Final Year Project Report

Notes Application Final Report

08872848 BSc in Computing

Conor Curran

Table of Contents

1. Executive Summary

Introduction

The core aim of this project is to provide a cloud based school administration system.

This document presents the final project report of the “Notes” application. Some changes have been made from previous documents such as the Requirements Specification. Overall the main characteristics and functionality of the application remain more or less the same. In addition to the previous proposal I completed work on a homework tool that allows students to confirm homework completion, render data on completion rates and other various interaction tools. Any changes are described below.

Attached in the Appendix is the original project proposal document as well as the monthly journals.

Background

During my internship in summer 2015 I gained a good insight in to the lack of standardised digital record keeping in Irish schools. The vast majority of schools don’t even have a website and if they do rarely are they maintained to any good standard. Below is a high level look at what I have implemented.

It is common practice to this day that record keeping and school disciplinary procedures are still carried out by the teacher, in the first instance, using traditional pen and paper. At a later date this information may be put in to a spreadsheet.

Whilst this method may represent an adequate way of maintaining records it could be streamlined to a much greater extent through the use of modern mobile web and cloud based technology.

By digitising this process it could save teachers and administrators time and the school money in terms of paper and storage requirements. They should be able to do on a smartphone, lap top or iPad what they would normally do on paper.

The purpose of this project is to provide a platform to enable them to do that.

The app also cuts out the need to send letters to student's homes by sending automatic messages to parents on the basis of any threshold (decided and implemented by the school through the app administration) being exceeded which would indicate the student is having difficulty or is being unreasonably disruptive.

The User Interface is designed to be as simple, quick and intuitive as possible with minimum input required from the user through the front end view for maximum output from the back end. The whole process only takes a few seconds.

In terms of practical implementation, I have implemented a small number of dynamic select dropdown menus while avoiding text fields.

This is a much slender, streamlined and efficient way of maintaining records but it also gives much greater latitude to the user in terms of time allocation.

The school decides what number of points are allocated to what area of disruption. They can do this via the app administration. For example, a case of bullying may carry 5 points and the school could decide at what point warrants a text to a parent or what action might be taken so that they are in full control of their own disciplinary procedures.

2. REQUIREMENT SPECIFICATION

Technical Approach

2.1 Core Requirements:

2.1.2 Actions (User Stories):

2.1.2.1 Actor: User

1. As a **"User"** I can "Log-in" so that I can “view/interact with my dashboard and the system depending on the role associated with my account – Guardian, Teacher, Student or School”

2.1.2.2 Actor: Guardian

1. As a **“Guardian”** I can “Log-in” so that I can “view my dashboard which contains metricised graphical visualisations”
2. As a **“Guardian”** I can “View” automated notifications to my mobile device/phone of remedies given to my child by the school through the system for breaches of discipline
3. As a **“Guardian”** I can “**Manage Notes**” so that I can:

    <manage notes use case>

    A. “Create” a note for the teacher of my child
    B. “View” notes sent to me by the teacher

4. As a **“Guardian”** I can “View Guardian Message” sent from teachers in the school so that I am involved
5. As a **“Guardian”** I can “View Grievance” record for my child so that “I am informed of my child’s discipline records”
6. As a **“Guardian”** I can “View Homework” so that I am aware of what homework is given, completed and outstanding

2.1.2.3 Actor: Student

1. As a **“Student”** I can “Log-in” so I can “view my dashboard which contains graphical visualisation of my individual behaviour metrics”
2. As a **“Student”** I can “View Student Message” so I can “view teacher messages through my inbox”
3. As a **“Student”** I can View Grievance so I can keep check of my behaviour metrics
4. As a **“Student”** I can View Homework so I can Manage Homework

2.1.2.4 Actor: Teacher

1. As a **“Teacher”** I can “Log-in” so that I can “view my dashboard which contains graphical visualisation of the behaviour metrics associated with all students in my class”

2. As a **"Teacher"** I can "View Notes" so I can "keep track of notes sent to me by guardians of students in my class"
3. As a **"Teacher"** I can "Manage Messages" so I can "communicate messages to guardians of students in my class"
4. As a **"Teacher"** I can "View Messages" so I can "read messages sent from guardians relating to their and respond/act accordingly"

2.1.2.5 Manage Homework Use Case and User Stories:

1. As a **"Student"** I can "**Manage Homework**" to:

A. "View Homework" assignments given to me by my teacher
B. "View Non-Interactive Homework Feedback" so I can view feedback I have given for individual assignments
C. "Confirm Homework Complete" so I can "click to confirm assignment completion for the teacher and my parent":
    i. Rate "Difficulty Score" so I can provide numerical rating feedback (1 – 10) for difficulty on assignment
    ii. Skip "Difficulty Score" so I do not need to provide "Difficulty Score" feedback
    iii. Rate "Interest Score" so I can provide feedback on my level of interest in the assignment
    iv. Skip "Interest Score" so I do not need to provide "Interest Score" feedback

2. As a "**Teacher**" I can "**Manage Homework**" to:

A. "View Bucket Items" so I can view homework items in my storage bucket:
    I. "Delete Bucket Items" so "I can remove homework items from my bucket"

B. "View Homework Feedback" so that "I can view feedback on homework assignments from my students":

    i. "Filter Homework Feedback" so I can filter through students in the class
    ii. "Select Homework Items for Storage" so I can choose homework items I wish to store
    iii. "Add Selected Items to Storage Bucket" so I can add above items to my bucket

C. "Create Homework" so I can create homework assignments for my class:

    I. "Create" Homework assignments and provide a due date
    II. "Edit" Homework assignments

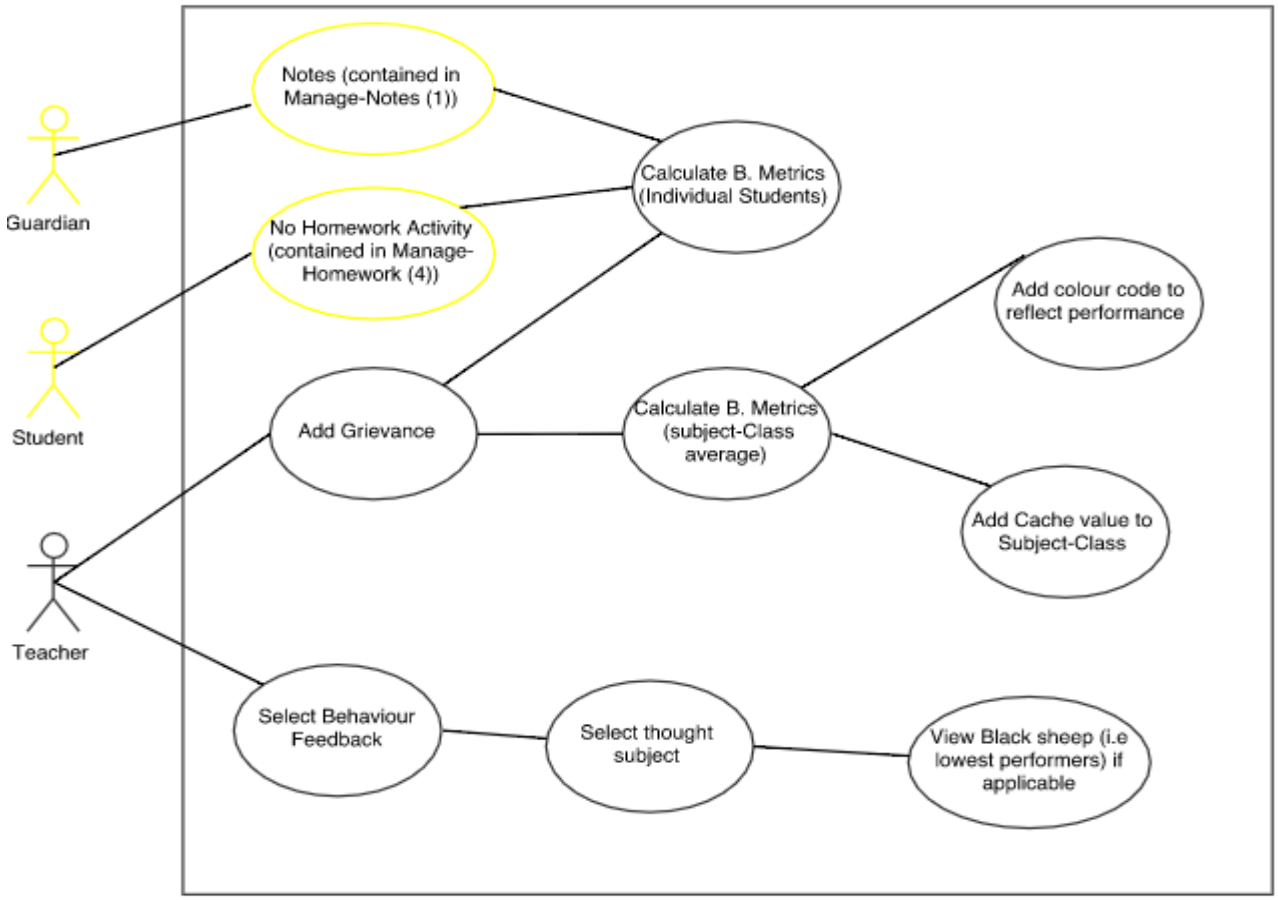
3. As a "**Guardian**" I can "**Manage Homework**" to:

A. "*View Homework Feedback*" so I can view homework feedback for the teacher from my child
B. "*View Non-interactive Homework Feedback*" ? <duplicate>
C. "View Homework" so that I can "view homework assignments given to my child by the teacher"

4. As the **"System"** used to **"Manage Homework"** to:

A. “Calculate Cache Average Score” so the average homework score for each student is calculated and stored in a cache

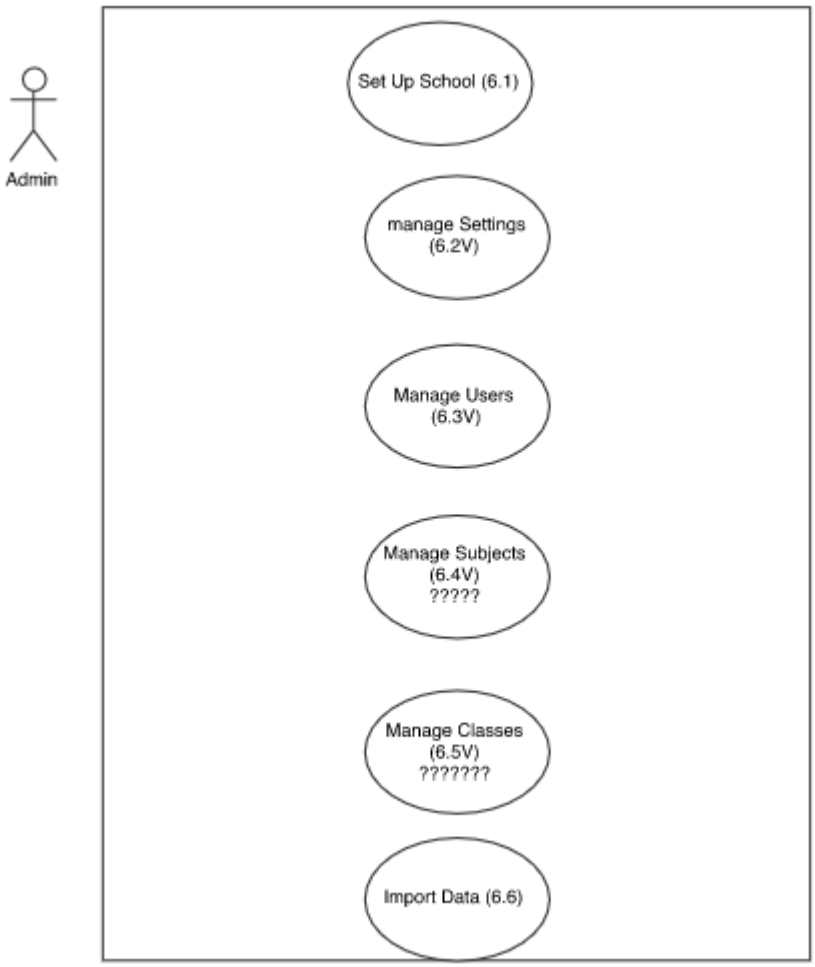
2.1.2.6: Manage Grievance Use Case and User Stories



5. As a “**Teacher**” I can “**Manage Grievance**” to:
    A. “Add Grievance” so I can “input a record through the form (choose student name, choose grievance) for a student in my class”
    B. “Select Behaviour Feedback” so I can “view class behaviour records”:
        I. “Select *thought* (through?) subject” so I can “filter the records based on subject”
        II. “View Black Sheep” so I can “view lowest 5 performing students in each subject”

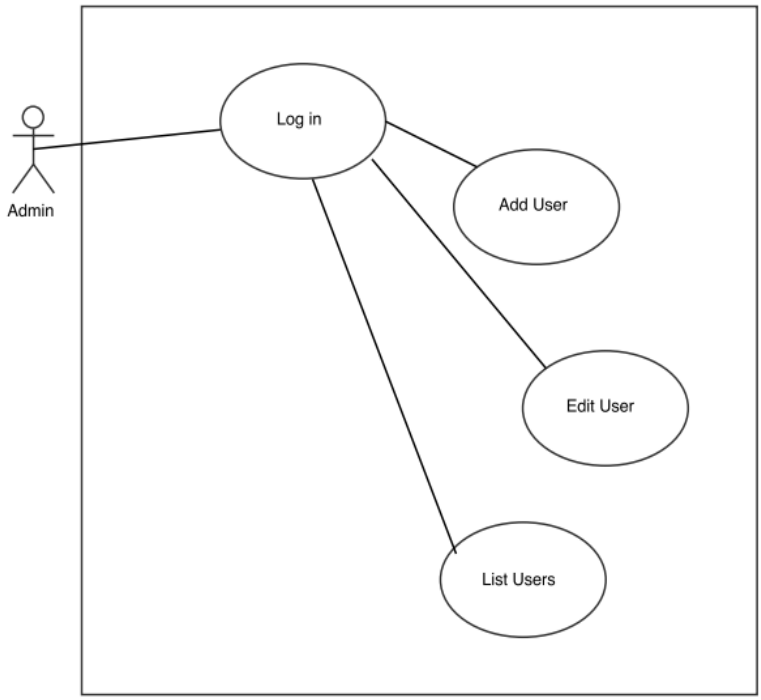
2.1.2.7 Actor: Administrator

Manage School, Overview:



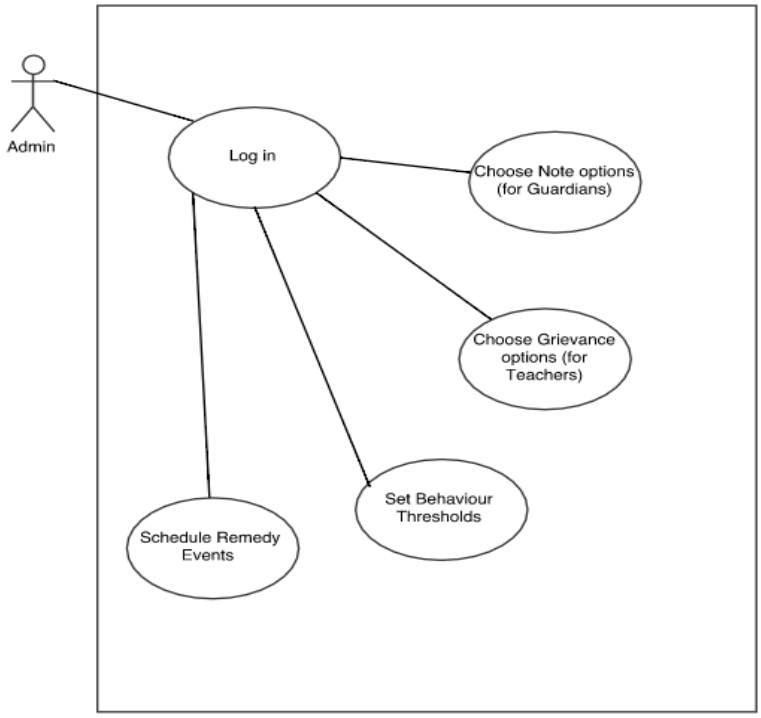
1. As an **"Administrator"** I can "Import Data" of users to the system through a CSV uploader
2. As an "**Administrator"** I can "Log-in" to my administration console to "administer records in the system"

2.1.2.8 Manage School – Manage Users



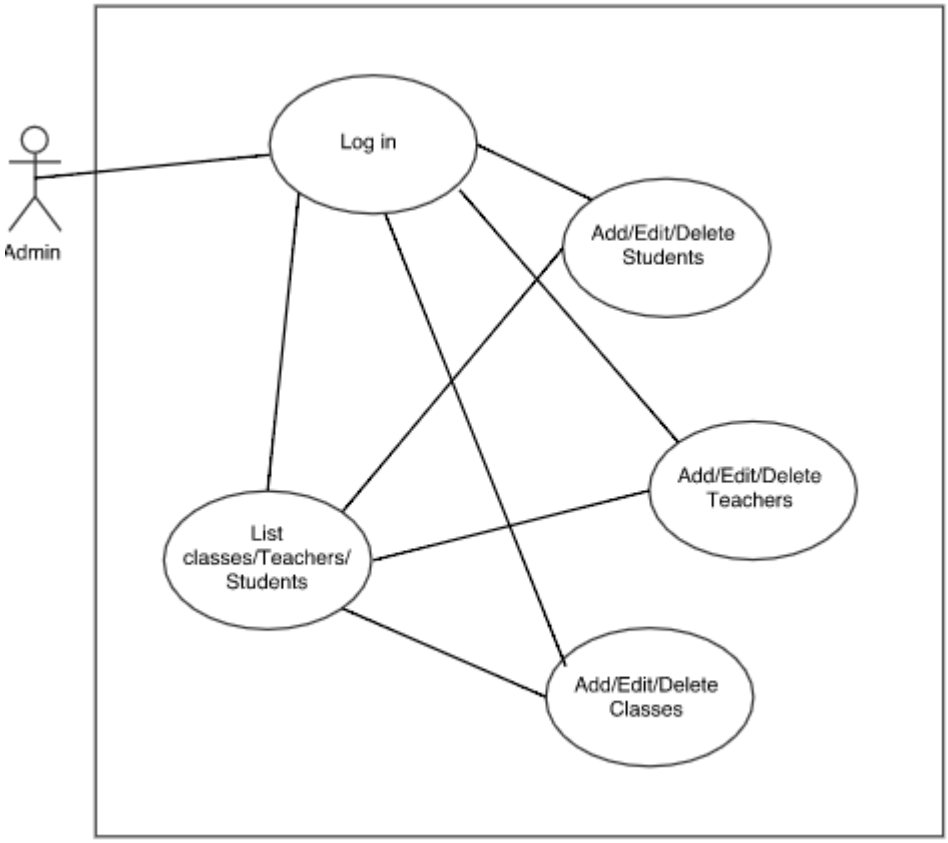
3. As an **“Administrator”** I can “**Manage School**” – **“Manage Users”** to:
    A. “Add User” so that I can “add a new user to the system and assign them a role within the system”
    B. “Edit User” so that I can “update user details where required”
    C. “List Users” so that I can “view a list of users and filter by their roles”

2.1.2.9: Manage School – Manage Settings



4. As an **"Administrator"** I can **"Manage School"-"Manage Settings"** so I can:
    A. "Choose Note Options" so that I can note options for guardians
    B. "Choose Grievance Options" so that I can "input the behaviour items to be measured"
    C. "Set Behaviour Treshold" so that I can "set the rate in percentage before automated communication is triggered with the guardian of a student for breach of discipline"

2.1.2.10: Manage School – Manage Classes



5. As an **"Administrator"** I can "Add/Edit/Delete Students" so I can add students to the system, edit their details or soft delete their records
6. As an **"Administrator"** I can "List classes/Teachers/Students" so I can view user group records and filter by role or other attributes associated with a user
7. As an **"Administrator"** I can "Add/Edit/Delete Teachers" so I can add teachers to the system, edit their details or soft delete their records
8. As an "**Administrator"** I can "Add/Edit/Delete Classes" so I can add classes (student_groups) and subjects associated with those classes to the system, edit these details or soft delete classes and subjects from the system

2.1.2.11 Actor: Super User

1. As a **"Super User"** I can "add" an administration user for the school

2.1.1: Bluemix:

Utilise IBM Bluemix platform to host project

2.1.2: Front end coding:

Ruby on Rails, jQuery, JavaScript, CSS and other attributes within framework

2.1.3: Back end:

MySQL using ClearDB service on IBM Bluemix (examining using NoSQL)

2.1.4: Twilio:

Integrate Twilio communication service through Bluemix thereby enabling SMS or messaging service.

