Skylab: NUS Orbital Project Platform

by

Gu Junchao

In partial fulfilment of the
requirements for the Degree of
Bachelor of Engineering (Computer Engineering)

NATIONAL UNIVERSITY OF SINGAPORE

B.Eng. Dissertation

Skylab: NUS Orbital Project Platform

by

Gu Junchao
NATIONAL UNIVERSITY OF SINGAPORE

2015/2016

Project No.: H0791220

Advisor: Assoc Prof Min-Yen Kan

Deliverable:

Report: 1 Volume

Declaration of Authorship

I, Gu Junchao, declare that this thesis titled, 'Skylab: NUS Orbital Project Platform' and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a bachelor's degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed: _____

Date: _____

"If debugging is the process of removing software bugs, then programming must be the process of putting them in."

Edsger Dijkstra

NATIONAL UNIVERSITY OF SINGAPORE

Abstract

Faculty of Engineering
Computer Engineering

Bachelor of Engineering (Computer Engineering)

by Gu Junchao

Orbital is the School of Computing's self-driven programming summer experience. It is designed to give first-year students the opportunity to self-learn and build something useful[1]. As more and more students join Orbital program, administration of the program and evaluations among teams are becoming increasingly complex and challenging. Therefore, Skylab is designed and implemented to help students to submit project logs and evaluate other teams with ease as well as to help the administrator of Orbital program overlook and manage this module well. In this project, we find and solve real-life problems faced by students, advisors, mentors, facilitators and administrators of Orbital program with an extensible system design and continuous contribution in an agile software development process.

Subject descriptors:

D.2 Software Engineering
D.2.11 Software Architectures
D.2.5 Testing and Debugging
H.5.2 User Interfaces
K.6.5 Security and Protection

Keywords:

Software Engineering, User Interface, Security, Usability, Ruby on Rails

Acknowledgements

I would like to express my most sincere appreciation to my project supervisor and project administrator, Assoc Prof Min-Yen Kan. Throughout this project, it was him who tirelessly provided me with significant support and assistance. I would also like to appreciate facilitators, advisers and students in Orbital program for suggesting many useful features and bringing up issues to make Skylab more usable.

Contents

Chapter 1

Introduction

Orbital is the School of Computing's self-driven programming summer experience. It is designed to give first-year students the opportunity to self-learn and build something useful. It is designed as a 4 modular credit (MC) module that is taken pass/fail (CS/CU) over the summer[1]. With its focus on hands-on experience, an increasing number of students are joining the program to gain practical coding and software engineering experience. During the academic year of 2015/2016, more than 250 students completed the Orbital program with 19 advisers evaluating these teams.

The *team* is the basic unit in Orbital. Teams have exactly two students. Teams start a project, submit project log and evaluate other teams. To express their interests in Orbital program, students have to register in Skylab (the official Orbital website) first. They can sign up as a team of two students if they have a partner in mind. If at the time of registration they do not have any one in mind whom they can partner with during the summer, they can sign up as individual first and a match making session is going to be conducted before Orbital program officially starts to help them find a good partner.

After registration, for those who are accepted into Orbital program, they will begin on their project ideas. At particular timings (checkpoints) throughout the summer (currently three), they report their progress by submitting their project's

README and project log. Assigned peer teams (each team will be assigned a number of peer teams, currently 3) then give feedback regarding their submission and the project built by the team. At the same time, there will be an adviser who oversees the whole process and also provides a similar evaluation of the team's submission as well. After the submissions and evaluations are done, the team provides feedback to its peer teams and adviser regarding the quality of evaluations received. Figure 1.1 illustrates the whole workflow.

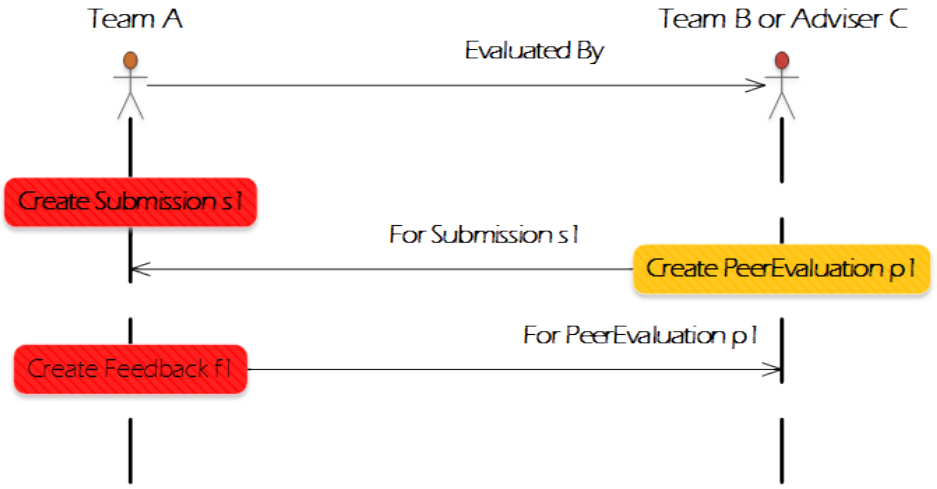

FIGURE 1.1: Illustration of evaluation workflow in Skylab

The scale and routine nature of the project management in Orbital necessitates the development of a project management framework to automate many functions. This defines the scope of Skylab – a development project built for administrators, facilitators, advisers, mentors and students and potential students in Orbital program. It also provides students with a real-life Software Engineering training ground to learn and sharpen programming and system design skills.

1.1 Challenges

1.1.1 Tight deadlines

As the implementation of Skylab was started only in March 2015 and students started using the system at May 2015, there was not much time for the core development. In fact, many features could only be delivered and put into production right before students in the 2015 cohort used them. Chasing deadlines and fixing urgent issues during implementation, especially in the summer when students were actively using Skylab, not only brought a lot of pressure to me but also led to difficulties in system design. I made some design decisions to compromise on certain deadlines, allowing me to postpone potential problems for later implementation.

