|  |
| --- |
| 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](#_Toc5659668)

[Client Needs 2](#_Toc5659669)

[Design Objectives 2](#_Toc5659670)

[Requirements 2](#_Toc5659671)

[Functional Requirements 2](#_Toc5659672)

[Non-functional Requirements 3](#_Toc5659673)

[Compatibility 4](#_Toc5659674)

[Performance 4](#_Toc5659675)

[Boundaries 4](#_Toc5659676)

[Project Plan 4](#_Toc5659677)

[Design Specifications 5](#_Toc5659678)

[Input-Process-Output and Algorithms 5](#_Toc5659679)

[General 5](#_Toc5659680)

[Authentication 6](#_Toc5659681)

[Admins 9](#_Toc5659682)

[Teachers 20](#_Toc5659683)

[Students 22](#_Toc5659684)

[Flow Charts and Graphs 23](#_Toc5659685)

[Routes 23](#_Toc5659686)

[System Flow Charts 23](#_Toc5659687)

[Storyboards 26](#_Toc5659688)

[Graphics 37](#_Toc5659689)

[Data Dictionary 37](#_Toc5659690)

[General 37](#_Toc5659691)

[Glossary of Key Terms 38](#_Toc5659692)

# 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

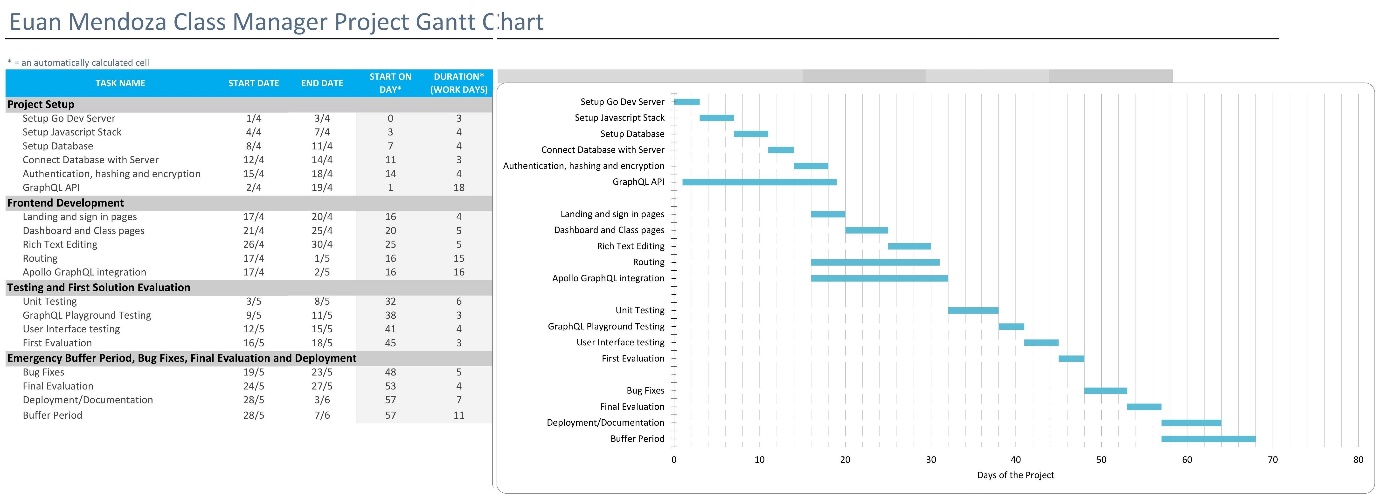
## Boundaries

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

Set School Year Bounds

|  |  |  |
| --- | --- | --- |
| **Input** | **Process** | **Output** |
| Weekly or bi-weekly  Array of Period: Begin, recess, lunch, end, 1 ….16  Start\_time  End\_time  Term\_one start\_time, end\_time  Term\_two, start\_time, end\_time  Term three, start\_time, end\_time  Term\_four, strat\_time, end\_time | 1. Select parameters 2. Add more periods 3. Send all inputs to server | void |

BEGIN SetBounds

display select school\_week = weekly | bi-weekly

display select period = begin | recess | lunch | end | 1 | 2 | 3 .... 16

display input start\_time

display input end\_time

IF add.button(click)

display select period = begin | recess | lunch | end | 1 | 2 | 3 | .... 16

display input start\_time = previous end\_time

display input end\_time

END IF

display input term\_one, start\_time, end\_time

display input term\_two, start\_time, end\_time

display input term\_three, start\_time, end\_time

display input term\_four, start\_time, end\_time

IF submit.event(click)

POST all fields TO server

END IF

END SetBounds

Store Year Parameters

|  |  |  |
| --- | --- | --- |
| **Input** | **Process** | **Output** |
| Weekly or bi-weekly  Period array: Begin, recess, lunch, end, 1 ….16  Start\_time  End\_time  Term\_one start\_time, end\_time  Term\_two, start\_time, end\_time  Term three, start\_time, end\_time  Term\_four, strat\_time, end\_time  Environment.json | Store all information in environment.json | Environment.json |

BEGIN StoreBounds

LISTEN FOR school\_week, [[period, start\_time, end\_time]], [term\_one, start\_time, end\_time]... SAVE IN environment.json

END StoreBounds

View List of Users

|  |  |  |
| --- | --- | --- |
| **Input** | **Process** | **Output** |
| User type | 1. Get list of users of user\_type from database 2. Display each users | List of users |

BEGIN ViewListOfUsers

GET list\_of\_users WITH user\_type, json\_web\_token FROM server

let success, users = server response

IF success

COUNT EACH user IN list\_of\_users

display user

END COUNT EACH

ELSE

display(access denied)

END IF

IF user.event(click)

REDIRECT TO /admin/<user\_type:string>/<username:string>

END IF

END ViewListOfUsers

View List of Users Server Side

|  |  |  |
| --- | --- | --- |
| **Input** | **Process** | **Output** |
| User\_type  Json\_web\_token | 1. Verify that the user is allowed access to the subprogram using a json web token 2. Get users of user\_type from database 3. Respond with users to database | List of users |

BEGIN GetUsersList

let user\_type, json\_web\_token = LISTEN FOR list\_of\_users get request

IF VerifyToken(json\_web\_token, admin)

let users = get users from user\_type table in database

RESPOND (true, users)

ELSE

RESPOND (false, null)

END IF

END GetUsersList

Create a User

|  |  |  |
| --- | --- | --- |
| **Input** | **Process** | **Output** |
| Void | 1. Add a username 2. Add a email 3. Add a password 4. Send information to server | New user |

BEGIN CreateUser

display username\_field

display email\_field

display password\_field

IF submit\_button.event('click') AND all\_fields NOT == null

POST (json\_web\_token, user\_type, username, email, password) TO server

RECEIVE is\_success, error

IF NOT is\_success

print(error)

ELSE

REDIRECT TO /admin/<user\_type:string>/

ELSE

highlight null fields

END IF

END CreateAdminUser

Create a User Server Side

|  |  |  |
| --- | --- | --- |
| **Input** | **Process** | **Output** |
| Json\_web\_token  User\_type  Username  Email  password | 1. Verify user permission with json web token 2. Hash password 3. Store username, email and password hash in database using user\_type | New user |

BEGIN CreateUser

let json\_web\_token, user\_type, username, email, password = LISTEN FOR json\_web\_token, username, email, password

IF VerifyUser(json\_web\_token, admin)

let password\_hash = SaltAndHash(password)

STORE username, email, password\_hash in database table account\_type

IF STORE Failed

IF username not unique error

RESPOND false, username not unique

ELSE IF email not unique error

RESPOND false, email not unique

ELSE

RESPOND false, store\_error

END IF

END IF

ELSE

RESPOND true

END CreateUser

View User

|  |  |  |
| --- | --- | --- |
| **Input** | **Process** | **Output** |
| User id | 1. Get user information from server using user\_id 2. If user information modified send new user information to server | User information |

BEGIN ViewUser

GET user WITH user\_type, user\_id FROM server

let username\_field = display user.username

let email\_field = display user.email

let password\_field = empty string

let confirm\_password\_field = empty string

let changed\_state = false

IF username\_field changed OR email\_field changed OR lengthof(password\_field) > 0

changed\_state = true

END IF

IF submit.event(click) AND changed\_state

POST (json\_web\_token, user.id, user.type, username\_field, email\_field, password\_field) TO server

END IF

IF delete.event(click)

POST json\_web\_token, user.type, user.id TO server

END IF

END ViewUser

Read User

|  |  |  |
| --- | --- | --- |
| **Input** | **Process** | **Output** |
| Json web token  User id  User type | 1. Verify user has access to the function through verifying the json web token 2. Get user using user id in user type table of database 3. Send to client | User |

