

National College of Ireland

Coronavirus Vaccination Management System - Technical Report

Cloud Application Development

April 2021

BSc (Honours) in Computing

Software Development

2020/2021

Aaron Reilly - X17124719

[X17124719@student.ncirl.ie](mailto:X17124719@student.ncirl.ie)

Table of Contents

[Introduction 2](#_Toc69024925)

[Development Strategy 2](#_Toc69024926)

[Database Design 2](#_Toc69024927)

[Implementation 3](#_Toc69024928)

[Design Patterns 4](#_Toc69024929)

[Observer Design Pattern 4](#_Toc69024930)

[Strategy Design Pattern 5](#_Toc69024931)

[Testing 5](#_Toc69024932)

[Unit Testing 5](#_Toc69024933)

[Functional Tests 6](#_Toc69024934)

[Integration Testing 6](#_Toc69024935)

[System Testing 7](#_Toc69024936)

[Deployment 8](#_Toc69024937)

[Bibliography 9](#_Toc69024938)

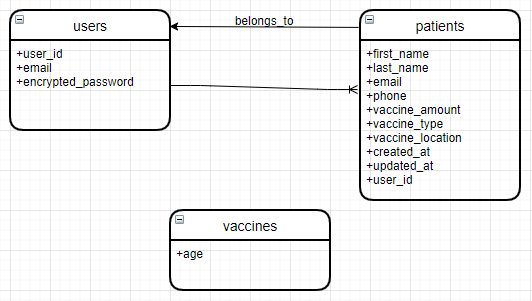
# Introduction

Coronavirus vaccination management system provides end users with the ability to add, update and delete patient details. Keeping track of who got vaccinated, at what location and what type of vaccination was used, with how many doses is information which needs to be kept. The application allows information such as First Name, Last Name, Email, Phone, Vaccine Amount, Vaccine Type and Vaccine Location to be stored on the database. An account can only see patients they have added to the system. Other functionalities include add, update, delete to the system while allowing users to search for patient by name. Account use cases are creating account, edit account, cancel account, sign in and sign out. Sessions and cookies are used throughout the application. Outside live data is also retrieved in json format. Running the class, it writes most recent data to its corresponding txt file this can be found in the data folder in the project. Custom gem has also been implemented allowing outside users to enter in their age in return they receive a time frame of when the vaccine is expected to be available to them.

# Development Strategy

## Database Design

Database contains three tables, names of the tables are users, patients, and vaccines. Two of the three tables have a relationship between which is a one-to-many relationship. Seen in the diagram below a user can have many patients. Using devise users has many Patients, while Patients belong to Users. Age table has no relationship with the other tables, this is used by our custom gem, end users without sign in privilege can enter in their age and get a time frame of when their vaccination is expected to be available.



**Users Table Commands**

* rails generate devise user
* rails db:migrate
* rails routes

**Patients Table Commands**

* rails g scaffold patients first\_name:string last\_name:string email:string phone:string vaccine\_amount:string vaccine\_type:string vaccine\_location:string
* rails db:migrate

**Vaccines Table Commands**

* rails g scaffold vaccines age:int
* rails db:migrate

## Implementation

The edit registration is implemented using an MVC approach. The functionality is implemented in the following files: vaccineApp/app/views/devise/registrations/edit.html.erb

The new registration is implemented using an MVC approach. The functionality is implemented in the following files: vaccineApp/app/views/devise/registrations/new.html.erb

The new session is implemented using an MVC approach. The functionality is implemented in the following files: vaccineApp/app/views/devise/sessions/new.html.erb

The edit password is implemented using an MVC approach. The functionality is implemented in the following files: vaccineApp/app/views/devise/passwords/edit.html.erb

The new password is implemented using an MVC approach. The functionality is implemented in the following files: vaccineApp/app/views/devise/passwords/new.html.erb

The patient functionalities show, new, create, update, search are implemented using an MVC approach. The functionality is implemented in the following files: vaccineApp/app/controllers/patients\_controller.rb

The partial patient form html is implemented using an MVC approach this feature is rendered on other pages. The functionality is implemented in the following files: vaccineApp/app/views/patients/\_form.html.erb

