SS-184-QUAEFACTA TECHNICAL REPORT

Version 0.2

*Quaefacta logo used with permission from Lea Dias.

Start Date: 29/10/2022

Authors:

- Richard Forsey, s3857811@student.rmit.edu.au
- Aman Khan, s3811422@student.rmit.edu.au
- Jiajie Lu, s3779102@student.rmit.edu.au
- Jack Allan, s3832293@student.rmit.edu.au
- Quoc Tran, s3827826@student.rmit.edu.au

Github link: https://github.com/RichardForsey89/QueafactaHealthMk2

0 - Document Control

Version	Implemented by	Implementation date	Reviewed by	Approval Date	Reason
0.1	Richard, Aman, Jiajie, Jack, Quoc	29/10/2022	Richard, Aman	29/10/2022	Document is needed for submission!
0.2	Richard, Aman, Jack, Quoc	30/10/2022	Richard, Aman	30/10/2022	Expanded the content in many areas.

Table of Contents

1 – Executive Summary

Quaefacta Health is a healthcare technology company that is focused on bringing solutions that are secure and user-centric to the healthcare and life sciences fields, with the eventual goal of including new technologies such as blockchain, AI, and the Internet of Things (IoT). However, before one can run, you must walk, and our project ended up being focused on more prosaic items and basic functionality.

The project was initially presented to us as being a final polishing up and fixing of a code base that was notionally already in a near-complete state for a React web app. Unfortunately, the code base was in fact in a state of near-complete uselessness due to the mismanagement of effort by prior groups. So, on a technical level our project went through a near-complete rebuild.

The inherited front-end code was a big mess of React JS with countless instances of in-line styling, repeated code, no code elegance and poor code organization. Furthermore, the code wasn't even connected to the back-end that was notionally present in any way. It was simply a repeat of the Figma design re-rendered in code, an empty and poorly made shell. Unfortunately, due to a mixture of member illness, student syndrome and an initial lack of leadership, the group took a few weeks to realize these problems. However, in the middle of the semester, the situation was understood by the team, and we sprang into action.

Looking upon the mess we had in front of us, the group made the tough decision to approach the client and offer the idea of a full rebuild of the application using a vertical slice strategy, and with the aim to make reusable, extendable and reliable code that follow-on groups could make good use of.

Our client was quite open to our concerns and accepted our proposal. In the course of a mere five weeks, we reconstructed the application in React TypeScript with improvements. The client was impressed and happy with our achievement and the work produced.

2 – Introduction

The project objectives and background will be explained in two sections, pre and post realization, due to the very different attributes of both.

Pre-realization

Our group was delayed in starting due to this being our second project for Capstone, as the first was found to be ethically unsuitable by the group. As a result, our start was delayed until week 4 of the semester. In week four we made contact with our client **Lea Dias**, and did the initial planning of our project. Around this time, our "Technical Lead" Richard got sick with Covid, which naturally pulled away his attention from the project.

The vision of our project was initially to focus heavily on end-to-end testing of the web app, improving the security features, user privacy and to re-do the AWS management and CI/CD pipeline of the application. A Jira board was created, documentation was reviewed and a sprint plan was drawn up. The project seemed like it would be a straightforward jog forward with no real surprises.

Then we started getting some warning signs. The web app wouldn't compile properly and had to be adjusted. The front-end didn't need to have the back-end running for it to appear to work. Quoc was the one making most of these discoveries, as he had taken on the responsibility of any back-end issues. At this point we were still thinking that things couldn't be that bad, but once Richard had recovered and taken a solid look at the React JS code, in week 6 an unfortunate realization was made: we would be better off starting again with a clean slate.

Post-realization

After this realization was made, the project scope and focus had to change completely and as part of this the responsibilities of the team had to be reallocated due to new requirements. We had also at this point realized that we needed to have clear leadership and responsibilities situation so that the group could start moving forward in good order. The team changed to the following organization:

- Scrum master / capstone leader – Aman
- Technical / Front-end lead – Richard
- Back-end developer – Quoc
- Front-end developer – Jiajie
- AWS / DevOps – Jack

These roles were parcelled out according to the capabilities and inclinations of each member against the needs of the project. For instance, Richard had initially been earmarked as the one to be developing the AWS / DevOps components but the group had the need for someone who was broadly across in understanding every part of our tech stack so that they could be well coordinated. As a result, this responsibility was instead handed over to Jack, as he was the 2nd most experienced with AWS in the group, and Richard ended up being the technical lead and main front-end developer, as the latter was the most critical aspect of the project.

Likewise for Quoc, he was the most suitable for the backend work and so he stayed in that Role. With Jiajie, we had originally had him set for doing testing, but with our expanded need for front-end work, he was reassigned to assist here. Aman was the most interested in the overall organisation, scrum planning documentation and so on, which would pair well with his security focus, and the team was very happy for him to take this role.

With this reorganization done, we had solved our initial problems of leadership and responsibilities.

The new project vision, goals and objectives

The new vision our team had was to deliver a "vertical slice" of the application, that is to say a full-stack, functional and live example of the program, or speaking informally, going "tall" instead of "wide". This was chosen because we felt that with the limited time left for us to deliver our capstone project, we would need to have a reasonable set of goals.

Furthermore, we also felt that it would provide a better basis for follow-on work, as there would be a fully integrated and functional example to build on and extend.

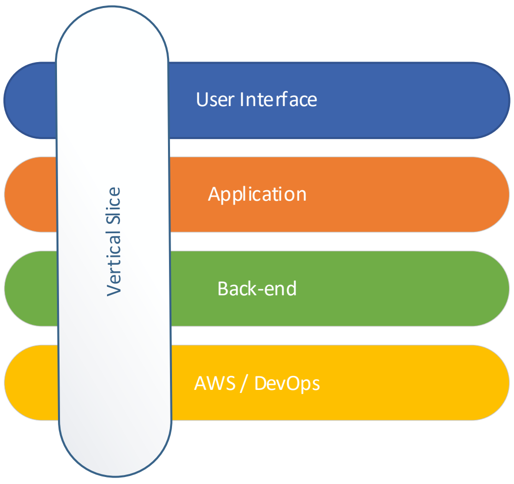


Figure 1. Vertical slice diagram.

With this in mind, our objectives were the following:

1. Implement the web-app in React with TypeScript, as per industry norm.
2. Implement the backend APIs needed for each screen of the web app we complete with Java Spring Boot.
3. Integrate items 1 and 2 with each-other.
4. Implement a complete AWS orchestration plan, at first for manual deployment, but later expanded to automated with Terraform and Ansible.
5. CI pipeline on the GitHub repo.

Stakeholder details

Society

Access to healthcare is an important issue, and with the recent data breaches with Optus and Medibank Private, being able to take control of one's data and store it securely is beneficial and of interest to many people. When we started building this application, we were reminded time and again by the client about the high stakes of making a secure platform for the society to upload their medical records and keep a track of their vaccines and medications.

The Client (Lea Dias, Queafacta Health)

Our client has seen the gap in the market for a product that answers the concerns raised in the previous stakeholder group and wishes to create an app which answers these concerns. The reputation of the client is at stake as due to the two big cybersecurity breaches in companies like Optus and Medibank, everyone is looking for a secure platform and if we fail to deliver a product which could safekeep the medical records of its users, we will put our reputation, our university's reputation and our client's trust as well as reputation at stake.

RMIT

RMIT will naturally be interested in showcasing the skills of the students who have received their education. RMIT has strong standards to uphold, so if a group of students there creates something that fundamentally alters the medical sector for good, RMIT would receive enormous praise. However, if something goes wrong, RMIT will also be held accountable, which would be catastrophic for all of the students there.

Our Team

We want to demonstrate our capabilities and advertise our competence to potential future employers. Creating a high-quality capstone project will be a positive addition to our portfolios. Our reputation and future career prospects are on stake as we thrive to make a product which could be used for the upliftment of the society rather than a product which could be misused to cause harm to their users.

3 – Requirements

Functional Requirements

After many direction changes in the project and multiple meetings with our client, we were able to extract the main functional requirements for this project. Seeing as this was a complete remake of the existing project, the main functional requirements are for new users being able to create a new account, users being able to sign in, users being able to edit their account, and users being able to upload documents. These would need to be functioning end to end, that is, have the front-end pages connect to a working backend database and storing the data in said database. For each of these requirements, we were able to break them down into the necessary tasks.

New users can sign up and create an account

- Create a sign up page
- Create the sign up form handling logic
- Create a User table to store id, email, and password
- Create API to insert new users into table
- Front end to send {email, password} to the backend API.
- Backend to store a hashed version of the password

Users can sign in to their account and view logged in user pages

- Create sign in page
- Create sign in form handling logic
- Create API to check if the user exists
- Create API to retrieve user from database on successful login
- Front end to send {email, password} to backend API
- Backend to check the password matches the stored hashed version of the password.

Users can update information in their profile

- Create user information page
- Create edit user information form handling logic
- Create user information table to store firstName, lastName, address, etc..
- Create API to check for updated fields and insert only the updated fields whilst keeping the unchanged fields the same
- Create API to retrieve entries from the user information table to display on frontend

Users can upload documents

- Create document upload page
- Create document handling logic via multipart file
- Create API to receive multipart file, connect to s3 storage bucket in AWS, and place uploaded document into correct directory for chosen file type.
- Create API to download stored files from s3 storage buckets

Upon completion of these requirements, our client was extremely pleased. We then discussed on what was possible for further development given the remaining time. We decided on a set of high priority pages for front end development. These would be fully functioning front end pages with interactive buttons, camera features, and further document upload features. These would not be connected to a backend as we did not have time to develop it and would be further developed by future groups. Alongside with these high priority pages would be a solution to automatically set up the AWS environment to run the application and to terminate all AWS resources when required. This requirement came after discussions about our client's previous experience with AWS where she was getting large monthly bills while having a non-operational application running. She requested a simplistic way to run and terminate all AWS resources with one click.