BEGIN ReadUser

let user\_type, user\_id = LISTEN FOR user\_type, user\_id

IF user\_id IN database table user\_type

RESPOND user\_type[user\_id]

ELSE

RESPOND user not found error

END IF

END ReadUser

Update User

|  |  |  |
| --- | --- | --- |
| **Input** | **Process** | **Output** |
| Json web token  User id  User type  Username  Email  password | 1. Verify user permission with json web token 2. Hash password 3. Store username, email and password hash in table user type using user id | Updated user |

BEGIN UpdateUser

let json\_web\_token, user\_id, user\_type, username, email, password = LISTEN FOR json\_web\_token, user\_id, username, email, password

IF VerifyToken(json\_web\_token, admin)

IF user\_id IN database table user\_type

let password\_hash = HashAndSalt(password)

STORE user\_id, username, email, password\_hash IN database table user\_type

END IF

END IF

END UpdateUser

Delete User

|  |  |  |
| --- | --- | --- |
| **Input** | **Process** | **Output** |
| Json web token  User type  User id | Verify permissions with json web token  Delete user from database in table user type | void |

BEGIN DeleteUser

let json\_web\_token, user\_type, user\_id = LISTEN FOR json\_web\_token, user\_type, user\_id

IF VerifyToken(json\_web\_token, admin)

DELETE user\_id FROM database table user\_type

END IF

END DeleteUser

View List of Classes

|  |  |  |
| --- | --- | --- |
| **Input** | **Process** | **Output** |
| void | 1. Get a list of classes from database 2. Count each class in list and display | List of classes |

BEGIN ViewClassesList

GET classes\_list USING json\_web\_token FROM server

COUNT EACH class IN classes\_list

display class

END COUNT EACH

END ViewClassesList

View List of Classes Server Side

|  |  |  |
| --- | --- | --- |
| **Input** | **Process** | **Output** |
| Json web token | 1. Verify user with json web token 2. Store all the classes in a list and send it to client | List of classes |

BEGIN ViewClassesList

LISTEN FOR json\_web\_token

IF VerifyToken(json\_web\_token, admin)

let classes\_list = []

COUNT EACH class IN database table Classes

classes\_list.append(class)

END COUNT EACH

RESPOND classes\_list

END IF

END ViewClassesList

Create Class

|  |  |  |
| --- | --- | --- |
| **Input** | **Process** | **Output** |
| Class id  Year group  Teacher  List of students  List of periods and time | 1. Get user input and store it in variables 2. Add students to student list 3. Add periods and time to the periods and time list 4. Send to server | Json web token  Class id  Year group  Teacher  List of students  List of periods and time |

BEGIN CreateClass

display input class\_id\_field

display select year\_group\_field

display select teacher\_field

let student\_array = []

COUNT EACH student IN student\_array

display select student\_field

END COUNT EACH

IF add\_student.event(click)

student\_array.append\_student

END IF

let period\_array = [[select\_period, select\_day]

COUNT EACH period IN periods

display select period[select\_period]

display select period[select\_day]

END COUNT EACH

IF add\_period.event(click)

period\_array.append([select\_period, select\_day])

END IF

POST json\_web\_token, class\_id\_field, year\_group\_field, teacher\_field, student\_array, period\_array TO server

IF RESPONSE == error

display error

END IF

END CreateClass

Create Class Server Side

|  |  |  |
| --- | --- | --- |
| **Input** | **Process** | **Output** |
| Json web token  Class id  Year group  Teacher  List of students  List of periods and time | 1. Verify users permission with json web token 2. Store the class id, year group, teacher in database table Classes 3. Store list of students with the classes primary key in table student\_class 4. Store list of periods and time with classes primary key in table timetable | New Class |

BEGIN CreateClass

LISTEN FOR json\_web\_token, class\_id, year\_group, teacher, student\_array, period\_array

IF VerifyToken(json\_web\_token, admin)

STORE class\_id, year\_group, teacher IN database table Classes

get class\_primary\_key WITH class\_id FROM database table Classes

COUNT EACH student IN student\_array

STORE class\_id, student.id IN database table student\_class

END COUNT EACH

COUNT EACH periods IN period\_array

STORE periods[period], periods[day] IN database table timetable

END COUNT EACH

IF error

RESPOND error

END IF

END IF

END CreateClass

View Class

|  |  |  |
| --- | --- | --- |
| **Input** | **Process** | **Output** |
| Class  Students list  Timetable list | 1. Get information from server 2. Display class information 3. Display students in class 4. Display date and period 5. If anything is modified by user, set state changed variable to true 6. If state changed variable is true send information to server | Class |

BEGIN ViewClass

GET Class, students\_array, periods\_array WITH json\_web\_token, class\_primary\_key FROM server

let changed\_state = false

display input class\_id\_field = class.class\_id

display select year\_group\_field = class.year\_group

display select teacher\_field = class.teacher\_id

let student\_array = []

COUNT EACH student IN student\_array

display select student\_field

END COUNT EACH

IF add\_student.event(click)

student\_array.append\_student

changed\_state = true

END IF

let period\_array = [[select\_period, select\_day]

COUNT EACH period IN periods

display select period[select\_period]

display select period[select\_day]

END COUNT EACH

IF add\_period.event(click)

period\_array.append([select\_period, select\_day])

changed\_state = true

END IF

IF changed\_state

POST json\_web\_token, Class.id, class\_id\_field, year\_group\_field, teacher\_field, student\_array, period\_array TO server

IF RESPONSE == error

display error

END IF

END IF

IF delete\_user.event(click)

POST User.id TO server

END IF

END ViewClass

Read Class

|  |  |  |
| --- | --- | --- |
| **Input** | **Process** | **Output** |
| Json web token  Class id | 1. Verify users permission using json web token 2. Get class from database class table using class id 3. Get students from database table student\_class using class id 4. Get periods and time from database table timetable using class id 5. Send to client | Class  Students in class  Class timetable |

BEGIN ReadClass

LISTEN FOR get\_request, json\_web\_token, class\_primary\_key

IF VerifyToken(json\_web\_token, admin)

get class, students\_list, periods\_list FROM database table Classes, student\_class, periods

RESPOND class, students\_list, periods\_list

ELSE

RESPOND error

END IF

END ReadClass

Update Class

|  |  |  |
| --- | --- | --- |
| **Input** | **Process** | **Output** |
| Json web token  Class id  Class class\_id  Class teacher  List of students in class  List of periods and days | 1. Verify users permission using the json web token 2. Store the class in table Class using the class id 3. Store the students in the table student\_class using class id 4. Store periods and days in table timetable using class id | Updated class |

BEGIN UpdateClass

LISTEN FOR json\_web\_token, class\_id\_int, class\_id\_string, year\_group, teacher, student\_array, period\_array

IF VerifyToken(json\_web\_token, admin)

get teacher\_id using teacher in dataabase table Teachers

STORE class\_id\_string, year\_group, teacher\_id IN database table Classes USING class\_id\_int

let student\_id\_array = []

COUNT EACH student IN student\_array

get student\_id using student from database table Students

student\_id\_array.append(student)

END COUNT EACH

COUNT EACH student\_id IN student\_id\_array

STORE class\_id\_int, student\_id IN database table student\_class

END COUNT EACH

COUNT EACH period IN period\_array

STORE period, class\_id\_int IN database table timetable

END COUNT EACH

IF error

RESPOND error

END IF

ELSE

RESPOND error

END IF

END UpdateClass

Delete Class

|  |  |  |
| --- | --- | --- |
| **Input** | **Process** | **Output** |
| Json web token  Class id | 1. Verify users permissions with json web token 2. Delete class from table Class 3. Delete all students with class id in table student class 4. Delete all periods and days with class id in table timetable | Void |

BEGIN DeleteClass

LISTEN FOR json\_web\_token, class\_id

IF VerifyToken(json\_web\_token, admin)

DELETE class class\_id FROM database table Classes

COUNT EACH item IN database table student\_class USING class\_id

DELETE item

END COUNT EACH

COUNT EACH item IN database table timetable USING class\_id

DELETE item

END IF

END IF

END DeleteClass

### Teachers

View Students List

|  |  |  |
| --- | --- | --- |
| **Input** | **Process** | **Output** |
| Void | Get list of students from server, for each student in list of students display student | List of students display |

BEGIN ViewStudentsList

GET list\_of\_students FROM server WITH json\_web\_token

COUNT EACH student IN list\_of\_students

display student

END COUNT EACH

END ViewStudentsList

View Students List Server Side

|  |  |  |
| --- | --- | --- |
| **Input** | **Process** | **Output** |
| Json web token | 1. Verify users permission using json web token 2. Get list of students from database | List of students |