2.1.5: User Login, Admin:

Devise and Active Admin to be used within Rails structure to form basis of user authentication and site administration.

2.1.6 Database:

The project incorporates a MySQL database and contains the following tables:

- Users (typical user table data)
- Teachers (teacher information)
- Students (student information and parent contact number)
  Guardians (guardian information and contact number)
- Penalty (Grievance + Penalty association and count)
- Notes (core app records)
- Admin (admin user data)
- Archive (for soft delete)

The database is much bigger at this point. I have attached an additional document to the application to reflect this.

It would be preferable to keep the number of tables under 8 to avoid having to implement a schema load when pushing the app to the Bluemix platform and thus losing data in the database with each upload.

2.1.7 Archiving: Due to the nature of the data intended to be collected these are not records that should be able to be simply deleted. Records deleted from the main Notes table should be automatically sent to archive which would only be accessed by admin.

2.1.8 Graphical Data Representation Dashboard: Highcharts JavaScript Library

2.2 Project Scope:

The scope of the project at the highest level is to design, build and deploy a functional web and mobile application that will allow the user to create, add, edit and delete records in a database.

The application has features built in, some from 3rd party sources, that provide for an intuitive and easy to use program tailored to the needs of the customers – in this case secondary schools and/or government agencies like the Dept of Education.

To achieve this within the constraints imposed by the subject matter that the application deals with several crucial points had to be addressed:

- Data Protection
- Robust and reliable functionality
- Security
- School administration requirements
- Schedule

Data Protection:

The data intended for use with this application is highly sensitive and therefore research will have to be completed involving the data protection commissioner and other parties to ascertain what, if any, data protection issues exist that could have a detrimental or negative impact on both the delivery and sustainability of the application.

Functionality:

The application has to be easy to use, quick, robust and reliable. The user interface must be as simple and intuitive for the user as possible. The database must be designed to work as efficiently and securely as possible.

Security:

As with data protection outlined above security must be to the highest standards to provide reassurance and the safest possible environment in which to store the collected data.

Administration Requirements:

As outlined in the project proposal document a key part of this application is allowing the customer to determine the rules and thresholds that trigger events such as email or messages to parents. It is most important that the school is able to decide it's own procedures and implement them through simple steps in site administration.

Schedule:

A project plan and gantt chart has been provided within the project proposal document.

2.3 User Requirements Definition:

The customer requirement revolves around ease of access, time saving and cost efficiencies.

- The User must be able to login (Priority – Highest)
- The User must be able to create records (Priority – Highest)
- Admin can update or destroy records (Priority – Highest)
- User must have access to Dashboards (Priority – Highest)
- Dashboard must be populated with graphical representation of data in database(Priority – High)
- An archive must be provided for "soft delete" of records(Priority – High)
- Application must automatically deliver messages to parent of student (Priority – High)
- Easy to learn and well laid out application documentation must be provided to users (Priority – High)

3. Functional Requirements

3.1 Functional requirements:

- **Requirement 1:** Admin User Registration

  It is intended that all new users will be registered through Admin. For the purposes of the prototype and testing users can register their own accounts.

- **Requirement 2:** Login

All users must be able to log in and log out.

- **Requirement 3:** User differentiation, roles

  Devise, Rolify and CanCan gems will be used to provide authentication. Admin sets roles. The two roles are “admin” and “teacher”. A guest account will have access to “about” only.

- **Requirement 4:** Form and dynamic select options

  Form to consist of collection select dropdowns using grouped_collection_select method:

```
<div class="field">
  <%= f.label :student_id, "Student by alias" %><br />
  <span class="glyphicon glyphicon-forward"> </span> <%= f.grouped_collection_select :
    studentname, Classmodule.order(:subject), :students, :subject, :s_alias, :s_alias, {prompt
    : "Student Name"}, {class: "form-control"} %>
</div>
```

- **Requirement 5:** Database

  My SQL to be used with connection to local MySQL Workbench via ClearDB service on the cloud platform. The application must be connected to the database.

  The local database settings are configured in the database.yml file.

- **Requirement 6:** Graphical Dashboards
  jQuery, JavaScript to be used to provide graphical representation of data.

- **Requirement 7:** Email, SMS or MMS

  The application must be able to automatically send messages to parents based on whether a student has breached a certain threshold in terms of points accumulated that will trigger a “send” event.

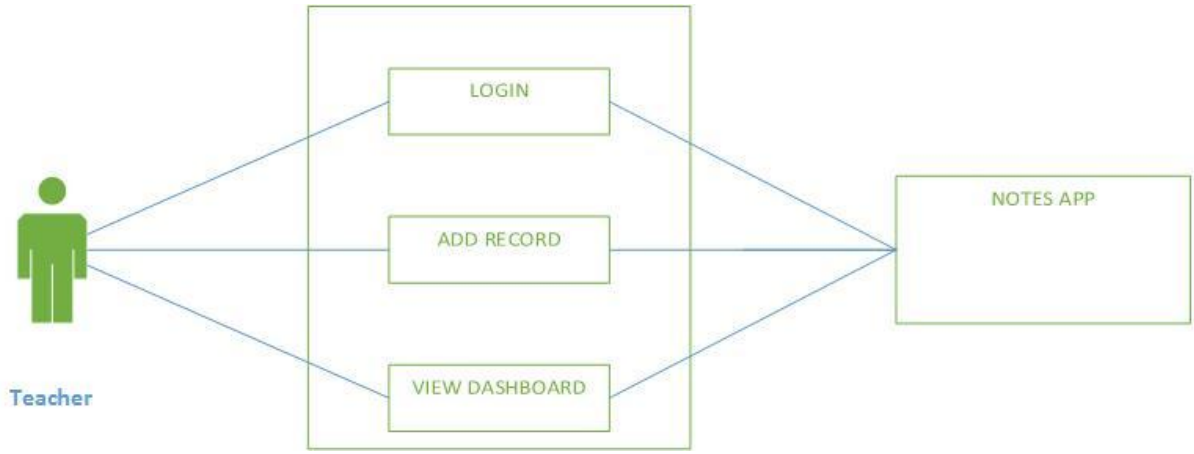
- **Requirement 8**: Homework Tool

  The application will contain an interactive Homework tool that will allow the teacher to enter in homework for students. Students can engage through comments, can mark their homework as completed and using a graphical quadrant can show whether they thought the homework was easy or difficult.

3.2 Local and Cloud based Technologies:

- JavaScript
- Twilio
- Ruby on Rails
- MySQL
- Cloud Foundry
- ClearDB
- Auto Scaling
- AppScan Security Services
- Sublime Text
- GitHub Version Control

3.2.1 Use Case Diagram(s) (Original):



Flow Description

Precondition: The User must be logged in to the mobile application with the correct privilege (teacher).

Activation: The User selects option from dynamic dropdown options.

Main Flow:

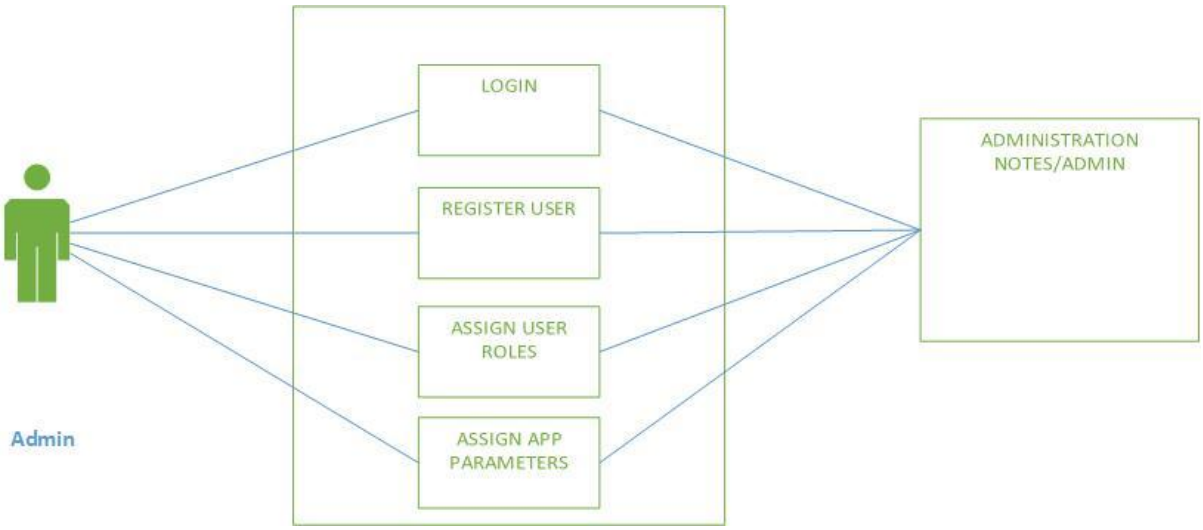
1. User selects applicable class module/subject
2. User selects student from automatically populated dropdown

3. User selects from a list of grievances
4. User selects “Submit Note” and verifies prompt
5. User routed to dashboard
6. User can enter new record or log out

Alternate flow: N/A

Exceptional flow: N/A

Termination: Automatic



Flow Description

Precondition: The User must be logged in with the correct privilege as an administrator.

Activation: The Administrator accesses the administration console.

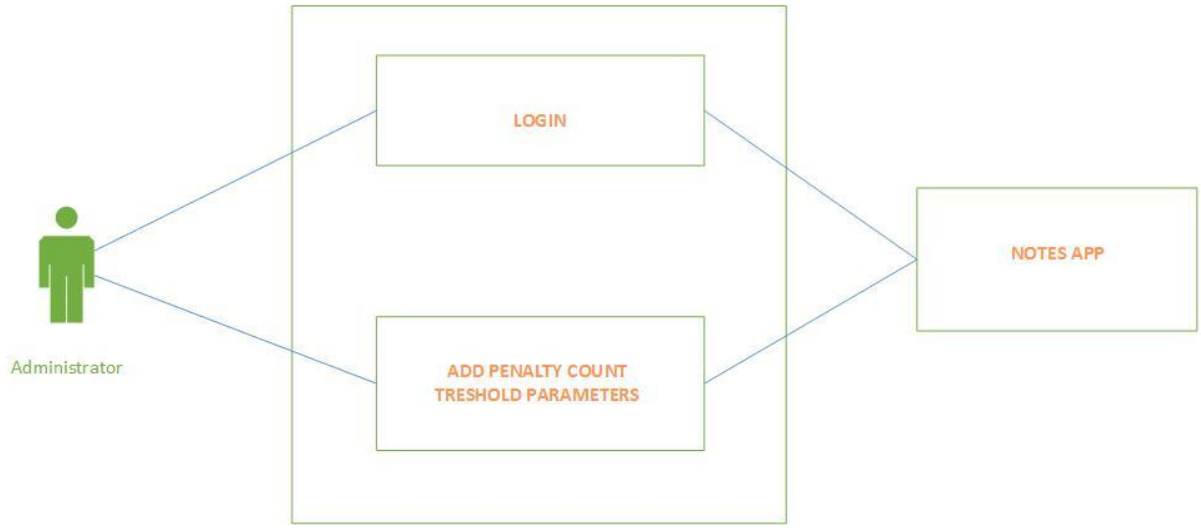
Main Flow:

1. Admin inputs user and student data and assigns roles
2. Admin assigns students to classes in accordance with model relationship restrictions
3. Admin sets “penalty” criteria for individual grievance options
4. Admin save changes
5. Admin logs out

Alternate flow: N/A

Exceptional flow: N/A

Termination: Automatic



Flow Description:

Precondition: Admin signed in to admin dashboard and has access to penalty table to set parameters.

Activation: School admin determines count threshold for that particular school and uses input field.

Main Flow:

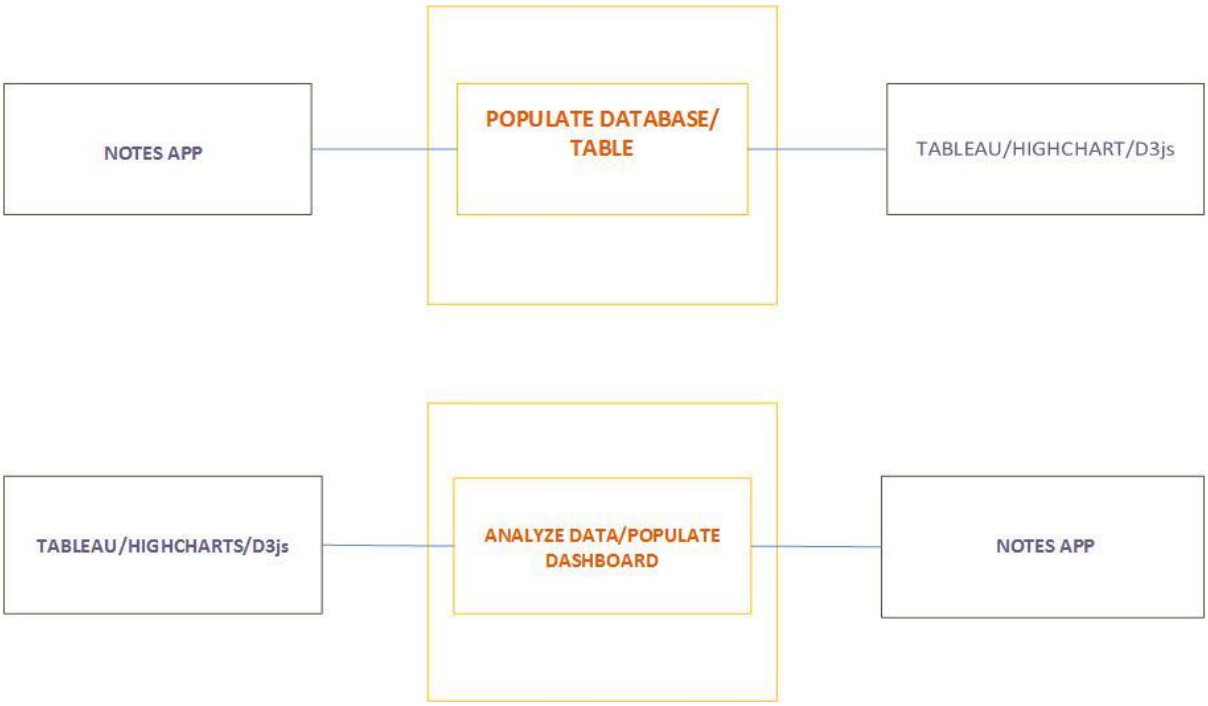
1. Admin signs in
2. Admin accesses penalty table dashboard to edit parameters
3. Admin edits and saves new parameters

Alternate Flow: N/A

Exceptional Flow: N/A

Termination: Automatic or admin signs out or presence is idle for a set period of time in which case they will be logged out automatically.

Description: This use case describes how the data added to the Notes record table is used to update the dashboard within a data visualisation programme file. It also describes how the Note record dashboard is subsequently updated in the web/mobile application.



Flow Description:

Precondition: The system is in constant initialisation mode.

Activation: Connect the chosen data visualisation tool to the 'Notes' record table.