1.1.2 System design

Skylab is built using Ruby on Rails, a mature convention-over-configuration web framework. So the first challenge is to get familiar with the conventions and recommended ways of doing things in the Rails community. Then Skylab can be designed into a web application on the top of the mainstream philosophy outlined by the Rails community. Another issue with the design of Skylab is that this is the first time for me to design such an application from ground up, without any guidance from any experienced Ruby on Rails developers. Therefore, it is all about trial-and-error and explore my own way. Reading books and browsing online tutorials helped a lot and luckily there are plenty of resources about Ruby on Rails development, due to its popularity.

1.1.3 Evolving requirements

Although the scope of the project is very clear and well-defined, changes in requirements are expected and did happen frequently due to the evolving features within the Skylab project and Skylab users' feedback. The challenge is to cope

with all changes and sometimes adjustments to the design of system have to be made to accommodate for extensions and new requirements. Therefore agility in development and adaptability of the system is essential.

1.1.4 Data migration

Sometimes a schema migration is required as a result of change in requirements. Therefore, dealing with old data and migration of data without affecting the use of application is another challenge during the development of Skylab. Extreme attention when migrating is required as data may not be clean enough and careless migration may even cause the system to be unusable for some users.

1.1.5 Security

Security is definitely a very important perspective in web development. Although the Rails framework already handles security well by taking measures against SQL injection, XSS attacks and CSRF attacks, there are still certain vulnerabilities when not implemented well. During development of Skylab, various techniques and practices are adopted to make Skylab more secure and trustable. For example, we implemented user roles within Skylab, allowing a role based access control system to permit users to permit fine-grained control to allowed actions.

1.1.6 Code quality/maintainability

Although currently there is only one developer constantly contributing to Skylab repository, a good development cycle which is agile enough is not only important to keep track of history and manage different issues but also convenient for developers who may join later to jump in and get started in a short time. Testing is also a very important factor when it comes to long-term maintenance. A continuous integration would also help in catching regression errors in development early and

easily. Besides all mentioned above, refactoring is helpful to the maintainability and growth of Skylab.

1.2 Importance of the work

As Orbital's evaluation of students' performance is largely based on evaluations from peers, it is very important to make the evaluation process among students to be smooth and user-friendly. Before Skylab is implemented, students were supposed to copy an HTML template and modify accordingly for their project's README and log – then post the submission in a forum visible to other teams. For peer evaluation, they need to repeat the whole process – only that this time the form is a lot more complicated and harder to copy and paste. Advisers on the other hand, have to manually remove critical sections from peer evaluations by all of his/her teams before make it visible to evaluated teams. And the worst part is that admin of Orbital program needs to do these manual work as well – only this time the admin has to do for all the teams (more than 100) in Orbital. Even with clear instructions, many submissions and peer evaluations were still submitted in incorrect formatting. Advisers and administrative users needed to manually fix those. Lack of standardization also leads to trouble when trying to compile a consolidated result from submissions and peer evaluations. A lot of time is wasted in the whole process and careless mistakes in copying, modifying and pasting are just inevitable.

So Skylab is designed and implemented to make life of students, advisers and administrators of Orbital program a lot easier with standardization and automation of many procedures in Orbital program. So students can just save a lot of time and effort now by simply filling and submitting required forms, instead of copying an HTML form, modifying, pasting and finally posting as a post in a forum; advisers do not have to manually take out critical sections from peer evaluation and republish posts as the whole process is taken care by the system; admins can easily

overlook and get statistics of evaluation results through administrative portal as well.

1.3 Objectives

In this project, I needed to:

- Enable users to login via NUS OpenID if they have NUS Net IDs already.
- Enable users to login via combination of email and password for those who do not have NUS Net IDs and also serve as a backup solution to NUS OpenID login.
- Enable "Forget password" feature for those who forget their password to reset their passwords.
- Enable users to register for Orbital program as a team.
- Enable users to register for Orbital program as an individual and recommend potential good partners based on their interested topics.
- Enable students to edit their own team's details including "Team Name" and "Project Level".
- Enable students to submit for Milestones or edit their previous submissions. Besides, students in the same team should be able to see changes made by teammates(For example, Team A is supped to be evaluated by Team B, Team C and Team D. Then Team B, Team C and Team D are evaluator teams to Team A and Team A is the evaluated Team to all 3 teams).
- Enable assignment of evaluation relationship among teams. Each team will be assigned 3 teams as evaluators.
- Enable students to view submissions from teams that they should evaluate(For example, Team A is supposed to evaluate Team B and Team C;

Then students in Team A should be able to view submissions from Team B and Team C).

- Enable students to evaluate submissions from teams that they are evaluating.
- Enable students to view peer evaluations from peer teams and their adviser(For example, Team B is supposed to be evaluated by Team A and Team C; Then students in Team B should be able to view peer evaluations submitted by Team A and Team C for Team B). For public part of a peer evaluation, students can see response with name of the team that submitted that evaluation; As for private part, students can only see a compilation of all private-part responses without team names.
- Enable students to submit feedback regarding the evaluations received(For example, Team B is supposed to be evaluated by Team A and Team C; Then students in Team B should be able to submit feedback to Team A and Team C regarding peer evaluations received from each team).
- Enable advisers to view a list of all teams under his/her supervision and edit any of these team's details including "Team name", "Project Level" and "Has Dropped".
- Enable advisers to submit evaluations to submissions from teams he/she supervises.
- Enable advisers to view feedback from his/her teams.
- Enable advisers to view a list of all teams and students in the current cohort.
- Enable advisers to view, edit and delete evaluation relationships among teams he/she supervises.
- Enable administrators to view, edit and delete any users, students, advisers, mentors, administrators, milestones and evaluation relationships.
- Enable administrators to login as any user into the system to debug and oversee things.

- Enable listing of all staff of Orbital program and display of their public profiles.
- Enable listing of all teams that passed Orbital program and display of the team's basic information.
- Add notion of cohort to Orbital program so that Skylab can serve for Orbital program for different cohorts.
- Enable advisers, mentors and admins to batch send reminder emails to students and keep a history of past emails sent for checking.