BEGIN ViewStudentsList

LISTEN FOR json\_web\_token

IF VerifyToken(json\_web\_token, teacher)

get list\_of\_students from database table Student

END IF

END ViewStudentsList

View Timetable

|  |  |  |
| --- | --- | --- |
| **Input** | **Process** | **Output** |
| void | 1. Get timetable from server, send json web token, teacher id and the date 2. If it’s a school day display each period in the day 3. Else display a holiday | Timetable |

BEGIN ViewTimetable

GET timetable FROM server WITH json\_web\_token, teacher\_id, date.now()

let school\_day = RESPONSE

IF school\_day

let periods, classes\_periods = RESPONSE

COUNT EACH period IN periods

IF period IN classes\_periods

display classes\_periods[period[0]]

display classes\_periods[period[1]]

display classes\_periods[period[2]]

ELSE

display free period

END IF

END COUNT EACH

ELSE

let holiday = RESPONSE

display holiday

END IF

END ViewTimetable

View Timetable Server Side

|  |  |  |
| --- | --- | --- |
| **Input** | **Process** | **Output** |
| Json web token  Teacher id  date | 1. Verify permissions using json web token 2. Get terms from environment.json 3. Match the term date is in and set the holiday to false, and term to that term 4. Else set holiday to true 5. If holiday is false and date in holidays from environment.json set holidays to true 6. If holidays is false find the day of the week using either the biweekly or weekly calander and respond to client with all the periods a teacher has for that particular day 7. Else respond with the holiday it is | Periods in day or holiday |

BEGIN ViewTimetable

LISTEN FOR json\_web\_token, teacher\_id, date

IF VerifyToken(json\_web\_token, teacher)

let term\_one = AccessPersistantConstants[term\_one]

let term\_two = AccessPersistantConstants[term\_two]

let term\_three = AccessPersistantConstants[term\_three]

let term\_four = AccessPersistantConstants[term\_four]

let holidays = AccessPersistantConstants[holidays]

let holiday: boolean

let term: term

MATCH date

CASE term\_one[start] > date AND term\_one[end] < date

holiday = false

term = term\_one

BREAk

CASE term\_two[start] > date AND term\_two[end] < date

holiday = false

term = term\_two

BREAK

CASE term\_three[start] > date AND term\_three[end] < date

holiday = false

term = term\_three

BREAK

CASE term\_four[start] > date AND term\_four[end] < date

holiday = false

term = term\_four

BREAK

ELSE

holiday = true

term = holidays

END MATCH

IF NOT holiday AND date IN holidays

holiday = true

END IF

IF NOT holiday

RESPOND NOT holiday

let week\_rotation = AccessPersistantConstant(weeks)

let day = 0

COUNT d = term.start\_day, d == date, d += 1

IF week\_rotation == biweekly

IF day >= 14

day = 1

END IF

day += 1

ELSE IF week\_rotation == weekly

IF day >= 7

day = 1

END IF

day +=1

END IF

END COUNT

get periods from database table timetable with teacher\_id

RESPOND periods, AccessPersistantConstant(periods)

ELSE

RESPOND holiday

RESPOND holidays[date]

END IF

END IF

END ViewTimetable

### Students

|  |  |  |
| --- | --- | --- |
| **Input** | **Process** | **Output** |
| Request roll status | 1. If absent display absent 2. 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 |

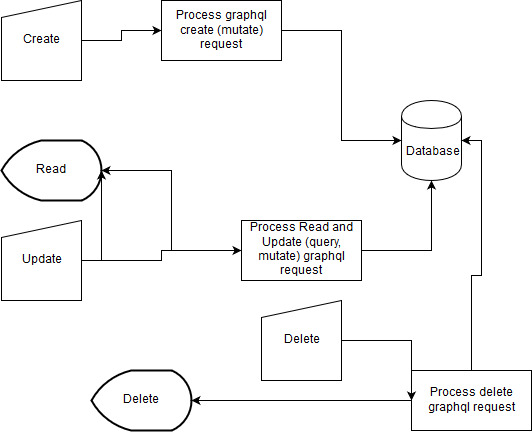
## Flow Charts and Graphs

### Routes

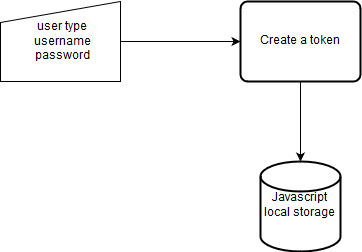
|  |  |
| --- | --- |
| **Route** | **Description** |
| / | The entry point for the application, displays a welcome page and redirects to login pages |
| /graphql | The route where the client can interact to send and receive data |
| /admin/login | Where admins login, reroutes to the dashboard |
| /admin/dashboard | Where admins can redirect to a place to manager admins, teachers, students and classes |
| /admin/admins | Where admins can view a list of admins, create, read, update and delete admin users |
| /admin/teachers | Where admins can view a list of teachers, create, read, update and delete a teacher |
| /admin/students | Where admin can view a list of students, create, read, update and delete a student |
| /admin/classes | Where admin can view a list of classes, create, read, update and delete a class |
| /teachers/login | Where teachers login, redirects to dashboard |
| /teachers/dashboard | Where a teacher views their timetable, and view a list of students and classes |
| /teachers/<class\_id> | A place where a teacher can view a class create, read, update and delete tasks, view task submissions and marks roll |
| /students/login | A place where students login |
| /students/dashboard | A place where students view their assigned tasks, their timetables and absents. |

