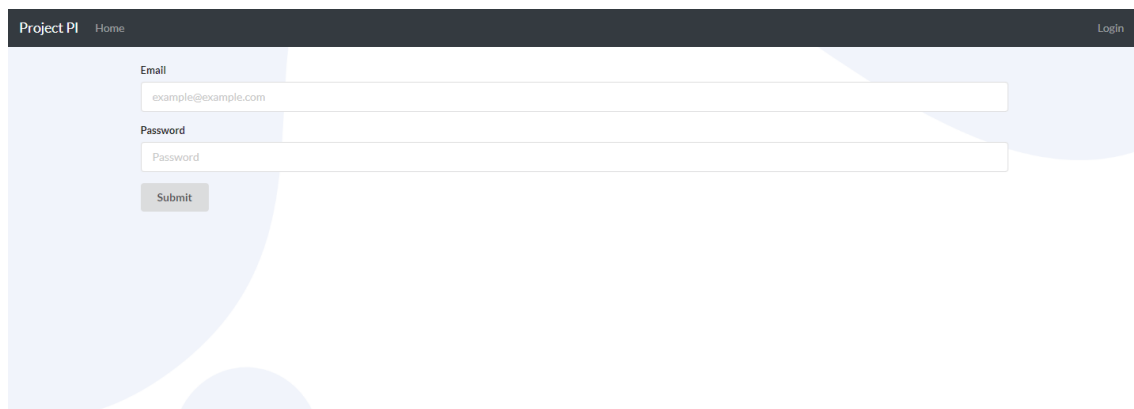


Figure 68: ReactJS login

In case the email or password are not correct, an error message is shown (Figure 69).