Camera functionality working

- Create camera page
- Make camera button display the device camera with the ability to take a photo
- When a photo is taken, return to the previous screen

Post signup page

- Create a page displaying a checklist of activities required after signup
- Have interactive buttons that change colours accordingly depending on which button is pressed

Health information page

- Create a health information page
- Create health information form handling logic
- Have a list of allergies and illnesses dynamically display on the page as the user enters them into the form

Automated AWS setup and shut down

- Create ansible scripts to spin up the AWS resources
- Create ansible script to destroy all AWS resources

With these functional requirements complete, our client was extremely pleased with the state of the application.

Non-functional Requirements

Within every client meeting, our client made it clear to follow the figma design as closely as possible. These designs were the results of previous UI research and development. The colour schemes were chosen specifically as they represent the medical field. As such, the non-functional requirements we came up with related mainly to the look and feel of the application and to ensure the implementation would follow the figma screens as closely as possible. To achieve this, we made reusable components in react, so that each button, form field, title, and any other front-end component would look like one another. This also enabled us to change the styling of each set of components by changing a single css field.

Another non-functioning requirement was to keep the cost of AWS as low as possible. As such we used free tier options where we could. In the end, our monthly cost was but a fraction of the previous year's implementation and our client was very pleased.

4 – Architecture

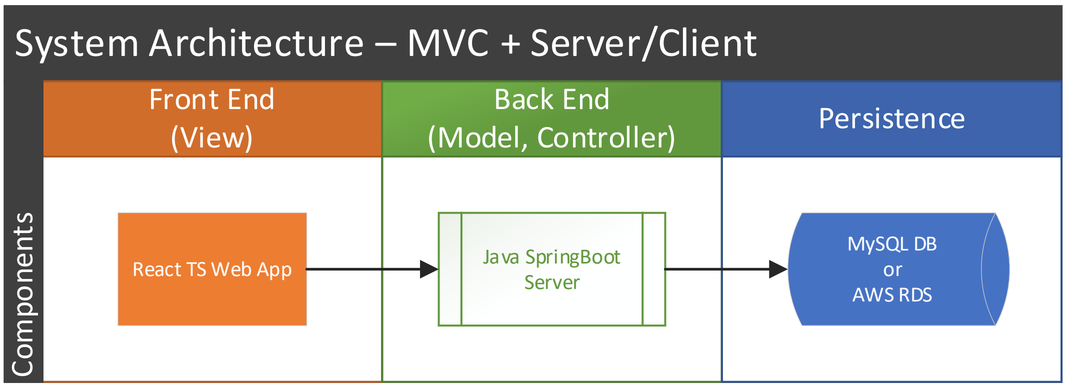


Figure 2.Diagram of the system architecture of our project.

In our project, we implemented the standard designs of Server/Client and MVC.

As React TS follows a functional paradigm instead of an object-oriented one, it would be inappropriate to render it in an entity-relationship diagram.

The SpringBoot backend followed the typical pattern of the framework, with models, controllers, services and repositories.

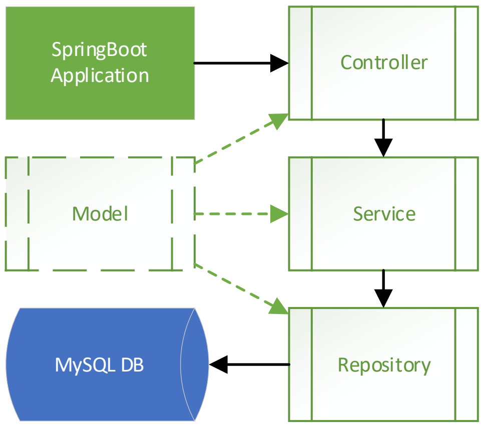


Figure 3. Class pattern of our backend.

5 – Technical Framework

The technical framework is a fairly typical arrangement:

- React Typescript front end.
- Java SpringBoot backend connecting to a local MySQL database for development, and to AWS services when deployed.
- Hosting provided by AWS.
  - EC2 instance
  - AWS RDB
  - S3 Bucket for files
- Development was done on Windows and Mac computers, and the deployment environment is Linux.
- CI Pipeline is run on the GitHub repo with GitHub actions.
- Postman was used to test the backend.
- The front end was tested using the React testing library.

Figure 4. Our tech stack components.

6 – Implementation

We've limited this section to only cover the more noteworthy pages for brevity, as many are quite similar to each other in terms of design and implementation.

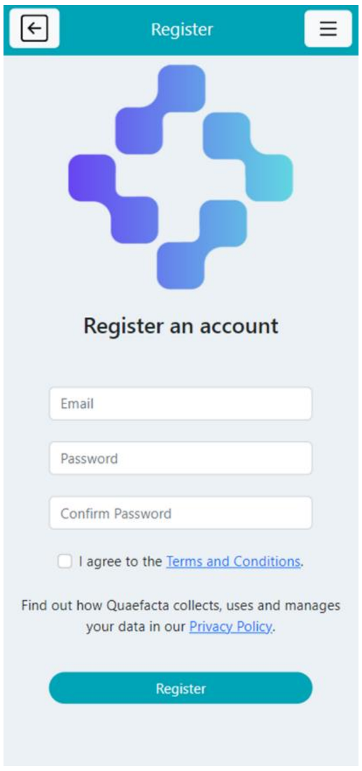

Register
Register an account
Email
Password
Confirm Password
I agree to the Terms and Conditions.
Find out how Quaefacta collects, uses and manages your data in our Privacy Policy.
Register

Figure 5. Registration Screen.

Register Account

This is the Register page, using typescript and in the Email password and Confirm Password if did not enter anything and just click the register, it will come up with the prompt to fill them in. We have done the signup page as well, but for this section we felt notable pages were better to show. For the terms and Conditions and privacy policy clicking the links will go to the pages for these items. When given a valid email and password, it will be stored in the database.

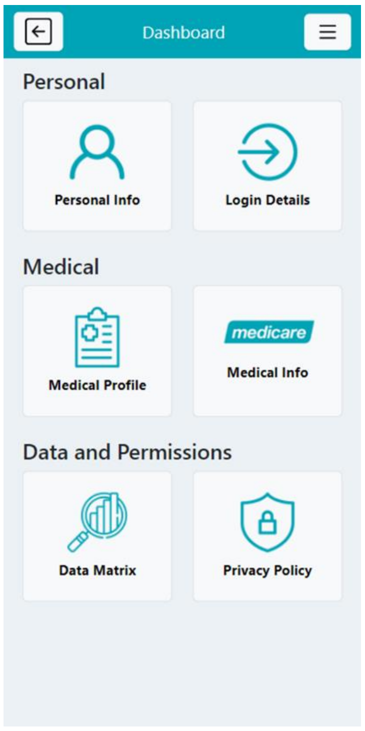


Figure 6. Main Dashboard screen.

Main Dashboard

This is the Main Dashboard once logged in to an account. Each button leads to a different page, and this screen will be expanded upon in future. The design philosophy here was to ensure that navigation is clear and easy with a flat and horizontal tile design.

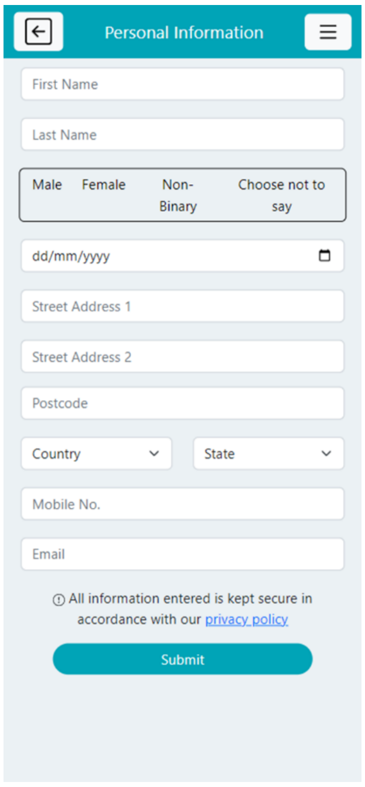


Figure 7. Personal information input form Screen.

Personal Information Form

This is the personal information form; each form needs to be filled in otherwise it cannot be submitted. It lets the user input all the personal information that might be needed when using the app. For the form we've made all the details as similar as possible to the Figma. There is of course validation, for example the date of birth, you can only choose dates from the past.

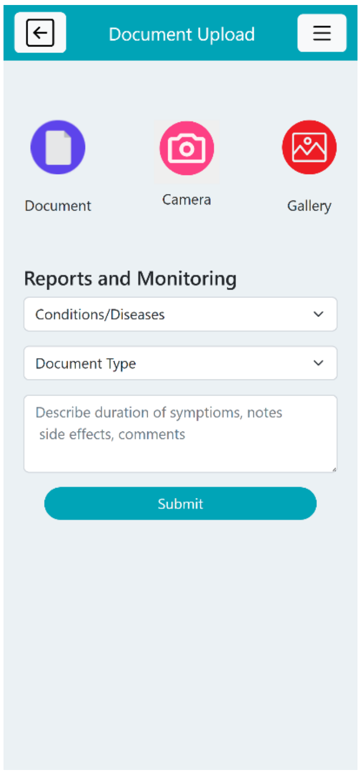
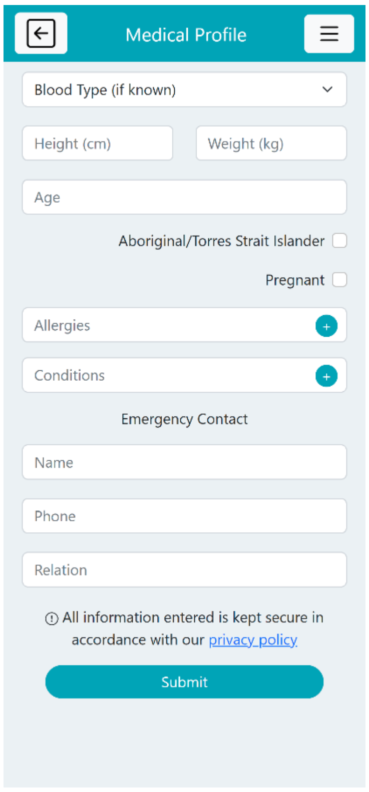


Figure 8. Document upload screen.