### System Flow Charts

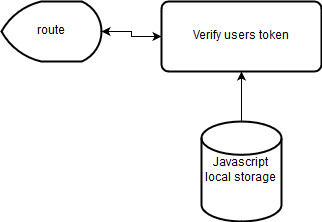
Graphql Api



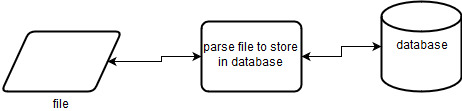
User Authentication



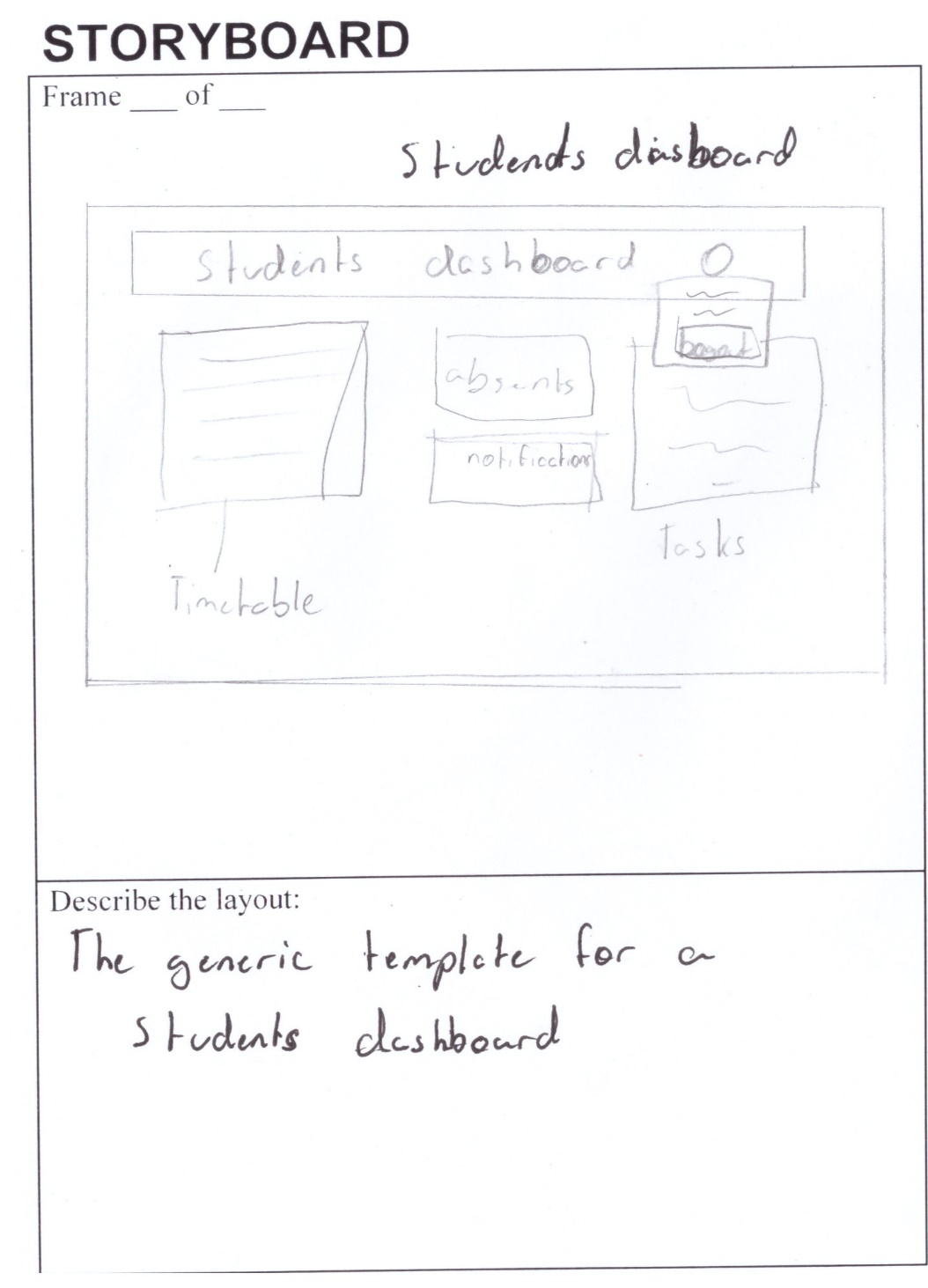
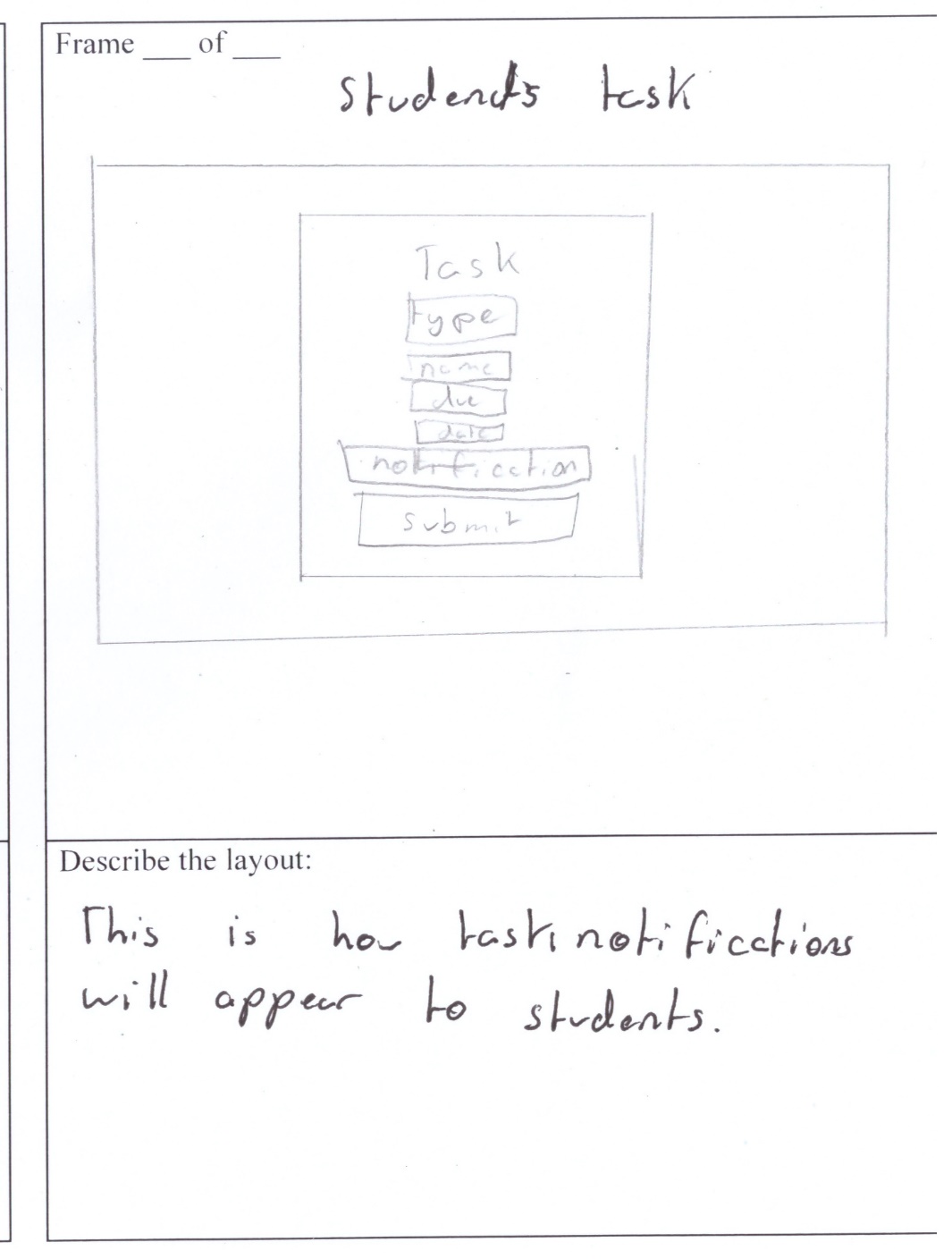
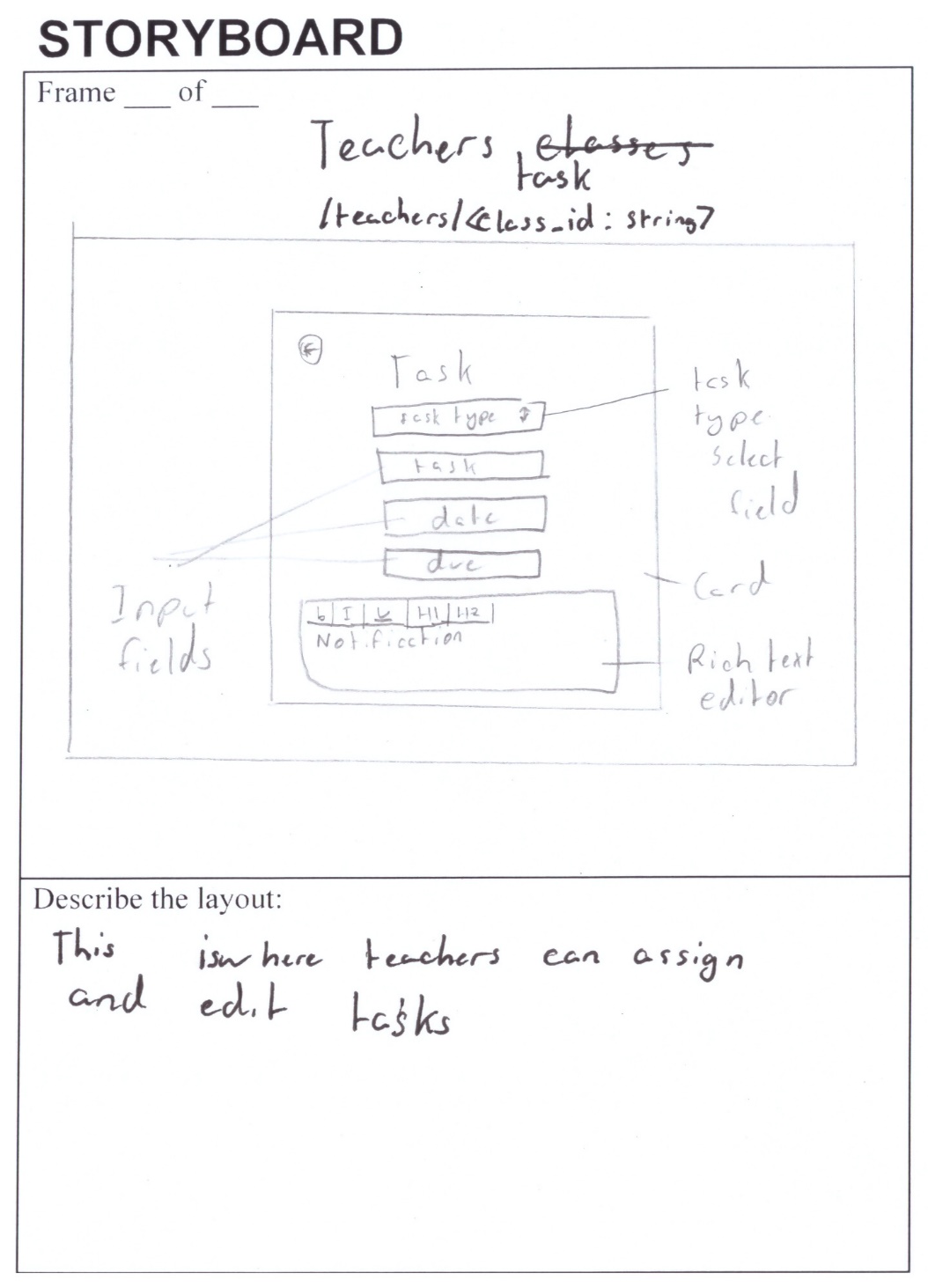
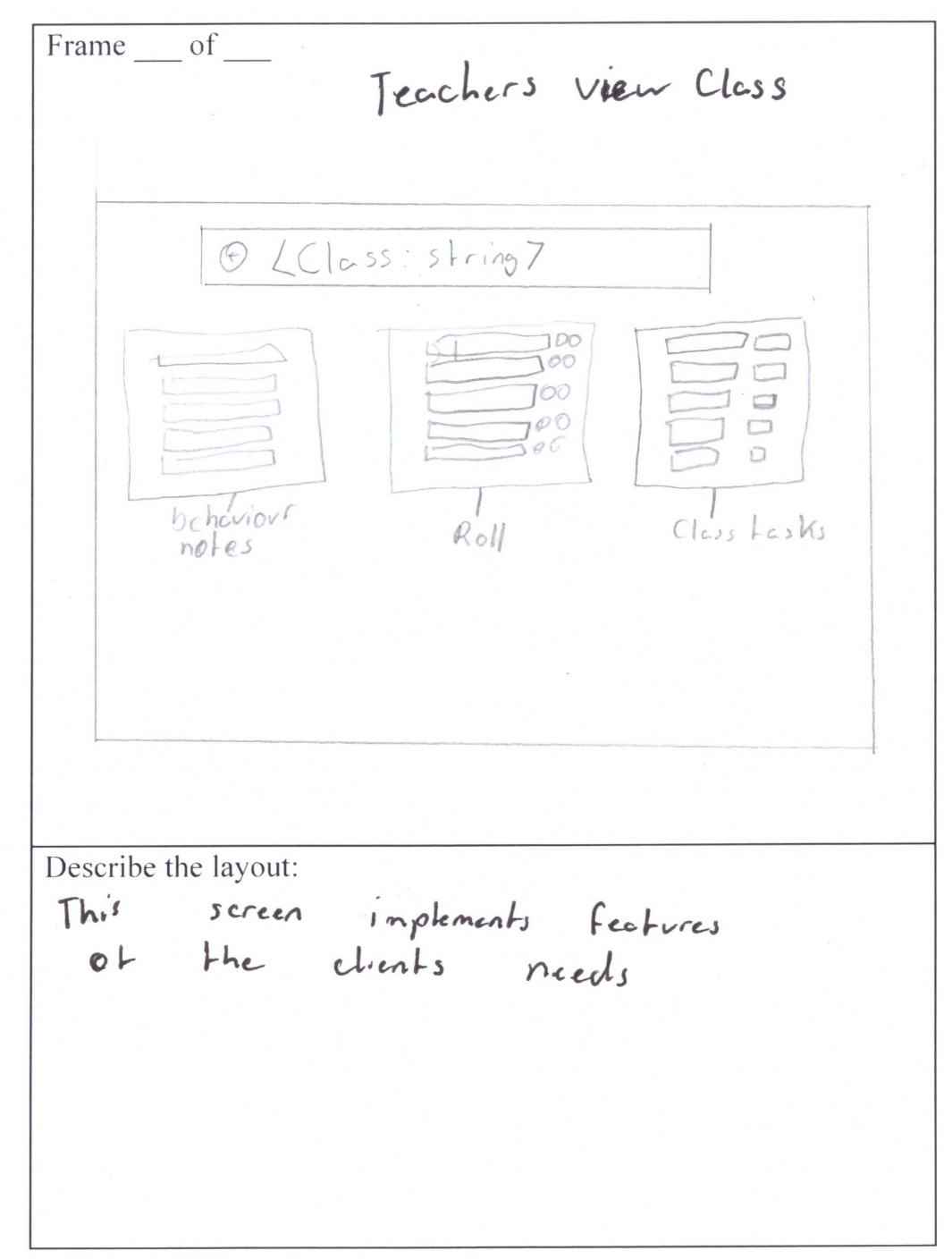