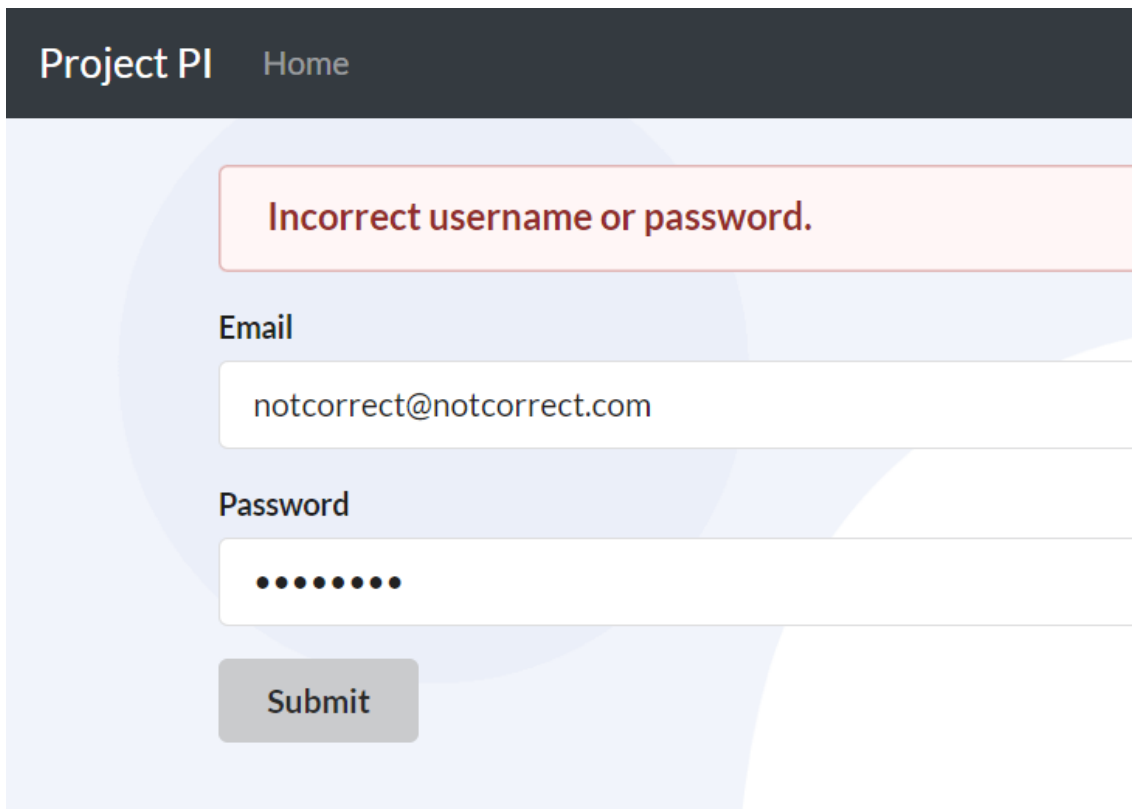


Figure 69: ReactJS wrong credentials

#### 4.5.6.3.3 My studies tab

For the logged-in users, they can find all their studies under the 'My Studies' tab. Each user can only see their own studies (Figure 70).

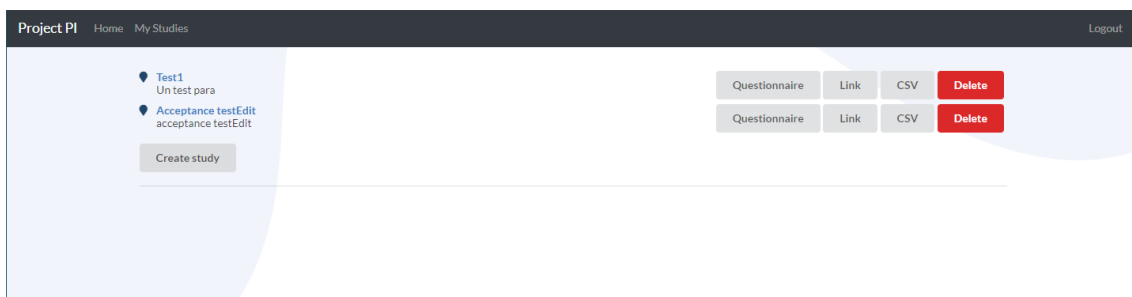
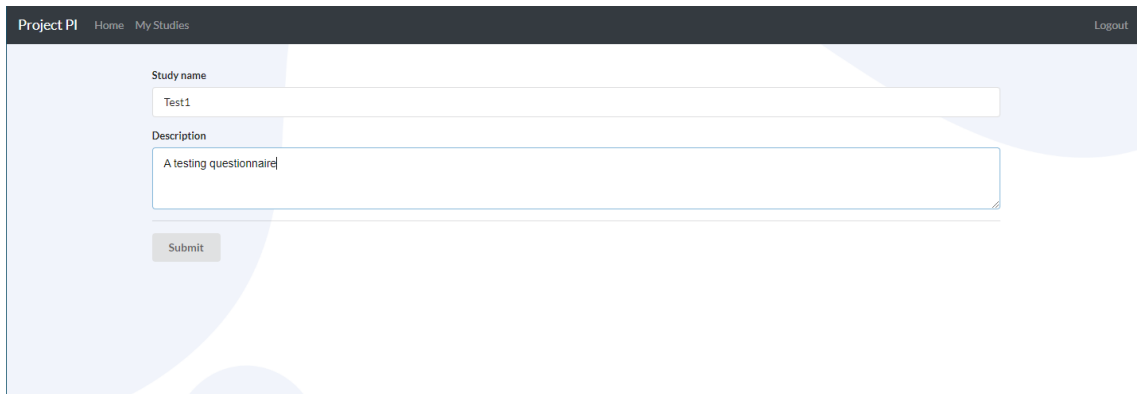


Figure 70: ReactJS my studies tab

#### 4.5.6.3.4 Create/modify study

Studies can be created or modified; the view remains the same for both use cases (Figure 71).