1.4 Outline

In this report, I will discuss various accomplishments I have done in the development of Skylab. Chapter 2 will be an overview of current architecture of Skylab, and methodologies I employed during the development. Chapter 3 will be talking about solutions to problems in the implementation of registration and evaluation process. Administrative portal will be discussed in details in Chapter 4. Chapter 5 will be focused on implementation of public profiles of staff and projects in Skylab. Then security related issues such as user authentication and access control will be discussed in Chapter 6. Chapter 7 is about how testing is done during the development of Skylab to ensure correct behaviors of various functionality and also feedback received from the focus group meeting with advisers in Orbital program. Last but not least, a summary of work and an overview of future development ends the report in Chapter 8.

Chapter 2

Background

The implementation of Skylab faced many design choices that shaped its fundamental development. We describe and justify important major design choices here.

Skylab is built using Ruby on Rails, a well-known web development framework with a great support from a large community, used widely in the industries by companies like Twitter, Groupon, Bloomberg, Airbnb and many more[2]. There are many reasons why we have chosen Ruby on Rails. Firstly, Ruby is clean, elegant and easy to read and readability and elegance of Ruby enables programmers to be more productive. Secondly, Ruby on Rails adopts many advanced industrial conventions and this enables contributors to have good exposure to programming in the industry. What is more, scaffolding and the contribution of many libraries (gems) by the Ruby community can significantly simplify programmers' work by saving time and effort of the development team from "reinventing the wheel". The popularity and activeness in Ruby on Rails' community ensures that getting continuous support would be easy and many resources can also be available online. Last but not least, Ruby on Rails community has a favor for open source contribution which aligns well with the open source nature of Skylab.

For the selection of database, we used PostgreSQL. Part of the reason is that it is open source and quite mature, with good drivers available in many languages[3].

Besides, we need full ACID (atomicity, consistency, isolation and durability) compliance for consistency of data and we do not need scalability to multiple servers in the foreseeable future[4]. And PostgreSQL has recently added implementation for rich data structures such as JSON which eases development[5].

Puma is the web server we have chosen for Skylab. Among common Ruby web servers such as Passenger, Unicorn, Rainbows!, Puma is considered to be fast and memory friendly according to online benchmarking[6]. Puma is built for high-concurrency and speed and we observe that more developers are switching to Puma within the Rails community[7].

We have selected Nginx as our web (HTTP) server. Nginx has grown in popularity since its release due to its light-weight resource utilization and its ability to scale easily with low memory usage. It excels at serving static content quickly and is designed to pass dynamic requests off to other software that is better suited for those purposes[8]. There are also some benchmarking results that indicate the superiority in Nginx handling concurrent access and of its small in-memory footprint[9].

A high-level overview of the architecture of Skylab is illustrated in Figure 2.1. Incoming requests to server will first be forwarded to Puma worker processes by Nginx. After that the corresponding Skylab code in the Ruby on Rails framework will be executed. Where database access is required, PostgreSQL serves the data.

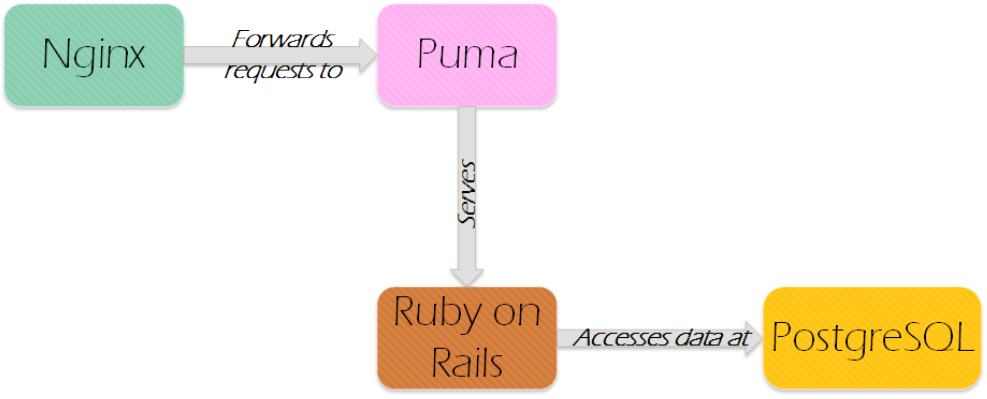

FIGURE 2.1: Architecture of Skylab

2.1 System design

2.1.1 MVC pattern in Ruby on Rails

The basic MVC structure of Rails is shown in Figure 2.2:

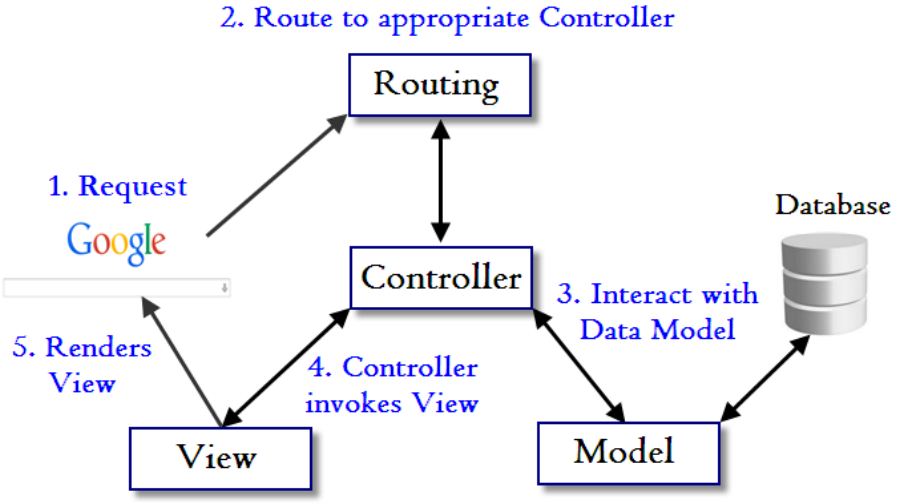

FIGURE 2.2: Illustration of how MCV works in Rails. Excerpted from [10]

After request is received by the Rails framework, the *router* will look at the pattern of the requested URL path and send it to the corresponding method of the target *controller* class. The *controller* is supposed to query *models* and gather necessary information for rendering the *view*. Last but not least, the response will be sent back to user for viewing.