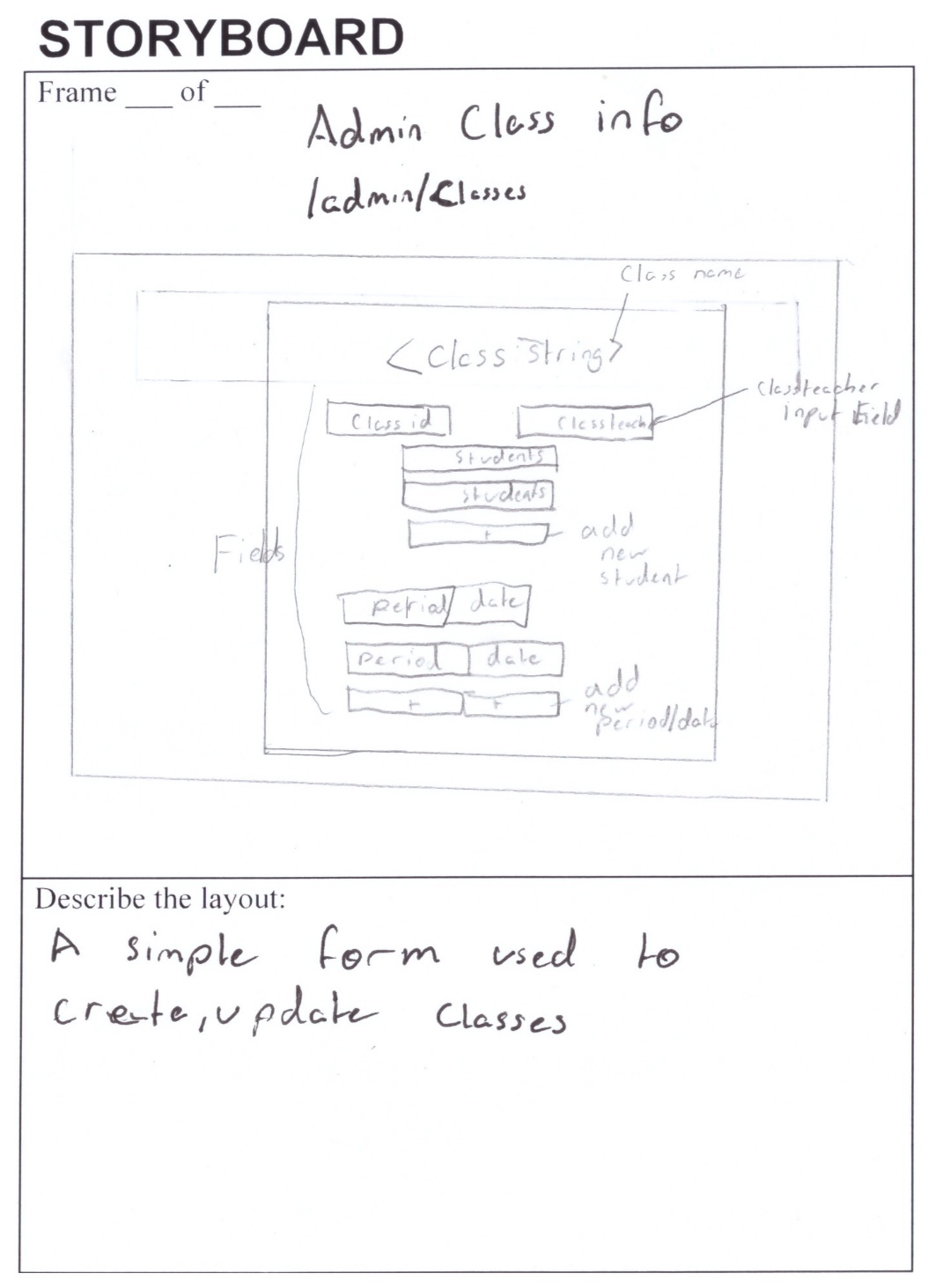
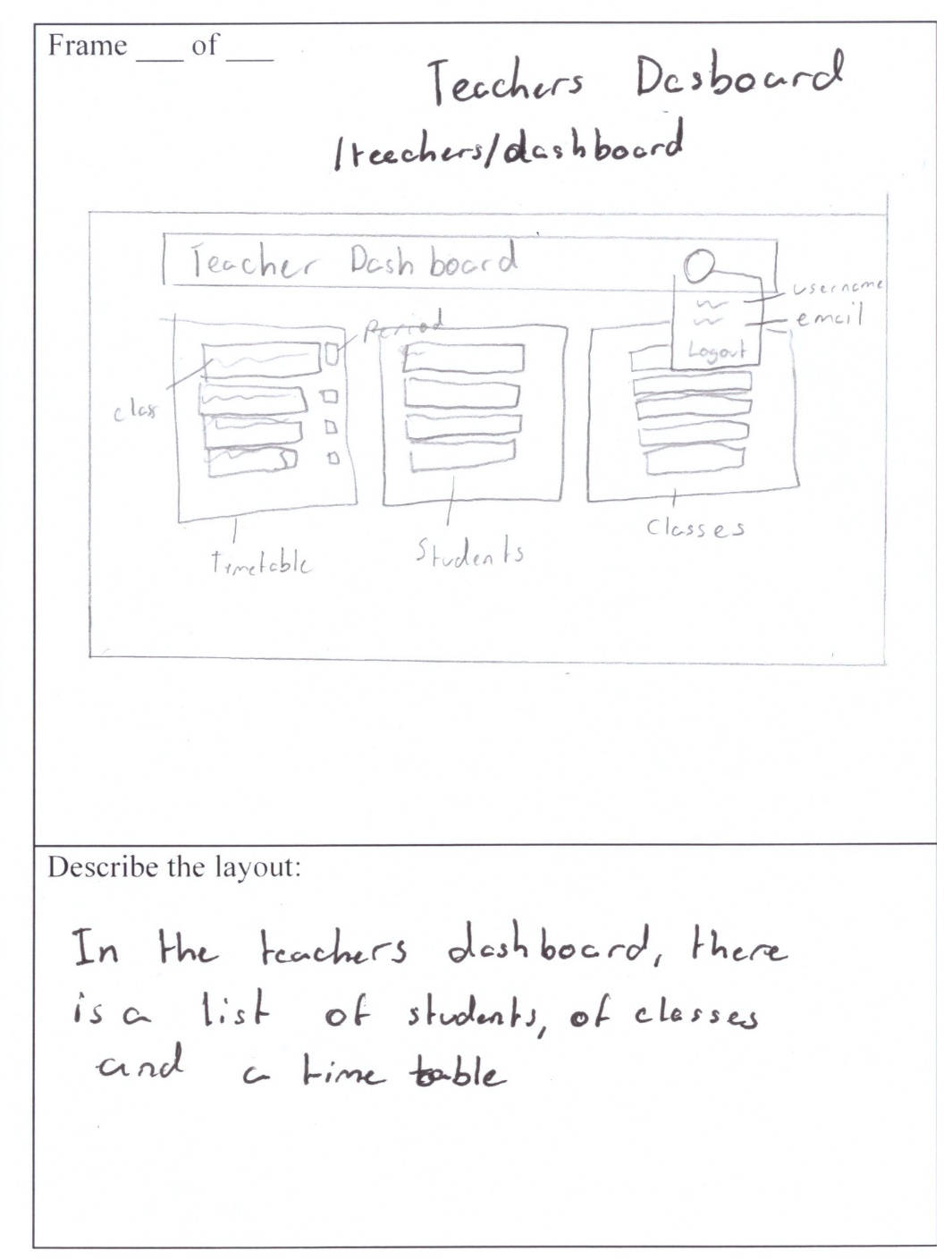
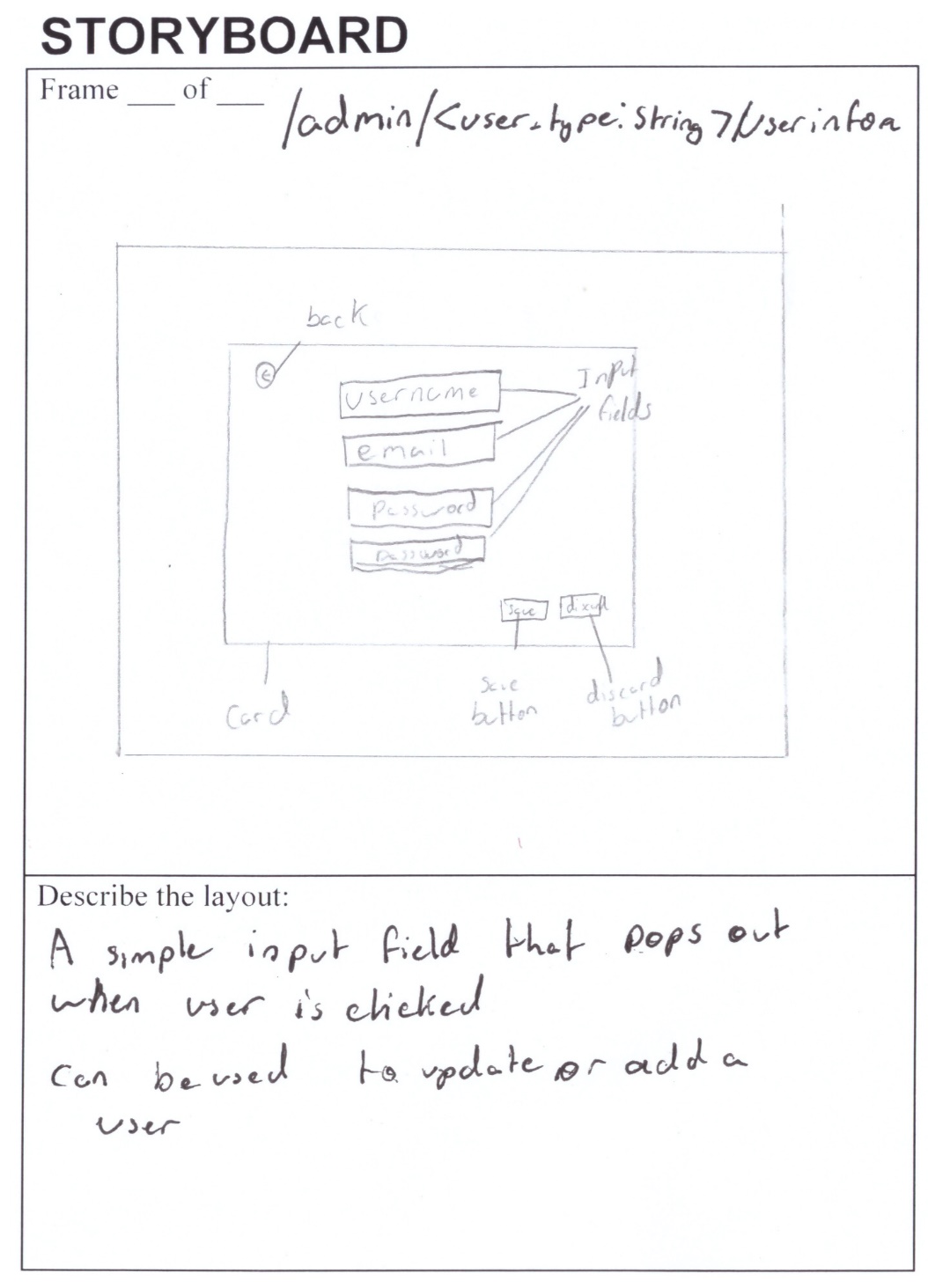
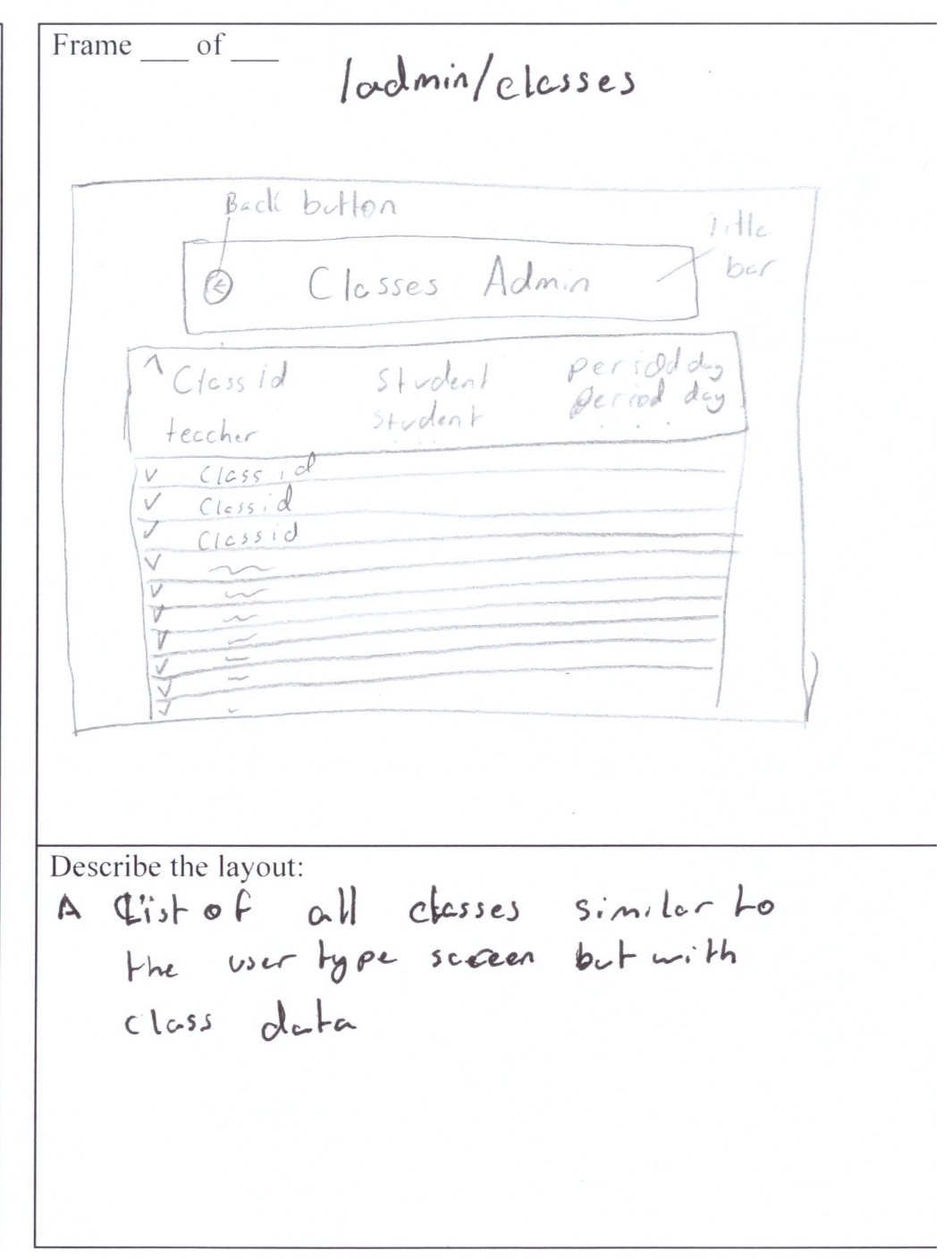
