

Name: **Rathnayake M Rathnayake**

Student Reference Number: **10747887**

Module Code: **PUSL3120**

Module Name: **Full stack development**

Coursework Title: **Project Report**

Deadline Date: **01/02/2022**

Member of staff responsible for coursework: **Dr. Mark Dixon**

Programme: **BSc (Hons) Computer Science**

Please note that University Academic Regulations are available under Rules and Regulations on the University website [www.plymouth.ac.uk/studenthandbook](http://www.plymouth.ac.uk/studenthandbook).

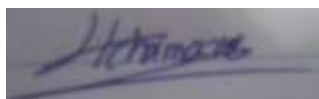
Group work: please list all names of all participants formally associated with this work and state whether the work was undertaken alone or as part of a team. Please note you may be required to identify individual responsibility for component parts.

***We confirm that we have read and understood the Plymouth University regulations relating to Assessment Offences and that we are aware of the possible penalties for any breach of these regulations. We confirm that this is the independent work of the group.***

Signed on behalf of the group:

Individual assignment: ***I confirm that I have read and understood the Plymouth University regulations relating to Assessment Offences and that I am aware of the possible penalties for any breach of these regulations. I confirm that this is my own independent work.***

Signed:



Use of translation software: failure to declare that translation software or a similar writing aid has been used will be treated as an assessment offence.

I \*have used/not used translation software.

If used, please state name of software.....

**Overall mark** \_\_\_\_\_ **%**      **Assessors Initials** \_\_\_\_\_      **Date** \_\_\_\_\_

## Table of Contents

Table Of Figures.....	1
1.INTRODUCTION.....	1
2.DESIGN .....	2
2.1 System Architecture.....	2
2.2 User Interactions.....	3
2.3 Domain Layer .....	4
2.4 Application Structure.....	5
2.4.1 Runtime.....	5
2.4.2 Application design.....	5
2.5 Advantages of Having Clean System Architecture.....	9
3.TESTING .....	10
3.1 Backend services testing.....	10
3.2 API Testing.....	12
4. DevOps .....	13
4.1 Development Environment.....	13
4.2 Version Control.....	13
4.3 Continues Integration and Development pipeline.....	14
5. PERSONAL REFLECTION.....	16
APPNDIX.....	17
Video Demonstration .....	17
GitHub Repository .....	17
Remote Host.....	17
Project Proposal.....	18
End.....	19

## Table Of Figures

Figure 2. 1 Use case Diagram. ....	3
Figure 2. 2 ER Diagram. ....	4
Figure 2. 3 Class Diagram Backend services.....	5
Figure 2. 4 Class Diagram Controllers.....	6
Figure 2. 5 Routes mapping. ....	6
Figure 2. 6 Backend folder structure .....	7
Figure 2. 7 Class Diagram Frontend Services.....	8
Figure 2. 8 View components. ....	9
Figure 3. 1 Test Cases .....	10
Figure 3. 2 Unit testing implementation .....	11
Figure 3. 3 Test results. ....	11
Figure 3. 4 Single API call to get all reports.....	12
Figure 4. 1 VSCode running on WSL. ....	13
Figure 4. 2 Cloud Build, build trigger.....	14
Figure 4. 3 Cloud Build, dashboard. ....	14
Figure 4. 4 Cloud Build, build summary.....	14
Figure 4. 5 app.yaml.....	15

## 1.INTRODUCTION

The world we live in is a dangerous place, and it's becoming increasingly important to ensure that our communities are safe. Crime watch is a web application based on incident management system that can help us do just that. With this innovative tool, registered users can report any criminal activities or any illegal incidents they witness directly to the proper authorities for investigation and judgment. This innovative tool provides an effective way for the law enforcement officials to investigate and judge the outcome of these reported crimes. The user-friendly interface makes it easy for anyone to quickly register and submit their reports with just a few clicks of the mouse.

Crime watch allows users to easily document their experiences with crime by providing an easy-to-use interface for creating reports of suspicious activity or other violations of the law. This allows citizens who have witnessed something illegal taking place around them to quickly alert local police departments so they can investigate further before more damage is done in their community. Additionally, registered users may add additional evidence such as photos or videos (the current version of Crime watch only supports photos) if available which helps provide even more information about what happened during the incident which will aid police officers when conducting investigations into these matters.

Crime Watch provides both citizens and law enforcement officers with an effective way of combating crime within our neighborhoods while also giving people peace of mind knowing there's another layer always protecting them from harm thanks to its user-friendly reporting system built right into its platform. It makes sense why this type of technology should be adopted across all cities as soon as possible. not only does it make everyone safer but also gives people greater confidence knowing those who break laws won't get away free anymore due its powerful tools allowing justice to be served accordingly when needed most.

Overall, Crime Watch is an essential security tool that helps promote safety within communities by providing individuals with real time updates regarding local crime trends while enabling authorities investigate suspicious activity faster than ever before through its comprehensive incident reporting system feature set. With more additional features added to the system, it could become increasingly popular among both citizens and law enforcement agencies, due to its great potential when used correctly.

## 2.DESIGN

This web application is a MEAN stack application, that is an innovative web application that combines MongoDB, Express.js, AngularJS and Node.js into a single, powerful web application. Using the four components of the MEAN stack together allows to create dynamic websites with interactive front-end functionality quickly and easily.

### 2.1 System Architecture.

The system architecture of Crime Watch includes following layers, each with a specific purpose and function.

1. Server layer: This is the base layer of the web application. For the development of the application, it has been built in the localhost and local environment. In the production ready build, the application will be hosted in google cloud.
2. Application layer: This is the processing layer of the web application. It is responsible for handling user communicating with the backend. In this MEAN stack application expressJs has been used. Addition to that, the application uses TypeScript, a superset of JavaScript.
3. Data layer: To store the data and retrieve data to the web application a database is required. This is the layer that handles the domain level communication with the application layer. For this application a mongodb database has been used.
4. Presentation layer: This is the user interface layer of the web application. It is responsible for displaying information to the user and accepting input. This is build using Angular.

## 2.2 User Interactions.

This application has two roles, Witness and Moderator. Witness can submit a report of an incident, and the report will be visible to the public after a moderator goes through a procedure to confirm the report. The confirmed report will be available for the public so any other registered witnesses can add more evidence to the report as they witness something else.