Main Flow:

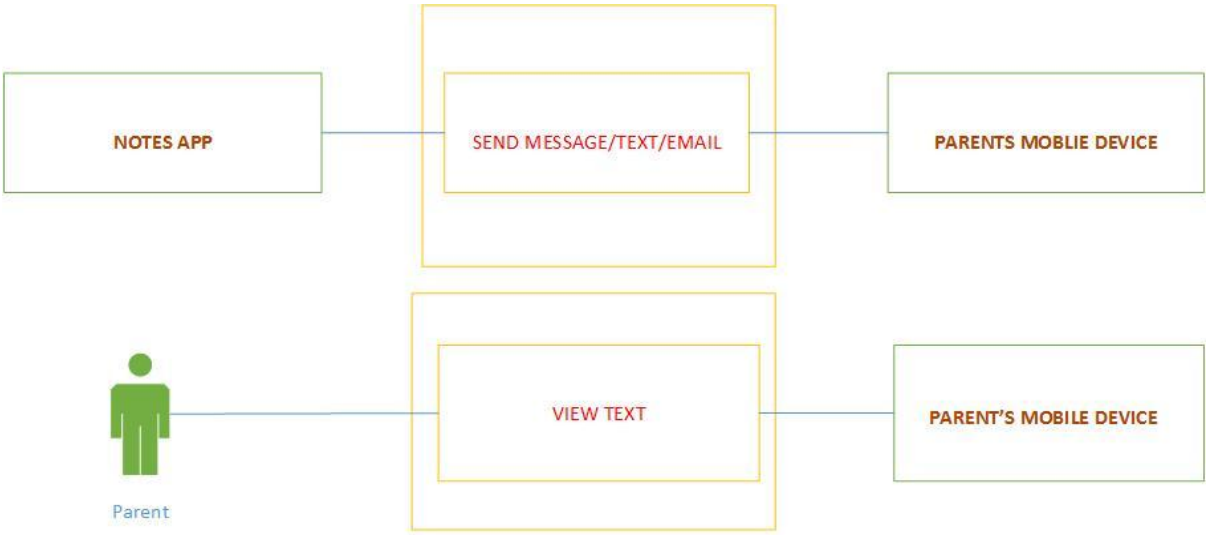
1. The data visualisation system shall update the graphs, tables and dashboard using the updated information from the user.

2. The data visualisation system shall automatically update the API embedded in the 'Notes' web/mobile application.

Alternate flow: N/A

Exceptional flow: N/A

Termination: Automatic



Flow Description:

Precondition: Child is in attendance at source school and there is an update available on the student's note record.

Activation: Automatic

Main Flow:

1. The system checks the database to see if there is an update to the student's note record.

2. If one or more updates exist that are greater than a defined threshold an email, message or SMS is sent automatically to the relevant parent's device.

Alternate Flow:

1. The system checks the database to see if there is an update to the student's records.
2. None exist, process ends.

Exceptional Flow: N/A

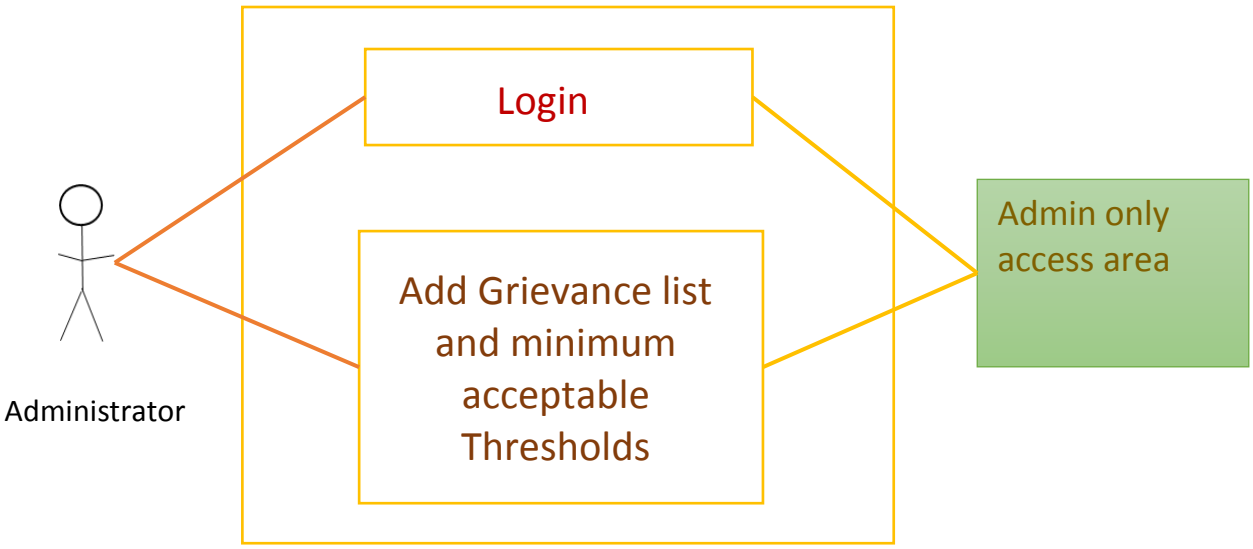
Termination: Automatic

3.2.2 Use Case Diagram(s):

Requirement 1: A User shall be able to log on to access resources based on their user group.

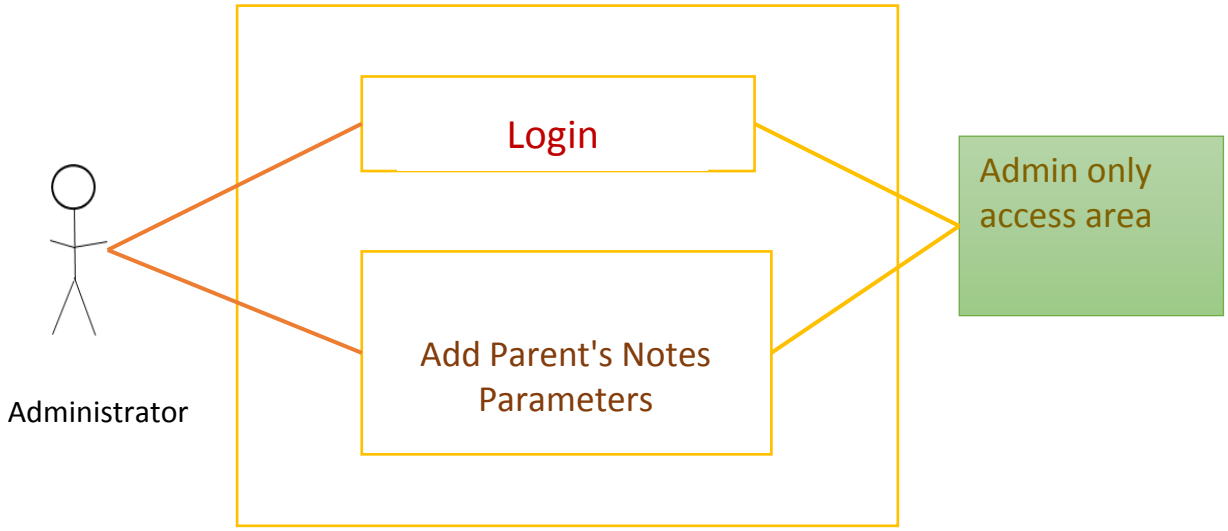
Stakeholder

Requirement 2: The administrator will be able to select the disciplinary actions that are equivalent to threshold breaches subject to the School's disciplinary procedure manual.



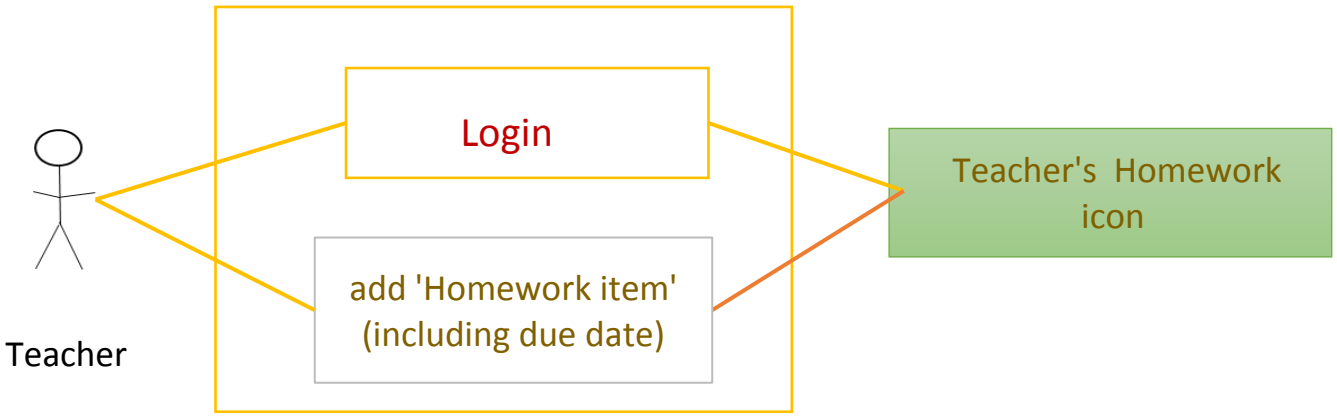
Administrator

Requirement 3: The administrator will be able to select the parameters for Parent's communication.

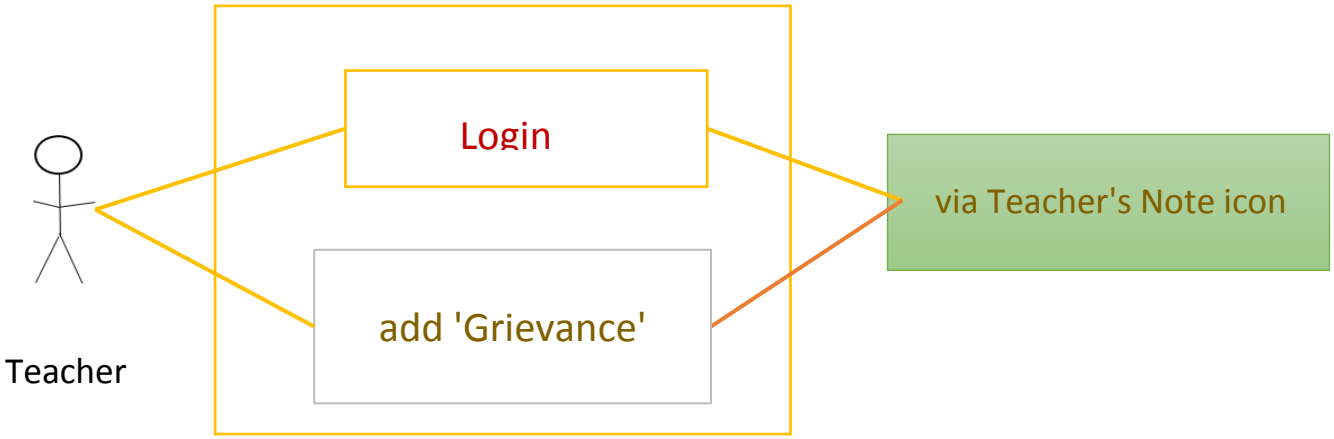


Requirement 4: Teacher can communicate relevant items, (currently homework and grievances).

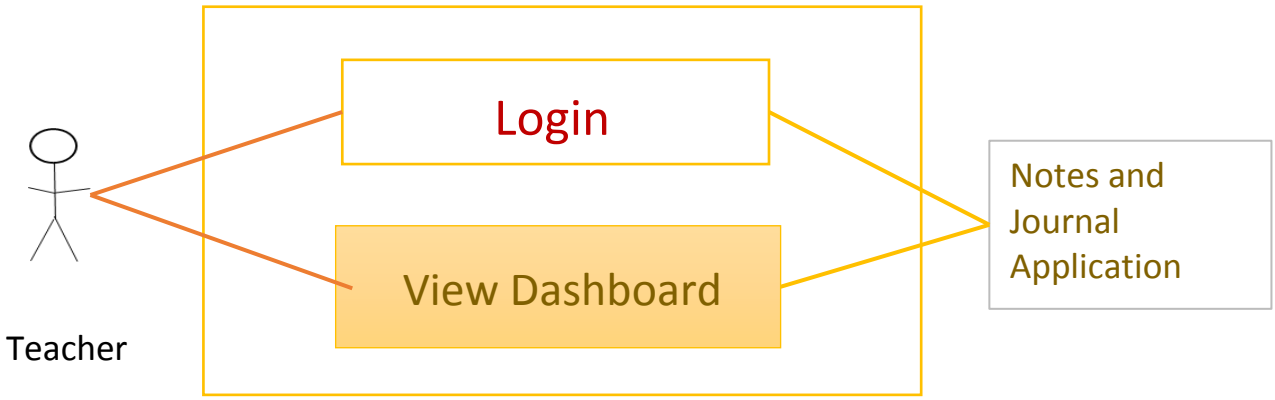
Requirement 4.1: Teacher can add a homework item to the system



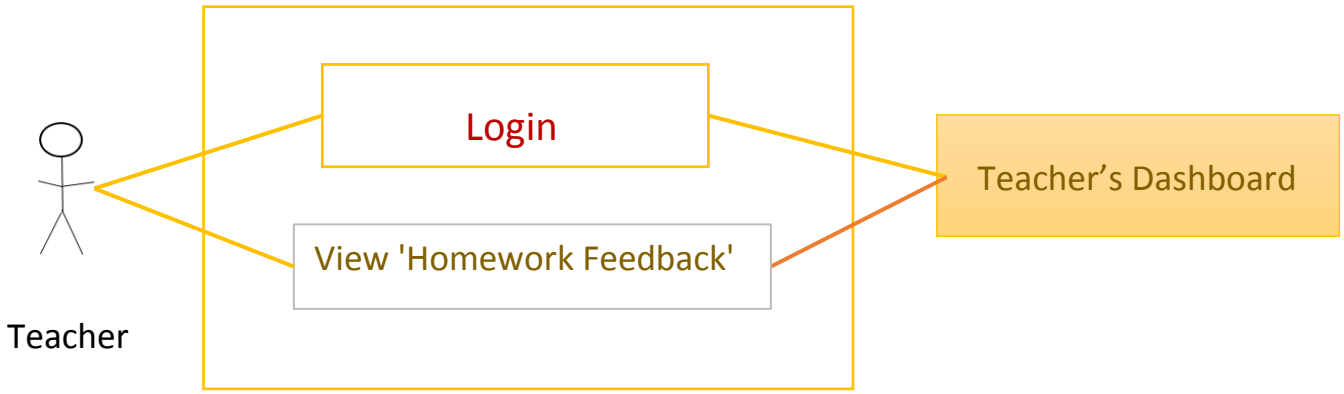
Requirement 4.2: Teacher can add a grievance item to the system



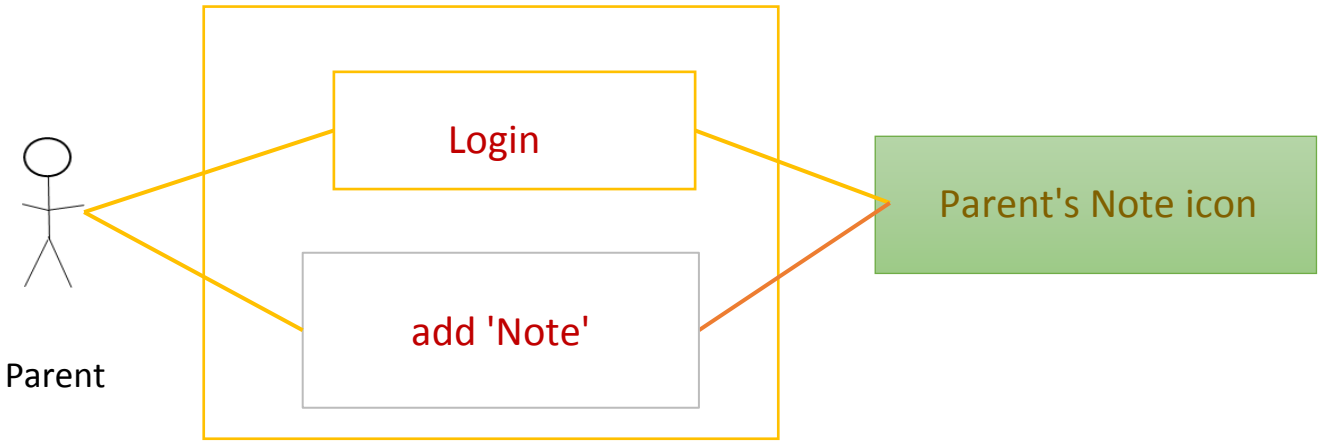
Requirement 5: Teacher can view a dashboard containing relevant information



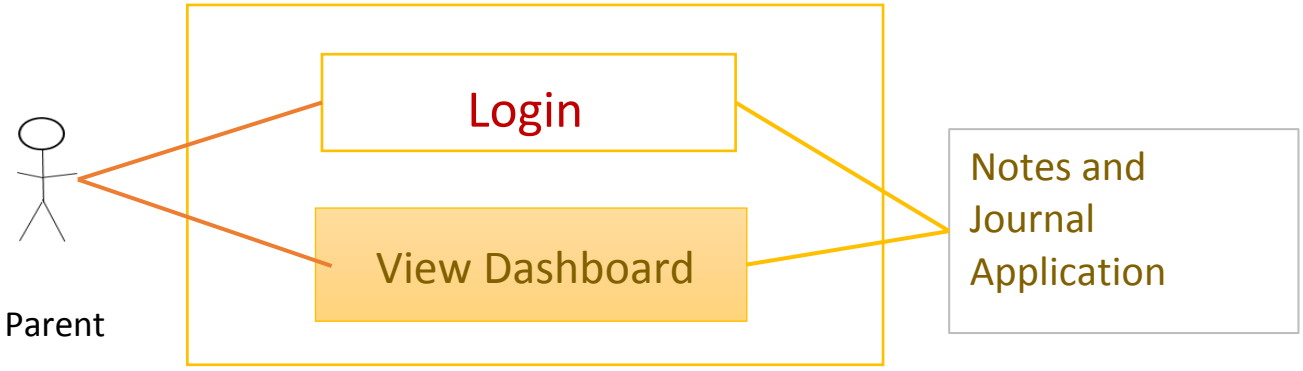
Requirement 5.1:Teacher can view homework feedback, provided by Students.