Document Upload

This is the document upload page, there are three main options here: document, camera and gallery. Pressing camera will navigate the user to a different screen for taking a picture. The user can also choose to upload files from their documents or gallery as well, using their device's native utilities. Categorization details and a personal note can be chosen or added below.


Medical Profile

Blood Type (if known)

Height (cm) Weight (kg)

Age

Aboriginal/Torres Strait Islander ☐

Pregnant ☐

Allergies +

Conditions +

Emergency Contact

Name

Phone

Relation

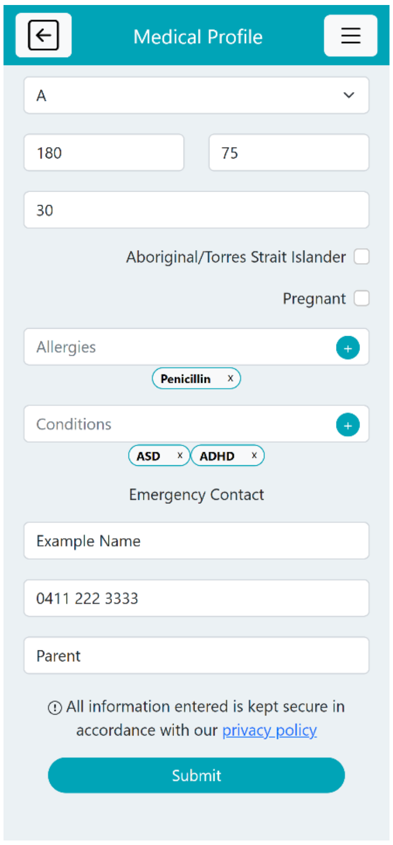
ⓘ All information entered is kept secure in accordance with our privacy policy

Submit

Figure 10. Medical profile form empty state.

Create Medical Profile

On this screen the user can input their medical information. Multiple Allergies and conditions can be added. Mistaken additions can also be removed. Once submission is pressed, the information is saved.


Medical Profile

A

180 75

30

Aboriginal/Torres Strait Islander ☐

Pregnant ☐

Allergies +

Penicillin x

Conditions +

ASD x ADHD x

Emergency Contact

Example Name

0411 222 3333

Parent

ⓘ All information entered is kept secure in accordance with our privacy policy

Submit

Figure 9. Medical profile form with data.

7 – Deployment Instructions

Terraform

Description

Terraform can be used to build and destroy AWS infrastructure.

This README will explain the process of setting up Terraform and using the main.tf Terraform script.

Part of this process includes installing Terraform and the AWS CLI, an AWS account is also required.

Install Terraform

Terraform can be downloaded from the following link: https://www.terraform.io/downloads

Depending on your operating system you may need to add it to the system path manually (for windows).

An in-depth tutorial on the installation process and use with AWS can be found here: https://learn.hashicorp.com/tutorials/terraform/install-cli?in=terraform/aws-get-started

Install and Setup AWS CLI

AWS CLI can be downloaded by following the instructions at: https://docs.aws.amazon.com/cli/latest/userguide/getting-started-install.html#getting-started-install-instructions

To use your AWS IAM credentials, you will need to set the AWS_ACCESS_KEY_ID and AWS_SECRET_ACCESS_KEY environment variables.

```
export AWS_ACCESS_KEY_ID=
```

```
export AWS_SECRET_ACCESS_KEY=
```

Terraform Script

Accessing the Terraform Script

Open main.tf in your text editor of choice to view the terraform script. This script contains all the information to create the AWS infrastructure for the Quaefacta health web application.

Setup the Terraform project directory

Before running the Terraform script it is a good idea to initialise the terraform-aws directory in which it is located.

This can be done with the following line in your command prompt:

```
terraform init
```

Creating and Updating AWS Infrastructure

You can create or update the AWS infrastructure outlined in the main.tf terraform script by running the following command:

```
terraform apply
```

In the case of updating the AWS infrastructure, any changes to the main.tf script will be used to update the necessary AWS resources.

You will be prompted with a list of actions that Terraform will perform and confirmation that you want to perform these actions you can answer with 'yes' to continue.

You have now created the AWS infrastructure for the Quaefacta Health web application, you can view it online on the AWS website.

When you run the terraform apply command you will receive an output with the public IP address for the ec2 instance. You can ssh into this ec2 instance using PuTTY which can be downloaded from https://www.putty.org/. The private key to connect to the ec2 instance is the QuaefactaKey.ppk file so be sure to add this file to the private key file for authentication section in PuTTY before connecting.

Destroying the AWS Infrastructure

To terminate the resources managed in this Terraform project you can use the command:

terraform destroy

Please note that this command will only terminate AWS infrastructure that was created by the terraform apply command so the rest of your AWS resources will not be affected.

Ansible

Description

Ansible can be used to deploy a series of commands to our AWS EC2 instance.

The Ansible playbook 'playbook.yml' contains the commands to upload the frontend application to an EC2 instance and make this application accessible over the internet.

This README will explain the process of installing ansible, updating the ansible script ip address and running the ansible script

Install Ansible

To install Ansible on an ubuntu system you can follow the instructions on this web page https://docs.ansible.com/ansible/latest/installation_guide/intro_installation.html

Updating Ansible Script IP

Before running the ansible script, you'll need to update the IP address. In the Ansible folder you can find a file named 'playbook.yml'. This file can be edditted in any simple text editor.

Go to the 7th line and you will see a web_ip variable, replace the numbers listed with the public IP address of the AWS EC2 instance you wish to connect to (the Terraform script will output the public IP address for the EC2 instance it creates, you can directly copy and paste this into the script).

Be sure to save the script before closing.

Adding Application to File Folder

The application needs to be added to the files directory located in the ansible directory for the script to run.

It is important that the application is compressed with the .tgz file extension - this can be done in your Ubuntu terminal with the command:

```
tar -cvzf application.tgz <path to application folder>
```

Now the ansible playbook can be run

Running the Ansible Playbook

To run the Ansible playbook access the Ansible folder in your terminal and enter the following command with the replaced with the EC2 instance public IP:

```
sudo ansible-playbook --private-key ../QuaefactaKey.pem -i ubuntu@<EC2 instance public IP>
playbook.yml
```

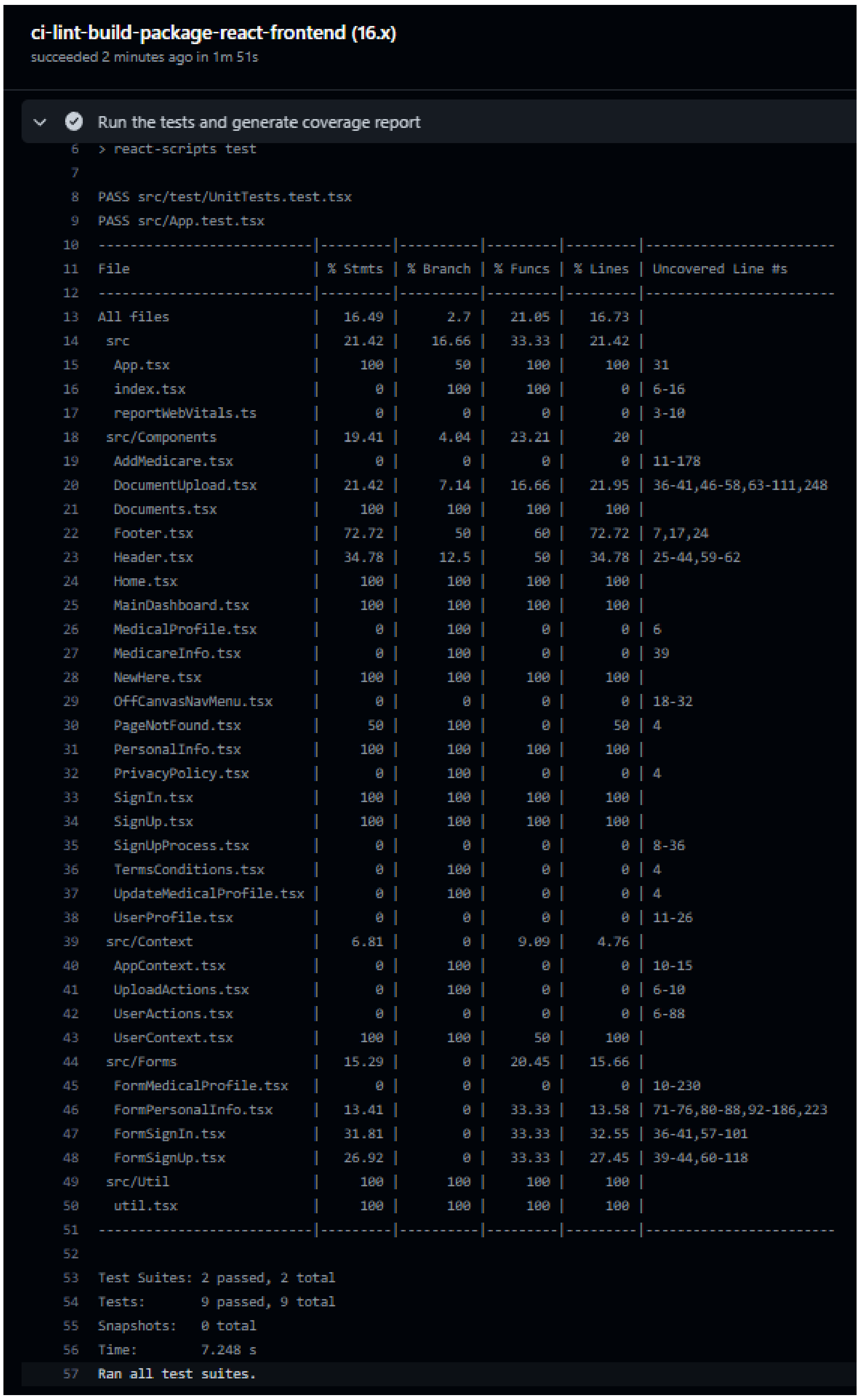
This command will connect to the EC2 instance using the QuaefactaKey.pem private key and run the ansible playbook - running the frontend application and making it accessible to the internet. To connect to this application, you can enter the EC2 instance public IP followed by ':5000' into your web browser. Or go to digitalhealthwalletapp.com

8 – Testing Results

At the moment, our testing strategy is somewhat limited. We only have unit tests in place for the front end, but we did also complete a full set of postman tests for the backend, and our earlier work with writing acceptance tests is still valid.