So the most fundamental component in this whole process is the *model*, which stores all the business data. A proper model design can save a lot of trouble when it comes to writing controllers.

2.1.2 Skylab Model Overview

Figure 2.3 (source available at [11]) gives an overview of the current model design.

As is seen from the entity relation diagram shown in Figure 2.3, the users' basic information is captured in the *User* model and each user may have one or more

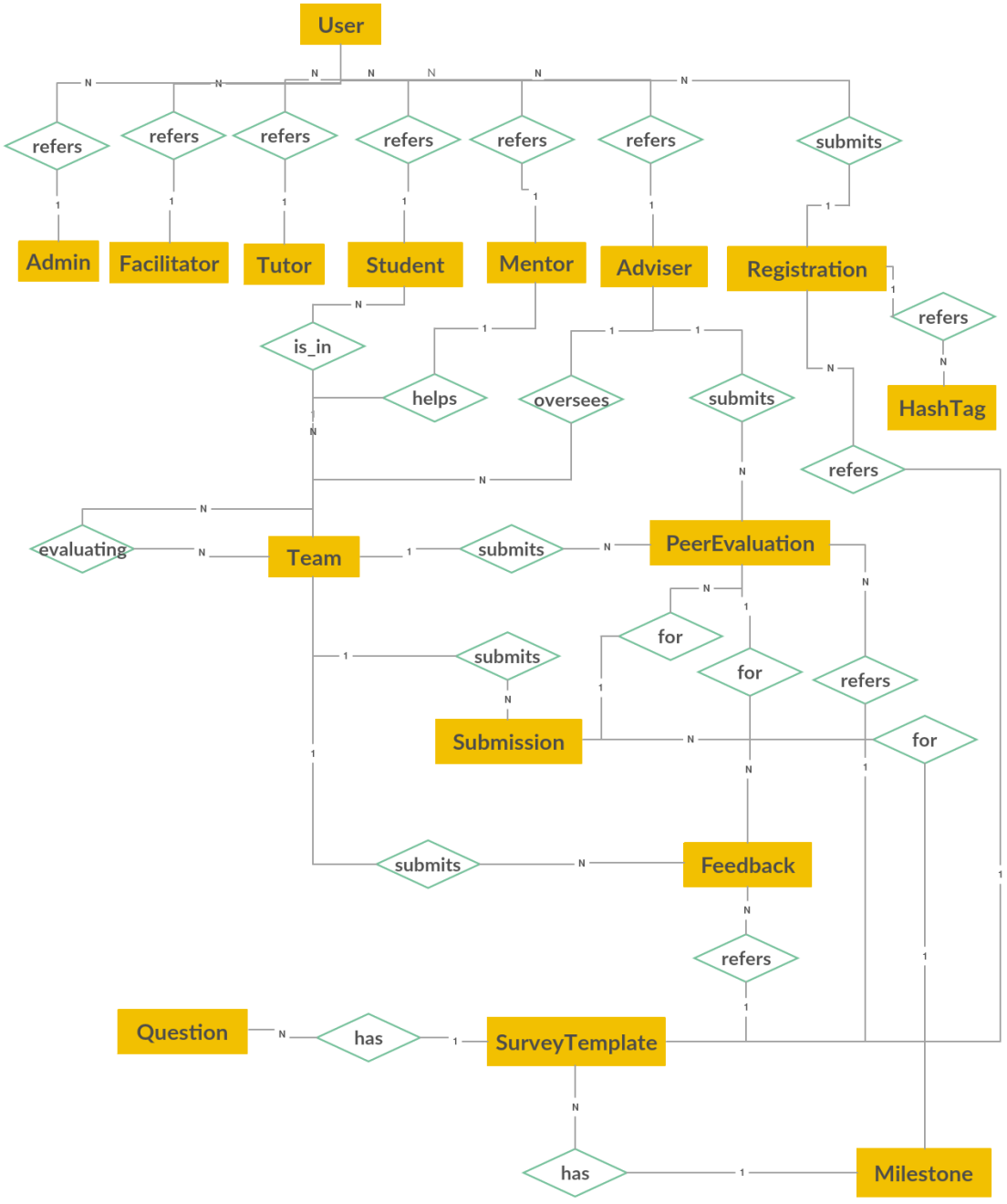

FIGURE 2.3: Current ER diagram for Skylab

different roles such as *Admin*, *Student*, *Adviser*, *Mentor*, *Tutor* and *Facilitator*. Each user can have only one of any type of role in a cohort – meaning that if *User* A is an *Adviser* B for cohort 2015, then his/her adviser role for cohort 2015 will only be B (and of course he/she can also be a mentor, a tutor or another role type in cohort 2015). But for cohort 2016 if *User* A is an adviser again, then it is another *Adviser* role C. Roles like *Facilitator*, *Tutor* do not have practical meanings in the

system but only for display in public staff page. *Admin* is supposed to overlook manage everything in Skylab. The remaining roles, *Adviser*, *Mentors* and *Student* are connected via different relationships.

Each student has a *Team* and a team has an *Adviser* and a *Mentor* (optional). A group of teams under the same adviser is called an *Evaluation Group* and evaluating relationships will be assigned from team to team in the same *Evaluation Group*. Each team will create *Submissions* to *Milestones* to report their progress and their adviser and assigned evaluator teams will submit *PeerEvaluations* to the team's *Submissions*. After receiving *PeerEvaluation* from evaluator teams and adviser, the team need to provide *Feedback* to rate the helpness of those *PeerEvaluations*.