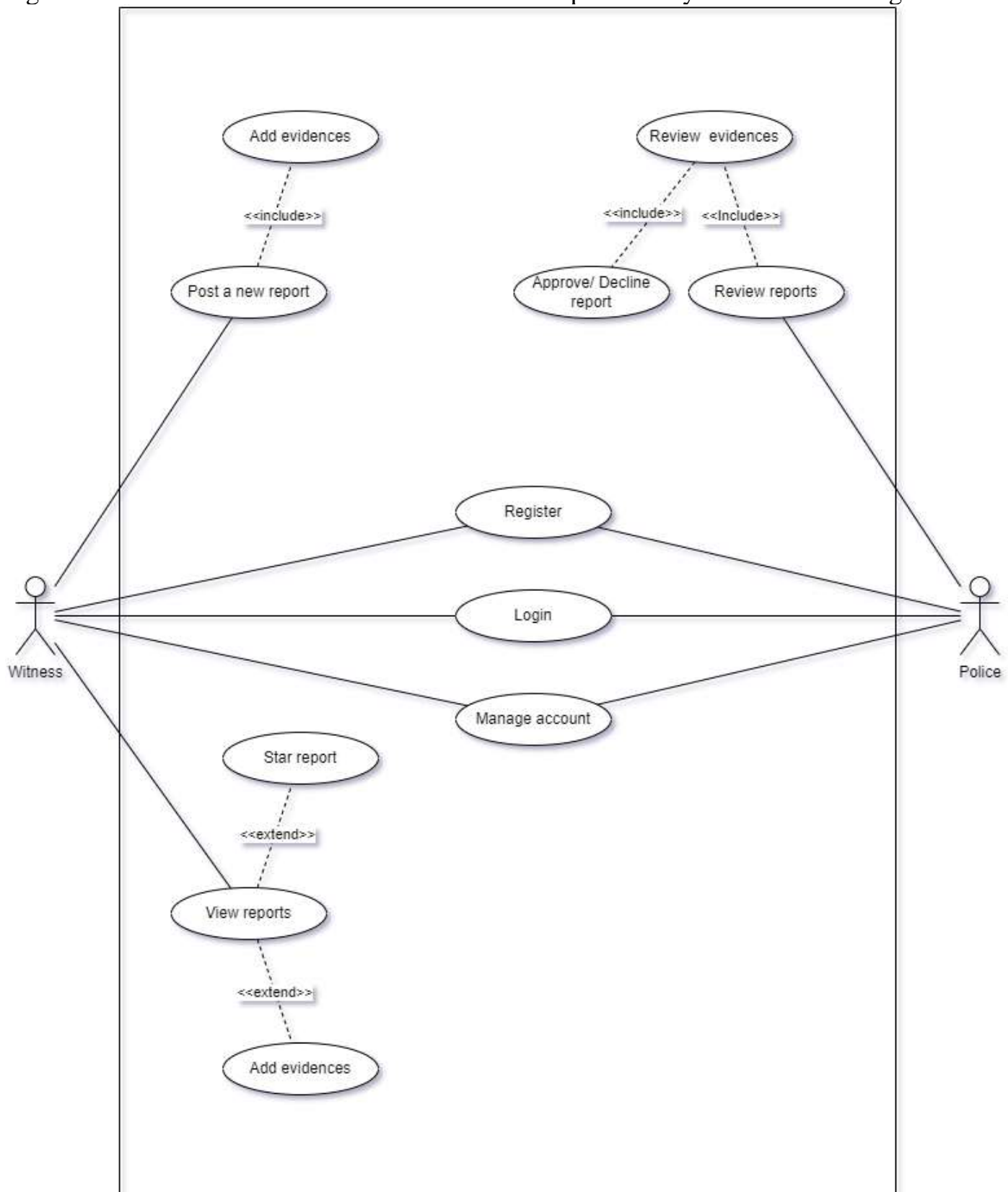


Figure 2. 1 Use case Diagram.

Addition to that witness can star the reports, so they will notify on updates on the stored report.