The edit patient html is implemented using an MVC approach, renders the partial form. The functionality is implemented in the following files: vaccineApp/app/views/patients/edit.html.erb

The display patients html is implemented using an MVC approach, allows the user to view, search and delete existing patients. The functionality is implemented in the following files: vaccineApp/app/views/patients/index.html.erb

The new patient html is implemented using an MVC approach, renders the partial patients form. The functionality is implemented in the following files: vaccineApp/app/views/patients/new.html.erb

The show patient html is implemented using an MVC approach, displays individual patient details, along with options to edit, delete and back to index. The functionality is implemented in the following files: vaccineApp/app/views/patients/show.html.erb

The partial alert html is implemented using an MVC approach, displays alerts when an action is taken on the application. The functionality is implemented in the following files: vaccineApp/app/views/layouts/alert.html.erb

The partial footer html is implemented using an MVC approach, displays the footer content which is used throughout the application. The functionality is implemented in the following files: vaccineApp/app/views/layouts/footer.html.erb

The isvaccinated custom gem is used to allow user who are not signed in to get a time frame of when the vaccination is available for their age group. The functionality is implemented in the following files: vaccineApp/isvaccinated/lib/isvaccinated.rb

The isvaccinated html is implemented using an MVC approach, displays the age and timeframe of the expected vaccination availability. The functionality is implemented in the following files: vaccineApp/app/views/vaccines/isvaccinated.html.erb

## Design Patterns

### Observer Design Pattern

Observer design pattern is used to broadcast new amounts of the vaccine doses for a patient. By running ruby displayer.rb in the observer folder. We can see a patient has initially one vaccine. When we run the command mentioned, we then notify observers with the new amount for that patient.

Files can be found:

* vaccineApp/observer/displayer.rb
* vaccineApp/observer/patients.rb
* vaccineApp/observer/subject.rb

### Strategy Design Pattern

Took the open/closed approach for our strategy design pattern. Taking this approach allows us to add a class or strategy every time we wish to add a new patient report format. Each strategy takes in the same method, outputting the report in a different format.

We report patient name, location, and vaccine amount in different formats such as json, html and plain text. Run ruby strategy-pattern.rb in the strategy folder you should get a patient in json format. Edit parameter on line 38 in strategy-pattern.rb to another class name available line 20 and 32, rerun the command mentioned, you should see patient report in the corresponding format.

Files can be found:

* strategy/strategy-pattern.rb

# Testing

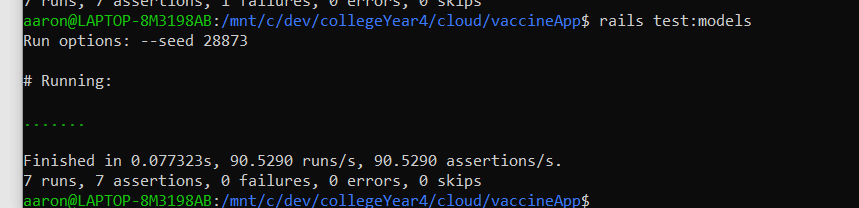
## Unit Testing

The approach taken for unit tests were to test all patient fields in which are mandatory. Our first test we run an assert on all fields to ensure if all fields are added the patient will be entered into the database. Our next tests we use an assert\_not, removing one fields to ensure the patient won’t be added if a given field is left blank. To run these tests locally I had to disable authentication on line 3 in file patients\_controller.rb and belongs\_to on line 2 in file patient.rb in models folder. All tests pass as expected seen in the results screenshot below.