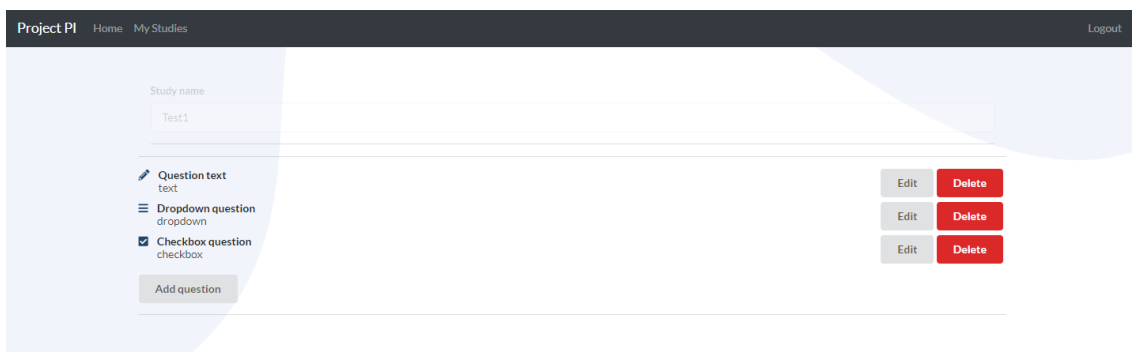


The screenshot shows a web interface for creating or modifying a study. At the top, there is a dark navigation bar with 'Project PI' on the left, 'Home My Studies' in the center, and 'Logout' on the right. Below the navigation bar, the main content area has a light blue background. On the left side, there is a form with two input fields: 'Study name' containing 'Test1' and 'Description' containing 'A testing questionnaire'. Below these fields is a 'Submit' button. The right side of the page is mostly blank.

Figure 71: ReactJS create/modify

#### 4.5.6.3.5 Questionnaire view

Each study has one questionnaire associated with it, for which the information such as the name is shown and all the questions which have been created for it (Figure 72).



The screenshot shows a web interface for viewing a questionnaire. At the top, there is a dark navigation bar with 'Project PI' on the left, 'Home My Studies' in the center, and 'Logout' on the right. Below the navigation bar, the main content area has a light blue background. On the left side, there is a form with a 'Study name' field containing 'Test1'. Below this field, there is a list of question types: 'Question text text', 'Dropdown question dropdown', and 'Checkbox question checkbox'. The 'Checkbox question checkbox' is selected with a checkmark. Below the list is an 'Add question' button. On the right side, there are three rows of buttons: 'Edit' and 'Delete' for each question type. The 'Delete' buttons are red, and the 'Edit' buttons are grey.

Figure 72: ReactJS questionnaire

#### 4.5.6.3.6 Add question

When creating a new question, apart from the question name and description which are common to all question types, once the type of question is selected a separate set of options are shown (Figure 73).

For the dropdown and checkbox types, the creator needs to add the options that will be shown to the respondent. With a Label (what the user will see) and a Value (what creators of the questionnaire will get in the CSV file).

Any of those values can be removed by clicking on the bullet point.

If the question type is text, no options are needed because it will show an input text.

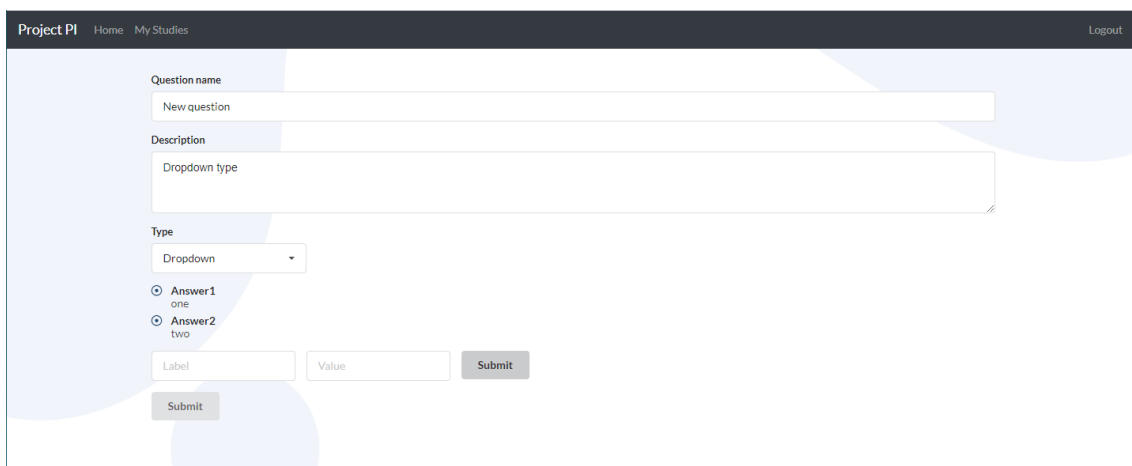


Figure 73: ReactJS add question

#### 4.5.6.3.7 Get access link

The link is available in a pop-up once the button 'Link' is clicked (Figure 74).