Unit Tests (React Testing Library)

With our current set of unit tests, they are mostly smoke tests to check that a given page is rendering properly, with only some function tests with input handling.



ci-lint-build-package-react-frontend (16.x)
succeeded 2 minutes ago in 1m 51s

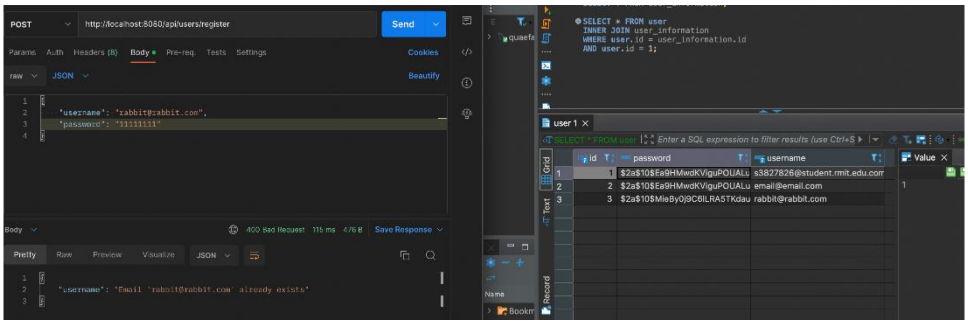
Run the tests and generate coverage report

```
> react-scripts test

PASS src/test/UnitTests.test.tsx
PASS src/App.test.tsx
---------------------------|---------|----------|---------|---------|-----------------------
File                       | % Stmts | % Branch | % Funcs | % Lines | Uncovered Line #s
---------------------------|---------|----------|---------|---------|-----------------------
All files                  |   16.49 |      2.7 |   21.05 |   16.73 |
 src                       |   21.42 |    16.66 |   33.33 |   21.42 |
  App.tsx                  |     100 |       50 |     100 |     100 | 31
  index.tsx                |       0 |      100 |     100 |       0 | 6-16
  reportWebVitals.ts       |       0 |        0 |       0 |       0 | 3-10
 src/Components            |   19.41 |     4.04 |   23.21 |      20 |
  AddMedicare.tsx          |       0 |        0 |       0 |       0 | 11-178
  DocumentUpload.tsx       |   21.42 |     7.14 |   16.66 |   21.95 | 36-41,46-58,63-111,248
  Documents.tsx            |     100 |      100 |     100 |     100 |
  Footer.tsx               |   72.72 |       50 |      60 |   72.72 | 7,17,24
  Header.tsx               |   34.78 |     12.5 |      50 |   34.78 | 25-44,59-62
  Home.tsx                 |     100 |      100 |     100 |     100 |
  MainDashboard.tsx        |     100 |      100 |     100 |     100 |
  MedicalProfile.tsx       |       0 |      100 |       0 |       0 | 6
  MedicareInfo.tsx         |       0 |      100 |       0 |       0 | 39
  NewHere.tsx              |     100 |      100 |     100 |     100 |
  OffCanvasNavMenu.tsx     |       0 |        0 |       0 |       0 | 18-32
  PageNotFound.tsx         |      50 |      100 |       0 |      50 | 4
  PersonalInfo.tsx         |     100 |      100 |     100 |     100 |
  PrivacyPolicy.tsx        |       0 |      100 |       0 |       0 | 4
  SignIn.tsx               |     100 |      100 |     100 |     100 |
  SignUp.tsx               |     100 |      100 |     100 |     100 |
  SignUpProcess.tsx        |       0 |        0 |       0 |       0 | 8-36
  TermsConditions.tsx      |       0 |      100 |       0 |       0 | 4
  UpdateMedicalProfile.tsx |       0 |      100 |       0 |       0 | 4
  UserProfile.tsx          |       0 |        0 |       0 |       0 | 11-26
 src/Context               |    6.81 |        0 |    9.09 |    4.76 |
  AppContext.tsx           |       0 |      100 |       0 |       0 | 10-15
  UploadActions.tsx        |       0 |      100 |       0 |       0 | 6-10
  UserActions.tsx          |       0 |        0 |       0 |       0 | 6-88
  UserContext.tsx          |     100 |      100 |      50 |     100 |
 src/Forms                 |   15.29 |        0 |   20.45 |   15.66 |
  FormMedicalProfile.tsx   |       0 |        0 |       0 |       0 | 10-230
  FormPersonalInfo.tsx     |   13.41 |        0 |   33.33 |   13.58 | 71-76,80-88,92-186,223
  FormSignIn.tsx           |   31.81 |        0 |   33.33 |   32.55 | 36-41,57-101
  FormSignUp.tsx           |   26.92 |        0 |   33.33 |   27.45 | 39-44,60-118
 src/Util                  |     100 |      100 |     100 |     100 |
  util.tsx                 |     100 |      100 |     100 |     100 |
---------------------------|---------|----------|---------|---------|-----------------------

Test Suites: 2 passed, 2 total
Tests:       9 passed, 9 total
Snapshots:   0 total
Time:        7.248 s
Ran all test suites.
```

Figure 11 11. Results of frontend unit tests.

Postman Tests

- Creating a user with an existing email fails with a "email already exists" message



POST http://localhost:8080/api/users/register

```
{
    "username": "rabbit@rabbit.com",
    "password": "11111111"
}
```

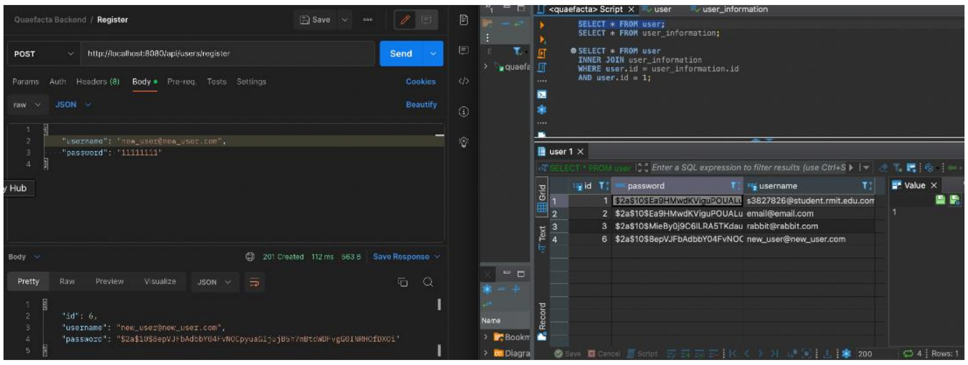
400 Bad Request 115 ms 476 B

```
{
    "username": "Email 'rabbit@rabbit.com' already exists"
}
```

```sql
SELECT * FROM user
INNER JOIN user_information
WHERE user.id = user_information.id
AND user.id = 1;
```

	id	password	username
1	1	$2a$10$Ea9HMwdKViguPOUALu	s3827826@student.rmit.edu.com
2	2	$2a$10$Ea9HMwdKViguPOUALu	email@email.com
3	3	$2a$10$MieBy0j9C6lLRA5TKdau	rabbit@rabbit.com

- Creating a user with a unique email creates an entry in the user table



Quaefacta Backend / Register

POST http://localhost:8080/api/users/register

```
{
    "username": "new_user@new_user.com",
    "password": "11111111"
}
```

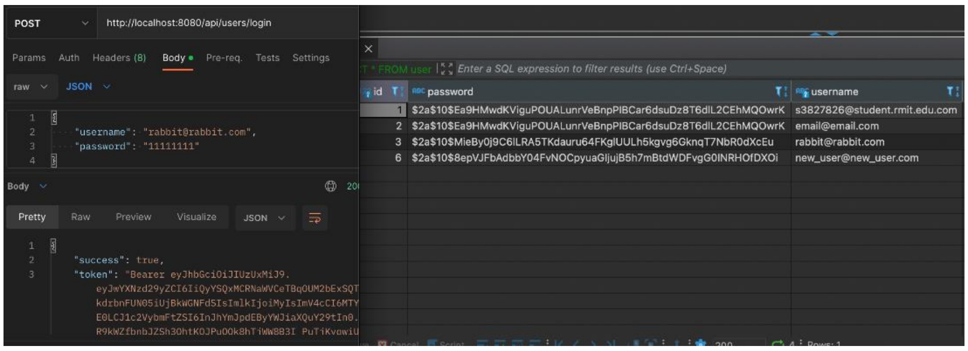
201 Created 112 ms 563 B

```
{
    "id": 6,
    "username": "new_user@new_user.com",
    "password": "$2a$10$8epVJFbAdbbY04FvNOCpyuaGljujB5h7mBtdWDFvgG0INRHOfDXOi"
}
```

```sql
SELECT * FROM user;
SELECT * FROM user_information;

SELECT * FROM user
INNER JOIN user_information
WHERE user.id = user_information.id
AND user.id = 1;
```

	id	password	username
1	1	$2a$10$Ea9HMwdKViguPOUALu	s3827826@student.rmit.edu.com
2	2	$2a$10$Ea9HMwdKViguPOUALu	email@email.com
3	3	$2a$10$MieBy0j9C6lLRA5TKdau	rabbit@rabbit.com
4	6	$2a$10$8epVJFbAdbbY04FvNOC	new_user@new_user.com

- Login. Logging in with valid credentials returns a success and token.