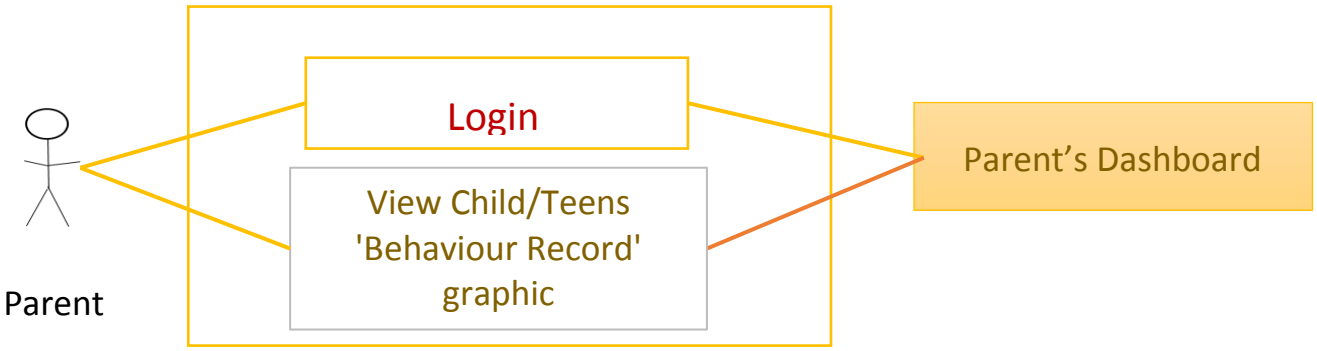
Requirement 6: Parents can communicate relevant items to School



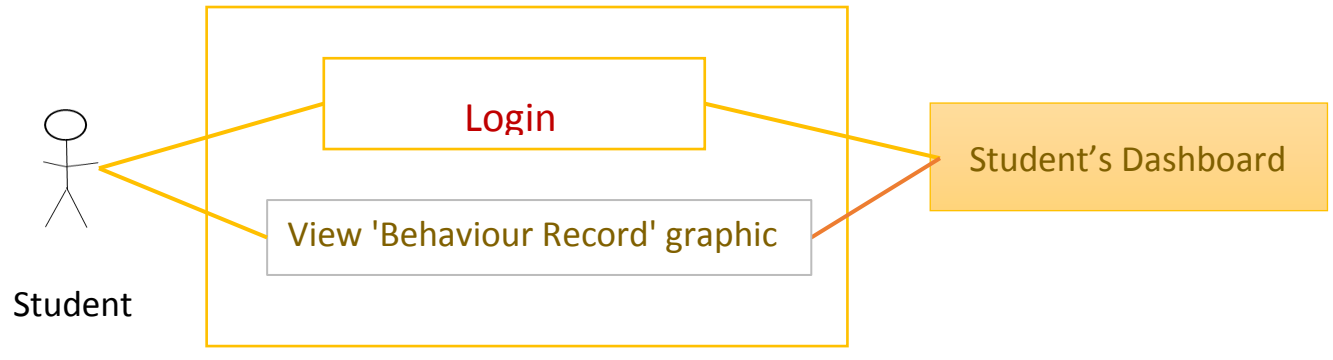
Requirement 7: Provide access to a Dashboard that provides a visual documentation of a Pupil's Grievance record and homework requirements.



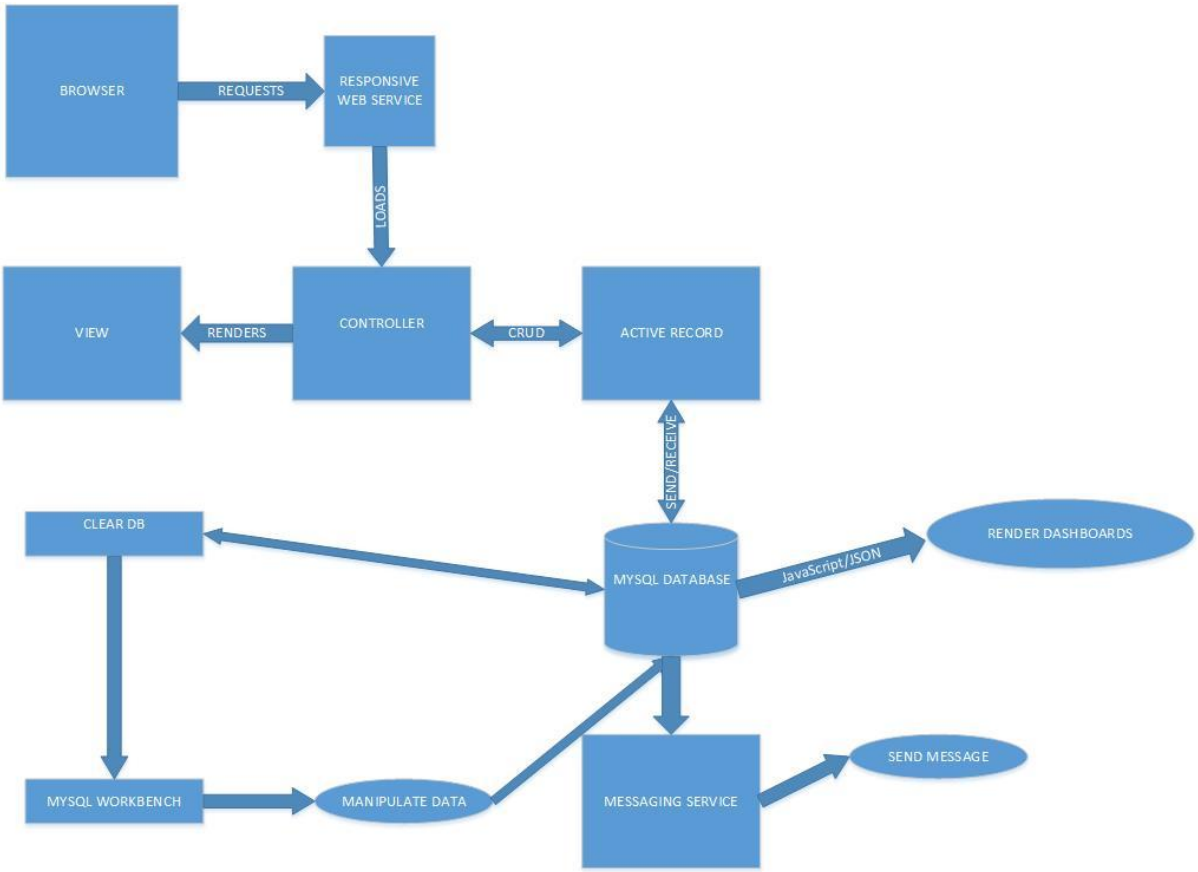
Requirement 7.1: Parents can view students behaviour record via an interactive graph



Requirement 8: Permit Students to view their behaviour record via an interactive graph

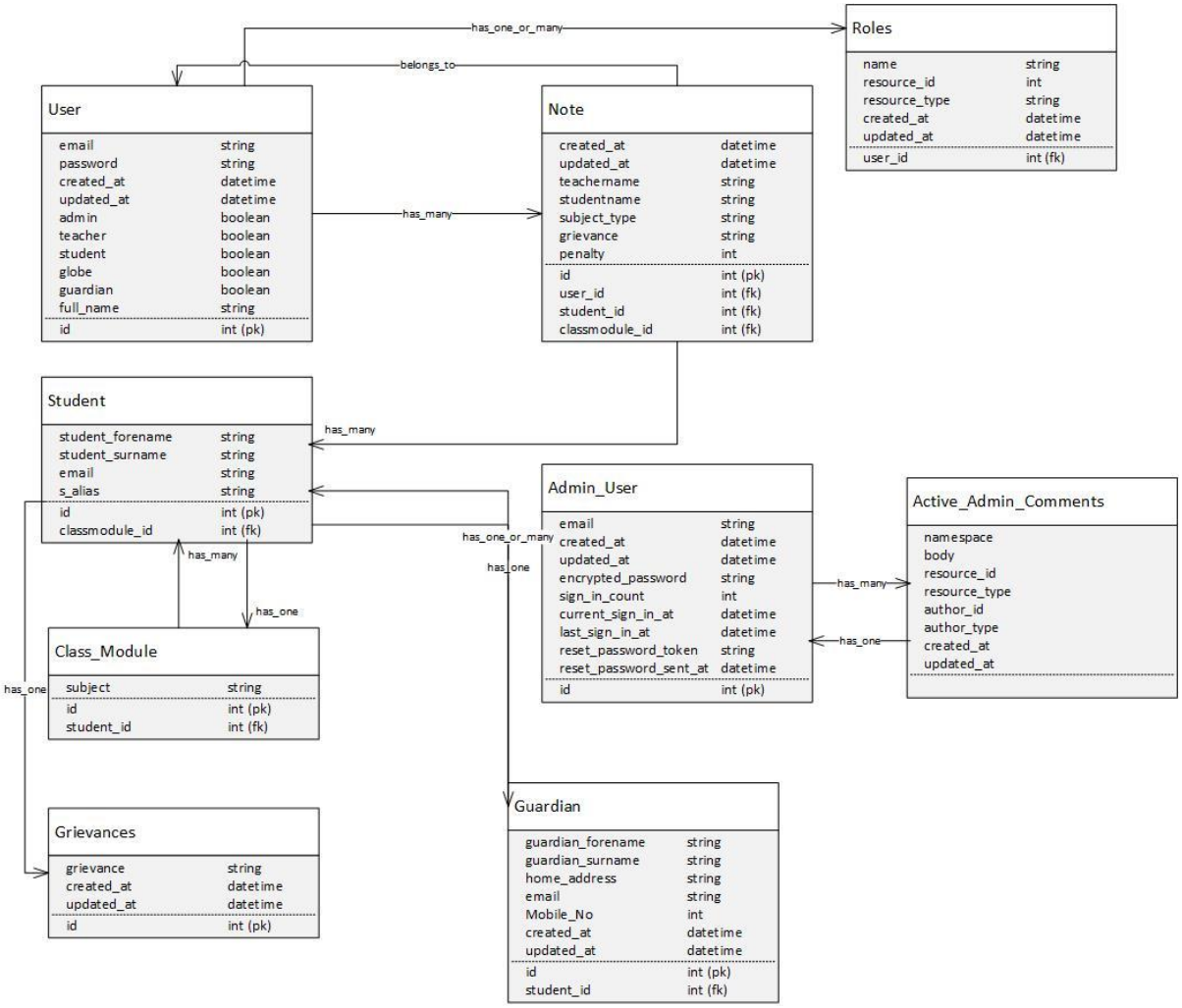


3.2.3 SYSTEM OVERVIEW:



The core system layout as above has not fundamentally changed in recent months.

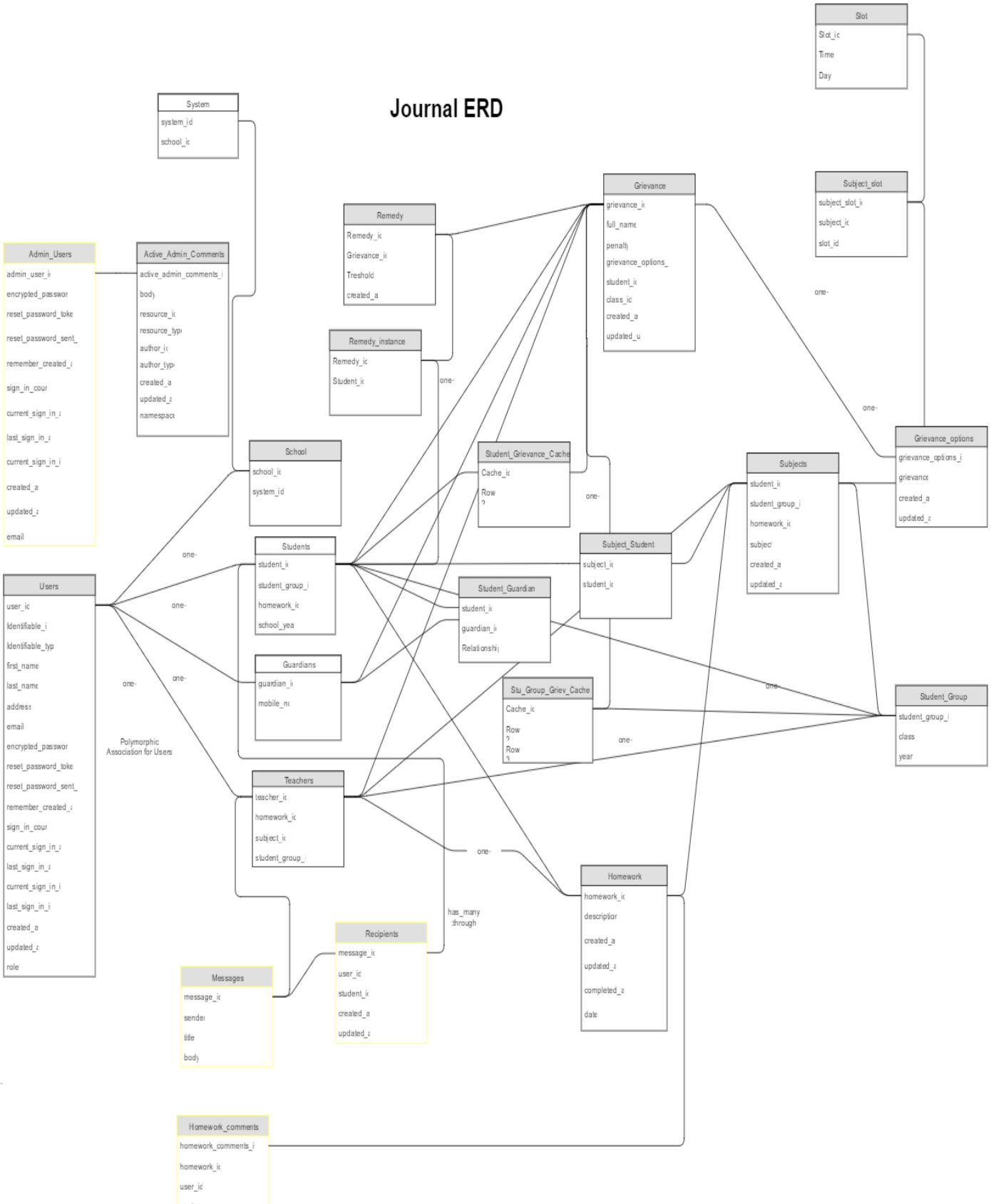
3.2.4 DATABASE OVERVIEW (as of November 2015)



Database May 2016:

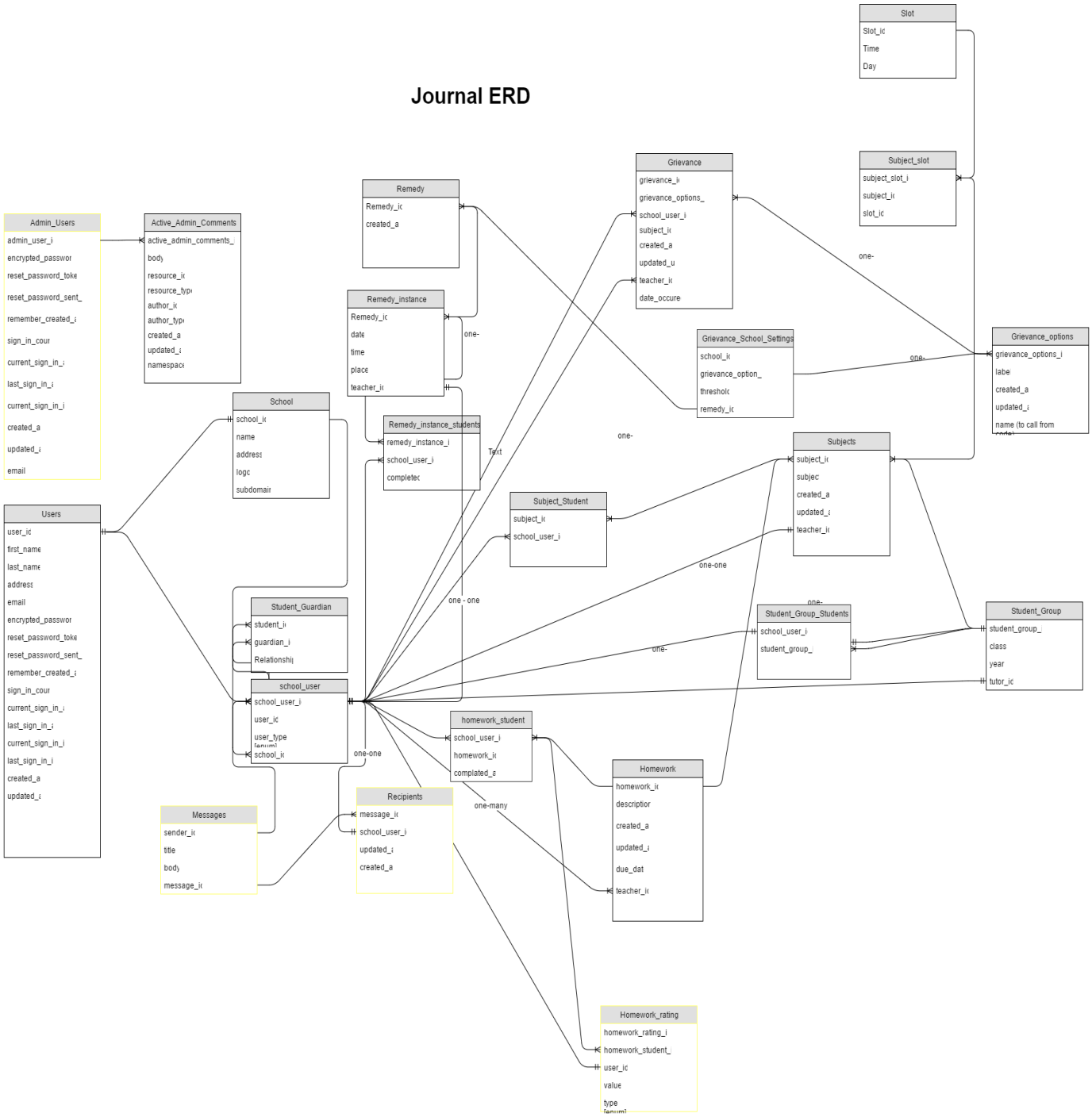
Relationships:
User - Teacher: one to one
User - Student: one to one
User - Guardian: one to one
User - School: one to one
School - System: one to one
Students - Student-Group: many to one
Greivance - Students: many to one
Greivance - Teachers: many to one
Grievance - Guardians: many-one
Teachers - Student Group: many to one
Active_Admin_Comments - Admin_Users: many to one
Homework - Teachers: many to one
Homework - Students: many to one
Subject_slot: many to one
Recipients - Messages: many to one
Guardians - Students: many to many through
Student_Guardian

Journal ERD



Future: (the above Polymorphic User relationship in the long run will be difficult to scale)

Relationships:
User - Teacher: one to one
User - Student: one to one
User - Guardian: one to one
User - School: one to one
School - System: one to one
Students - Student-Group: many to one
Greivance - Students: many to one
Greivance - Teachers: many to one
Grievance - Guardians: many-one
Teachers - Student Group: many to one
Active_Admin_Comments - Admin_Users: many to one
Homework - Teachers: many to one
Homework - Students: many to one
Subject_slot: many to one
Recipients - Messages: many to one
Guardians - Students: many to many through Student_Guardian



3.3 Non Functional Requirements:

3.3.1 Performance/Response time requirement

The user should spend no more than 10 – 15 seconds creating a record. From log in to log out should be no more than 1 minute. This is a key reason for avoiding input text fields in favour of dynamic dropdowns. The interface will be designed to be as intuitive and easy to use as possible. The application will provide instant response times to user actions.