Before Orbital officially starts, interested students need to register in Skylab first. A *Registration* will be created to record information about the student and his/her interested topics of project in the summer for recommending potential teammates if they do not have a partner in mind at the time of registration.

PeerEvaluation, *Feedback* and *Registration* can be considered as *Surveys*. Basically a *Survey* contains some instructions and consists of questions to be answered. So in Skylab, a *SurveyTemplate* would record information like instructions to students and have many *Questions* to form a *Survey*. Responses of *Survey* would be stored in *PeerEvaluation*, *Feedback* and *Registration*.

2.1.3 Overview of Controllers in Skylab

Controllers will query models to get information needed to present views or make modifications to models if requested. Therefore they are crucial in powering a Rails application and in Skylab we have well-defined controllers to carry out different tasks. A class diagram of all controllers in Skylab is shown in Figure 2.4.

ApplicationController is the base of all other controllers. Utility methods such as access control, page title and handling of exceptions are included in *ApplicationController* and it also includes auxiliary modules such as *CohortHelper* to

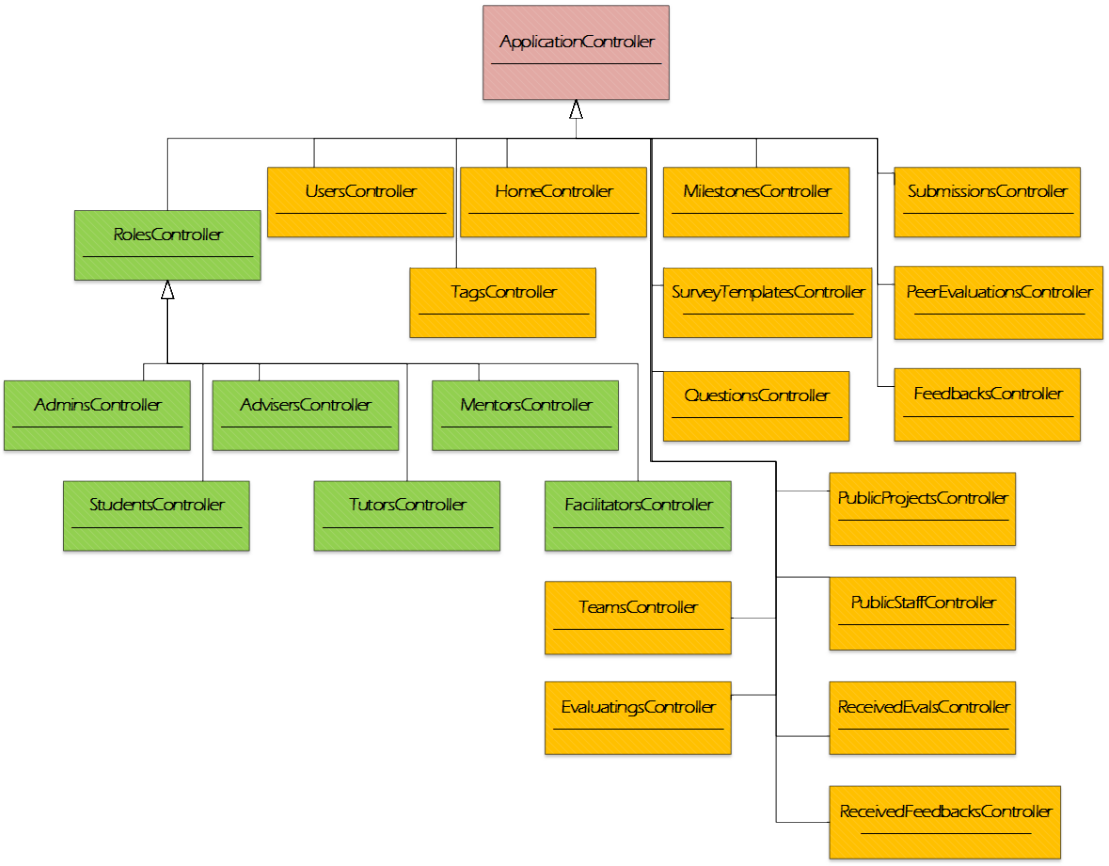

FIGURE 2.4: Current Class diagram of controllers in Skylab

enrich its functionality. *RolesController* is the base class of all roles' controller classes: *AdminsController*, *AdvisersController*, *MentorsController*, *StudentsController*, *FacilitatorsController* and *TutorsController*. These roles share quite a lot in common so they will share request processing methods and they will override data related methods or provide some handlers for special cases. Other controllers are as follows:

- **HomeController**: serves home page to user only.
- **UsersController**: manages listing, creation, editing and display of users and also includes functionality for admin to masquerade as any user, for new users to register as students in Orbital.
- **TagsController**: lists tags based on specified filters.
- **MilestonesController**: handles listing, creation, editing and display of milestones.

- **SurveyTemplatesController**: handles listing, creation, editing and display of survey templates and also serves as interface for editing of questions belonging to current survey templates.
- **QuestionsController**: manages Ajax requests to create, edit or delete a question.
- **SubmissionsController**: manages creation, editing and display of submissions by students.
- **PeerEvaluationsController**: handles creation, editing and display of peer evaluations from evaluator teams or advisers to evaluated teams.
- **FeedbacksController**: manages creation, editing and display of feedback from evaluated team to evaluator teams or advisers.
- **ReceivedEvalsController**: compiles a set of received peer evaluations for a team in one milestone.
- **ReceivedFeedbacksConroller**: compiles a set of received feedback for a team or an adviser in one milestone.
- **PublicProjectsController**: serves completed teams' projects to the public.
- **PublicStaffController**: serves staff of Orbital program to the public.
- **TeamsController**: manages listing, creation, editing and display of teams.
- **EvaluatingsController**: handles listing, creation, editing and display of evaluating relation among teams.

