|  |
| --- |
| Software Design and Development |
| Major Project: Classroom Manager |
| Analysis and Design Report |

|  |
| --- |
| Euan Mendoza  2-3-2019 |
| Version: 0.3.0 |

Table of Contents

[Introduction 2](#_Toc4878780)

[Client Needs 2](#_Toc4878781)

[Design Objectives 2](#_Toc4878782)

[Requirements 2](#_Toc4878783)

[Functional Requirements 2](#_Toc4878784)

[Non-functional Requirements 3](#_Toc4878785)

[Compatibility 4](#_Toc4878786)

[Performance 4](#_Toc4878787)

[Boundaries 4](#_Toc4878788)

[Project Plan 4](#_Toc4878789)

[Design Specifications 4](#_Toc4878790)

[Input-Process-Output 4](#_Toc4878791)

[Authentication 4](#_Toc4878792)

[Admins 5](#_Toc4878793)

[Flow charts and Graphs 6](#_Toc4878794)

[Design 6](#_Toc4878795)

[Algorithms 6](#_Toc4878796)

[Storyboards 6](#_Toc4878797)

[Graphics 6](#_Toc4878798)

[Database 6](#_Toc4878799)

[Data Dictionary 6](#_Toc4878800)

# Introduction

The classroom manager is a classroom management solution made for St Pius X College and is completely open source software; it aims to be a modern yet simple and practical solution for schools. The classroom manager aims to be a free but used in a proprietary/corporate environment hence the choice of the License agreement. The project aims to be a foundation that with improvement can serve as a foundation core of more complex and full-fledged solutions for classroom management.

This project uses the Apache 2.0 License in order to allow companies and corporations to expand the project to suit their needs. This comes from the Apache License’s explicit and lengthy agreement which protects the software in more legal cases than other licenses and while striking a balance between copyleft software and proprietary license agreements.

# Client Needs

The client needs are as follows:

This program must allow a teacher to manage the day-to-day classroom tasks, which must include.

* class lists
* Recording of marks
* recording of timetable
* Behaviour notes

# Design Objectives

* To be able to accomplish end users goals
  + Admins: manage information and users
  + Teachers: manage classes and students
  + Students: see what classes they have
* To be easy to use for non-technical users
* To be fast, type-safe and a stable platform
* To be easy to maintain without risking users information by separating front end and backend, by modularising code, and by having readable code.
* To be modular with the ability to add, remove or replace features as needed.

# Requirements

## Functional Requirements

This software will allow Users to:

1. Contains multiple types of user accounts, each with different functions
2. Admin accounts
   1. Add, remove, and update classes, students, teachers and personal information
   2. Update and change timetables and class lists
   3. See notifications about students who are marked as absent
3. Teacher accounts
   1. See all classes in timetable format
   2. Sign in as present at school or not
   3. Mark students as either in class or not in class
   4. Record behaviour notes of students
   5. Assign class tasks, homework tasks and assessment tasks
   6. Assign marks and/or marked works
4. Student accounts
   1. See if the student is marked as in class or not
   2. See a timetable of their current classes
   3. See class tasks, homework tasks and assessment tasks
5. Each user has a
   1. Username
   2. Password
   3. Email
6. Class Timetables can change per person and per year group
7. Save data so that it is persistent and secure

## Non-functional Requirements

The following is a list of requirements that do not relate directly to the business functions of the software

* General Application
  + SQLite database, GraphQL API and react frontend
  + Password and username authentication
  + Uses SQL a language standard used for interfacing with databases
  + Uses GraphQL a language standard used for API and web client interaction
  + Uses original code with the exception of certain libraries listed
  + Common design patterns (Client/API) used
  + Uses git modules, git flow and Travis continuous integration
  + Used JetBrains IDE’s Goland and Webstorm
  + Used GitKraken to manage repositories and project versions
  + Tested with graphql playground and custom unit tests
* Frontend
  + JavaScript frontend but served within the go server
  + Frontend user interface follows material design standards
  + Libraries used
    - React – A component-based JavaScript UI library
    - React Router – A React extension library used to route HTML webpages and to route for authentication and security
    - Apollo Client – A JavaScript library that is used to interface with GraphQL APIs
    - Material UI – A React component library that follows material design standards
    - Draft JS – A rich text editor component library for React
    - JSON Web Tokens for secure information transfer over http
* Backend
  + Go web server used
  + Serves static files (HTML, javascript, CSS and images)
  + Handles data and controls flow between frontend and database
  + Encrypts all data with AES encryption
  + Hashes all passwords with a bcrypt password hash and salt
  + Libraries used
    - Standard libraries from Go
    - The Crypto Library
    - go-graphql
    - go-sqlite3
    - json-web-tokens
* Database
  + SQLite database used
  + Stores all personal information

## Compatibility

Needs to be able to work on Microsoft Edge as the baseline.

Google Chrome for phones.

## Performance

Needs to display webpages less than 3 seconds after request.