3.3.2 Availability requirement

The application will be available during school hours with any required downtime, maintenance or upgrading carried out outside of these periods.

Geographically the application will be available in and around school campuses as a first priority but can be accessed anywhere with the right credentials.

3.3.3 System Capacity

Initially the system will have to support a minimum of 25 simultaneous users per school (average size) and 50 transactions.

As the system gets bigger an auto-scaling service binded to the application will ensure that the usage requirements are always met without any excessive drawdown of capacity. This works the other way too meaning service is never interrupted because capacity constraints are never exceeded.

3.3.4 Recover requirement

In case of system failure it is imperative that any downtime is kept to a minimum and that recovery protocols are in place for all aspects of the application in the operations manual on the back and front ends.

A development team must be provided with the proper tools and infrastructure to carry out recovery operations particularly in relation to the database.

3.3.5 Robustness requirement

The application has a high level of fault and stress tolerance. Certain failures will not result in the app failing to work. All external and internal interfaces of the system will have to undergo frequent random testing.

The dashboards are fault tolerant to the extent that even if one is broken the page will still load, it just won't show the broken chart or graph. This is an important design characteristic given the prominence of the dashboards within the application as a whole.

Load times will be instant.

3.3.6 Security requirement

All user accounts will be secure through the admin registration process. All users will be required to login. A default password will be provided to each user which they will have to change themselves by editing their profile.

Data integrity tests will be routinely carried out.

The highest standard security services will be binded to the application via the cloud platform as well as local security features already added like data encryption and user records.

3.3.7 Reliability requirement

The application must be running at all times during school hours. An application failure during these periods would be very damaging.

The database will contain data that can be considered sensitive. There will be backup and recovery systems to cover any potential compromise and provide reassurance to users as well as persistence of service.

The role of the cloud provider in conjunction with the administrator will be to maintain services at all times. This means, in practice, if an instance of the application fails another takes it's place immediately or if the cloud region in which the application is hosted is experiencing downtime another region will be utilised to replace it automatically without interruption of service.

3.3.8 Maintainability requirement

The application will be hosted on the cloud and so a lot of the heavier maintenance load in terms of databases and security is handled off site by the provider. The application and database will require ongoing maintenance both in house and off site.

3.3.9 Portability requirement

The system uses portable languages like JavaScript and is designed to be used on different operating systems or host machines as well as being 100% responsive to mobile design. The percentage of code that is host dependent is minimal.

3.3.10 Extendibility requirement

The application must be scalable to handle new features, a growing user base and increased traffic. With the aid of the cloud this is easier to achieve with Auto Scaling which means that as new capacity is required it will come on stream automatically.

3.3.11 Reusability requirement

The code, documentation, design and interface will be reusable up until any upgrading of the architecture is required.

Build: The application can be rebuilt as many times as required with the code and architecture using version control.

Distribution: The application can function as an API that can be plugged in and used on various operating systems or host machines.

Deployment: The application can be deployed through the cloud platform.

Upgrading: The application can be upgraded to take account of levels of demand, usage, geographical spread and the integration of new web based technologies.

3.3.12 Resource utilization requirement

The application requires an administrator for maintenance and the registration of new teachers.

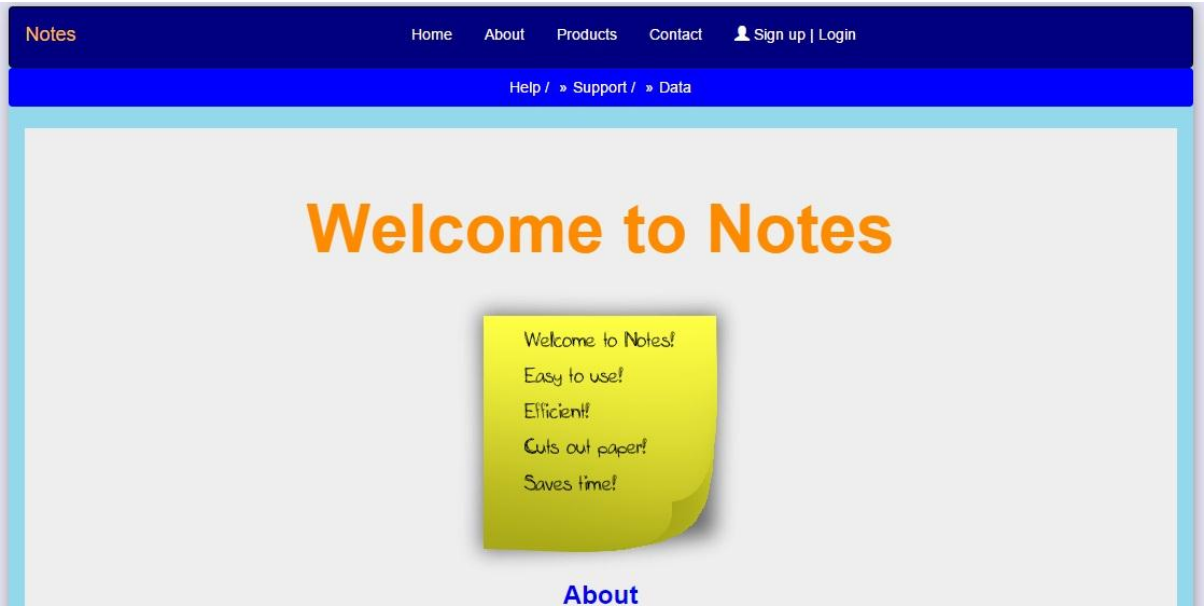
The application and associated database will require continuous maintenance. The personel required will have a competent knowledge of Ruby on Rails, JavaScript and Cloud Deployment and maintenance.

The allocation of physical space to support this system is minimal to non-existent. The premise is that a PAAS cloud solution will be utilised to host the application so most of this requirement is off site and technical support is available off site also.

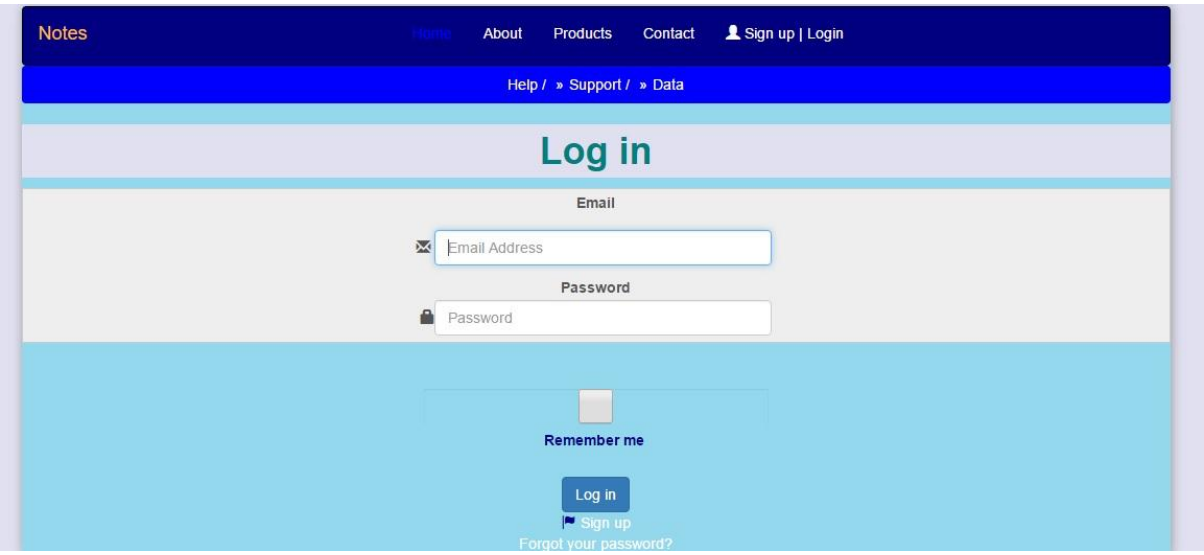
3.4 GRAPHICAL USER INTERFACE

The UI is designed to be as simple and intuitive for navigation and user input as possible. Below are some selected screenshots.

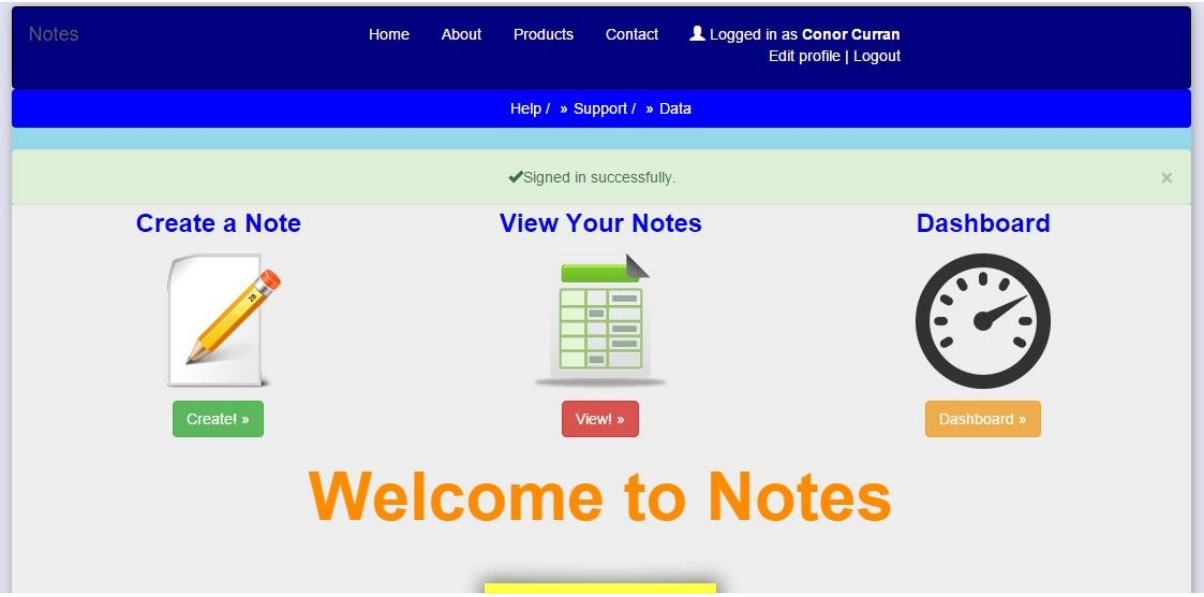
Before Log in:



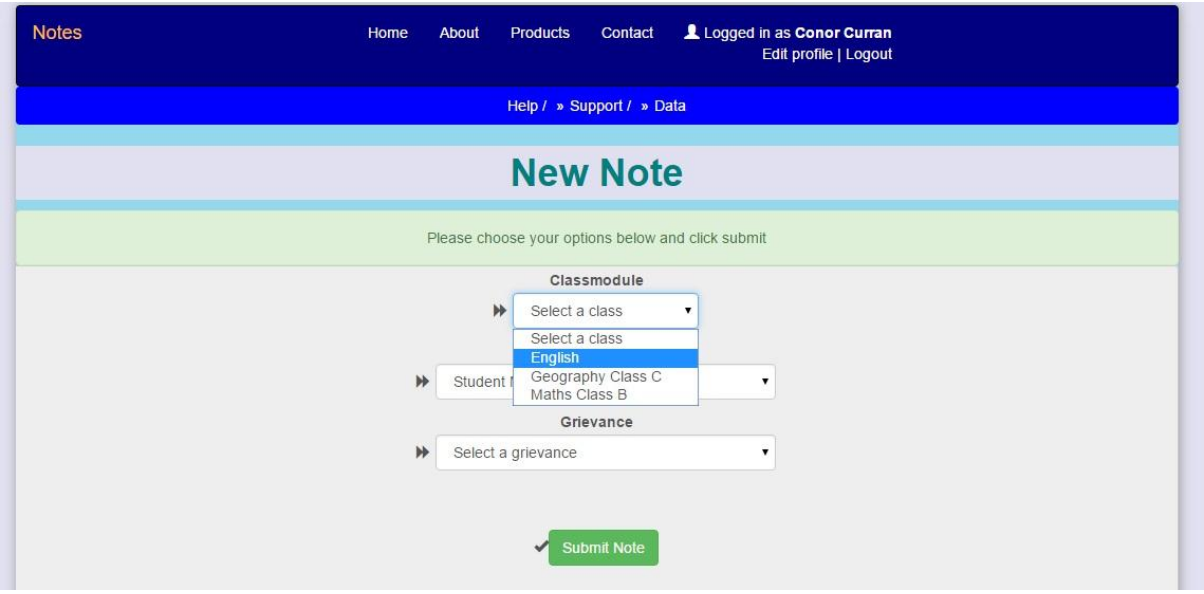
Log in:



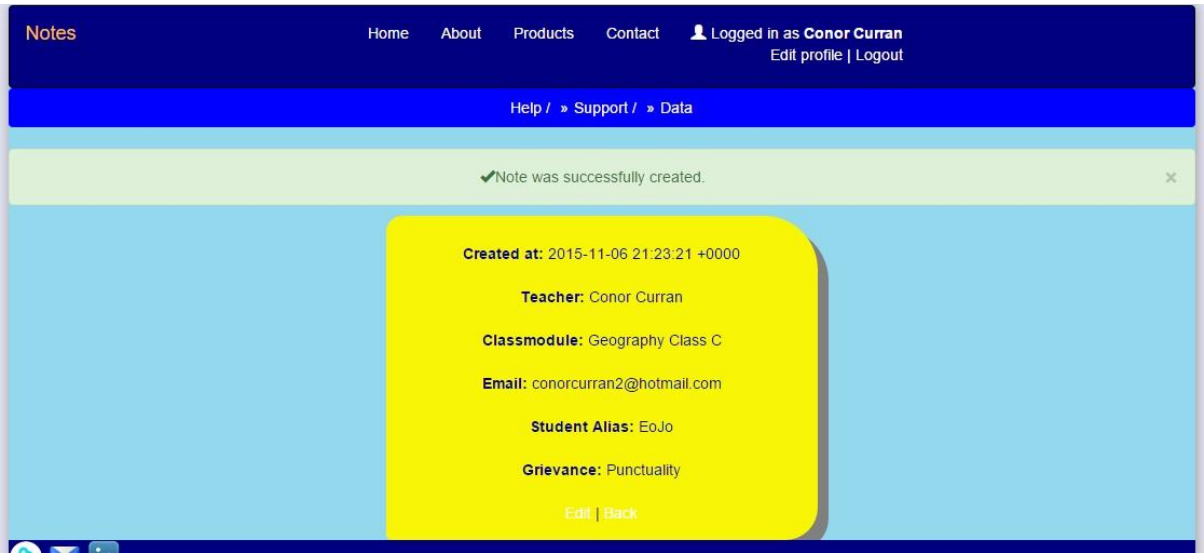
After Log in:



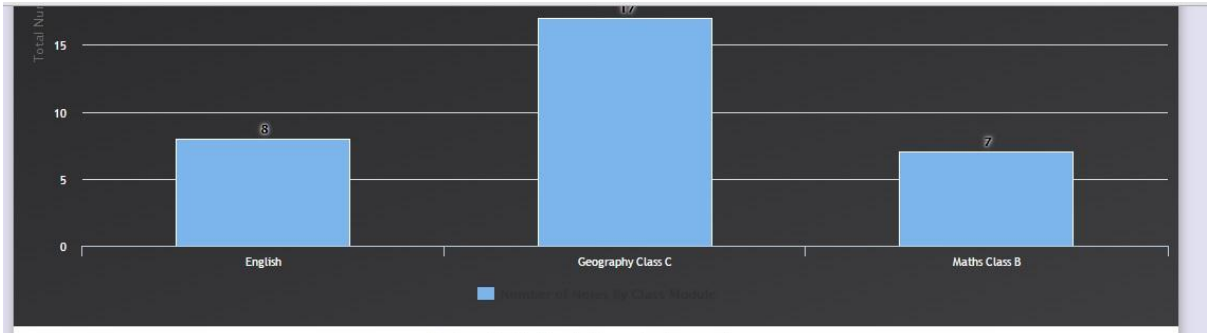
Create Note (just 3 dynamic dropdowns):