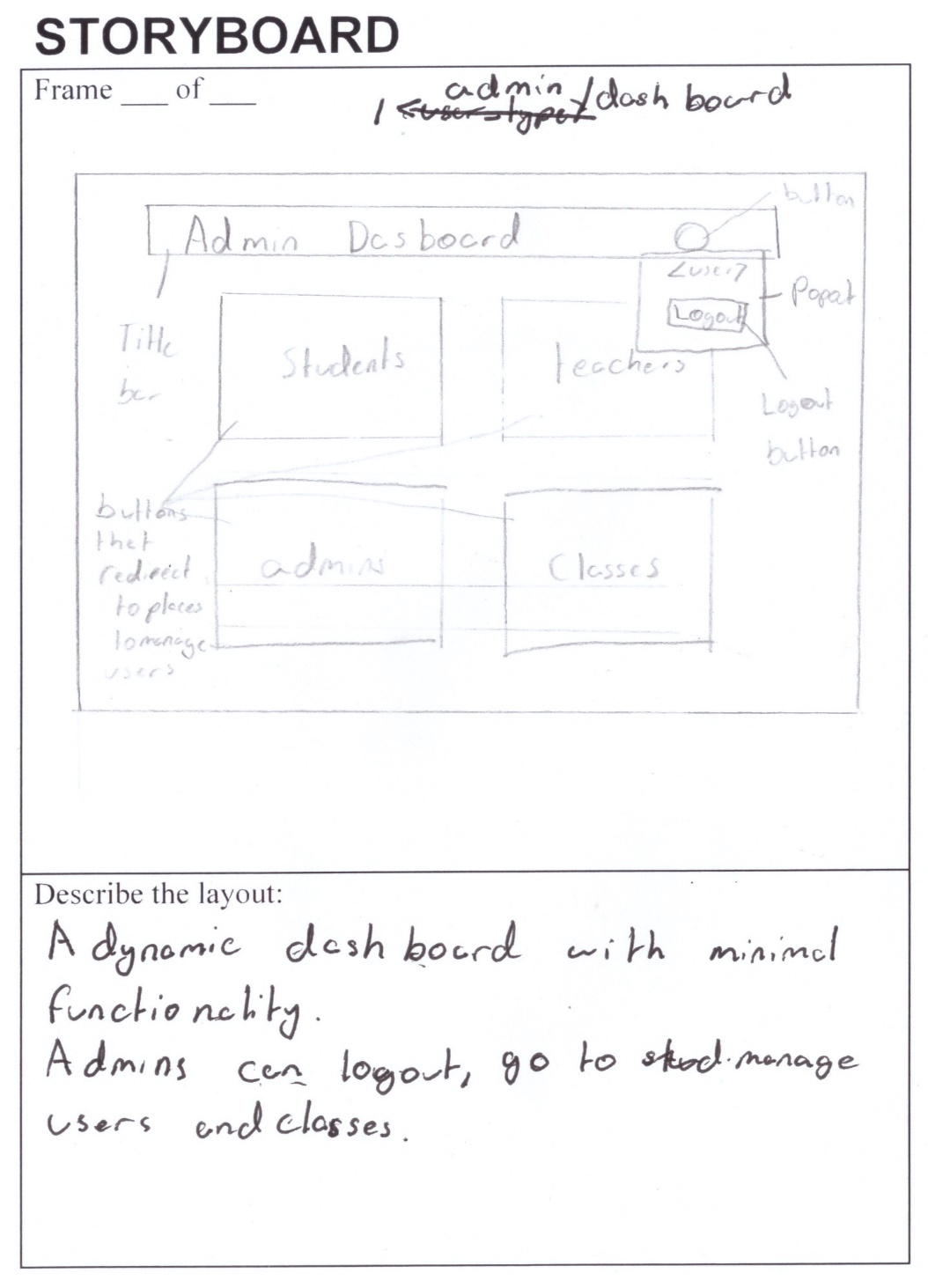
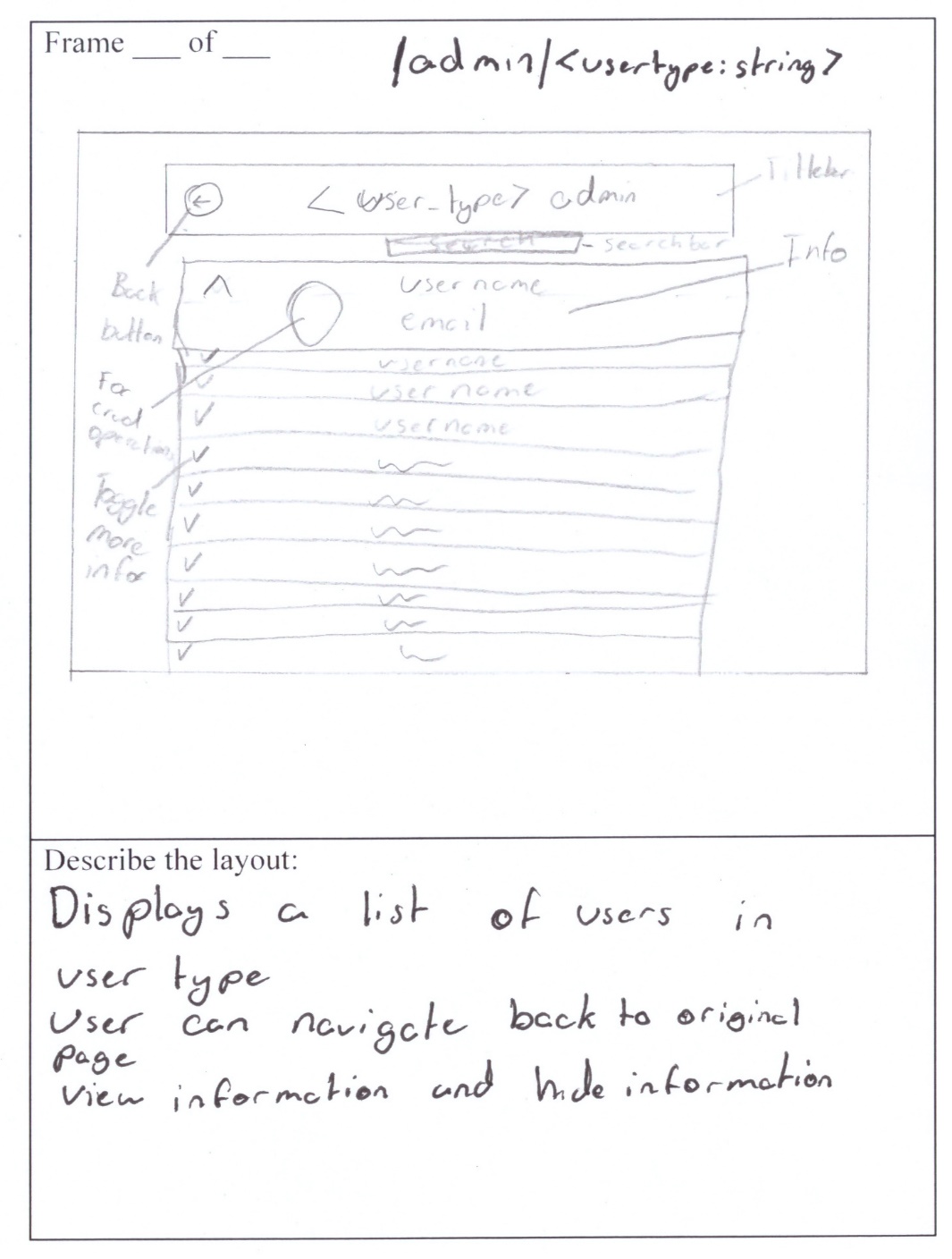
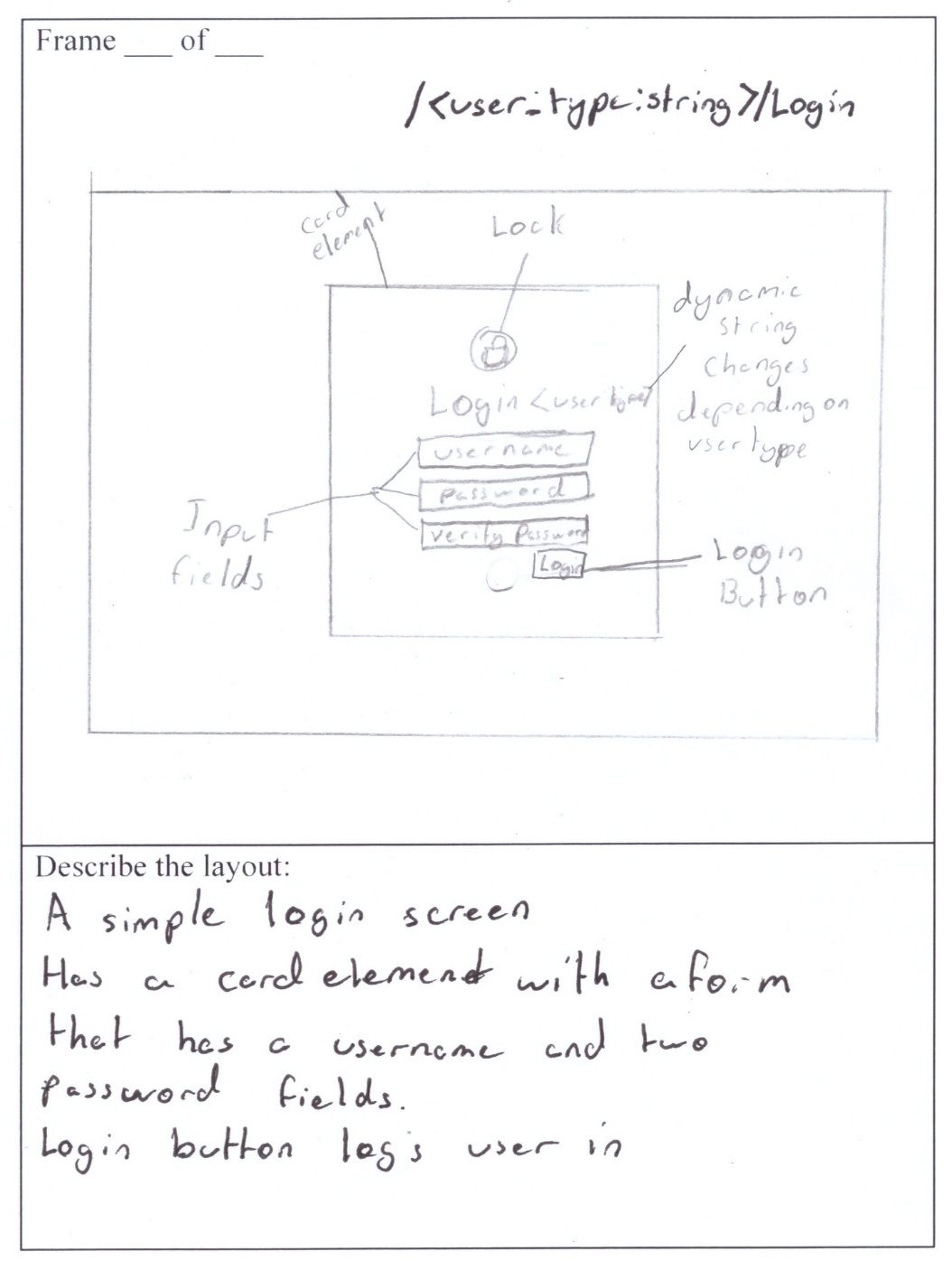
User Authorization