## 2.3 Domain Layer

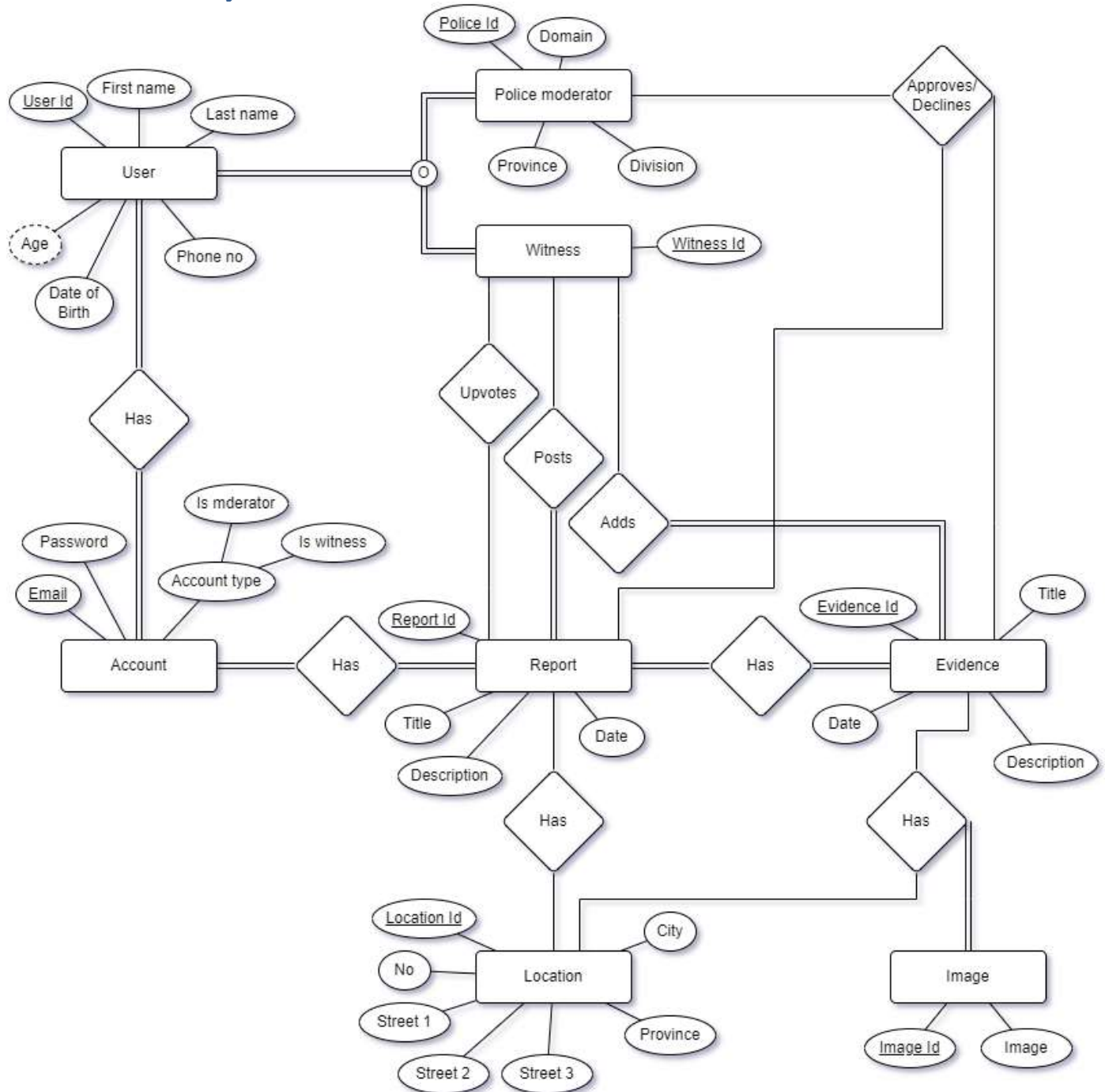


Figure 2. 2 ER Diagram.

The figure 2.2 shows how the data are mapped in this application. Even though this application used non-relational database somewhat relationship can be achieved in the data. As for an example, The User model can embed Account model.

```
class User {
  FirstName!: string;
  LastName!: string;
  Gender!: Gender;
  DOB!: Date;
  Age!: number;
  PhoneNumber!: number;
  Account!: Account;
}
```

## 2.4 Application Structure.

### 2.4.1 Runtime.

The runtime of this application is Node.js: The backend code is written in TypeScript. The application uses a package called ts-node, and it allows to run TypeScript code directly without the need to first transpile it to JavaScript, which can make development faster and more efficient. To run the server, the application used ExpressJs. Using types for express allows to code the requests and responses, more type strict way.

### 2.4.2 Application design

#### 2.4.2.1 Backend

As for the design, this uses several design patterns to promote separation of concern. Model-View-Controller (MVC) and repository pattern are used in this application.

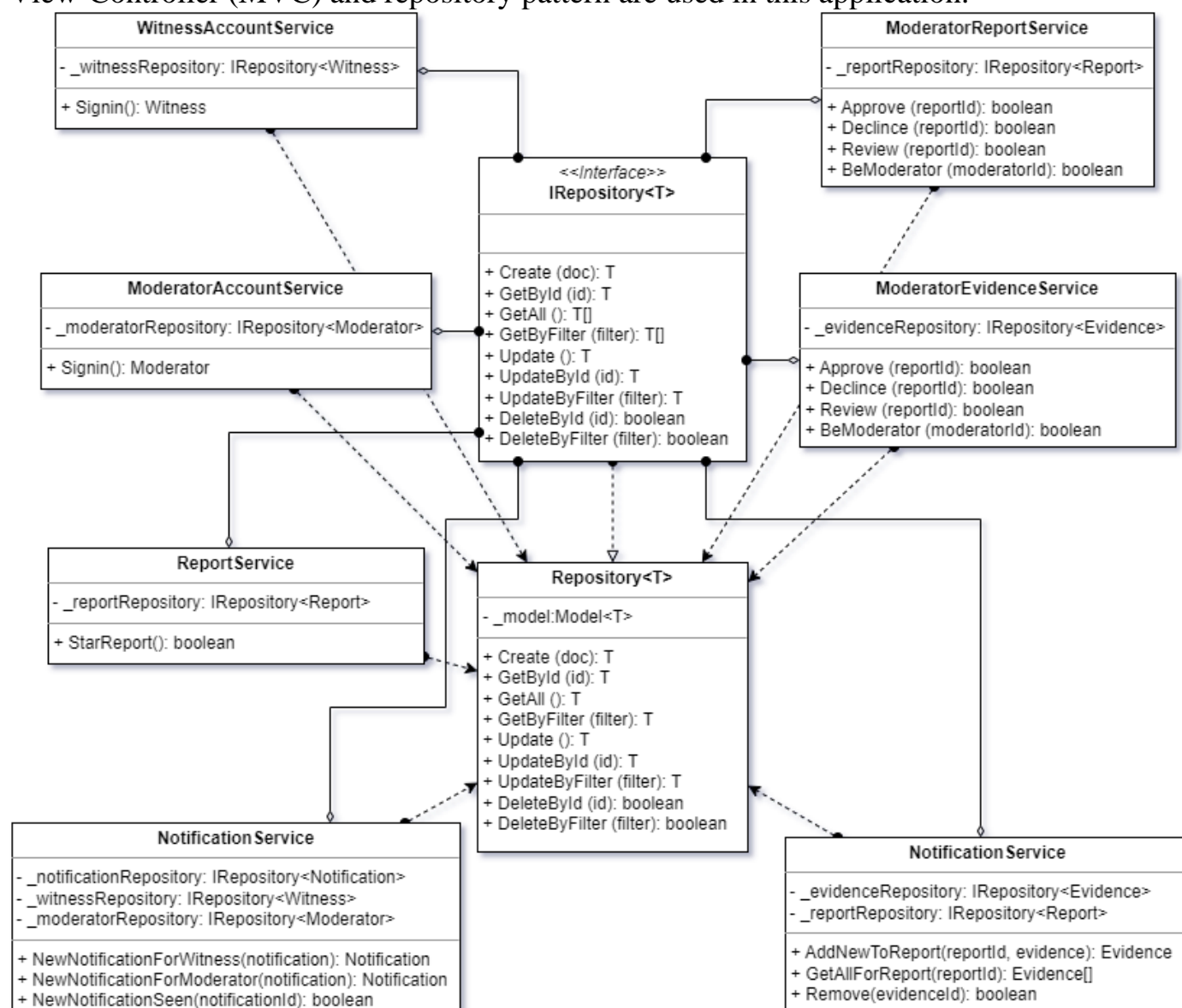


Figure 2. 3 Class Diagram Backend services.

The figure 2.3 shows the class diagram of the backend services that implements the repository pattern. The repository is a generic repository, so it can be used on every model to create a separate instance of repository. These services are connected with the models that is showing



in figure 2.2 and the models are connected with the mongodb. So, through services and repository, domain level operations can be done against the database.