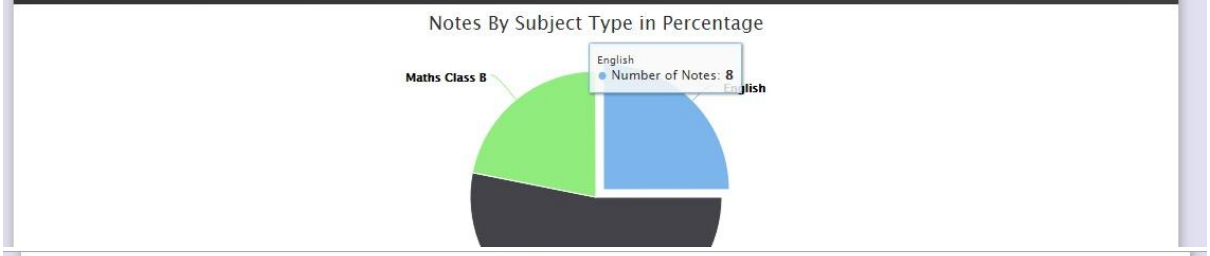


Note submitted:



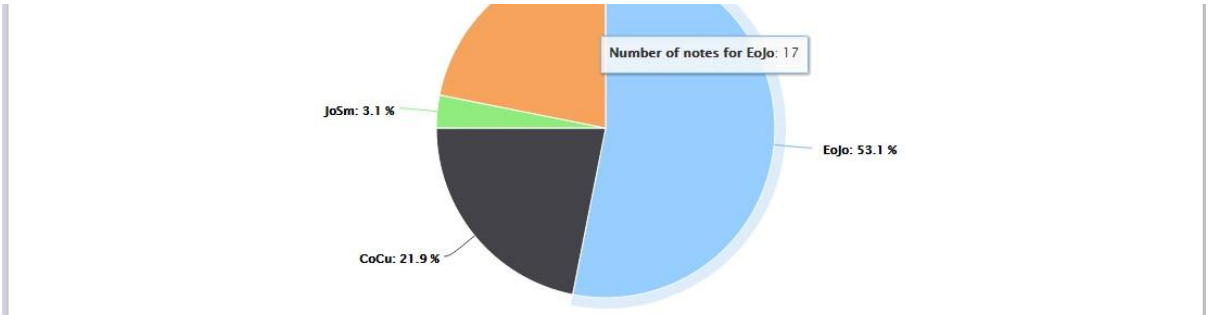
Dashboard examples:

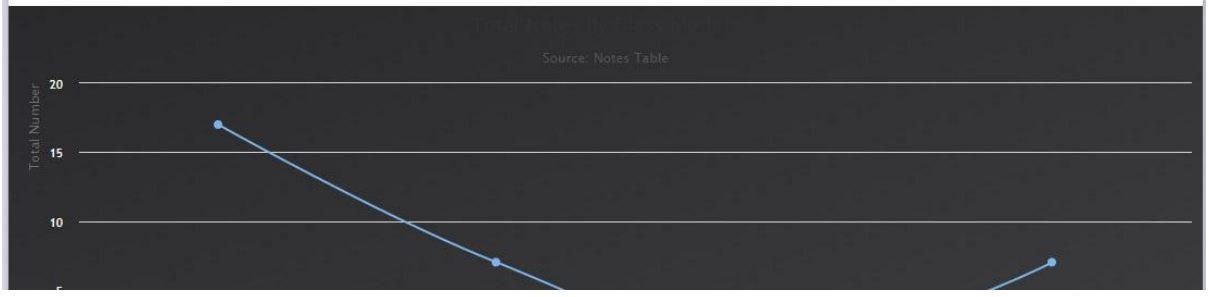

15
10
5
0
8
17
7
English
Geography Class C
Maths Class B


Notes By Subject Type in Percentage
Maths Class B
English
Number of Notes: 8


Total Notes By Grievance
Source: Notes Table
Total Number
20
15
10
5
0
17
17
8
5
1
Geography Class C
Number of Notes By Grievance: 3
English
Geography Class C
Maths Class B
3
Number of Notes By Grievance
Number of Notes by Class Module

Source: Notes Table
20
15
17


Number of notes for EoJo: 17
JoSm: 3.1 %
EoJo: 53.1 %
CoCu: 21.9 %


Source: Notes Table
Total Number
20
15
10

4. REFERENCES

1. **Bluemix (Cloud Deployment)**

   Accessed throughout

   Accessed at www.bluemix.net

2. **Twilio (Communications)**

   Accessed throughout, automated text messaging

   Accessed at: https://www.twilio.com/docs/tutorials

3. **Stackoverflow.com (code queries)**

   Accessed occasionally

   Accessed at: www.stackoverflow.com

4. **Ruby on Rails Guides**

   Accessed throughout

   Accessed at http://guides.rubyonrails.org/

5. **Ruby on Rails Docs**

   Accessed throughout

   Accessed at http://ruby-doc.com/docs/ProgrammingRuby/

6. **JSFiddle**

   Accessed throughout

   Accessed at https://jsfiddle.net/

7. **Rails Casts, Ryan Bates**

   Accessed throughout

   Accessed at www.railscasts.com

8. **railstutorial.org**

Accessed throughout

Accessed at https://www.railstutorial.org/book

9. **Youtube tutorials**

Accessed occasionally

10. **Pluralsight Ruby on Rails resources**

Accessed occasionally

Accessed at https://www.pluralsight.com/

11. **GitHub (Gemfiles, Version control and application repository)**

Accessed throughout

Accessed at www.github.com

12. **Heroku (cloud hosting)**

Accessed occasionally

Accessed at www.heroku.com

13. **Rubygems.org (Gemfiles)**

Gemfile repository links and statistics

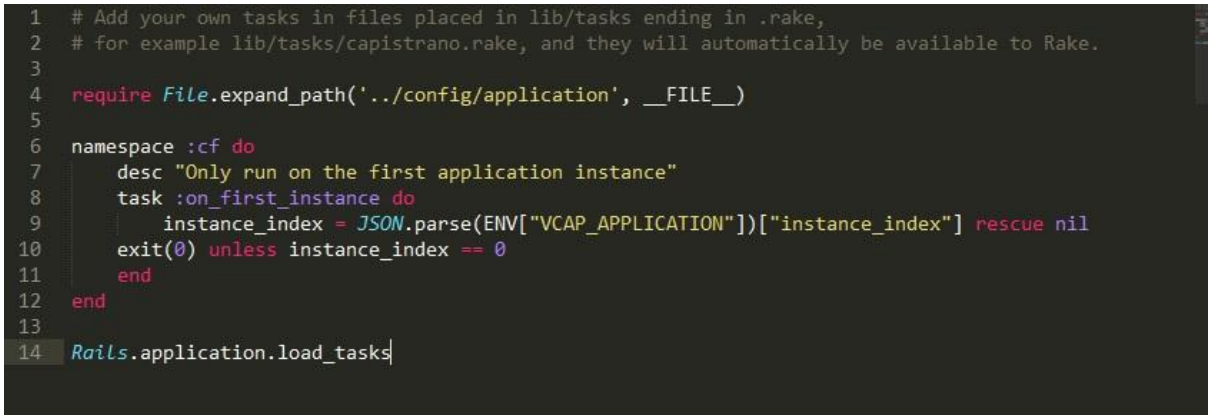
Accessed at www.rubygems.org

5. APPENDIX

5.1 Cloud Deployment:

Steps for deploying to Cloud Foundry (assumes Cloud Foundry command line tool has been installed):

1. Prepare rake file:



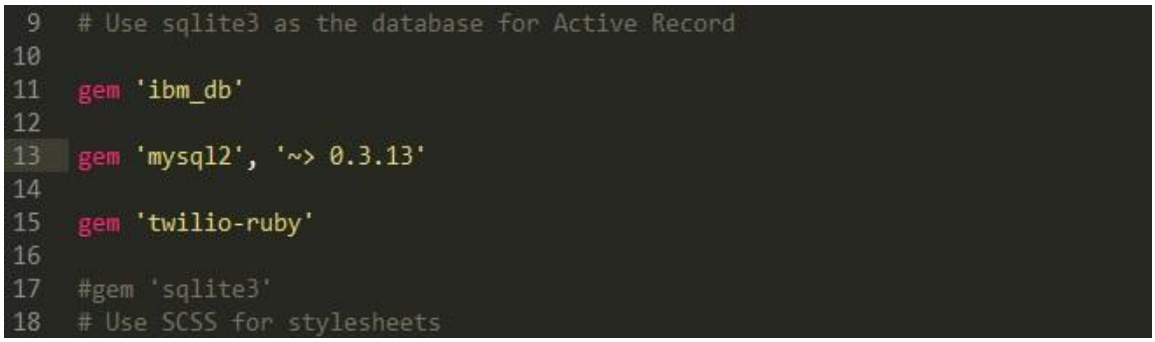
```
# Add your own tasks in files placed in lib/tasks ending in .rake,
# for example lib/tasks/capistrano.rake, and they will automatically be available to Rake.

require File.expand_path('../config/application', __FILE__)

namespace :cf do
    desc "Only run on the first application instance"
    task :on_first_instance do
        instance_index = JSON.parse(ENV["VCAP_APPLICATION"])["instance_index"] rescue nil
    exit(0) unless instance_index == 0
    end
end

Rails.application.load_tasks
```

2. Prepare Gem File (comment out sqlite gem as we won't be using it), Bundle Install:



```
# Use sqlite3 as the database for Active Record

gem 'ibm_db'

gem 'mysql2', '~> 0.3.13'

gem 'twilio-ruby'

#gem 'sqlite3'
# Use SCSS for stylesheets
```

3. Configure Manifest.yml, provide root to buildpack:

```
applications:
- services:
  - notesdb
  - twilio-52
  disk_quota: 1024M
  host: notes2
  name: notes2
  path: .
  domain: eu-gb.mybluemix.net
  instances: 1
  memory: 256M
  buildpack: https://github.com/cloudfoundry/ruby-buildpack.git#v1.3.0
```

4. Ensure following configuration settings in production.rb environment:

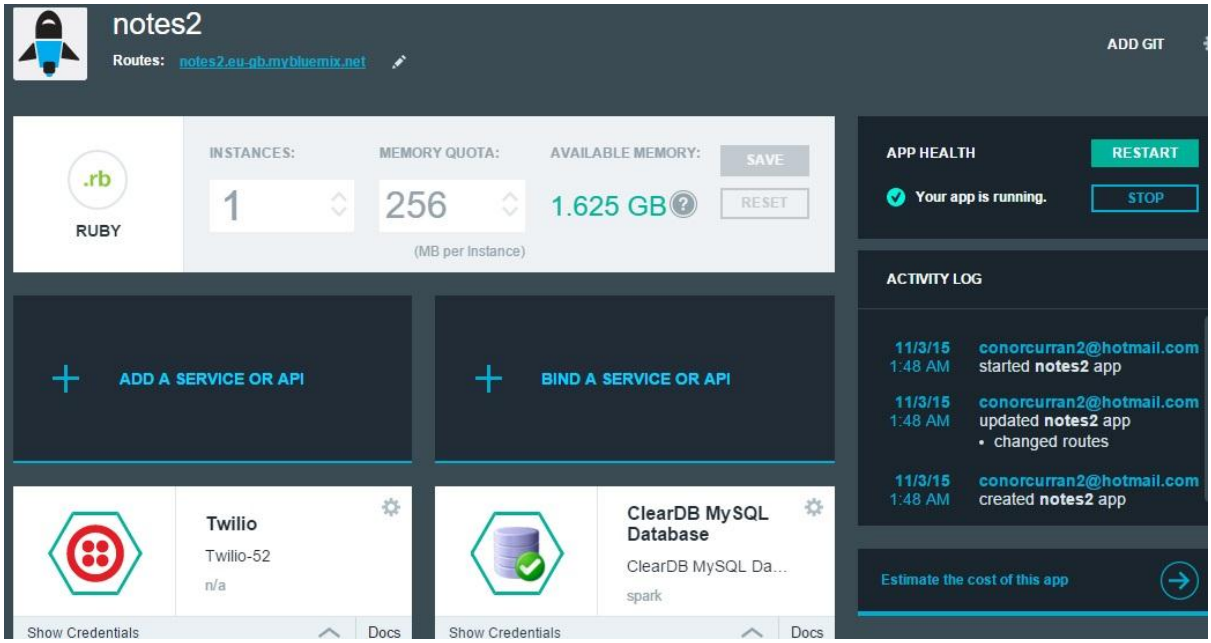
```
  # Apache or NGINX already handles this.
  config.serve_static_assets = true

  # Compress JavaScripts and CSS.
  config.assets.js_compressor = :uglifier
  # config.assets.css_compressor = :sass

  # Do not fallback to assets pipeline if a precompiled asset is missed.
  config.assets.compile = true

  # Asset digests allow you to set far-future HTTP expiration dates on all assets,
  # yet still be able to expire them through the digest params.
  config.assets.digest = true
```

5. Stage application on cloud platform and bind relevant services:



6. Open command line, go to root directory of project and enter:

```
C:\Users\owner\Twilio>cf push notes2 -c "bundle exec rake db:migrate"
```

This will fail so after bundler is complete stop the deployment with ctrl C. Run the following command to finish deployment:

```
C:\Users\owner\Twilio>cf push notes2 -c "null"
```

5.2 Twilio Implementation for messaging

1. Create Twilio account
2. Set up Twilio phone number (with required capabilities like SMS)
3. Copy id and authentication codes for account
4. Define new action in relevant controller (in this case notes_controller.rb)

```
88
89  def send_text_message
90    number_to_send_to = params[:number_to_send_to]
91
92    twilio_sid = "AC0ff247f88b671b6eef93208c2bc027fe"
93    twilio_token = "148cbba33c50a38b2abd566c00d36c74"
94    twilio_phone_number = "+14804188101"
95
96    @twilio_client = Twilio::REST::Client.new twilio_sid, twilio_token
97
98    @twilio_client.account.sms.messages.create(
99      :from => "+14804188101",
100     :to => "+353862413430",
101     :body => "School x message from Notes. Conor has been given detention for Monday 02/11/15 for 'truancy' and accumulating 12 disciplin
102   )
103 end
```

5. In view apply:

```
<%= button_to "Send Message", {:controller => :notes, :action => 'send_text_message'}, :method => :get, class: 'btn btn-danger btn-md',
  data: { confirm: 'You are about to send a Notes text message, are you sure?' } %>
```

5.3 Project Proposal Document

"NOTES" Project Proposal

Student Name: Conor Curran

Number: 08872848

Email: x08872848@student.ncirl.ie

BSc (Hons) in Computing

Specialisation: Cloud Computing

Date: 02/10/15

1 Table of Contents:

2 Objectives

The core objective of this project is to develop a functional web/mobile API for secondary school teachers hosted on a cloud platform utilising various cloud based services.

The application is intended to assist in the digitisation of school record keeping: in this case specifically student disciplinary records.

In practice a teacher should be able to update and maintain disciplinary records on their smart phones. The data stored in the database could then be used to produce graphical information which would be readily accessible to the school or other stakeholders and could aid in planning, identifying trends and, if needed, direct intervention with the student.

In the app it is intended that various tresholds in terms of points accumulated by individual students once reached would trigger an event. Mostly this would be an automatic SMS and/or email to the student's parents. However other tresholds may include and won't be limited to, for example, a detention list whereby the student record is automatically placed in a new and separate table within the database updated weekly which would list students in detention that week. There are various options I am considering on what such output should be.

It is important that the data is presented in a positive light rather than a negative one. This is not about punishing students, it's about getting a better insight and method of recording behavioral issues, and identifying problems that need to be addressed.

The UI shall be as simple and intuitive as possible. The two user groups/roles will be teachers and administration.

Key objectives:

- Provide fully functional and tested cloud based web/mobile API
- If possible have the product tested by end user group
- Utilise relevant cloud based services to achieve desired outcomes
- Ascertain, in as much as possible, any issues that may pertain to data protection

3 Background

During my internship in summer 2015 I gained a good insight in to the lack of standardised digital record keeping in Irish schools. The vast majority of schools don't even have a website and if they do rarely are they maintained to any good standard. Below is a high level look at what I am trying to implement.

It is common practice to this day that record keeping and school disciplinary procedures are still carried out by the teacher, in the first instance, using traditional pen and paper. At a later date this information may be put in to a spreadsheet.

Whilst this method may represent an adequate way of maintaining records it could be streamlined to a much greater extent through the use of modern mobile web and cloud based technology.

By digitising this process it could save teachers and administrators time and the school money in terms of paper and storage requirements. They should be able to do on a smartphone, lap top or iPad what they would normally do on paper.

The purpose of this project is to try and provide a platform to enable them to do that.

The app would also cut out the need to send letters to students homes by sending automatic messages to parents on the basis of any treshold (decided and implemented by the school through the app administration) being exceeded which would indicate the student is having difficulty or is being unreasonably disruptive.

The User Interface will be designed to be as simple, quick and intuitive as possible with minimum input required from the user through the front end view for maximum output from the back end. Ideally the whole process should only take a few seconds.

In terms of practical implementation I am of the view a small number of dynamic select dropdowns would be ideal as an interface. It would be preferable to avoid, if possible, input text fields.

This is a much more slender, streamlined and efficient way of maintaining records but it also gives much greater latitude to the user in terms of time allocation.

I think it is important that the school would decide what number of points would be allocated to what area of disruption. They could do this via the app administration. For example a case of bullying may carry 5 points and the school could decide at what point warrants a text to a parent or what action might be taken so that they are in full control of their own disciplinary procedures.

4 Technical Approach

2.1 Core Requirements:

2.1.1: **Bluemix**:

Utilise IBM Bluemix platform to host project

2.1.2: **Front end coding**:

Ruby on Rails, jQuery, Javascript, CSS and other attributes within framework

2.1.3: **Back end**:

MySQL using ClearDB service on IBM Bluemix (examining using NoSQL)

2.1.4: **Twilio:**

Integrate Twilio communication service through Bluemix thereby enabling SMS or messaging service.

2.1.5: **User Login, Admin**:

Devise and Active Admin to be used within Rails structure to form basis of user authentication and site administration.

2.1.6 **Database**:

It is currently envisaged that MySQL or NoSQL would be used in the project which would consist of 7 tables (open to change):

- Users (typical user table data)
- Teachers (teacher information)
- Students (student information and parent contact number)
- Penalty (Grievance + Penalty association and count)
- Notes (core app records)
- Admin (admin user data)
- Archive (for soft delete)

It would be preferable to keep the number of tables under 8 to avoid having to implement a schema load when pushing the app to the Bluemix platform and thus losing data in the database with each upload.

