|  |
| --- |
| 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](#_Toc5539298)

[Client Needs 2](#_Toc5539299)

[Design Objectives 2](#_Toc5539300)

[Requirements 2](#_Toc5539301)

[Functional Requirements 2](#_Toc5539302)

[Non-functional Requirements 3](#_Toc5539303)

[Compatibility 4](#_Toc5539304)

[Performance 4](#_Toc5539305)

[Boundaries 4](#_Toc5539306)

[Project Plan 4](#_Toc5539307)

[Design Specifications 5](#_Toc5539308)

[Input-Process-Output and Algorithms 5](#_Toc5539309)

[General 5](#_Toc5539310)

[Authentication 6](#_Toc5539311)

[Admins 6](#_Toc5539312)

[Teachers 7](#_Toc5539313)

[Students 7](#_Toc5539314)

[Flow Charts and Graphs 8](#_Toc5539315)

[Routes 8](#_Toc5539316)

[Storyboards 8](#_Toc5539317)

[Graphics 8](#_Toc5539318)

[Data Dictionary 8](#_Toc5539319)

[General 8](#_Toc5539320)

[Glossary of Key Terms 8](#_Toc5539321)

# 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. Manage the database and user accounts
   2. Do not have access to class tasks and marks
   3. Create, Read, Update and Delete all users, and classes
   4. Add Term dates and public holidays
3. Teacher accounts
   1. See all classes in timetable format
   2. Mark students as either in class or not in class
   3. Record behaviour notes of students
   4. Assign class tasks, homework tasks and assessment tasks
   5. 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
  + Hashes all passwords with a bcrypt password hash and salt
  + Libraries used
    - Standard libraries from Go
    - The Crypto bcrypt 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

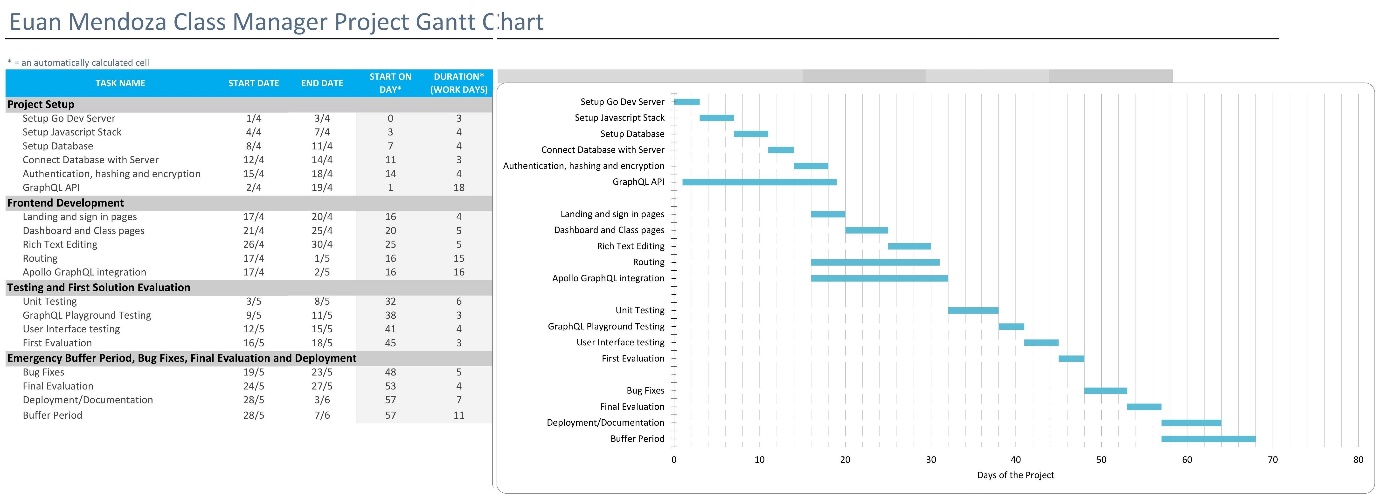
## Boundaries

Needs to be run and tested on windows computers during development

Needs to be able to deploy on a linux server with nginx

# 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 and Algorithms

### General

Routes

|  |  |  |
| --- | --- | --- |
| **Input** | **Process** | **Output** |
| Website path, website, graphql handler | Create Routes for main website and graphql handler, entrypoint for the application | Web Server |

BEGIN StartWebserver

let website = path\_to\_website

http.handle("/", website)

let graphql\_handler = graphql\_queries

http.handle("/graphql", graphql\_handler)

http.ListenAndServe()

END StartWebserver

Access Persistent Constants

|  |  |  |
| --- | --- | --- |
| **Input** | **Process** | **Output** |
| Persistent constant key | 1. If environment json file does not exist, create environment json file and call create environment constants function 2. return constant using contestant key | Environment constant |

BEGIN AccesssPersistentConstants(constant\_request)

IF environment.json NOT exists

Create environment.json

CreateEnvironmentConstants()

END IF

let env\_constants = get constants from environment.json

return env\_constants[constant\_request]

END AccessPersistentConstants

Create Environment Constant

|  |  |  |
| --- | --- | --- |
| **Input** | **Process** | **Output** |
| void | Randomly generate a secret key and store in environment json file | void |

BEGIN CreateEnvironmentConstants

let SECRET\_KEY = generate random string

store SECRET\_KEY in environment.json

END CreateEnvironmentConstants

### Authentication

Login Form

|  |  |  |
| --- | --- | --- |
| **Input** | **Process** | **Output** |
| void | Create input fields for username and password  If submit button is clicked call login function | Username, password |

BEGIN LoginForm

display input username\_field

display input password\_field

display input confirm\_password\_field

submit\_button.event('click', Login)