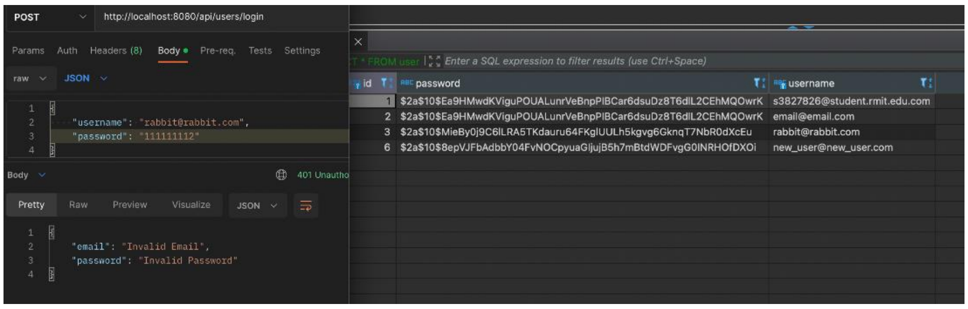
POST http://localhost:8080/api/users/login

```
{
    "username": "rabbit@rabbit.com",
    "password": "11111111"
}
```

```
{
    "success": true,
    "token": "Bearer eyJhbGciOiJIUzUxMiJ9.
        eyJwYXNzd29yZCI6IiQyYSQxMCRNaWVCeTBqOUM2bExSQT
        kdrbnFUN05iUjBkWGNFd5IsImlkIjoiMyIsImV4cCI6MTY
        E0LCJ1c2VybmFtZSI6InJhYmJpdEByYWJiaXQuY29tIn0.
        R9kWZfbnbJZSh3OhtKOJPuOOk8hTiWW8B3I PuTiKvowiU
```

id	password	username
1	$2a$10$Ea9HMwdKViguPOUALunrVeBnpPIBCar6dsuDz8T6dlL2CEhMQOwrK	s3827826@student.rmit.edu.com
2	$2a$10$Ea9HMwdKViguPOUALunrVeBnpPIBCar6dsuDz8T6dlL2CEhMQOwrK	email@email.com
3	$2a$10$MieBy0j9C6lLRA5TKdauru64FKglUULh5kgvg6GknqT7NbR0dXcEu	rabbit@rabbit.com
6	$2a$10$8epVJFbAdbbY04FvNOCpyuaGljujB5h7mBtdWDFvgG0INRHOfDXOi	new_user@new_user.com

- Logging in with invalid credentials gives an incorrect email or password error.



POST http://localhost:8080/api/users/login

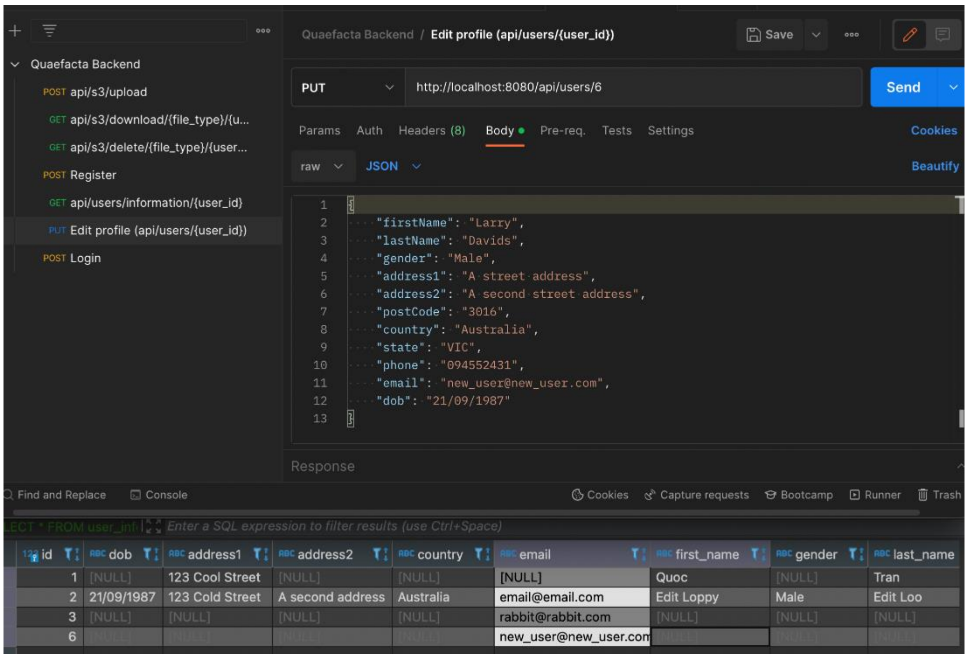
```
{
    "username": "rabbit@rabbit.com",
    "password": "111111112"
}
```

401 Unautho

```
{
    "email": "Invalid Email",
    "password": "Invalid Password"
}
```

id	password	username
1	$2a$10$Ea9HMwdKViguPOUALunrVeBnpPIBCar6dsuDz8T6dlL2CEhMQOwrK	s3827826@student.rmit.edu.com
2	$2a$10$Ea9HMwdKViguPOUALunrVeBnpPIBCar6dsuDz8T6dlL2CEhMQOwrK	email@email.com
3	$2a$10$MieBy0j9C6lLRA5TKdauru64FKglUULh5kgvg6GknqT7NbR0dXcEu	rabbit@rabbit.com
6	$2a$10$8epVJFbAdbbY04FvNOCpyuaGljujB5h7mBtdWDFvgG0INRHOfDXOi	new_user@new_user.com

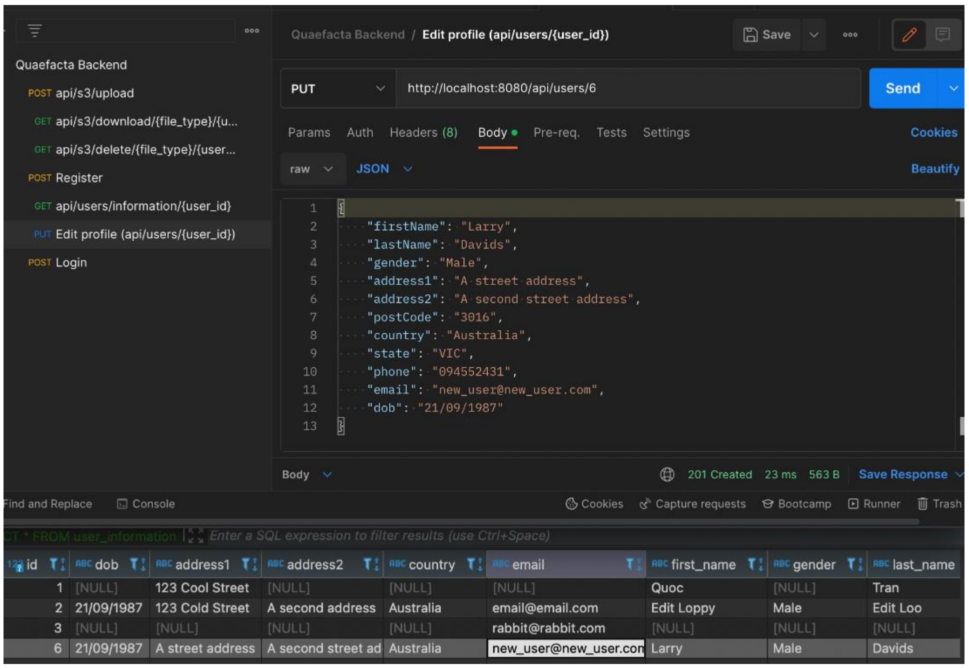
- Edit a user. Before we send the request to edit profile, we can see the fields for this user with email new_user@new_user.com are null. After we send the edit request, the fields are populated accordingly in the database.


Quaefacta Backend / Edit profile (api/users/{user_id})

PUT http://localhost:8080/api/users/6

```
{
    "firstName": "Larry",
    "lastName": "Davids",
    "gender": "Male",
    "address1": "A street address",
    "address2": "A second street address",
    "postCode": "3016",
    "country": "Australia",
    "state": "VIC",
    "phone": "094552431",
    "email": "new_user@new_user.com",
    "dob": "21/09/1987"
}
```

id	dob	address1	address2	country	email	first_name	gender	last_name
1	[NULL]	123 Cool Street	[NULL]	[NULL]	[NULL]	Quoc	[NULL]	Tran
2	21/09/1987	123 Cold Street	A second address	Australia	email@email.com	Edit Loppy	Male	Edit Loo
3	[NULL]	[NULL]	[NULL]	[NULL]	rabbit@rabbit.com	[NULL]	[NULL]	[NULL]
6	[NULL]	[NULL]	[NULL]	[NULL]	new_user@new_user.com	[NULL]	[NULL]	[NULL]


Quaefacta Backend / Edit profile (api/users/{user_id})

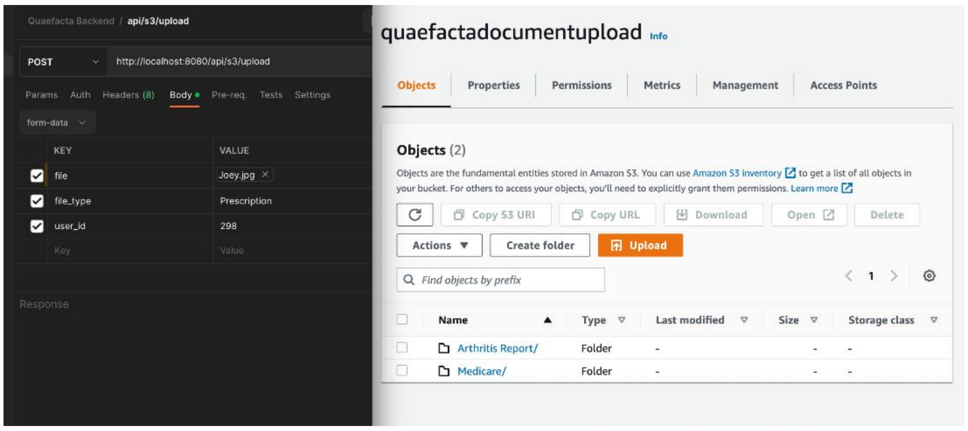
PUT http://localhost:8080/api/users/6

```
{
    "firstName": "Larry",
    "lastName": "Davids",
    "gender": "Male",
    "address1": "A street address",
    "address2": "A second street address",
    "postCode": "3016",
    "country": "Australia",
    "state": "VIC",
    "phone": "094552431",
    "email": "new_user@new_user.com",
    "dob": "21/09/1987"
}
```

201 Created 23 ms 563 B

id	dob	address1	address2	country	email	first_name	gender	last_name
1	[NULL]	123 Cool Street	[NULL]	[NULL]	[NULL]	Quoc	[NULL]	Tran
2	21/09/1987	123 Cold Street	A second address	Australia	email@email.com	Edit Loppy	Male	Edit Loo
3	[NULL]	[NULL]	[NULL]	[NULL]	rabbit@rabbit.com	[NULL]	[NULL]	[NULL]
6	21/09/1987	A street address	A second street ad	Australia	new_user@new_user.com	Larry	Male	Davids

- Uploading a new document type called “Prescriptions”. The following screens show the folder is created and the document placed inside the folder under the user_id.