The controllers are called by the routes that defined in the application. When the client reaches a specific API end point, it fires up an instance of the controller.

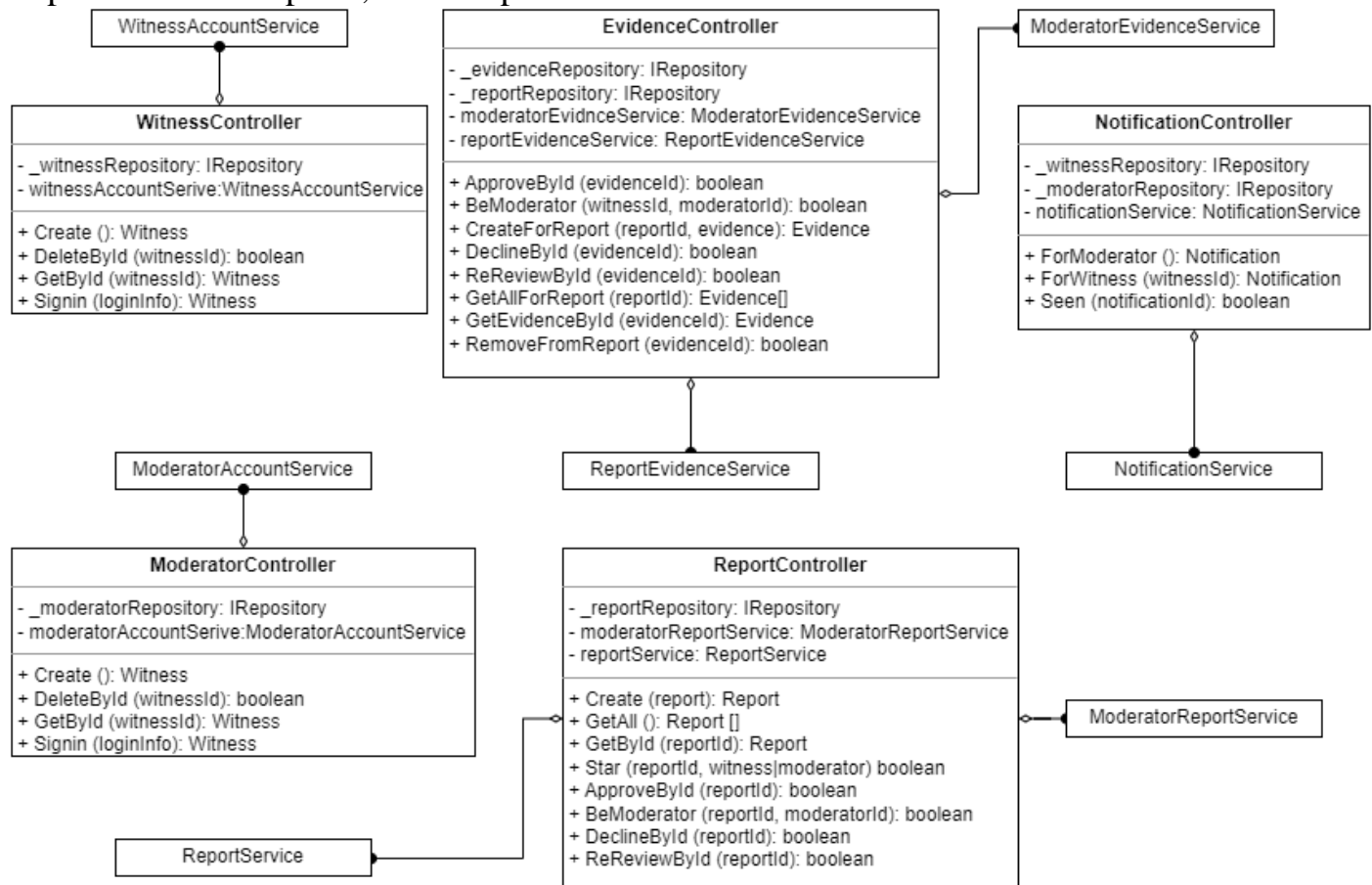


Figure 2.4 Class Diagram Controllers.

As can be seen in the figure 2.4 once the controllers are reached, they call specific repositories and services.

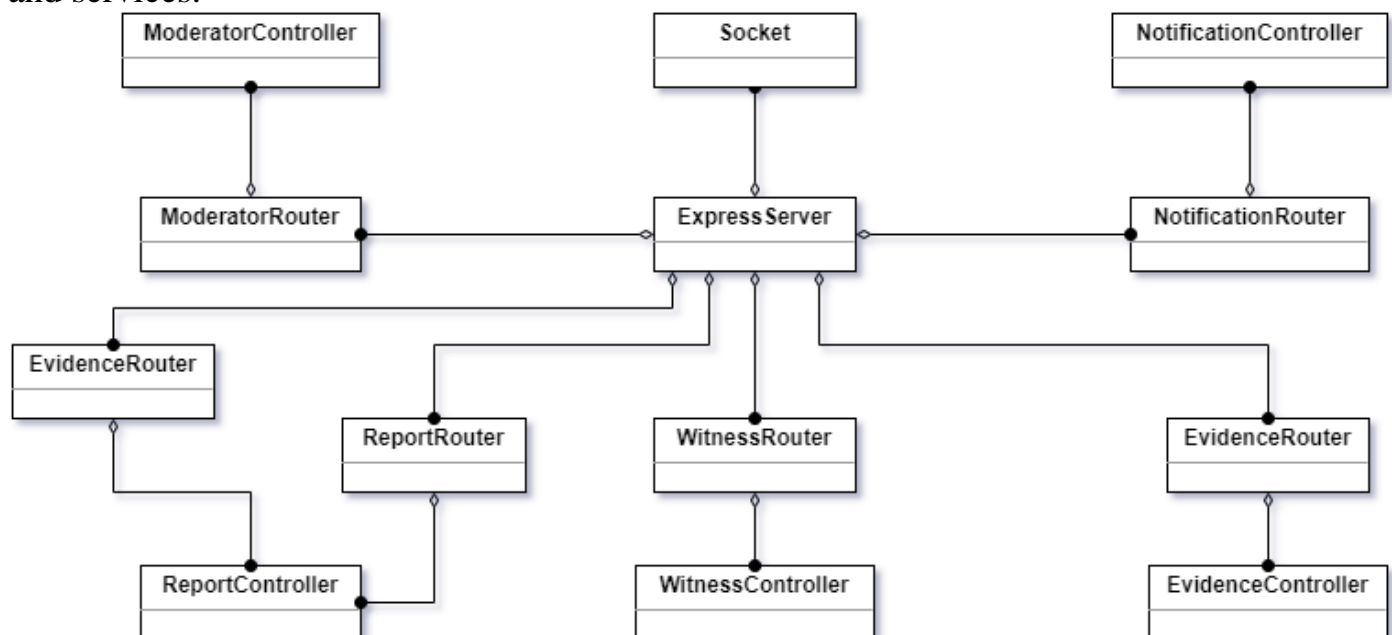


Figure 2.5 Routes mapping.