Unit Tests



Unit Test Results

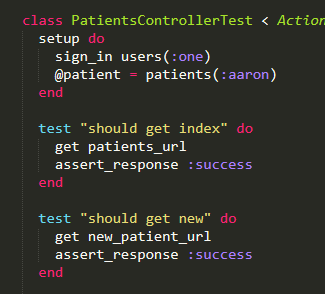


## Functional Tests

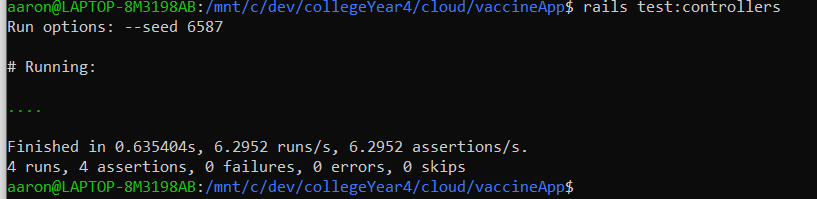
After adding devise to my project, functional tests started to fail. By adding a user using “sign\_in”, in our setup method resolved in some tests to pass. Test patients were also used in the functional tests. Feeding in patient aaron which is available in vaccineApp/test/fixtures/patients.yml.

Functional tests cover mostly the expected url for index, new, show and edit. Removed update create and delete as they kept failing since devise got added. These tests are covered in the system tests.

Functional Tests

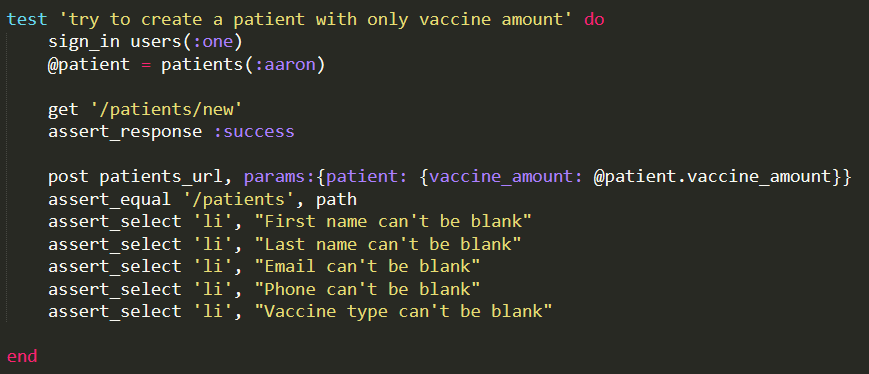


Functional Test Results



## Integration Testing

Integration tests were implemented with how our end users would use the application in mind. Integration tests work like functional tests, here I was able to cover areas I couldn’t in our functional tests. If a field in the form is missing, we expect an error message to return in a list. Here we’re ensuring the correct message are returned also validating that all mandatory fields are throwing an error. We try to add a patient with just vaccine amount, then in the next test we try to add a patient with everything but the vaccine amount.



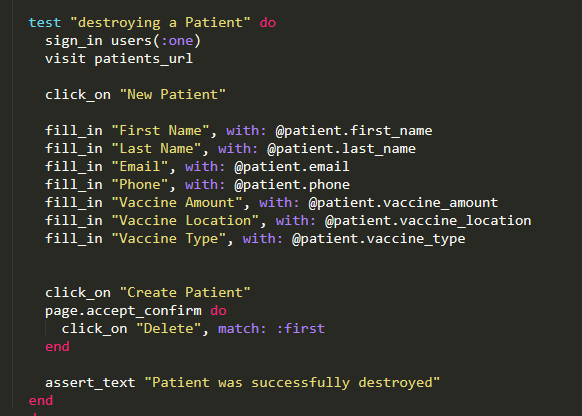
## System Testing

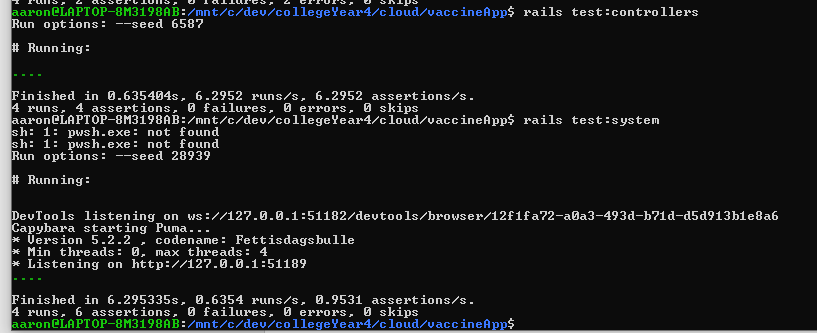
System tests are using selenium driver to run full browser testing on our project. Implementation of these tests were based on how our end users would use the application in mind. System tests are covering areas which are not covered in our unit, functional, and integration tests, this is why I believe we have good coverage in our tests.