Controllers manage a collection of *resources* in Skylab, meaning that each controller is managing one model class, to facilitate higher cohesion. If there is a necessary dependency on other models — for example, SurveyTemplate will manage questions belonging to current SurveyTemplate — the requests will be handled by Ajax requests to separate concerns.

Skylab follows the Ruby on Rails convention for controller classes. A basic controller that manages a collection has the following structure illustrated in Figure 2.5.

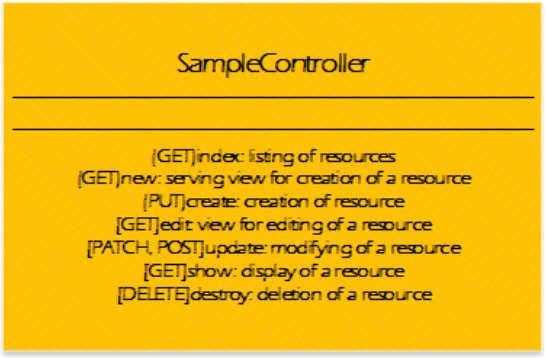

FIGURE 2.5: Sample controller class in Skylab

These methods – *index*, *new*, *create*, *edit*, *update*, *show* and *destroy* – have included all the basic actions to be done with a collection of resources. Following this convention makes controller class design in Skylab consistent and easy to read, and also aligns well with the goal of creating a more human-readable RESTful URL design.

2.2 Development / source code management process

GitHub flow is used in Skylab's development process. It is simple but agile enough for project management. Skylab is a web application for which releases of new versions do not require any user action. Each release can be deployed to server without users noticing it. This flow is quite different from traditional Git flow, which is far more complex and puts a lot of focus on each release[12]. In fact, in GitHub flow, each commit on master branch should be releasable and continuous deployment is recommended[12]. Each feature will be developed on a feature branch and once the feature is ready, a pull request is created against master. After code review and running test suite, pull requests are merged into the master branch and is then ready for deployment (Figure 2.6).

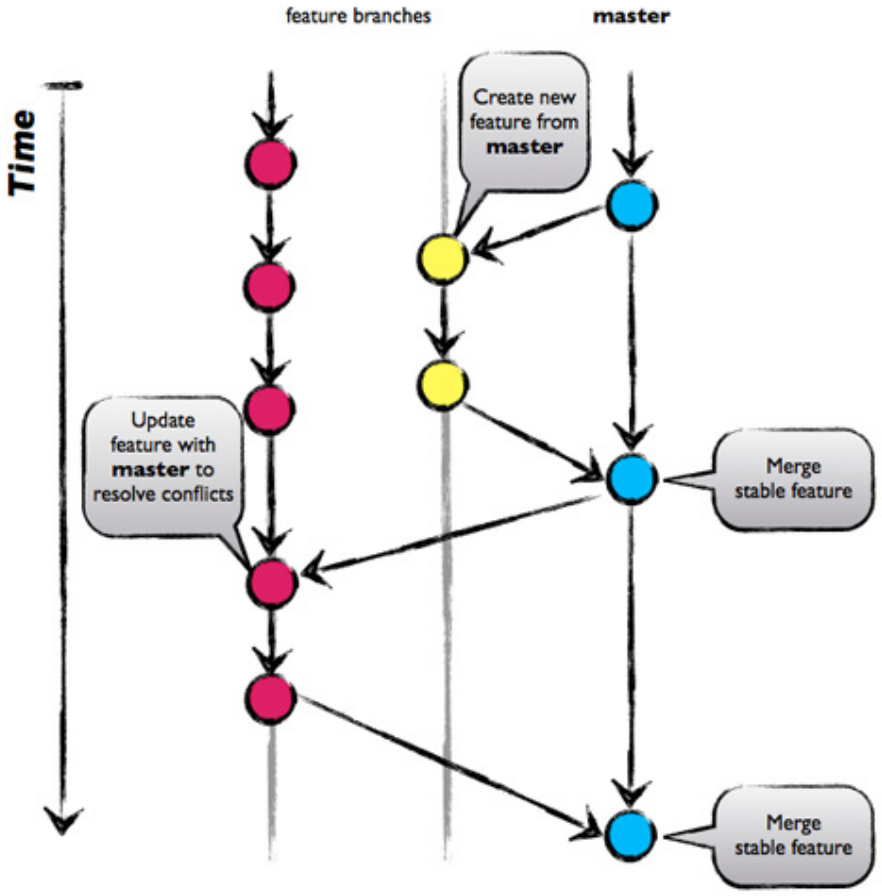

FIGURE 2.6: Branching diagram for GitHub Flow. Excerpted from [12]

By using GitHub flow and services such as Travis as the continuous integration service[13], CodeClimate as the code quality monitoring service[14], we keep the development of Skylab agile and fast. The use of GitHub flow enables Skylab to deployed regularly, which is essential for timely improvement of user experience[15].

Chapter 3

Orbital workflow

To appreciate the scope of the design and implementation Skylab, we detail the major work processes that the various user roles need to accomplish during an iteration (one year's cohort) of Orbital. We describe the workflow for Orbital students first, and circle back to describe the workflows of other supporting roles in the process.

3.1 Registration

To enroll in the Orbital program, students will need to register in Skylab first. When a user first logs into Skylab, a *User* object will be created for user. And since it is the first time the user logs in to the system, there is no existing role assigned to current user in current cohort. Then Skylab assumes that the user is trying to register and therefore a link to registration will be displayed. This workflow works for most users who are students in Orbital program. As for roles like *Adviser*, *Mentor* and *Tutor*, admin users will create those roles manually. This is because there are only a few such roles and they have higher privileges and therefore creation of such roles should definitely be controlled by admin users.

For students' registration, after a user click the registration link, a form consisting of different questions will be presented to the user. Among these questions, several

are to get a sense of what sort of interests the user has for the project over the summer. The user is supposed to select interested topics from different categories like programming language, web framework or mobile platform, IDE or text editor, e-commerce, cloud platform, purpose, embedded system, game framework, content management system and audience. These information will be recorded for match-making among students who need to find a teammate in Skylab.