2.1.7 Archiving: Due to the nature of the data intended to be collected these are not records that should be able to be simply deleted. Records deleted from the main Notes table should be automatically sent to archive which would only be accessed by admin.

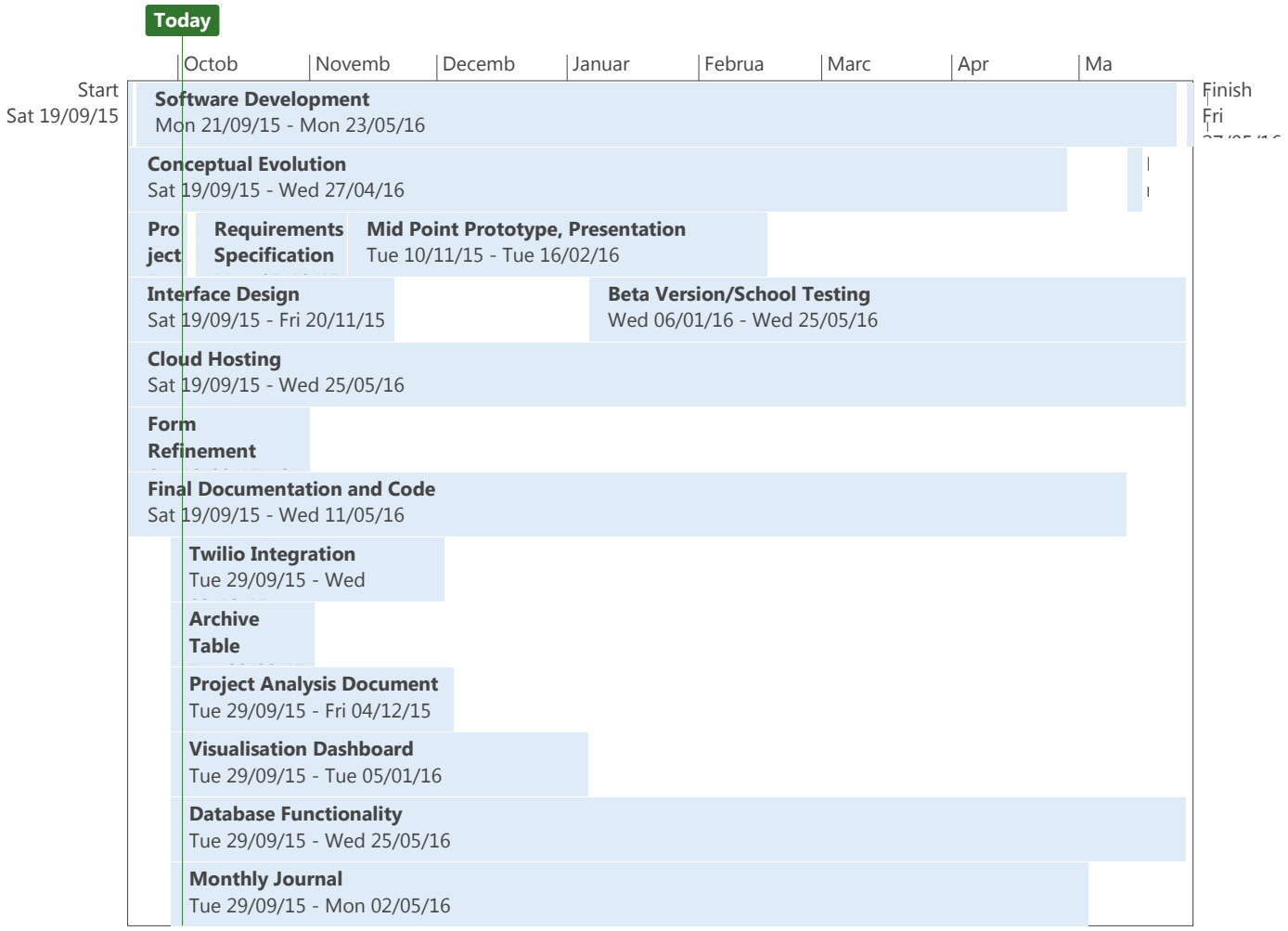
2.1.8 Graphical Data Representation Dashboard: jQuery or embedded Tableau views, exploring "High Charts" as potential use.

5 Special resources required

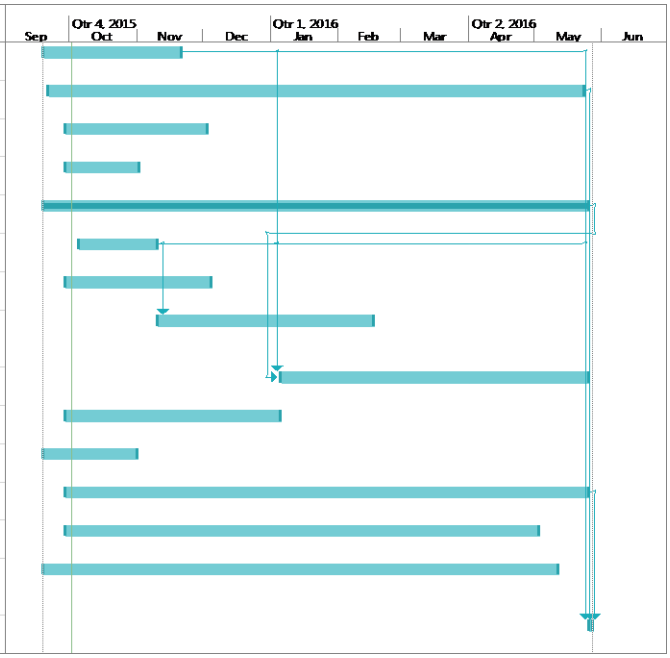
- Extended Bluemix access
- Twilio account
- MySQL Workbench
- Tableau account

6 Project Plan

4.1 Timeline:



ID	Task Mode	Task Name	Duration	Start	Finish	Predecessors
4		Interface Design	46 days	Sat 19/09/1	Fri 20/11/15	
5		Software Development	176 days	Mon 21/09/15	Mon 23/05/16	1
6		Twilio Integration	47 days	Tue 29/09/1	Wed 02/12/	
7		Archive Table	25 days	Tue 29/09/1	Sun 01/11/1	
8		Cloud Hosting	179 days	Sat 19/09/1	Wed 25/05/	
9		Requirements Specification	26 days	Mon 05/10/15	Mon 09/11/15	3
10		Project Analysis Document	49 days	Tue 29/09/15	Fri 04/12/15	
11		Mid Point Prototype, Presentation	71 days	Tue 10/11/15	Tue 16/02/16	1,3,9
12		Beta Version/School	101 days	Wed 06/01/16	Wed 25/05/16	8,1,4,9
13		Visualisation Dashboard	71 days	Tue 29/09/15	Tue 05/01/16	
14		Form Refinement	32 days	Sat 19/09/1	Sat 31/10/1	
15		Database Functionality	172 days	Tue 29/09/15	Wed 25/05/16	
16		Monthly Journal	155 days	Tue 29/09/1	Mon 02/05/	
17		Final Documentation and Code	169 days	Sat 19/09/15	Wed 11/05/16	
18		Presentation	2 days	Thu 26/05/1	Fri 27/05/16	1,15,4,9,5,3



7 Technical Details

- Implementation language : Ruby on Rails – (HTML, CSS, jQuery, Javascript, Ruby etc)
- Editor: Sublime Text 3, Jet Brains
- Cloud platform: **Bluemix**
- Cloud services: Twilio, Clear DB and/or IBM DB, potential to utilise more services like Watson. Bind security services.
- Libraries : jQuery, javascript, Minitest

8 Evaluation

Evaluation and testing will be ongoing in development and production environments. The dedicated standard rails test environment will be used to run regular tests on every model and controller. The scaffold provides for a very comprehensive test environment. Integration tests will be run to ensure controller and model interactions are sound and robust.

The database will also be routinely tested by running the migrations through the test database schema against the development schema.

Unit testing will be conducted by providing one test for every validation and method in the models.

Functional testing will take the traditional request approach. Rails provides for 6 or these:

- get

- post
- patch
- put
- head
- delete

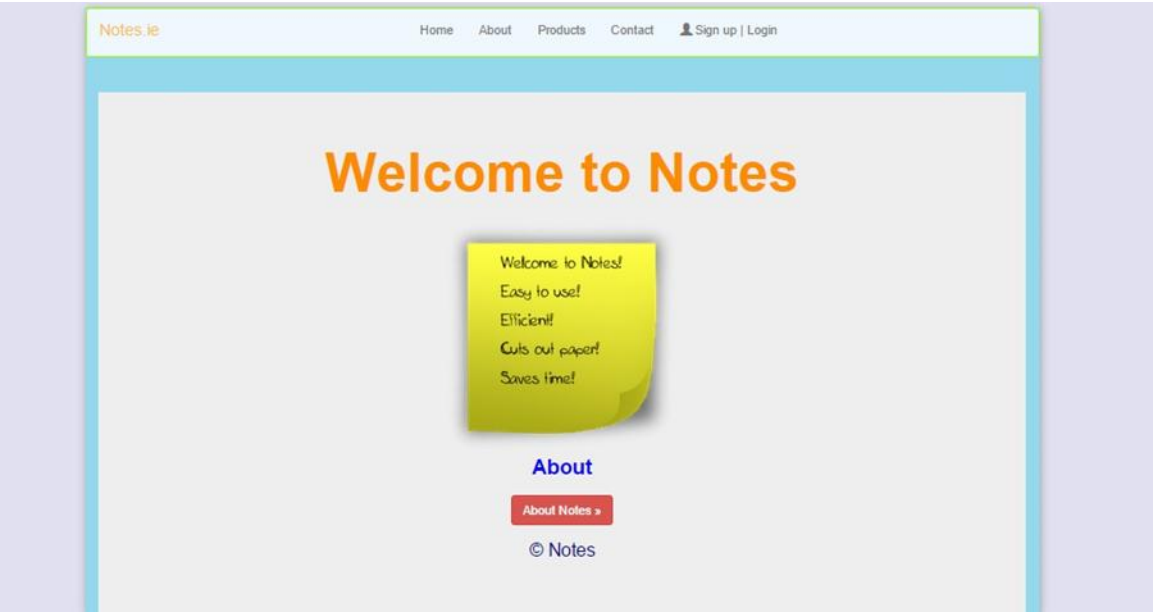
Assertions will be used with the default test library.

If possible the ideal outcome for user testing and evaluation would be to have this app in use in a school from January.

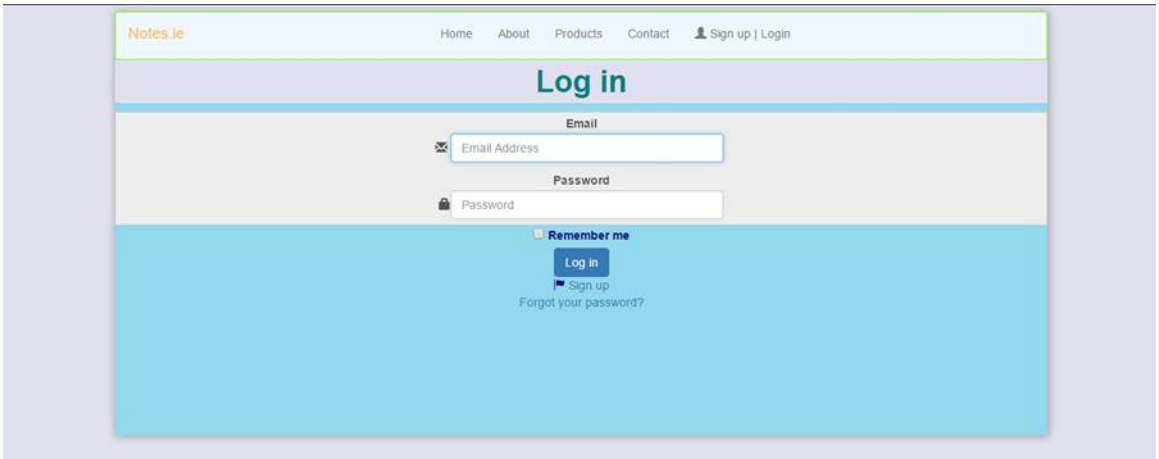
Below are some indicative screenshots of how the app is currently envisaged. This will probably change as the project progresses.

UI Indicative Screenshots:

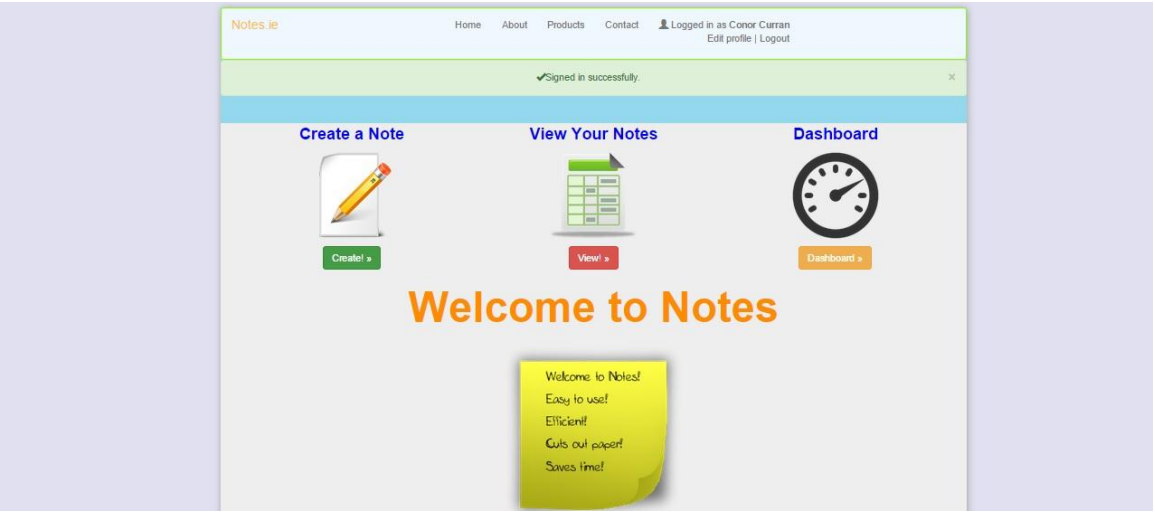
User Interface (before log-in)



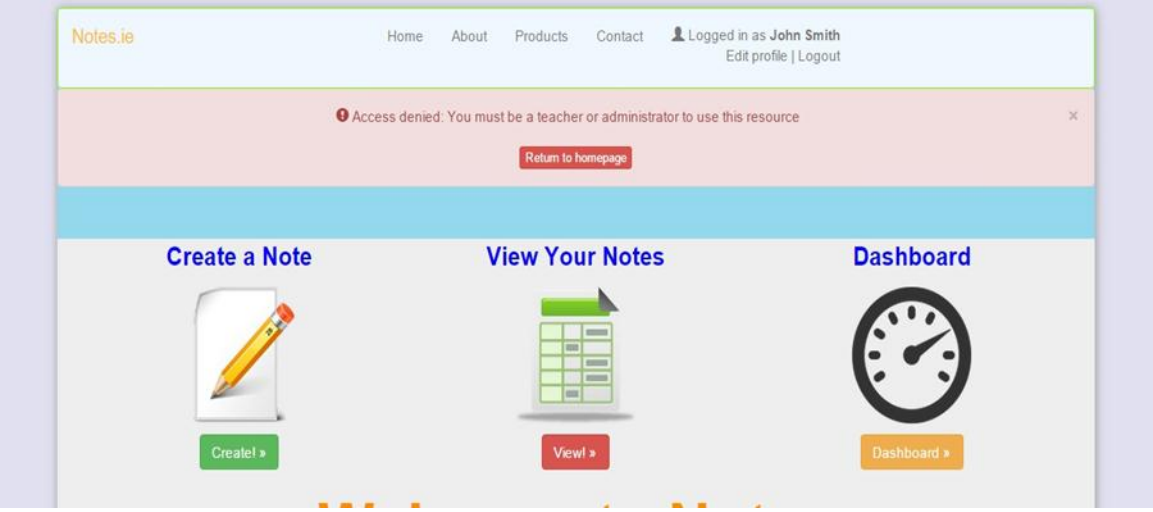
Log-in Screen:



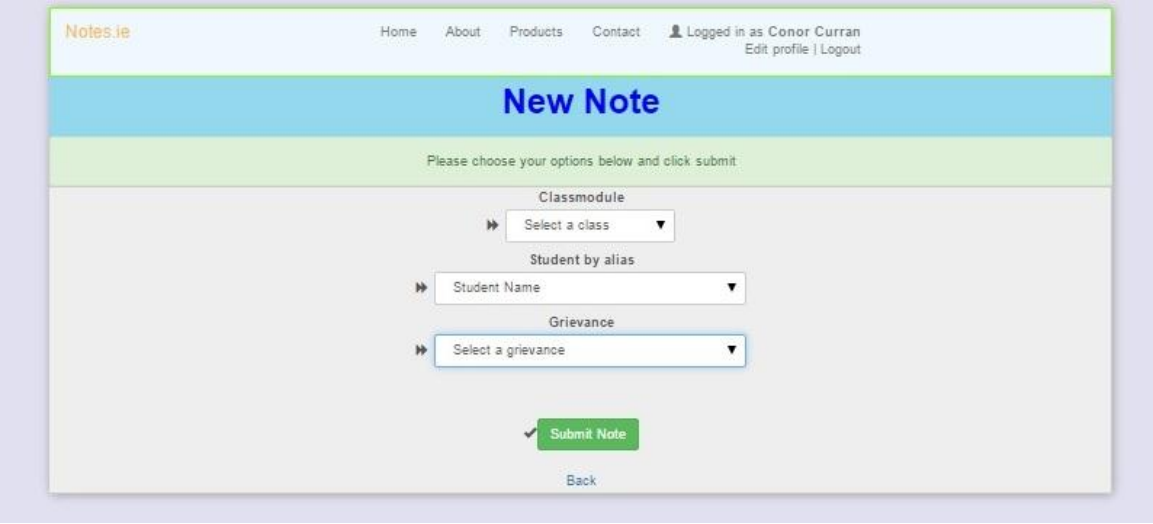
After Log In, additional options and privilages appear:



User must be a teacher or administrator to access resources – “About” is open to all.