Needs to run on 64-bit servers

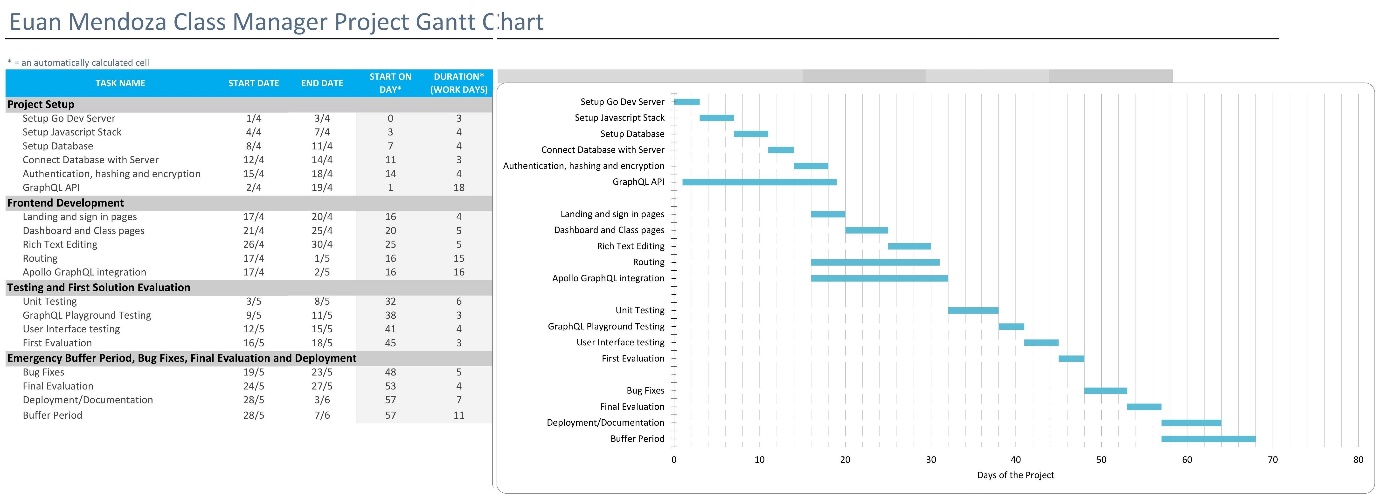
## Boundaries

Windows operating system for server-side deployment.

Variety of screen types and operating systems for general use.

# Project Plan

The following is the Gantt chart for the project. The project splits the application development time and the testing time for the project. The project will be developed in four distinct phases. Project setup will involve setting the groundwork and creating the database. Phase 2 will involve developing the frontend and integrating state, graph QL and routing, this will be the most tedious and error-prone part of the project. Phase 3 is testing, which will also assess how well the project fulfils the requirements. Phase 4 is about last minute changes, final evaluation and testing on intended systems.



# Design Specifications

## Input-Process-Output

### Authentication

|  |  |  |
| --- | --- | --- |
| **Input** | **Process** | **Output** |
| Account Type | Redirect to account type sign in page | Sign in page |
| Username, Password and secret key | 1. Send username and password to server 2. Check if the user is in the database 3. Check if users password hash matches the stored hash from the database 4. If match create a json web token using the secret key and send it to client, then redirect to dashboard 5. else if not a match display error message | Json web token, or error message |
| Json web token | 1. Send web token to server 2. Server decodes token 3. If secret key does not match secret key, server returns error and stops 4. Else server continues 5. Server verifies user, and their privileges 6. If user is not found server returns an error and quits 7. Return access grant | Access not/ is granted |

### Admins

|  |  |  |
| --- | --- | --- |
| **Input** | **Process** | **Output** |
| All Users Request | * Create a User in the database with relevant information * Read User information in the database * Update relevant user information in the database * Delete users in the database | New list of all users |
| All Classes Request | * Create a new class with students and a teacher a period and a start and end time * Read class information * Update Classes period, and start and end time * Delete classes, start time and end time | New list of all classes |
| Marked roll and timestamp | * If student marked absent push message to admin dashboard | Absent notification |

### Teachers

|  |  |  |
| --- | --- | --- |
| **Input** | **Process** | **Output** |
| Teacher ID | * View a timetable of all associated classes | Timetable |
| Teacher ID, Timetable Request | * View a dashboard of classes | Class Dashboard |
| Class period, time and unmarked roll | * If student is present record present * Else if student is absent record absent | Present or absent and timestamp |
| Class ID, behaviour notes request, time | * Edit a text document and add information relating to behaviour | Behaviour note file, timestamp |
| Class ID, Task type, task information and due date | * Assign a task to all students | Task |
| Task ID, Student ID, and mark | * Assign a mark for students task | Marked Task |

### Students

|  |  |  |
| --- | --- | --- |
| **Input** | **Process** | **Output** |
| Request roll status | * If absent display absent * Else display present | Present or absent |
| Student ID, day, class request | * View list of classes | Class Timetable |
| Task ID, Class ID | * View Task | Task |
| Task ID, Student ID, Date | * Submit task response | Submission |
| Task ID, Student ID, mark request | * View marked task | Marked Task |

## Design

### Flow Charts and Graphs

#### Routes

### Algorithms

#### Authentication

##### Hash Password

BEGIN SaltAndHash(password)

let hash = bcrypt.GenerateFromPassword(password, bcrypt.MinCost)

as string

return hash

END SaltAndHash

##### Compare Hash and Password

BEGIN CompareHashAndPassword(hash, password)

let check = bcrypt.CompareHashAndPassword(hash, password) as bool

IF !check

return true

END IF

return false

END CompareHashAndPassword

### Storyboards

### Graphics

<https://material.io/tools/icons/?style=baseline>

### Data Dictionary

#### General

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Data Item** | **Data Type** | **Format** | **Storage Requirements** | **Display Size** | **Description** | **Example** | **Validation** |
|  |  |  |  |  |  |  |  |

# Glossary of Key Terms