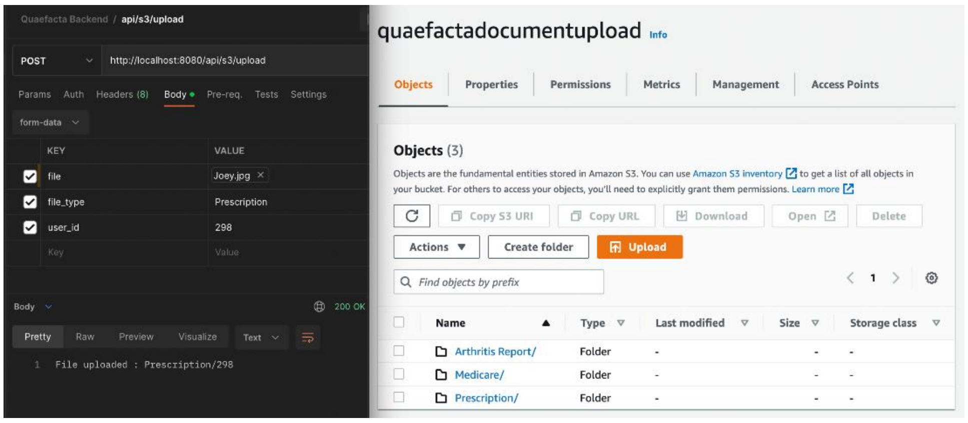
POST http://localhost:8080/api/s3/upload

KEY	VALUE
file	Joey.jpg
file_type	Prescription
user_id	298

Objects (2)

Objects are the fundamental entities stored in Amazon S3. You can use Amazon S3 inventory to get a list of all objects in your bucket. For others to access your objects, you'll need to explicitly grant them permissions. Learn more

Name	Type	Last modified	Size	Storage class
Arthritis Report/	Folder	-	-	-
Medicare/	Folder	-	-	-



POST http://localhost:8080/api/s3/upload

KEY	VALUE
file	Joey.jpg
file_type	Prescription
user_id	298

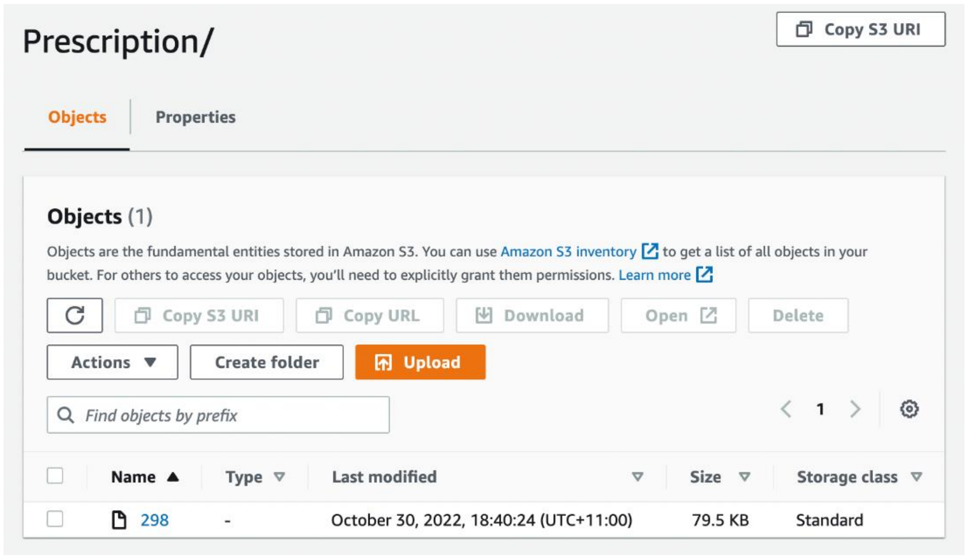
200 OK

```
File uploaded : Prescription/298
```

Objects (3)

Objects are the fundamental entities stored in Amazon S3. You can use Amazon S3 inventory to get a list of all objects in your bucket. For others to access your objects, you'll need to explicitly grant them permissions. Learn more

Name	Type	Last modified	Size	Storage class
Arthritis Report/	Folder	-	-	-
Medicare/	Folder	-	-	-
Prescription/	Folder	-	-	-



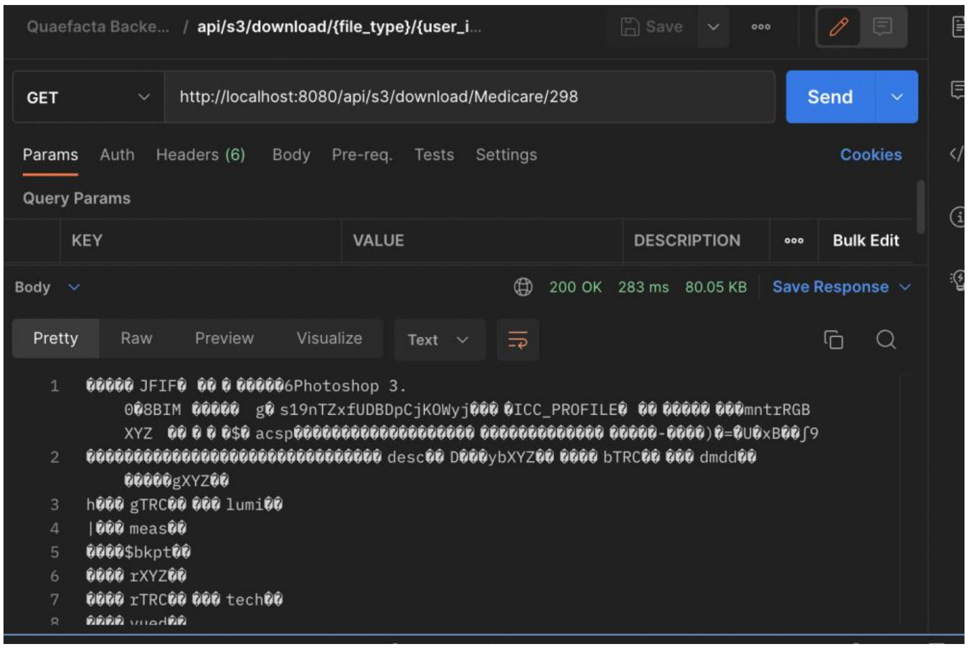
Prescription/

Objects (1)

Objects are the fundamental entities stored in Amazon S3. You can use Amazon S3 inventory to get a list of all objects in your bucket. For others to access your objects, you'll need to explicitly grant them permissions. Learn more

Name	Type	Last modified	Size	Storage class
298	-	October 30, 2022, 18:40:24 (UTC+11:00)	79.5 KB	Standard

- Get document. Returns the document if it exists



Quaefacta Backe... / api/s3/download/{file_type}/{user_i...

GET http://localhost:8080/api/s3/download/Medicare/298

Params | Auth | Headers (6) | Body | Pre-req. | Tests | Settings | Cookies

Query Params

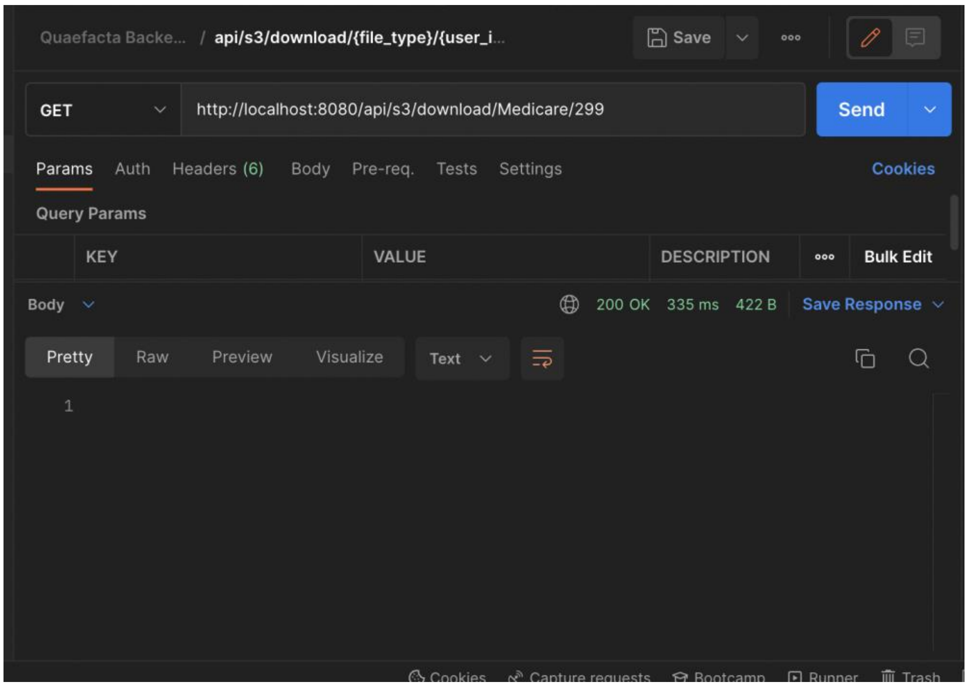
KEY	VALUE	DESCRIPTION

Body — 200 OK 283 ms 80.05 KB — Save Response

Pretty | Raw | Preview | Visualize | Text

```
����� JFIF�  �� � �����6Photoshop 3.
    0�8BIM  �����   g� s19nTZxfUDBDpCjKOWyj��� �ICC_PROFILE�  �� ����� ���mntrRGB
    XYZ  �� � � �$� acsp������������������� ������������ �����-����)�=�U�xB��ʃ9
������������������������������ desc�� D���ybXYZ�� ���� bTRC�� ��� dmdd��
    �����gXYZ��
h��� gTRC�� ��� lumi��
|��� meas��
����$bkpt��
���� rXYZ��
���� rTRC�� ��� tech��
���� vued��
```

- Get document returns nothing if it doesn’t exist



Quaefacta Backe... / api/s3/download/{file_type}/{user_i...