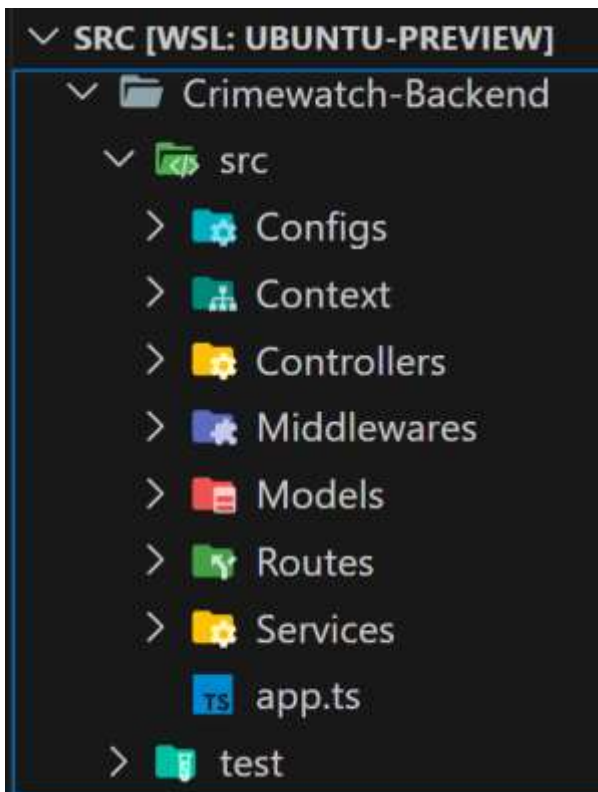


Figure 2. 6 Backend folder structure

### 2.4.2.1 Frontend

As for the view, this MEAN stack project uses Angular front-end. Angular is a powerful JavaScript framework that has powerful features such, data binding, dependency injection and templating.

To get the advantages of dependency injection, a service layer has been implemented in the angular application.

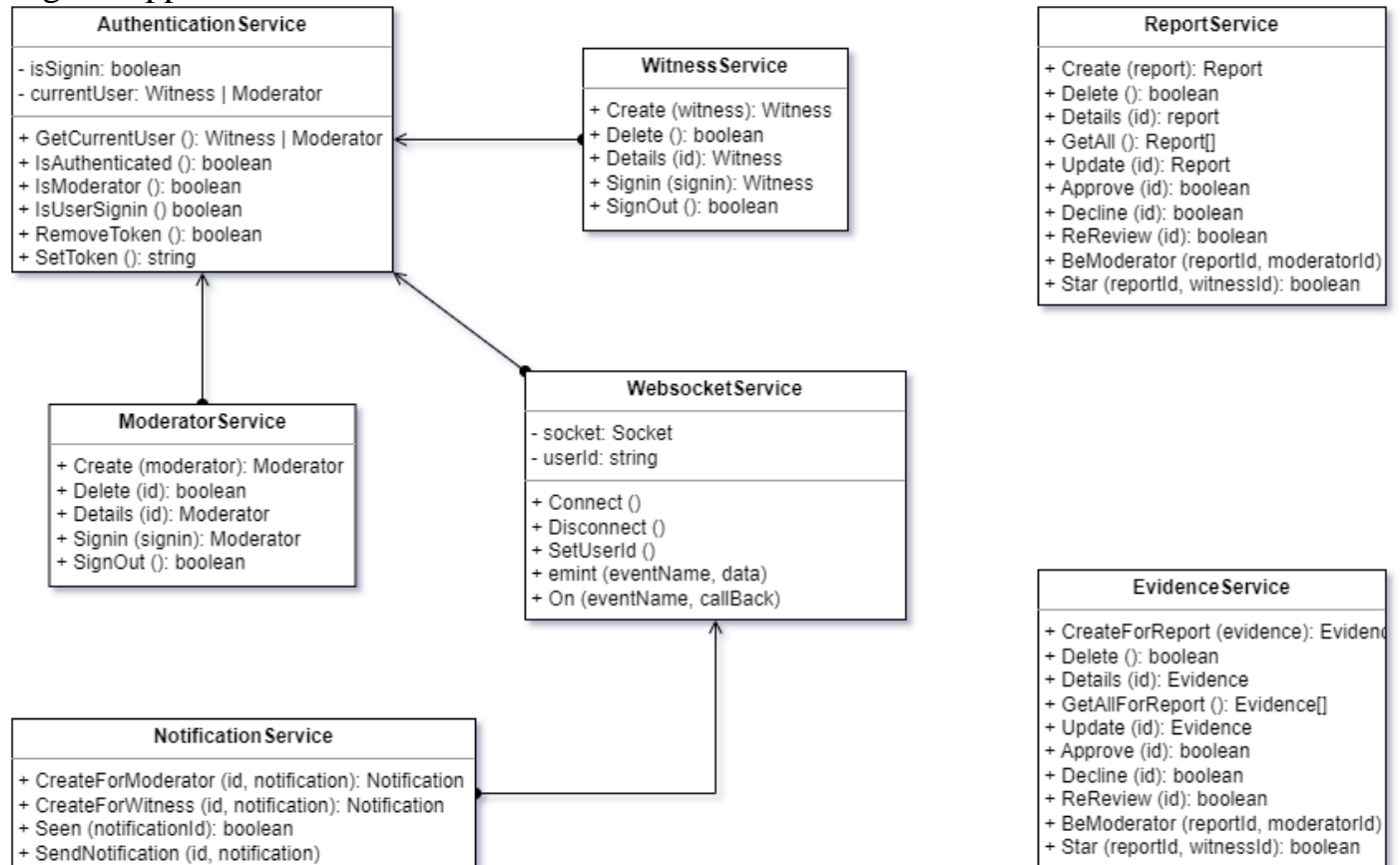


Figure 2. 7 Class Diagram Frontend Services.