Dropbox function



## 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 | ‘satohusaohushou’ |
| 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 | eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiaWF0IjoxNTE2MjM5MDIyfQ.SflKxwRJSMeKKF2QT4fwpMeJf36POk6yJV\_adQssw5c |
| User\_type | Enum | Used to verify what table data should be stored in | Admin | Teacher | Student |
| User id | Integer | The primary key for a user in a database | 1 |
| username | string | Identifier of a student | ROEU01 |
| email | string | A email address that is helpful for teachers | ROEU01@stpiusx.nsw.edu.au |
| Password\_hash\_and\_salt | Byte array | A secure way to validate passwords without storing passwords | $2b$10$//DXiVVE59p7G5k/4Klx/ezF7BI42QZKmoOD0NDvUuqxRE5bFFBLy |
| Class id | Integer | The primary key for a class | 1 |
| Class class id | string | The identifier of a class | 12SDD |
| Term start | date | Identifies the start of a term | 1/1/2019 |
| Term end | Date | Identifies the end of a term | 30/12/2019 |
| periods | enum | Identifies the period on a timetable | Start | recess | lunch | end | 1 | 2 |
| mark | int | Identifies the mark a student received for a task | 31/45 |
| Task notification | string | A notification for a task | Do this task by this day |
| Task submission | file | A students submission for a task | Example.docx |

# Glossary of Key Terms

* GraphQL API: GraphQL stands for Graph Query Language, it is a language specification similar to SQL, which defines a certain way for client a server interactions. A client will submit a GraphQL request to route /graphql and receive a json file containing requested information. Some benefits of using GraphQL over rest, is that coding routes is much simpler because you only need to worry about client side routing, and state management, because GraphQL clients, such as Apollo handle state.
* State: state is simply locally stored variables within the application, handling state refers to storing variables when a screen or route changes.
* ES6, ES7 and Javascript: Javascript is a scripting language used to modify documents (html) on the web. Javascript code executes in web browsers rather than servers. In the past Javascript was hard to read, tedious to write, and had problems like block scoped variables, which a quick google search will explain. Javascript gets updated periodically, and a new version called ES6 was released in order to update syntax and get rid of some of the aforementioned issues. ES7 brought further support for client server interactions with fetch and async functions. The only problem is that some browsers do not work with these new javascript versions.
* Babel: babel is a javascript compiler that compiles ES6 and react jsx into code that all browsers can understand.
* React: React is a Javascript library with the intent of creating single paged applications, it is component based.
* React JSX: jsx is a way of writing html like code in javascript, and then injecting it using react into a website page. Through this a website can completely change but only requires a single html document. Jsx needs to be compiled using a javascript compiler such as babel.
* React Router: A react extension that allows routing react components, to create a separate page for things like login screens.
* Webpack: webpack does things like compile javascript, minimize it, lint it, and other various things in one build step.
* Yarn: yarn is a dependency manager for javascript
* Apollo client: Apollo client is a javascript library used to connect javascript to graphql apis
* Golang: Go is a programming language developed by google that has various features making it ideal for web development such as modularity and in build http handling.
* Hash: a hash is a function that takes an input and generates a unique output based on the input. It is ideal for storing passwords because if you hash a password, you can turn the password into the hash, but you cant turn the hash into the password.
* Bcrypt: a hashing function that is good for making passwords because its slow, fast enough that a user wont notice if the password is correct, but slow enough that if someone tries to brute force random passwords, it will take ages before the hash is cracked.
* Salt: a salt is a hash of a hash, if you bcrypt hash 8 times, the amount of times a potential attacker will have to hash random passwords will increase to the power of 8, so it is a beneficial practice.