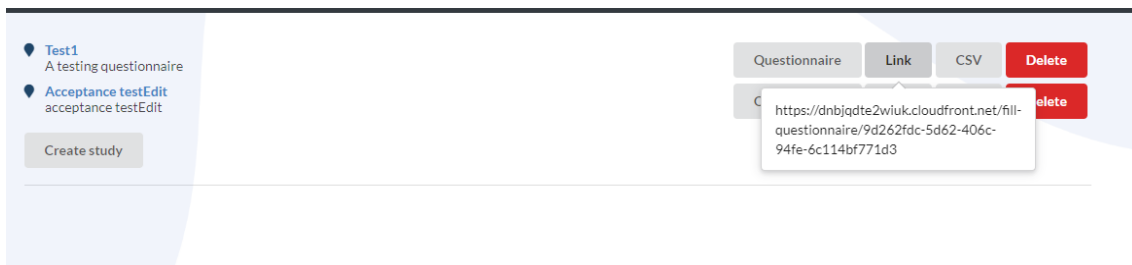
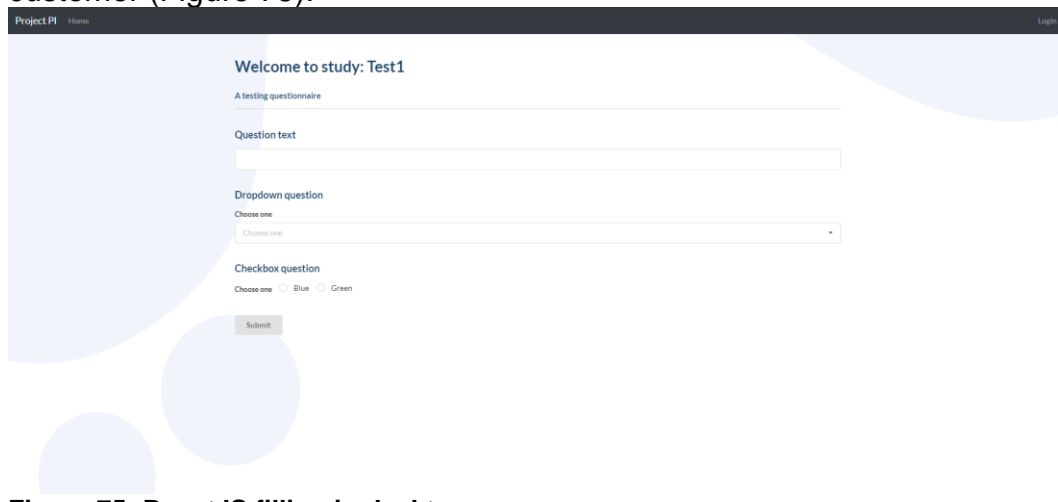


Figure 74: ReactJS questionnaire link

#### 4.5.6.3.8 Filling in questionnaire data in desktop

This is an example of the questionnaire view filled in by an anonymous customer (Figure 75).