GET http://localhost:8080/api/s3/download/Medicare/299

Params | Auth | Headers (6) | Body | Pre-req. | Tests | Settings | Cookies

Query Params

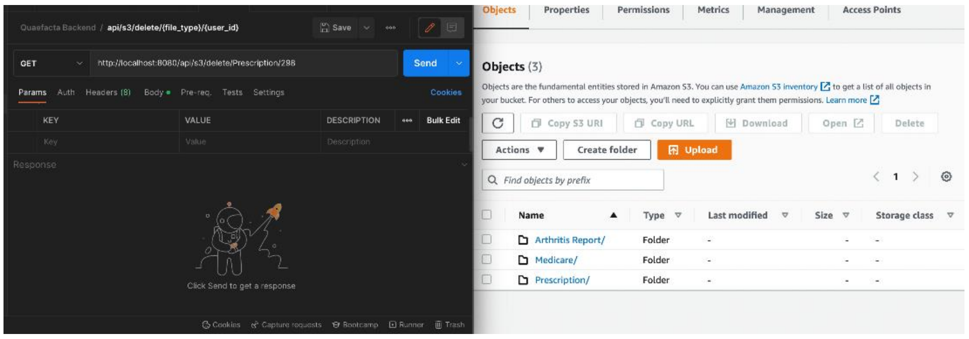
KEY	VALUE	DESCRIPTION

Body — 200 OK 335 ms 422 B — Save Response

Pretty | Raw | Preview | Visualize | Text

Cookies | Capture requests | Bootcamp | Runner | Trash

- Delete Document



Quaefacta Backend / api/s3/delete/{file_type}/{user_id}

GET http://localhost:8080/api/s3/delete/Prescription/298

Params | Auth | Headers (8) | Body | Pre-req. | Tests | Settings | Cookies

KEY	VALUE	DESCRIPTION
Key	Value	Description

Response

Click Send to get a response

Objects | Properties | Permissions | Metrics | Management | Access Points

Objects (3)

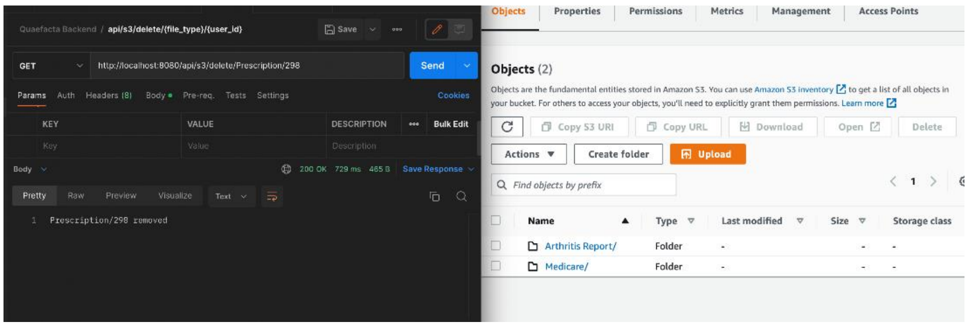
Objects are the fundamental entities stored in Amazon S3. You can use Amazon S3 inventory to get a list of all objects in your bucket. For others to access your objects, you'll need to explicitly grant them permissions. Learn more

Copy S3 URI | Copy URL | Download | Open | Delete | Actions | Create folder | Upload

Find objects by prefix

Name	Type	Last modified	Size	Storage class
Arthritis Report/	Folder	-	-	-
Medicare/	Folder	-	-	-
Prescription/	Folder	-	-	-

- Deletes the document when it exists. In this case there was only one document in the folder, so the folder is also removed.



GET http://localhost:8080/api/s3/delete/Prescription/298

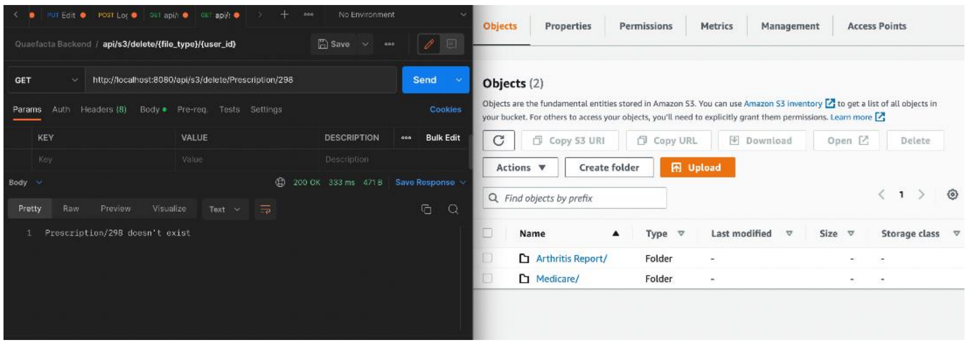
```
Prescription/298 removed
```

Objects (2)

Objects are the fundamental entities stored in Amazon S3. You can use Amazon S3 inventory to get a list of all objects in your bucket. For others to access your objects, you'll need to explicitly grant them permissions. Learn more

Name	Type	Last modified	Size	Storage class
Arthritis Report/	Folder	-	-	-
Medicare/	Folder	-	-	-

- Sending the request again returns a “File does not exist” message.



GET http://localhost:8080/api/s3/delete/Prescription/298

```
Prescription/298 doesn't exist
```

Objects (2)

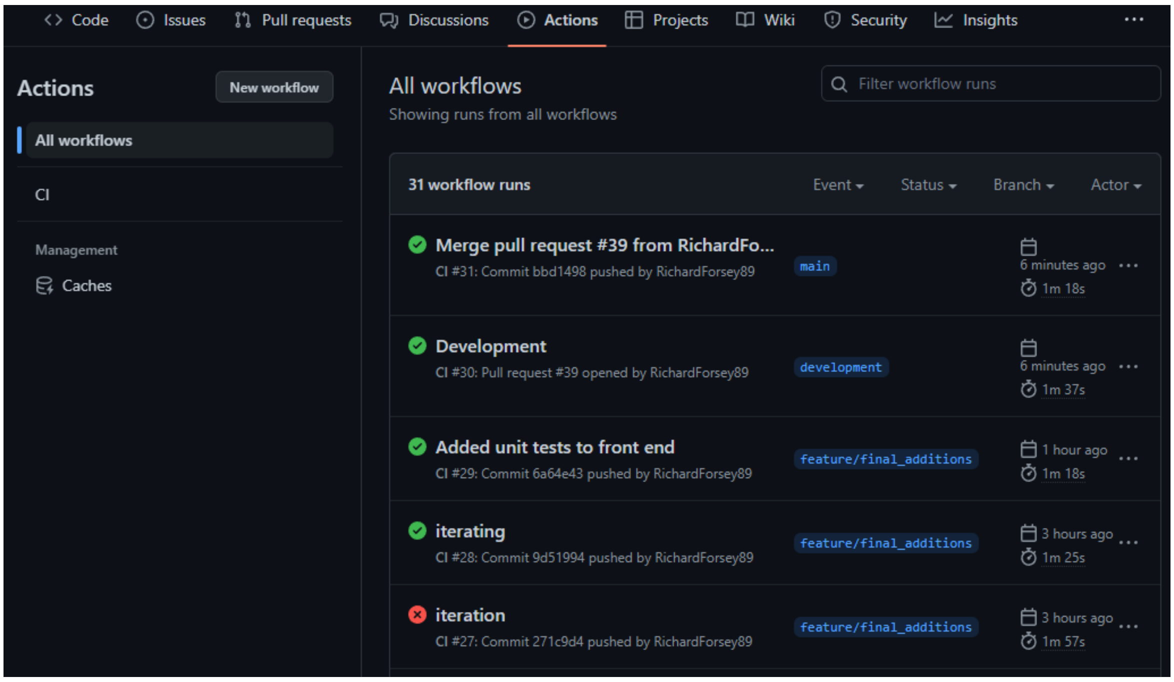
Objects are the fundamental entities stored in Amazon S3. You can use Amazon S3 inventory to get a list of all objects in your bucket. For others to access your objects, you'll need to explicitly grant them permissions. Learn more

Name	Type	Last modified	Size	Storage class
Arthritis Report/	Folder	-	-	-
Medicare/	Folder	-	-	-

9 – Other Items

CI Pipeline

It would be remiss to not mention the CI pipeline. It will run on the main, development, “feature/**” and “update/**” branch pushes, on pull requests to main and development, or can be manually triggered. Ideally, this pipeline should have been integrated with the AWS orchestration, to create a full CI/CD pipeline, but due to time constraints, this feature wasn’t able to be implemented.