After completing the registration form, if the user already has a teammate in mind, he/she can send an invitation to the potential teammate to form a team. Some users, on the other hand, do not have anyone to partner with for Orbital yet and a match-making algorithm will be run offline by admin user based on the interested topics they have selected during registration. An illustration of the whole workflow is as Figure 3.1.

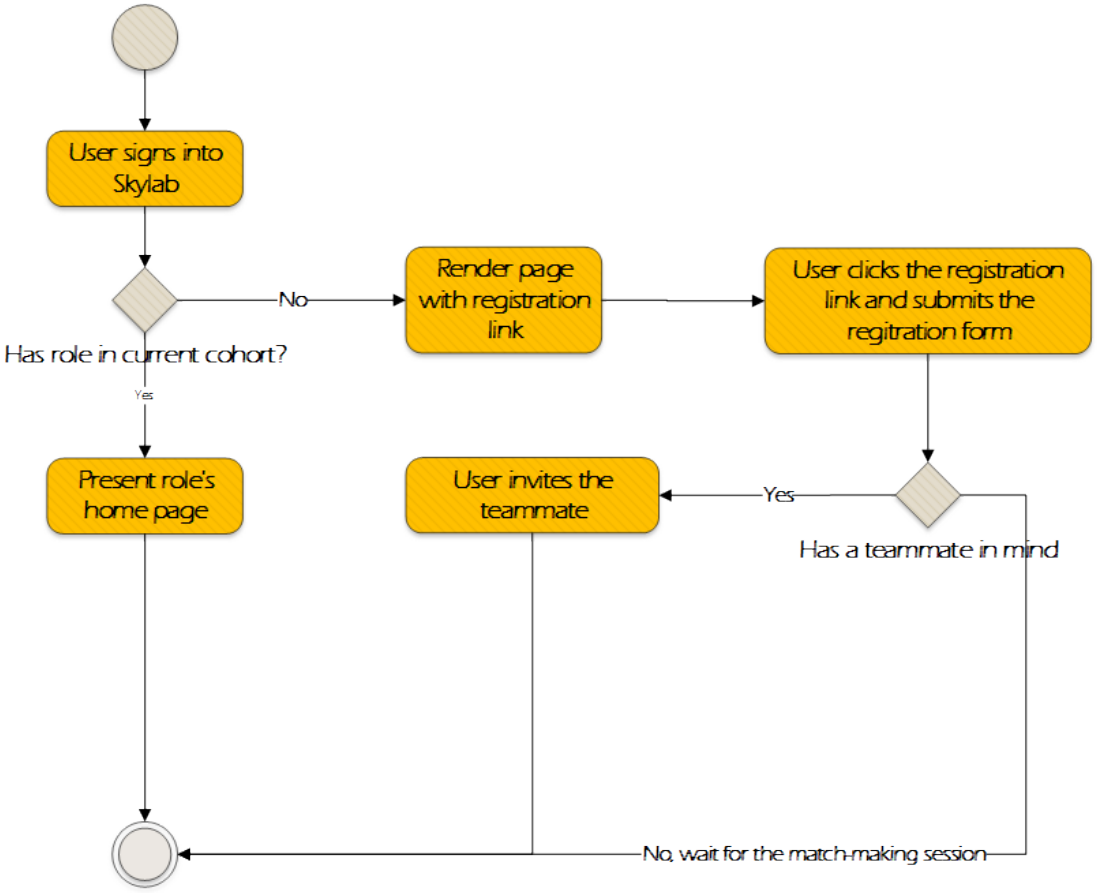

FIGURE 3.1: Illustration of registration workflow in Skylab

3.1.1 Match making algorithm

For students who do not have potential teammates at the time of registration, we will run a matching making algorithm based on the interested topics recorded in registration form submission. The algorithm composes of mainly two steps:

- Calculate inverse document frequency of tags: inverse document frequency is to reflect how rare a term is in the corpus[16]. In Skylab's match making's case, a document is a user's selection of interested topics, or tags; the corpus is the collection of users' interested topic sets; a term is a tag in a users' selections. If a term(tag) is rare among registration form results, we will be more confident that if both students choose this term and they will have higher chance of forming a team. The formula of calculation is as Formula 3.1.

$$idf(t, D) = \log \frac{N}{\{d \in D : t \in d\} + 1} \tag{3.1}$$

  with N as total number of documents in the corpus $N = |D|$ and $\{d \in D : t \in d\}$ as number of documents where the term t appears[16].

- Construct a vector space model based on students' selected tags and each tag's weight calculated in previous step and calculate similarity(cosine) of different students. The formula for calculating cosine of two vectors is as Formula 3.2[17].

$$\cos\theta = \frac{d_2 \cdot q}{\|d_2\|\|q\|} \tag{3.2}$$

  And we represent a user's interested topics with tags' weights and the similarity between 2 users will be the cosine between 2 vectors representing the 2 users. The formula for calculating cosine of two vectors is as Formula 3.3[17].

$$sim(d_j, q) = \frac{d_j \cdot q}{\|d_j\|\|q\|} = \frac{sum_{i=1}^{N} \omega_{i,j}\omega_{i,q}}{\sqrt{sum_{i=1}^{N} \omega_{i,j}^2}\sqrt{sum_{i=1}^{N} \omega_{i,q}^2}} \tag{3.3}$$

As matching making only needs to be run once before Orbital officially starts for those students who do no have a team yet, the algorithm is currently designed to run offline as a rake task. Currently it is only for students but in the future with recoding of mentors' and advisers' interested tags, this algorithm can be easily extended to students and mentors or students and advisers as well.

3.2 Submission

Students in Orbital are supposed to report their progress regarding their project at each milestone via *Submission*. Basically a submission contains 3 parts:

- README: highlights what are the changes and new features in the project.
- Project Log: a summary of work done during the phase and time used for each task.
- Video link: a link to a video introducing the project to evaluators.