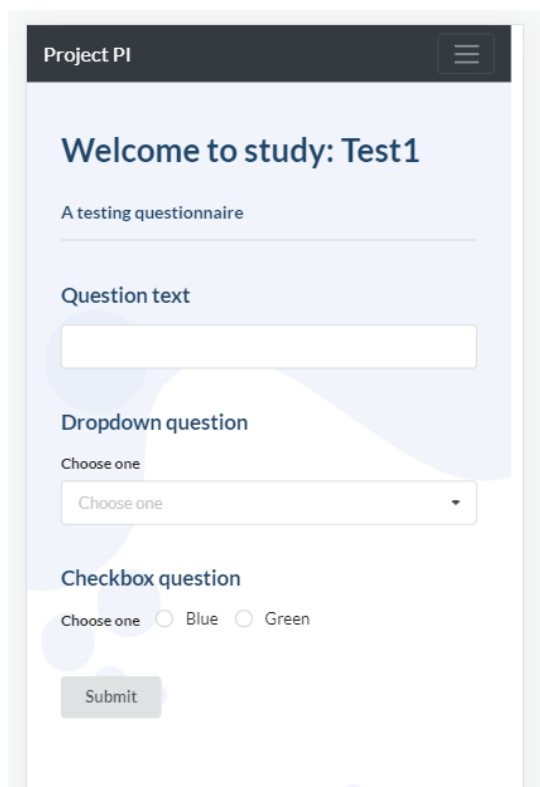


The screenshot shows a desktop view of a questionnaire titled "Welcome to study: Test1". The form includes a header with "Project PI" and "Home" on the left and "Login" on the right. Below the title, it says "A testing questionnaire". The form contains three main sections: "Question text" with a text input field, "Dropdown question" with a "Choose one" label and a dropdown menu, and "Checkbox question" with a "Choose one" label and two radio buttons labeled "Blue" and "Green". A "Submit" button is located at the bottom of the form.

Figure 75: ReactJS filling in desktop

#### 4.5.6.3.9 Filling in questionnaire data in mobile

The filling in of the questionnaire data is also available for mobile devices (Figure 76).



The screenshot shows a mobile view of the same questionnaire. The layout is adapted for a smaller screen, with the "Project PI" header and "Home" link on the left, and a hamburger menu icon on the right. The title "Welcome to study: Test1" is prominently displayed. Below it, the text "A testing questionnaire" is followed by a horizontal line. The form sections are: "Question text" with a text input field, "Dropdown question" with a "Choose one" label and a dropdown menu, and "Checkbox question" with a "Choose one" label and two radio buttons labeled "Blue" and "Green". A "Submit" button is at the bottom.

Figure 76: ReactJS filling in mobile



#### 4.5.6.3.10 Download CSV file

The CSV button will download all the answers for a particular study (Figure 77).

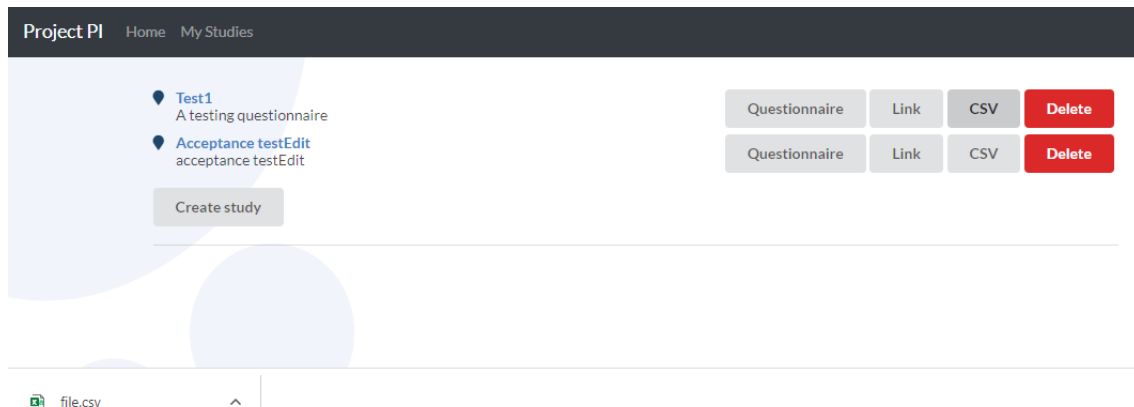


Figure 77: ReactJS .csv file

## 4.6 Python

### 4.6.1 Development environment

For the development of the Python application, I used Python 3.10.4 which was already installed in my mac OS.

### 4.6.2 Setting up CodePipeline

The configuration of CodePipeline was the same as shown for the NodeJS application to keep configurations consistent across all the applications.