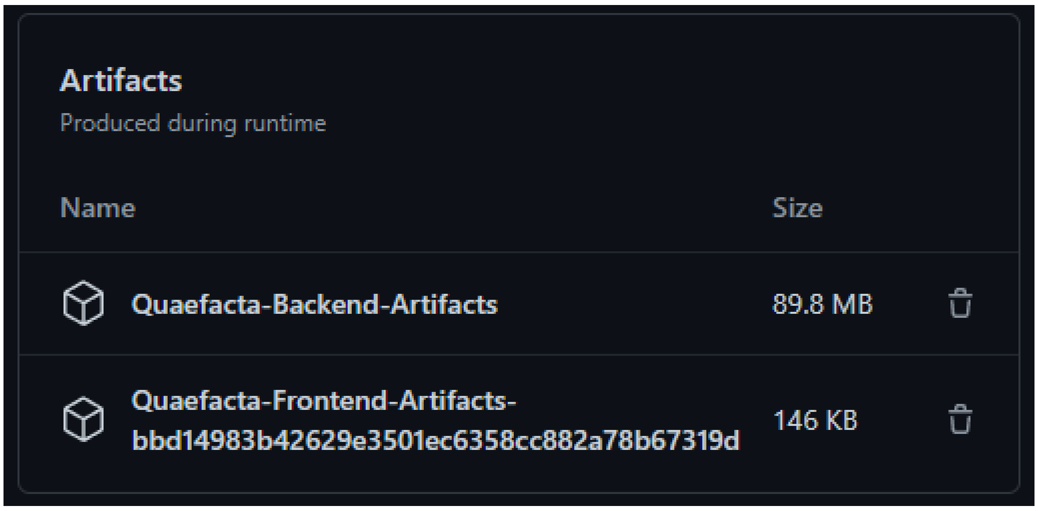


31 workflow runs

Run	Details	Branch	When	Duration
✓ Merge pull request #39 from RichardFo...	CI #31: Commit bbd1498 pushed by RichardForsey89	main	6 minutes ago	1m 18s
✓ Development	CI #30: Pull request #39 opened by RichardForsey89	development	6 minutes ago	1m 37s
✓ Added unit tests to front end	CI #29: Commit 6a64e43 pushed by RichardForsey89	feature/final_additions	1 hour ago	1m 18s
✓ iterating	CI #28: Commit 9d51994 pushed by RichardForsey89	feature/final_additions	3 hours ago	1m 25s
✗ iteration	CI #27: Commit 271c9d4 pushed by RichardForsey89	feature/final_additions	3 hours ago	1m 57s

Figure 1212. GitHub repo actions page showing the CI pipeline running with both successes and failures.

There are two jobs currently, one for the backend and one for the frontend. In both cases they have the same purpose: check the code via linting and then running test suites on the build. If the action is running on the main branch, both jobs will package the build artifacts into zips.



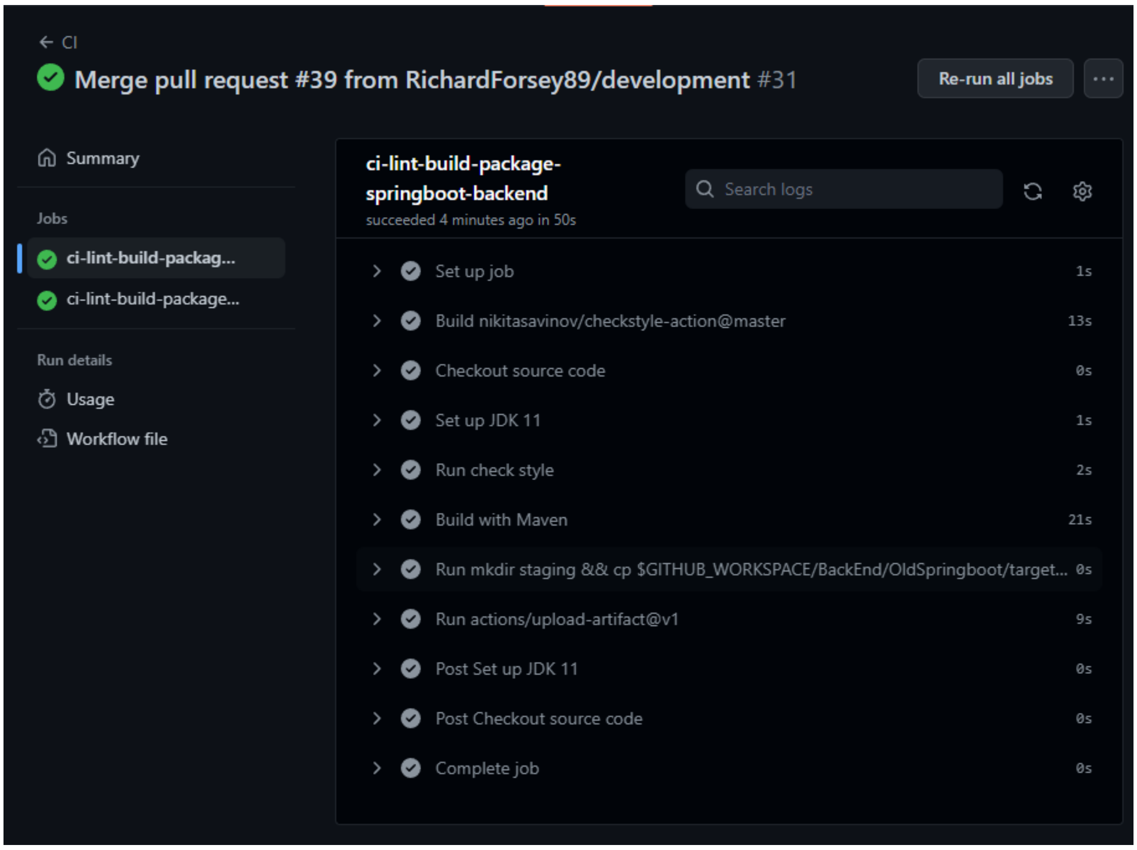
Artifacts

Produced during runtime

Name	Size
Quaefacta-Backend-Artifacts	89.8 MB
Quaefacta-Frontend-Artifacts-bbd14983b42629e3501ec6358cc882a78b67319d	146 KB

Figure 1313. Artifacts produced by package step on main branch.

To see the steps of the two pipeline jobs, please view the two figures on the following page.



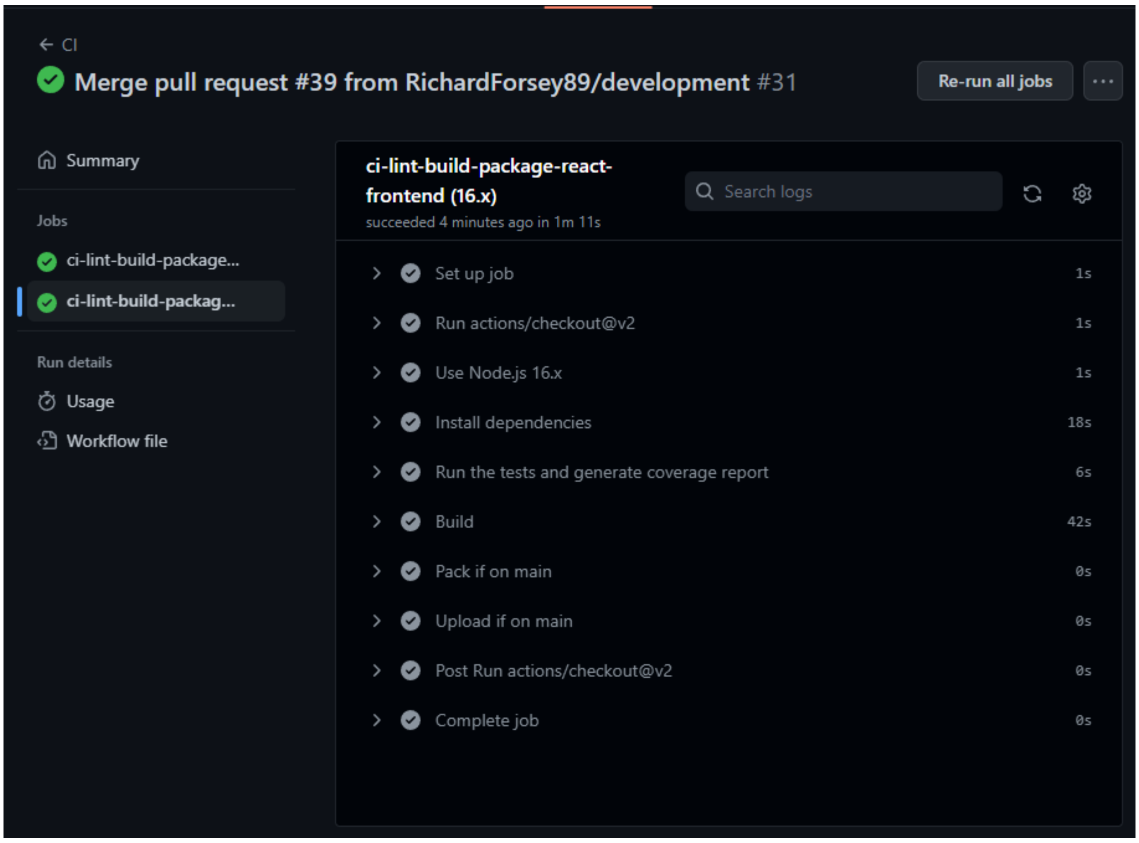
← CI

Merge pull request #39 from RichardForsey89/development #31

ci-lint-build-package-springboot-backend
succeeded 4 minutes ago in 50s

```
Set up job                                                              1s
Build nikitasavinov/checkstyle-action@master                           13s
Checkout source code                                                    0s
Set up JDK 11                                                           1s
Run check style                                                         2s
Build with Maven                                                       21s
Run mkdir staging && cp $GITHUB_WORKSPACE/BackEnd/OldSpringboot/target... 0s
Run actions/upload-artifact@v1                                          9s
Post Set up JDK 11                                                      0s
Post Checkout source code                                               0s
Complete job                                                            0s
```

Figure 1414. Steps of the backend job in the CI pipeline.



← CI

Merge pull request #39 from RichardForsey89/development #31

ci-lint-build-package-react-frontend (16.x)
succeeded 4 minutes ago in 1m 11s

```
Set up job                                     1s
Run actions/checkout@v2                        1s
Use Node.js 16.x                               1s
Install dependencies                          18s
Run the tests and generate coverage report     6s
Build                                         42s
Pack if on main                                0s
Upload if on main                              0s
Post Run actions/checkout@v2                   0s
Complete job                                   0s
```

Figure 1515. Steps of the frontend job in the CI pipeline.

10 – References

We have made no citations in this report, so this section is intentionally blank.