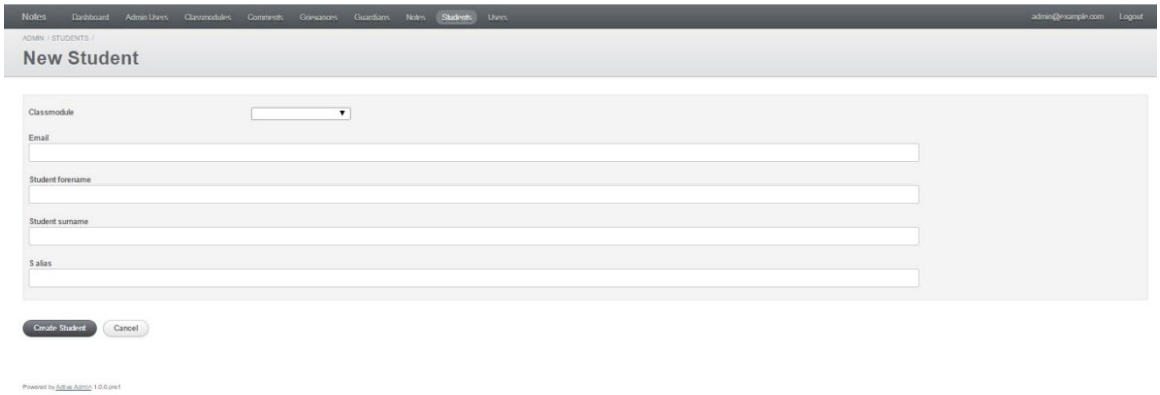


"Create Note" screen, currently 3 dynamic dropdowns using collection_select

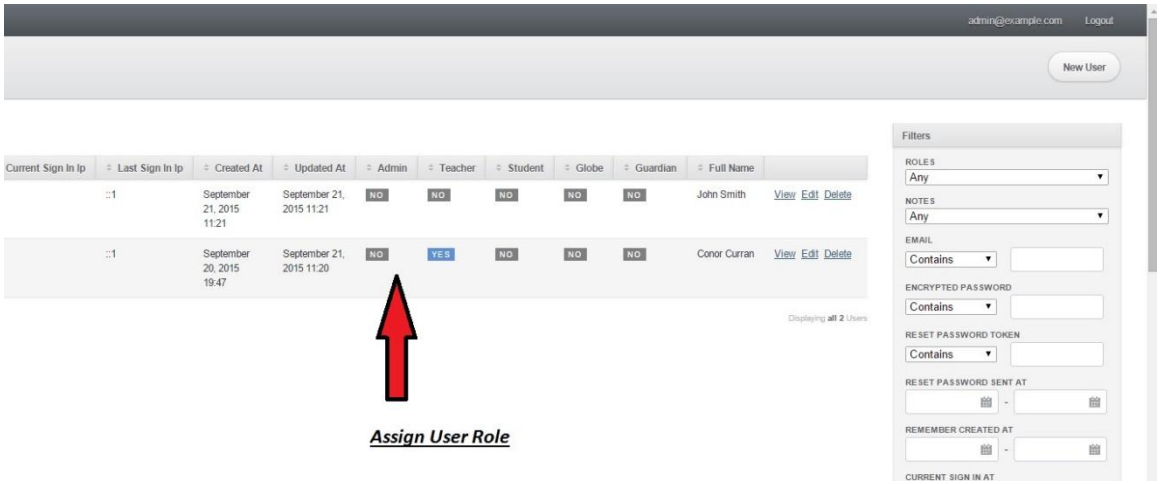


"Note submitted", details of record

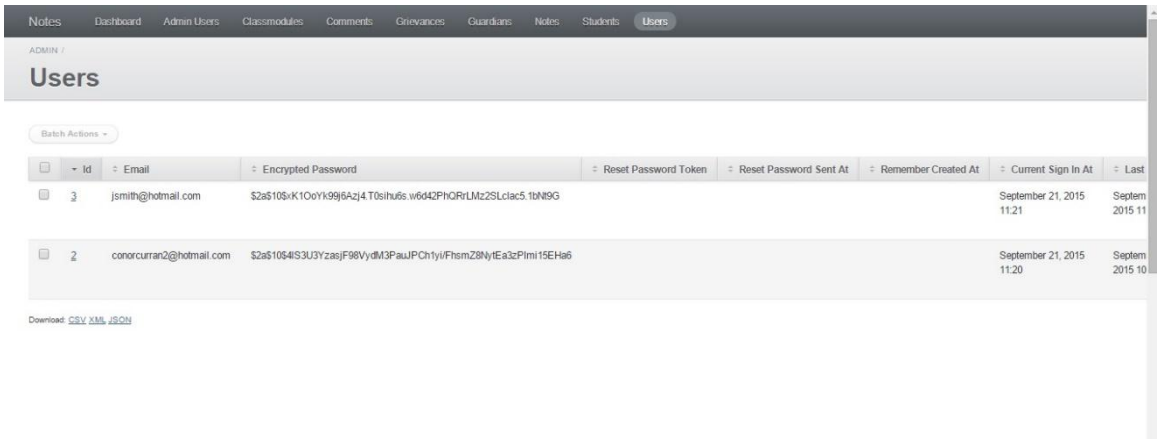
Administration, creating student record:



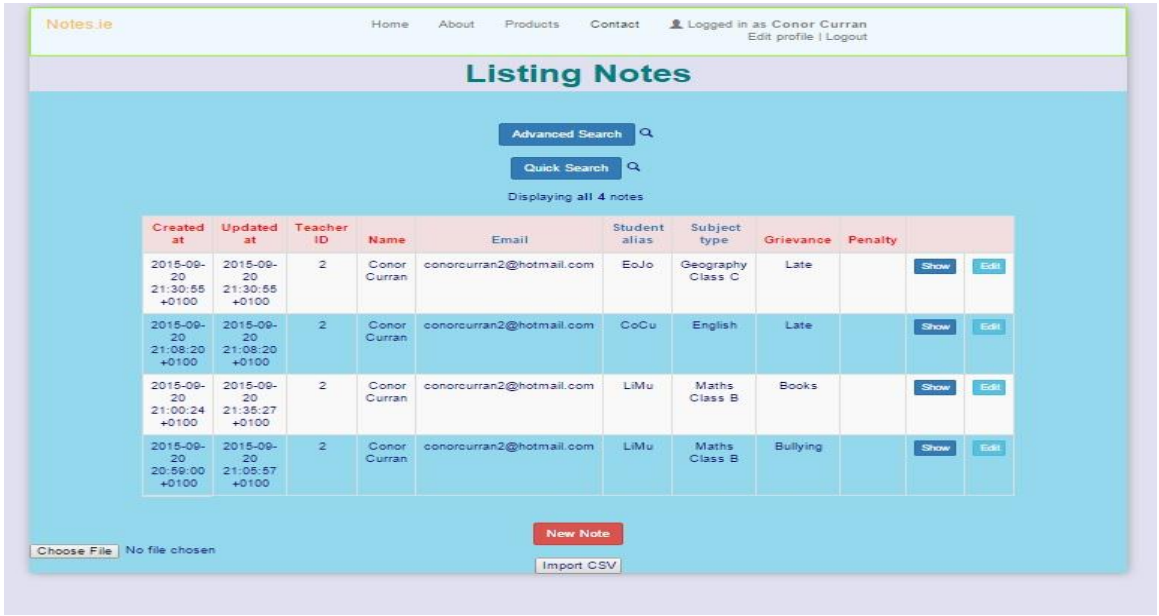
Administration, users:



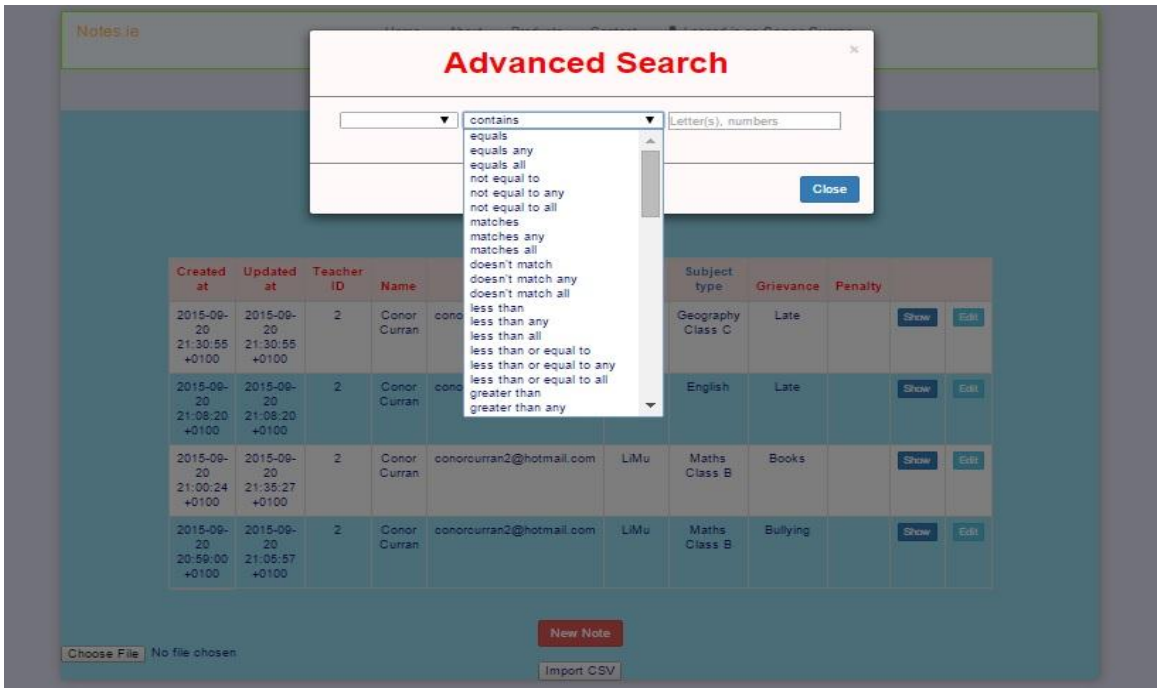
Assign User Role



Notes Index, restricted access, note owner can edit, admin can send record to archive:



Indicative Index Search:



All images indicative.

Conor Curran 02/10/15

6. MONTHLY JOURNALS

September 2015

This month I spent most of my time planning the next steps in how to successfully deliver the project. I have been investigating Twilio and looking at how to incorporate it in a Ruby on Rails application with the emphasis on how I want to use it (i.e on submit, perform action). I have been looking at the best method to deliver the visual dashboard as indicated in the proposal document.

I have encountered a big error when pushing my app to the cloud in the last week, one that I am still trying to fix. This is from command prompt when trying to push the app to cloud foundry:

"…downloading and untarring DB2 CLI driver....

setting DB2 ODBC driver ENV variables

Running: bundle install --without development:test --path vendor/bundle --binstubs vendor/bundle/bin -j4

Could not load OpenSSL.

You must recompile Ruby with OpenSSL support or change the sources in your

Gemfile from 'https' to 'http'. Instructions for compiling with OpenSSL using

RVM are available at http://rvm.io/packages/openssl.

Bundler Output:

Could not load OpenSSL.

You must recompile Ruby with OpenSSL support or change the sources in your

Gemfile from 'https' to 'http'. Instructions for compiling with OpenSSL using

RVM are available at http://rvm.io/packages/openssl.

!

! **Failed to install gems via Bundler.**

!

Staging failed: Buildpack compilation step failed

FAILED

BuildpackCompileFailed…"

I have no idea what is wrong. It was working fine until a week ago and suddenly it won't work. This is extremely frustrating. I have tried to do this on multiple machines and the same error occurs. I have changed the sources of my gemfiles as the error suggestion states, I have deleted completely my Rails and Ruby versions and reinstalled. I have been trying to use RVM to recompile Ruby with Openssl to get a solution but it's not clear to me how to do this on Windows.

Clearly if I can't host my applications this is a serious problem but I'm hopeful of getting a solution ASAP and maybe my supervisor might be able to help.

So that is really annoying and it has sucked up a lot of time in the last week.

Other than that I have been focused on documentation as well, so the requirements spec specifically I have been working on.

It's going well overall so far.

October 2015

This month I concentrated on getting Data visualisation dashboards and text messaging working on my application. I used Twilio for the text messaging and javascript for the graphs on the dashboards. I just need to find a way to update the charts automatically from the table as currently I am having to hard code the stats in. This is not sustainable in the long run.

I succeeded with both to an extent in that something works but I still have a lot to do to ensure it works as it is intended to work in the final application. I also progressed the requirement spec and updated the application interface.

I have developed a good understanding of Heroku platform in recent weeks and have staged a few versions of the app there successfully utilising some services.

I have been researching more on the platform(s) I'd like to use to host the application. I am impressed with Heroku. Very intuitive, easy to use. I'm still going to use Bluemix but for added learning will choose one more too.

My supervisor recommended building a CV uploader gemfile which I thought was a bit daunting but I will try because I think a CV uploader is important for this particular application as a lot of school records will be recorded in Excel documents.

November 2015

This month I made reasonably good progress with my application. I attended various meetups around town like ruby groups and IBM Cloud groups to get a better understanding and some new ideas on the framework I am using.

I also did some market research on the product so I have spoken with teachers and parents. I also shared through social media a couple of Surveymonkey surveys and actually got a good response. Most respondents, both teachers and parents, were broadly enthusiastic about the idea. Some specifically mentioned they would like to see more functionality in the application.

From a research perspective it has been a good month.

On the coding side I did stutter a little bit – specifically with the graphical information on the dashboards – but I consulted with my supervisor Johnathan and made good progress here. This has allowed me to progress other elements of the application.

Overall happy with progress but a lot of work to do.

December 2015

This month I started adding a tool that allows students to check their homework and teachers to submit homework for them. The student knows when the homework is due.

I have also added a calendar that shows when homework is due or completed.

In addition I have added a table for the teacher to be able to see which students have completed their homework.

I did encounter a deployment issue this month when support for my Ruby version was removed suddenly from the build pack I was using. I targeted a different build pack but had to revert to Ruby 2.0. In the long run this is not ideal so hopefully a new build pack will be available shortly for later Ruby versions.

January 2015

This month I concentrated more on the homework tool element of the application making some refinements. I also cleaned up the Administration user interface as this is one of the most important parts of the application.

The idea is that the schools should have full control over all parameters within the application relating to discipline. To this extent it must be intuitive and easy to use.

I continued carrying out market research with some interesting findings. Most responses amongst the targeted stakeholder appear to validate some assumptions I had made and overall the general response has been positive regarding this application/service.

February Progress Report

The application is on track as per the original timeline in the project schedule however end user testing can now not begin until June at the earliest and even then only in a primary school as secondary schools will be closing for the summer. The intention would be to approach a secondary school in June with the intention of starting user testing over a 3 month period from September 2016.

I am looking at doing a transition year design workshop for students shortly to get feedback on design from both teachers and students.

I have carried out market research through meetings with the various stakeholders (students, teachers and parents). I have carried out surveys through Facebook where the response was positive to the idea. I also took part in an education hackathon over a weekend in November to carry out further research.

March 2016

This month I focused on making the rendering of the data on the dashboards dynamic which was not an easy task. I had to do research on the Highcharts API to find out exactly how to dynamically render the data within the rails framework. This involved using a hash as below after defining other calls:

```
def self.getData
   data = []
   self.subject_types.each do |type|
     data << Hash["name", type, "y", self.type_count(type)]
   end
   data
 end
```

I was awarded “Inventor of the month” by NCI for the project which was a nice recognition.

April 2016

This month I implemented additional functionality in the application including making refinements to the homework tool and began the process of future proofing for commercialisation to help ensure the application scales more easily and that I would not have to come back and do a lot of repair work on what I have already done. In this regard I scaled the database beyond what is currently used in the application and have started coding new functionality.

May 2016

With exams this month the application had to take a back seat for most of the time and my focus has been preparing for the project show case. I still managed to do some refinement

and additional work on the user interface. I stopped making big changes to the application just in case these might be too ambitious and could cause problems at the show case if not completed properly.

7. TESTING

During development of the application testing was undertaken in both the development and production environments and I also used the dedicated test database Rails provides in the config directory.

The following tests were carried out using the Rake_test command:

Model Unit Testing (for example:)

“rake test test/models/note _test.rb test_the_truth”

```
test "should not save note without student_name" do
  note = Note.new
  note_not note.save, "Saved the note without a student_name"
end
```

And more were carried out using the available assertions within the Rails framework.

Functional Controller Testing (for example:)

```
class HomeControllerTest < ActionController::TestCase
  test "should get index" do
    get :index
    assert_response :success
    assert_not_nil assigns(:homes)
  end
end
```

I used most of the available test types throughout development on the controllers and also the @request and @response instance variable tests. This was very useful in identify issues that may not have been a problem at the time but could have become serious problems later on.

View Based Assertion Testing

I tested the integrity of the views occasionally with commands like:

```
“assert_select 'header', "Welcome to Notes"
assert_select "ol" do |elements|
  elements.each do |element|
```

```
    assert_select element, "li", 4
  end
end

assert_select "ol" do
  assert_select "li", 8
end”
```

Interaction Controller Testing – Integration Testing (for example:)

```
“class UserFlowsTest < ActionDispatch::IntegrationTest
  test "login and browse site" do
    https!
    get "/login"
    assert_response :success
end”
```

Routes Testing (for example:)

```
“class NoteRoutesTest < ActionController::TestCase
  test "should route to note" do
    assert_routing '/notes/1', { controller: "notes", action: "show", id: "1" }
  end

  test "should route to create note" do
    assert_routing({ method: 'submit', path: '/notes' }, { controller: "notes", action: "create" })
  end
end”
```

Aside from the above I used R-Spec to simplify the testing process.