The only difference is, instead of using NodeJS runtime, I used the latest available version in AWS: Python 3.9 (Figure 78).

```
install:
runtime-versions:
python: 3.9
```

Figure 78: CodePipeline

### 4.6.3 Development

This was a quite simple Python Lambda, so all the code is contained in the same file (Figure 79). This function is responsible for querying all the answers for a particular questionnaire, creating a csv file, and returning the file.

```
def lambda_handler(event, context):
    questionnaire_id = event['pathParameters']['id']

    dynamodb = boto3.resource('dynamodb')
    table = dynamodb.Table('StudyData')

    response = table.query(
        KeyConditionExpression=
            Key('pk').eq('QUESTIONNAIRE#'+questionnaire_id) &
            Key('sk').begins_with('ANSWER#')
    )

    csv_buffer = StringIO()
    writer = csv.writer(csv_buffer, delimiter=',', quoting=csv.QUOTE_ALL)
    writer.writerow(["AnswerId", "QuestionId", "Value"])

    for item in response['Items']:
        writer.writerow(
            [item['sk'], item['questionId'], item['value']]
        )

    csv_file = csv_buffer.getvalue()

    return {
        'statusCode': 200,
        'headers': {
            'Content-Type': 'text/csv',
            'Content-disposition': 'attachment; filename=testing.csv',
            'Access-Control-Allow-Origin': '*'
        },
        'body': csv_file
    }
```

Figure 79: Python Lambda

## 4.7 Acceptance testing

For the acceptance testing, I did a manual test for each required action.

### ADMIN

Action	Expected result	Result
Log in	Admin is logged in and can see the 'My studies' tab	Ok
Create study	Admin can create a new questionnaire	Ok
Edit study	Admin can edit the questionnaire title and description	Ok
Delete study	Admin can delete any questionnaire	Ok
Create question	Admin can create a new question for the questionnaire	Ok
Edit question	Admin can edit any question	Ok
Delete question	Admin can delete any question	Ok
Get link for study	Admin can get a public link to the study	Ok
Get csv file	Admin can download a csv file with the answers	Ok

### USER

Action	Expected result	Result
Access public study	User can access a study by using the public link	Ok
Form validation	User will get error messages if any of the questions is not answered	Ok
Answer questions	User can answer any type of question	Ok
Answers are stored	User answers are stored in database	Ok

## 4.8 Design changes

There have been 2 major changes and 1 minor change.

### 4.8.1 Major changes

#### CloudFront addition to AWS architecture

As mentioned previously in this document: for ReactJS to be able to work together with React routing, CloudFront was needed (Figure 80).

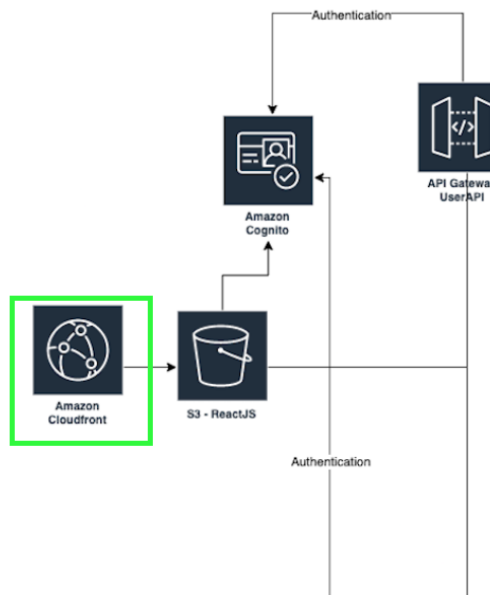


Figure 80: Major changes

#### API Gateway + Lambda for the Python application

The original idea was to have an EC2 instance for the Python application, however, during the development of this part I found it was an overkill solution for a simple problem such as fetching data and creating a csv file. The deployment of an EC2 instance with all the server configurations would have taken too much time. Instead, I did the same approach as I did with NodeJS.

Lambdas are short lived, 15 min max, but this should be sufficient for processing a csv file.

### 4.8.2 Minor changes

1. All questions are required
  - a. All questions part of a questionnaire are required instead of the admin being able to decide which ones are and which are not.
  - b. The code can be extended to allow this functionality nevertheless.

## 5 Financial evaluations

AWS offers a free tier for most of their services; if the free tier limit is not reached the costs are 0.

During the development I had barely reach that limit in 1 service (CodeBuild); for the rest I always kept below that threshold.