Each services that can be seen in the figure 2.6, has a property called `HttpClient` expect for the **AuthenticationService**. So these services can call the rest API to communicate between the server and the frontend. Using dependency injections these services can be inject to the frontend view components. Lastly, **AuthenticationService** will store and retrieve the token that contains the current user details to authenticate the user to automatically sign in or, call the protected APIs.

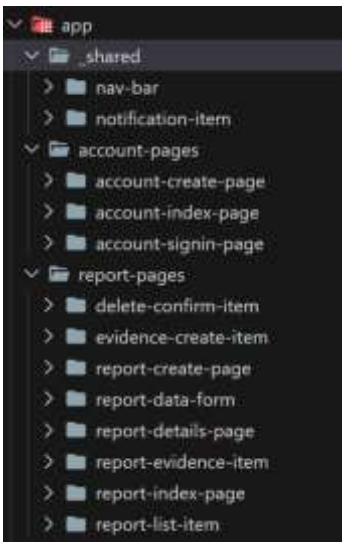


Figure 2. 8 View components.

The view components are grouped as figure 2.7. Additionally, Angular materials are used in the application.

#### 2.4.2.1 Serving the frontend to backend.

Finally, the angular frontend can be served in the express server.

```
app.use(
  express.static(
    path.join(
      __dirname,
      "../../Crimewatch-Frontend/dist/crimewatch-frontend"
    )
  )
);
```

After running “npm start” in the Crimewatch-Frontend directory, the built version of the angular app will deploy in the “dist” directory, so it can be used in the express server.

## 2.5 Advantages of Having Clean System Architecture.

**Easier maintenance:** A well-designed architecture makes it easier to understand the system's structure and makes it simpler to make changes or fix bugs.

**Better scalability:** A clean architecture allows for the easy addition of new features and the ability to handle increased load.

**Improved code reusability:** A modular architecture makes it easier to reuse code across different parts of the system.

### 3.TESTING

This application promotes separation of concerns, allowing for easy implementation of unit testing. Each component can be tested separately without having to worry about interference from other components. In this coursework, Unit Testing has been used and implemented to test the services layer which uses repository pattern. This allows for easier verification of functionality within the service by testing a single instance model as a model repository. By separating each component and using Unit Testing, developers are able to easily verify that their applications are functioning properly with minimal effort required on their part.

#### 3.1 Backend services testing.

To implement unit testing chai and faker npm packages have been used. The followings are testcases for the service layer.

Test Case Id	Test Description	Prerequisite	Input Data	Expected Result	Actual Result	Stat
1	Connecting to the local test database	Must have install mongo-server in the local system	Connection string: mongodb://localhost/IMSTEST	The returned connection should not be null	Same as expected	Pass
2	Creating a new account for witness	Database should be connected	Faker generated: User:{ Witness: {faker generated} }	should return the user and should match the data with the actual data	Same as expected	Pass
3	Creating a new account for moderator	Database should be connected	Faker generated: User:{ Moderator: {faker generated} }	should return the user and should match the data with the actual data	Same as expected	Pass
4	Create a report	1. A database should be connected 2. Witness account should be exist 3. Get the witness Id	Faker generated: Report:{ WitnessId:witnessId }	Report should be posted and should be returned	Same as expected	Pass
5	Add evidences	1. A database should be connected 2. Witness account should be exist 3.Report should be existed and should return the report id	Faker generated: evidence:{ Report:reportId WitnessId:witnessId }	Evidence should be posted and should be returned	Same as expected	Pass
6	Approve a report as moderator	1. A database should be connected 2. Moderator account should be exist 3.Report should be existed and should return the report id	Update the report with moderator id { ModeratorId: moderatorId }	Report should be approved and return boolean true	Same as expected	Pass

Figure 3. 1 Test Cases

The Chai and Faker packages are two of the most popular tools for implementing unit testing. They provide a wide range of test cases that can be used to ensure that all components of the service layer are working correctly. The tests provided by these packages include assertions, mocks, spies, and stubs which help to validate each component's functionality as well as any potential bugs or errors in the codebase.

The following is a single usage in the unit testing process.

```
0 references
describe(description: "Report services test", specDefinitions: () => {
  0 references
  it(expectation: "Should get a witness id and create a new report", assertion: async () => {
    const _witnessRepository = new Repository<WitnessDocument>({
      _model: WitnessModel
    });
    const _reportRepository = new Repository<ReportDocument>(_model: ReportModel);

    const witnessId = await (await _witnessRepository.GetAll()).pop()?._id;
    const newReport = reportPayload;
    newReport.Author = witnessId;

    const insertReport = await _reportRepository.Create(doc: newReport);
    expect(val: newReport.Body).to.equals(value: insertReport.Body);
  });
});
```

Figure 3. 2 Unit testing implementation

```
Connected to mongodb
  ✓ Should connect to the test db (71ms)

create a new witness
{
  User: {
    FirstName: 'Willy',
    LastName: 'Haley',
    DOB: 1972-12-22T15:10:48.598Z,
    Age: 89,
    PhoneNumber: 7139039281,
    Account: {
      Email: 'Godfrey.Heidenreich61@gmail.com',
      Password: '4PJEQ73NZkgbQne',
      IsModerator: false
    },
    Gender: 'Female'
  }
}
  ✓ Should insert a witness and returns the document (77ms)

create a new moderator
  ✓ Should insert a moderator and returns the document

Report services test
  ✓ Should get a witness id and create a new report (127ms)

Adding evidences to report
new ObjectId("63d1fdd7c09e3a6bb8ce51ea")
  ✓ Should add a new evidence to report (147ms)

Test moderator services
  ✓ Should approve the report (127ms)

6 passing (595ms)
```

Figure 3. 3 Test results.

### 3.2 API Testing.

To test the API of the project it is more into manual than automated. Test the API will verify the functionality of the backend routes and controllers. To achieve these testing, an API tools can be used. In this project a vscode extension called “Thunder client” has been used to test API calls.