Test’s coverage include:

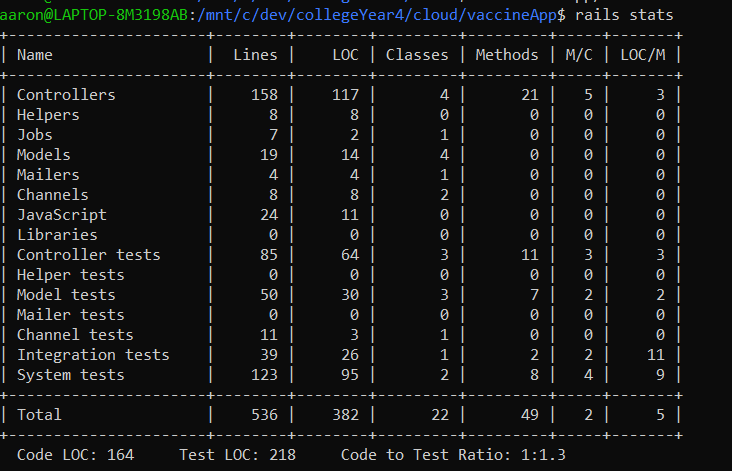
* visit index after user signs in
* creating a Patient,
* updating a Patient,
* deleting a Patient

System Tests



System Test Results

Rails stats



# Deployment

**Cloud Service:** Heroku

Successfully deployed my application and data needed to get full functionality on Heroku. Being very new to cloud platforms I chose to go with the provider covered in the module. Only change needed to be made on my production environment was to switch databases. To deploy the full application successfully, we need to use Postgres database.

Our application is deployed to Heroku our Production environment is using Postgres database.

**Review three platforms**

|  |  |  |
| --- | --- | --- |
| **Platform** | **Advantages** | **Disadvantages** |
| AWS | * Widely used in the IT industry * Wide range of features * Cheaper than competitors (Azure, Google) | * Lack of knowledge haven’t covered it in modules. * Technical support fees * Some security limitations |
| Azure | * Provides good scalability options. * Allows you to use any language, framework, or tools of your chosen | * Cost slightly more than AWS * Expertise is required |
| Google Cloud | * Improvement on the performance * Provides security at a high level. * Cheaper than competitors (AWS, Azure) | * Provides less features compared to other providers. * Inexperienced compared to other major providers. |

**Project url:** <https://rails-vaccine.herokuapp.com/users/sign_in>

**User Accounts**

|  |  |
| --- | --- |
| **Username** | **Password** |
| SarahFoley@gmail.com | cloudTest |
| JohnTest@gmail.com | cloudTest123 |

# Bibliography

Guru99.com. 2021. Google Cloud vs AWS: Difference Between AWS and GCP. [online] Available at: <https://www.guru99.com/google-cloud-vs-aws.html> [Accessed 9 April 2021].

Jackson, B., 2021. Top 7 Advantages of Choosing Google Cloud Hosting (2021). [online] Kinsta. Available at: <https://kinsta.com/blog/google-cloud-hosting/> [Accessed 9 April 2021].

Gaille, B., 2021. 15 Microsoft Azure Advantages and Disadvantages. [online] BrandonGaille.com. Available at: <https://brandongaille.com/15-microsoft-azure-advantages-and-disadvantages/> [Accessed 9 April 2021].

DataFlair. 2021. AWS Advantages & Disadvantages | Advantages of Cloud Computing - DataFlair. [online] Available at: <https://data-flair.training/blogs/aws-advantages/> [Accessed 9 April 2021].

European Centre for Disease Prevention and Control. 2021. ECDC. [online] Available at: <https://opendata.ecdc.europa.eu/covid19> [Accessed 9 April 2021].