The free tier for the services I used are the following:

API Gateway

- 1 million API calls received per month.

AWS CodeBuild

- 100 build minutes free per month.

AWS CodePipeline

- 1 active pipeline per month.

AWS Cognito

- 50,000 MAUs per month.

AWS DynamoDB

- 25GB of storage and 200 million read/write requests per month.

AWS Lambda

- 1 million requests per month.

Month	API Gateway	CodeBuild	CodePipeline	Cognito	DynamoDB	Lambda
April	0\$	0\$	0\$	0\$	0\$	0\$
May	0\$	0.5\$	0\$	0\$	0\$	0\$
<b>Total</b>		<b>0.5\$</b>				
<b>21% VAT</b>		<b>0.105\$</b>				
<b>Total inc. VAT</b>		<b>0.605\$</b>				

As can be seen, the total cost amount for setting up the infrastructure and development was 0.605\$

It is important to mention that if the tier limit is not reached, the website will be fully functional at 0 cost. Once the limit is reached, the billing for those extra services will start, but because the infrastructure is completely serverless the cost will remain exceptionally low.

## 6 Conclusions

### 6.1 Lessons learnt

During this project I learnt how to deploy an entire cloud infrastructure in AWS and deep dived into its diverse types of services and how to interconnect them.

Regarding other technologies, I grasped the power of front end frameworks such as ReactJS to build interfaces, as well as using NodeJS for the back end side.

Going through all the development processes, starting with the design, planning and culminating with this document also taught me how to be more organized, methodical and appreciative of the technical documentation.

### 6.2 Goals achieved

All main and secondary goals have been achieved during this project and it was possible thanks to:

- Using the knowledge that was obtain during my studies as a software engineer student as well as using my professional experience as a developer.
- Being aware of the time that was available to develop the code, focusing on developing only the most important aspects of an MVP (Minimum Viable Product) and setting up the basis for what it could become in the future: a marketable product.
- Using existing technologies and frameworks available in the market.

### 6.3 Planning

The methodology used during this project was of a pseudo-scrum approach. Because I was the only one working on it, I thought it unnecessary to do a full scrum approach and instead I did a more lightweight version where I was working on small achievable stories and re-evaluating the results and the next steps after each one was completed.

The planning was followed as specified in Figure 1 and achieved successfully.

## 6.4 Future vision

A big part of this project was setting up the infrastructure, including a development life cycle with the inclusion of a CI/CD and automatic deployment. This was done with the idea of setting up the basics for what could potentially become a marketable product.

There are still improvements that can be made:

### Improvements

- Investigate the speed of the site to make it faster.
- Having also testing and acceptance environments.
- Having \*optional questions in the questionnaires.
- Being able to reorganize the order of the questions.
- Add the secrets (such as the API gateway URLs) as part of the deployment so they do not need to be hardcoded in the code itself.

### Future vision

- Add a reports system with more options to download the data such as pdfs, raw data, ...etc
- Use machine learning to analyse the data automatically for each study.
- Add new types of questionnaires, such as randomized studies.

# 7 Glossary

## 7.1 AWS

Amazon Web Services (AWS) is the world's most comprehensive and universally adopted cloud platform, offering over 200 fully featured services from data centres globally. Millions of customers—including the fastest-growing start-ups, largest enterprises, and leading government agencies—are using AWS to lower costs, become more agile, and innovate faster.

## 7.2 Cloud

Cloud computing is the delivery of different services through the Internet. These resources include tools and applications such as data storage, servers, databases, networking, and software.

## 7.3 Microservices

Microservices - also known as microservice architecture - is an architectural style that structures an application as a collection of services that are:

- Highly maintainable and testable.
- Loosely coupled.
- Independently deployable.
- Organized around business capabilities.
- Owned by a small team.

## 7.4 IaC

Infrastructure as code (IaC) is the process of managing and provisioning computer data centres through machine-readable definition files, rather than physical hardware configuration or interactive configuration tools.

## 7.5 Tech stack

A tech stack is the combination of technologies a company uses to build and run an application or project

## 7.6 Framework

An abstraction in which software, providing generic functionality, can be selectively changed by additional user-written code, thus providing application-specific software. It provides a standard way to build and deploy applications and is a universal, reusable software environment that provides functionality as part of a larger software platform to facilitate the development of software applications, products and solutions.