END LoginForm

Login callback

|  |  |  |
| --- | --- | --- |
| **Input** | **Process** | **Output** |
| Account type  Username  Password  Password 2 | 1. if the first password and the second password match send all data to server 2. else redirect to login 3. if the server responds successful with token, redirect to dashboard 4. else redirect back to login | Dashboard or login screens |

BEGIN Login

let account\_type = student | teacher | admin

let username, password = get\_username\_state('username\_field')

let password = get\_password\_state('password\_field')

let confirm\_password = get\_password\_state('confirm\_password\_field')

IF password == confirm\_password

POST username, password, account\_type TO server

ELSE

REDIRECT TO /<usertype:string>/login

print(passwords dont match)

END IF

IF response bool == true

print(response msg)

let user\_access\_token = response token

REDIRECT TO /<usertype:string>/dashboard

ELSE

print(response msg)

END IF

END Login

Login Server Side

|  |  |  |
| --- | --- | --- |
| **Input** | **Process** | **Output** |
| Account\_type  Username  password | 1. Check if username is in the database table of user\_type 2. Get password hash of user from database 3. Compare hashed password with password 4. Return json web token with the secret key account\_type and username | Json\_web\_token |

BEGIN Login

let username, password, account\_type = LISTEN FOR username, password

IF username NOT IN table user\_type

RESPOND (false, msg: User not found, empty token)

END Login

END IF

let password\_hash = account\_type[username\_cypher]

IF CompareHashAndPassword(password\_hash, password)

let user\_email = get username from table user\_type

json\_web\_token = jwt(AccessPersistantConstants(SECRET\_KEY), [ account\_type, username, user\_email )

RESPOND (true, msg: Login Success, json\_web\_token)

END Login

END IF

RESPOND (false, msg: Login failed, empty token)

END Login

Logout

|  |  |  |
| --- | --- | --- |
| **Input** | **Process** | **Output** |
| Json\_web\_token | Clear json web token from users storage  Redirect to home page | void |

BEGIN Logout

local.user\_access\_token = ''

Redirect to /

END Logout

Access Protected URL

|  |  |  |
| --- | --- | --- |
| **Input** | **Process** | **Output** |
| Json\_web\_token | 1. Send the json web token and url to server to call server function 2. If the server responds with true allow access to page 3. Else redirect to homepage | Protected Route, or homepage |

BEGIN RequestProtectedRoute

POST json\_web\_token, requested url TO server

IF RESPONSE == true

access granted

ELSE

redirect to login

END IF

END RequestProtectedRoute

Access Protected URL Server Side

|  |  |  |
| --- | --- | --- |
| **Input** | **Process** | **Output** |
| Json\_web\_token | 1. If url name matches with the user type in the json web token, and the json web token is verified return true 2. Else return false | True or false |

BEGIN RequestProtectedRoute

let json\_web\_token, requested url = LISTEN FOR json\_web\_token, requested url

IF VerifyToken(json\_web\_token)

RESPOND true

ELSE

RESPOND false

END IF

END RequestProtectedRoute

Verify User with Json Web Token

|  |  |  |
| --- | --- | --- |
| **Input** | **Process** | **Output** |
| Json\_web\_token, user | 1. Decode the json web token 2. If the secret key matches the stored secret key and the user type is permitted return true 3. Else return falst | True or false |

BEGIN VerifyToken(json\_web\_token, permitted\_usertype)

let secret\_key, usertype, username = decode json\_web\_token

IF secret\_key == AccessPersistantConstant(SECRET\_KEY) AND usertype == permitted\_usertype

let username = username

IF username IN database table user\_type

RETURN true

END IF

END IF

RETURN false

END VerifyToken

Salt and Hash

|  |  |  |
| --- | --- | --- |
| **Input** | **Process** | **Output** |
| Password | Create a bcrypt salt and hash of the password using as minimal computer resources as possible | Password hash |

BEGIN SaltAndHash(password)

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

return hash

END SaltAndHash

Compare Hashes

|  |  |  |
| --- | --- | --- |
| **Input** | **Process** | **Output** |
| Hash  password | Checks if the hash and salt and the password are the same | True or false |

BEGIN CompareHashAndPassword(hash, password)

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

IF NOT check

return true

END IF

return false

END CompareHashAndPassword

### 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 |

## Flow Charts and Graphs

### Routes

|  |  |
| --- | --- |
| **Route** | **Description** |
| / |  |
| /graphql |  |
| /admin/login |  |
| /admin/dashboard |  |
| /admin/teachers |  |
| /admin/students |  |
| /admin/classes |  |
| /teachers/login |  |
| /teachers/dashboard |  |
| /teachers/classes |  |
| /teachers/classes/<class\_id> |  |
| /students/login |  |
| /students/dashboard |  |

## Storyboards

## Graphics

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

## Data Dictionary

### General

|  |  |  |  |
| --- | --- | --- | --- |
| **Data Item** | **Data Type** | **Description** | **Example** |
| SECRET\_KEY | string | A randomly generated string used to verify json web tokens | If SECRET\_KEY == auth\_SECRET\_KEY |
| Json\_web\_token | Json object or string | A encoded string of various items used when placed in the head of http requests to verify users | If json\_web\_token == user json\_web\_token |
| User\_type | Enum -  Admin, Teacher  Student | Used to verify what table data should be stored in | Get user from database table user\_type |
| username | string | Identifier of a student | Search for username |
| email | string | A email address that is helpful for teachers | Email this person using email |
| Password\_hash\_and\_salt | Byte array | A secure way to validate passwords without storing passwords | Check password with password\_hash\_and\_salt |
|  |  |  |  |

# Glossary of Key Terms