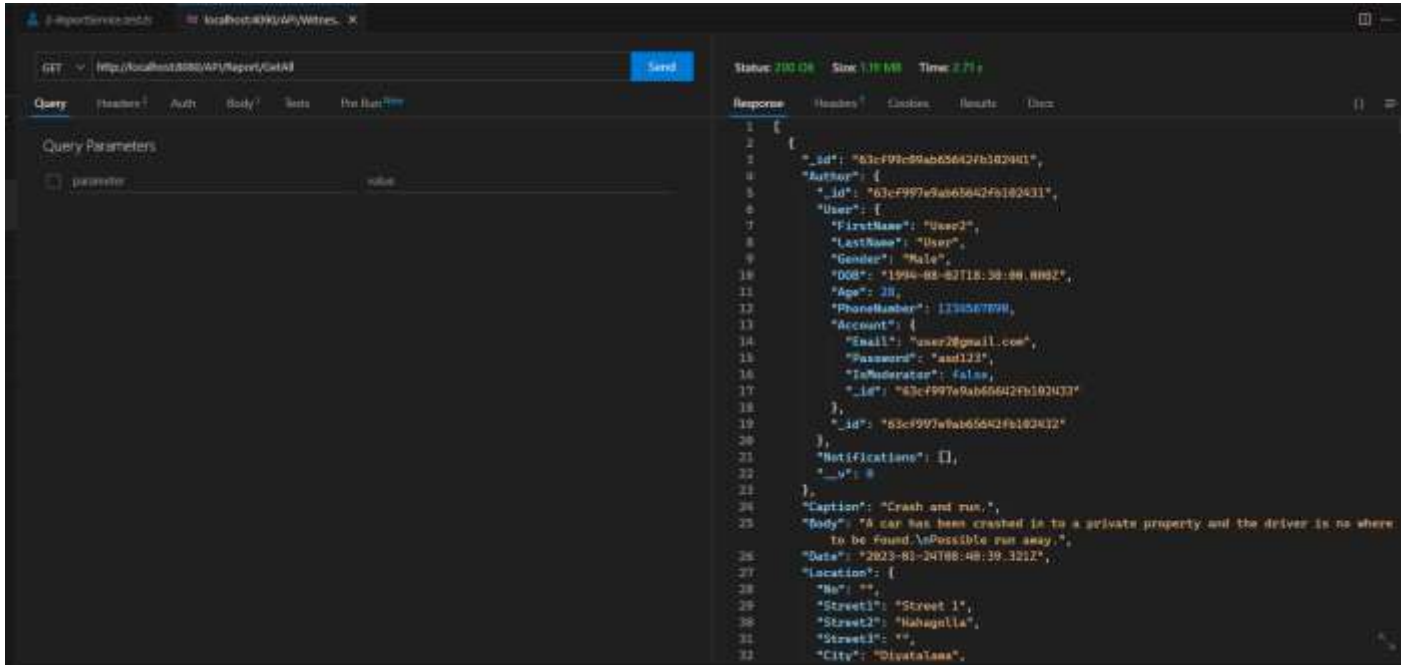


Figure 3. 4 Single API call to get all reports.



## 4. DevOps

### 4.1 Development Environment.

The development environment for a Mean Stack project is now easier than ever before with the Windows Subsystem for Linux (WSL) Ubuntu Preview. This allows developers to debug their projects in both Windows and Linux file systems, making it much more efficient and convenient.

Visual Studio Code is the preferred IDE as it offers all the provides all the necessary tools needed to develop a Mean Stack project such as Node NPM extensions. The integrated terminal within VSCode allows to access the localhost server running in WSL, which makes development process easier.

With this setup, developers can easily create powerful applications that are cross-platform compatible without having to worry about any compatibility issues or other difficulties associated with using different operating systems simultaneously.



Figure 4. 1 VSCode running on WSL.

### 4.2 Version Control.

A mono-repository is a single repository for all the code related to a project, including both frontend and backend. This approach allows to easily manage source code in one place, making it easier to track changes. The advantages of using version control are that it helps keep track of every change in the project's codebase over time, allowing to work more efficiently without having duplicate versions. Additionally, version control makes continuous delivery/continuous integration (CD/CI) much simpler as this project uses automated cloud building. Using GitHub these advantages can be achieved.



### 4.3 Continues Integration and Development pipeline.

Continuous Integration and Development pipeline of this Mean Stack project is using Google Cloud. It is a great way to ensure that the application is updated with the latest version. By connecting to a GitHub repository, Google Cloud Build can be used to automate builds, tests and deployments. It could be trigger whenever an event occurs in the repository.

Filter	Enter property name or value					?	III
Name ↑	Description	Repository	Event	Build configuration	Status		
Build-Deploy	-	haritha99ch/Crime-Watch	Push to branch	src/cloudbuild.yaml	Disabl	RUN	:

Figure 4. 2 Cloud Build, build trigger.



Figure 4. 3 Cloud Build, dashboard.

Steps		Duration	Build Log	Execution Details	Build Artifacts
Build Summary (3 steps)		00:10:21			
0	gcr.io/cloud-builders/gcp	00:01:41			
1	gcr.io/cloud-builders/gcp	00:01:18			
2	gcr.io/cloud-builders/gcp	00:08:16			
gcp-deploy --project crime-watch-375407					
Step 0: gcr.io/cloud-builders/gcp		00:01:41			
Step 1: gcr.io/cloud-builders/gcp		00:01:18			
Step 2: gcr.io/cloud-builders/gcp		00:08:16			
gcp-deploy --project crime-watch-375407					

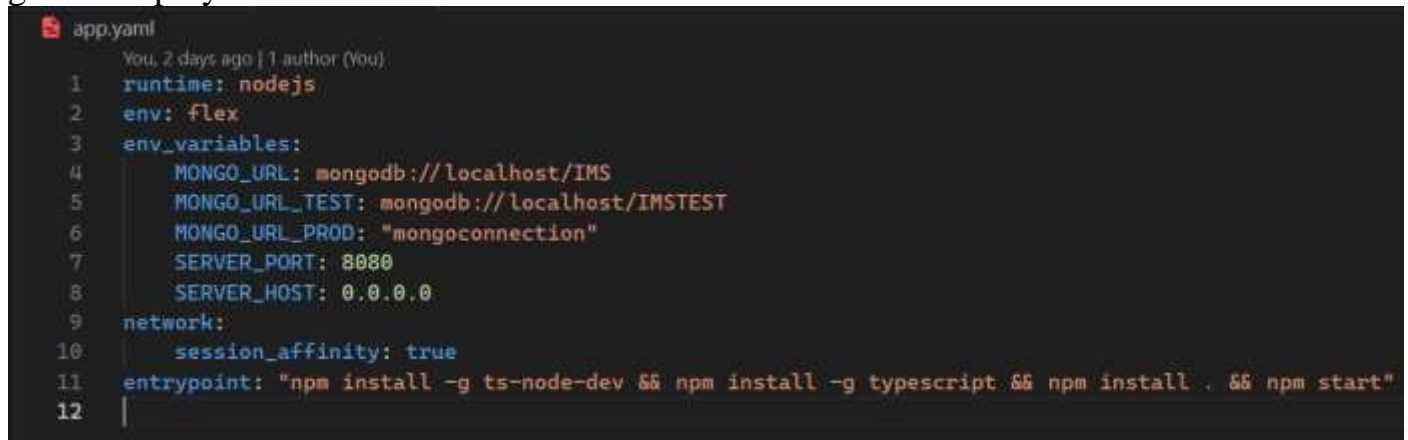
Figure 4. 4 Cloud Build, build summary.