## 7.7 API

API stands for Application Programming Interface. A Web API is an application programming interface for the Web. A Browser API can extend the functionality of a web browser. A Server API can extend the functionality of a web server.



## 8 Bibliography

- [1] react.org, "Getting started," 2022. [Online].  
Available: <https://reactjs.org/docs/getting-started.html>.
- [2] semantic-ui.com, "Getting started," 2022. [Online].  
Available: <https://semantic-ui.com/introduction/getting-started.html>.
- [3] babeljs.io, "What is Babel?," 2022. [Online].  
Available: <https://babeljs.io/docs/en/>.
- [4] webpack.js.org, "Concepts," 2022. [Online].  
Available: <https://webpack.js.org/concepts/>.
- [5] nodejs.org, "About Node.js," [Online].  
Available: <https://nodejs.org/en/about/>.
- [6] python.org, "About," [Online].  
Available: <https://www.python.org/about/>.
- [7] aws.amazon.com, "Building applications with serverless architectures," [Online].  
Available: <https://aws.amazon.com/lambda/serverless-architectures-learn-more/>.
- [8] L. Gupta, "What is REST," [Online].  
Available: <https://restfulapi.net/>.
- [9] docs.aws.amazon.com, "What is Amazon DynamoDB," [Online].  
Available:  
<https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/Introduction.html>.
- [10] docs.aws.amazon.com, "What is AWS Lambda," [Online].  
Available: <https://docs.aws.amazon.com/lambda/latest/dg/welcome.html>.
- [11] docs.aws.amazon.com, "What is Amazon API Gateway," [Online].  
Available: <https://docs.aws.amazon.com/apigateway/latest/developerguide/welcome.html>.
- [12] docs.aws.amazon.com, "What is Amazon Cognito?," [Online].  
Available: <https://docs.aws.amazon.com/cognito/latest/developerguide/what-is-amazon-cognito.html>.
- [13] docs.aws.amazon.com, "What is Amazon S3?," [Online].  
Available: <https://docs.aws.amazon.com/AmazonS3/latest/userguide/Welcome.html>.
- [14] docs.aws.amazon.com, "What is Amazon CloudFront?," [Online].  
Available:  
<https://docs.aws.amazon.com/AmazonCloudFront/latest/DeveloperGuide/Introduction.html>.
- [15] docs.aws.amazon.com, "What is AWS CodePipeline?," [Online].  
Available: <https://docs.aws.amazon.com/codepipeline/latest/userguide/welcome.html>.
- [16] docs.aws.amazon.com, "What is AWS CloudFormation?," [Online].  
Available:  
<https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/Welcome.html>.
- [17] jestjs.io, "Getting Started," [Online].  
Available: <https://jestjs.io/docs/getting-started>.
- [18] J. Beswick, "Creating a single-table design with Amazon DynamoDB," aws.amazon.com, 26 07 2021. [Online].  
Available: <https://aws.amazon.com/blogs/compute/creating-a-single-table-design-with-amazon-dynamodb/>.
- [19] sumologic.com, "What is CRUD?," [Online].  
Available: <https://www.sumologic.com/glossary/crud/>.

- [20] developer.mozilla.org, "Cross-Origin Resource Sharing (CORS)," [Online].  
Available: <https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS>.
- [21] en.wikipedia.org, "Comma-separated values," [Online].  
Available: [https://en.wikipedia.org/wiki/Comma-separated\\_values](https://en.wikipedia.org/wiki/Comma-separated_values).
- [22] S. J. Bigelow, "Infrastructure as code," [Online].  
Available: <https://www.techtarget.com/searchitoperations/definition/Infrastructure-as-Code-IAC>.

## 9 Annex

Website url: <https://dnbjqdt2wiuk.cloudfront.net/>

Available users

<b>Username</b>	<b>Password</b>
<b>user1@user.com</b>	User1Pass!
<b>user2@user.com</b>	User2Pass!
<b>user3@user.com</b>	User3Pass!
<b>user4@user.com</b>	User4Pass!