To achieve these automated tasks, a remote environment is required such as Google cloud, Travis Ci. After creating a remote environment, a file can be used to configure the automation depend on the remote environment. In this case a cloudbuild.yaml can be used. The file contains the steps to be executed to build the project. Following is a single step in the file.

```
steps:
- name: "gcr.io/cloud-builders/gcloud"
  dir: "/workspace/src"
  args: ["app", "deploy", "--project", "crime-watch-375407"]
```

The file should be stored in the root directory of the repository or the path to cloudbuild.yaml should be specified in the remote environment.

To execute the deployment a similar processes can be used with a different file. In this project gcloud deploy has been used.



```
app.yaml
You, 2 days ago | 1 author (You)
1 runtime: nodejs
2 env: flex
3 env_variables:
4     MONGO_URL: mongodb://localhost/IMS
5     MONGO_URL_TEST: mongodb://localhost/IMSTEST
6     MONGO_URL_PROD: "mongoconnection"
7     SERVER_PORT: 8080
8     SERVER_HOST: 0.0.0.0
9 network:
10     session_affinity: true
11 entrypoint: "npm install -g ts-node-dev && npm install -g typescript && npm install . && npm start"
12 |
```

Figure 4. 5 app.yaml

Both steps will install the required dependencies and run the build and deploy it to the cloud.

## 5. PERSONAL REFLECTION

The project uses repository pattern, which I have learned from DotNet and using to DotNet projects. After researching and understanding the concept of a generic type repository pattern in the DotNet environment, I decided to implement it in this MEAN stack project. Programmatically speaking, it worked well not surprisingly due to its efficient design and structure. This implementation has allowed us to make use of the best practices within software development while creating a robust system that is easy for developers to work with and maintain over time.

I have also implemented a web socket in this project after reading the Socket.io documentation, and it worked as expected on localhost. However, when deployed to the cloud, I encountered connection issues that resulted in an unreliable connection and frustration with its performance. By the time of reporting the document, I have still cloud not find a solution for it. So, the WebSocket did not worked as I expected.

This project has been hosted with Google Cloud Services, which offers a free trial with \$300 credit to be used on their services for a certain period. As I am new to cloud services, my economy management was not the best and in just two days I had already spent \$10 - an expense that would have been costly without the free trial. Fortunately, thanks to the generous offer from Google Cloud Services, I did not get charged for it and learnt an important lesson about budgeting when using cloud-based solutions. Addition to that I should learn more JavaScript frameworks to build more powerful web application.

## APPNDIX

### Video Demonstration

Video: <https://youtu.be/pbSQ2VQAau0>

Backup: <https://youtu.be/5NJmlPCQTXo>

### GitHub Repository

Repository: <https://github.com/haritha99ch/Crime-Watch>

### Remote Host

Host: <https://crime-watch-375407.as.r.appspot.com/>

**\*\*note\*\***, I may stop the service incase of a cost alert send by the google cloud, so this might be not available. Please refer the video demonstration to see the build.

**PUSL3120 Full-Stack Development**

---

Proposal title: **Incident Management System**

Student number: **10747887**

---

**Problem**

In the public there are lot of regulations and law breaking happen every day. Most of these actions remains as unpunished because of lack of evidence to prove them. There could be many reasons to have lack of evidence. One of them is people do not have spare time to report the evidence to the police, because people are busy minded and go to the police and reporting evidence is time consuming.

**Solution**

The proposing project is a solution for this time-consuming task, develop a web solution to report any regulations and law breaking activities online. This way people can report these illegal activities as soon as they witness. This will also keep the freshness of the evidence. These reports will go on public after the approval by the respective authorities such as the police department, and other registered people can add more evidence to the report and authorities can act fast and more accurately.

**Functionality****Interactivity**

This system will allow users to register and allows them to report any law-breaking activities that they witness. It could be any illegal activities or incidents such as high speeding, car crashes, robberies or any activities that are wrong according to the law.

After users report these activities with enough evidence, the authorities, in this case of scenario the Police department can approve these reports to go on public and allow other users to add more evidence if they have some. Police agents should also be register to the system with their employee details, and they will be notified of any reports posted by other users in their region.

**Websockets**

The authorities should be notified for any reports that posting to the system, and it must be fast an efficient. This process will use Websockets, so the police agents that are assign to a certain region will be immediately notified whenever a user in the same region post a report.

After the initial report approval, the report will go on public and if there are more witnesses to the incident, they can also add more evidence to the report. This new evidence should also go through an approval process and the same police agent must be notified who initially approve the report. As these examples, Websockets will be used in this system.

**Planned Work****Structure**

## Structure

The development process will follow a good application architecture to promote good separation of concern. MVC is the preferred design pattern to keep all the components loosely coupled.

Mainly, there will be a Data Access Layer to that handle the Mongo Db operations that communicate through an ORM (Sequelize, Prisma ...TBD), and that layer will inherit the data services module that directly communicate with the web application.

Node.js will be heavily used in this project alongside with TypeScript and Angular will be used to develop the frontend.

## Resources

- Laptop (Windows 11 with Windows Subsystem for Linux)
- Identity verification API – free/costly

## Testing

As explained in the structure section, MVC design pattern will be used in the application. So, the most suitable test methodology is unit testing, and a single testing module will be deployed for every module that are mentioned in the structure section.

## Work plan

In the first 10 hours I will be go through all the documentation of each third-party module such as selected ORM framework and then the initial project plan will be done by drawing the essential user cases and ER diagram in less than 10 hours, then I will start the development starting from the back end of the application. During this stage I will do booth development and testing the data services as I develop the backend. This will take about 40 hours and the rest of 20 hours will be spent on developing the front end.

End