As the structure of *README* and *Project Log* is free and we should allow creativity of students when it comes to describing their own projects, we decided to support rich text for submissions and there are some issues coming along with this decision such as SQL injection and XSS attacks. What is more, during use of Skylab, many good suggestions regarding user experience were brought up by students and advisers, like uploading of images and auto-expanding textareas during editing.

3.2.1 Handling of rich text

We used TinyMCE to support rich text editing feature in Skylab as it is a very popular WYSIWYG(what-you-see-is-what-you-get) editor with a rich set of features[18]. There are some libraries with markdown syntax supported such as EpicEditor, Vue.js and Hallo.js which are more lightweight. However, as Skylab is built for freshmen to get more hands-on experience with coding, we do not expect students

to be equipped with much prior knowledge such as markdown syntax. In the contrast, markup based editors such as TinyMCE and CKEditor do not require any learning and are easily to get started with. What is more, the large community using TinyMCE has made various plug-ins available for different features. There are also quite good resources online about integrating TinyMCE editor in a Rails project. Therefore, we decided to use TinyMCE for *Submissions*' rich text support.

One particular disadvantage of using TinyMCE is that the size of this editor is pretty large and loading of the page will be slowed down because of it. Therefore, TinyMCE related resources is only loaded if the page is requiring rich text editing. This is done by configuration to disable auto loading of all JavaScript, which is the default behavior of Rails. In this way, most pages can be loaded within a very short time. What is more, assets whose names start with "tinymce" are set to expire in a year to reduce frequent requesting of such resources from clients — which basically means students only need to load these resources once in one year.

Rails has built-in checking against SQL injection attacks and therefore Skylab is safe from such attacks when storing submissions' contents[19]. However, there is still currently a known bug in the implementation when it comes to viewing of submissions. As contents like *README* and *Project Log* should be rendered as rich text, it is possible for students to perform XSS attacks by injecting executable JavaScript code in the submission. This sort of vulnerabilities will be fixed in the future.

3.2.2 Usability

Skylab is a software engineering project and therefore improvements in user experience is one key part when it comes to implementation. During the use of Skylab, many suggestions were brought up by students and advisers about usability. And by addressing these issues, Skylab is serving users better with a smoother user experience.

3.2.2.1 Target Milestone Selection

When Skylab was used for the first time, students were expected to choose the target milestone, which the submission is for. However, many students reported that it is just a redundant step as every time they will only submit to the currently active milestone. After hearing this, a quick fix of automatically selecting the current milestone for students using JavaScript functions was done, while the manual selection is still possible. And then during the *Adviser Focus Group Meeting*, some advisers further pointed out that the selection should not even be presented to users as it is of completely no use and therefore the whole selection was completely removed from Skylab after the meeting by moving the task to the backend logic.

3.2.2.2 Rich Text Editing

As quite some students want to insert image to *README* sections, an image uploading feature was soon added to submission page for users' convenience. Behind the scene Skylab is using a third-party API from Imgur. This was done mainly for 2 reasons: using Imgur is relatively easy to implement; we can also avoid heavy server load due to file uploading and possible attacks in uploaded files.

Another improvement over user experience is auto expanding of TinyMCE editing area as feedback from students mentioned that their *README* and *Project Log* are usually quite lengthy. So auto expanding would not require too much scrolling during creating/editing a submission.

3.3 Peer Evaluation

After students have submitted to milestones, peer evaluation process can begin. Teams will look through evaluated teams' projects and submissions and then evaluate their performance in *PeerEvaluation*, which is a very important component in

determining whether the evaluated teams can pass Orbital or not. Although there are different questions for each peer evaluation, all evaluations contains essentially 2 parts:

- Public: a section with general feedback on how well the evaluated team has done and the response will be viewed by target team with evaluator team name available.
- Private: a section with critiques and overall rating. Critiques will only be viewed by target team without any evaluator team information while overall rating is only for grading purpose and not viewable by target teams.

3.3.1 Loading of Peer Evaluation templates

When Skylab was first developed, the public part of a *PeerEvaluation* templates will be one HTML form and the private part is another one. This means all questions in the public part are coded in one predefined html template and same goes for questions in private part.

This approach does not seem to have much problem at first and it really served Skylab's purpose well by delivering perfectly workable features in time. However, since questions are different for different milestones, separate templates have to be created for each new milestone. And when user wants to view/submit/edit a *PeerEvaluation*, the corresponding templates will have to be loaded based on properties of the milestone. In this way, the system is not really open to extension and clearly it violates Open-Close principle. After realizing this, a system for dynamically creating questions has been implemented for *Feedback* and migration of *PeerEvaluations* to the survey-template-and-questions based system has been done as well. A more detailed description for how the dynamic loading of questions is in Section 3.4.

3.4 Feedback

Feedback is for evaluated teams to evaluate the *PeerEvaluations* they have received and it is also a very important component in determining whether the evaluator team can pass or not. After realizing the lack of extensibility in the design for *PeerEvaluations* and with more time available for implementation of *Feedback*, a survey-template-and-questions system has been set up, which made the system open to extensions for more questions and even question types.

3.4.1 Survey-template-and-questions system

The basic flow when a request to create a *Feedback* is illustrated in Figure 3.2.

So when a student clicks the button to create feedback, *FeedbacksController*'s *new* action will be invoked and inside the method, the corresponding *SurveyTemplate* and all *Questions* belonging to the *SurveyTemplate* will be sent to view. Then instruction for the *Feedback* and all questions will be rendered as response to the student. After the student completed and pressed submit button, responses to all questions will be sent to server and a Feedback instance containing all submitted responses will be created.

There currently 4 types of *Questions* in the system as Figure 3.3:

- **TextQuestion**: a text question which will be rendered in the format of textarea and text-based responses are expected.
- **RichTextQuestion**: a rich text question which will be rendered in the format of TinyMCE editable area and rich text is expected as response.
- **MultipleChoiceQuestion**: a multiple choice question which will be rendered as radio buttons followed by option descriptions and one option value is expected.

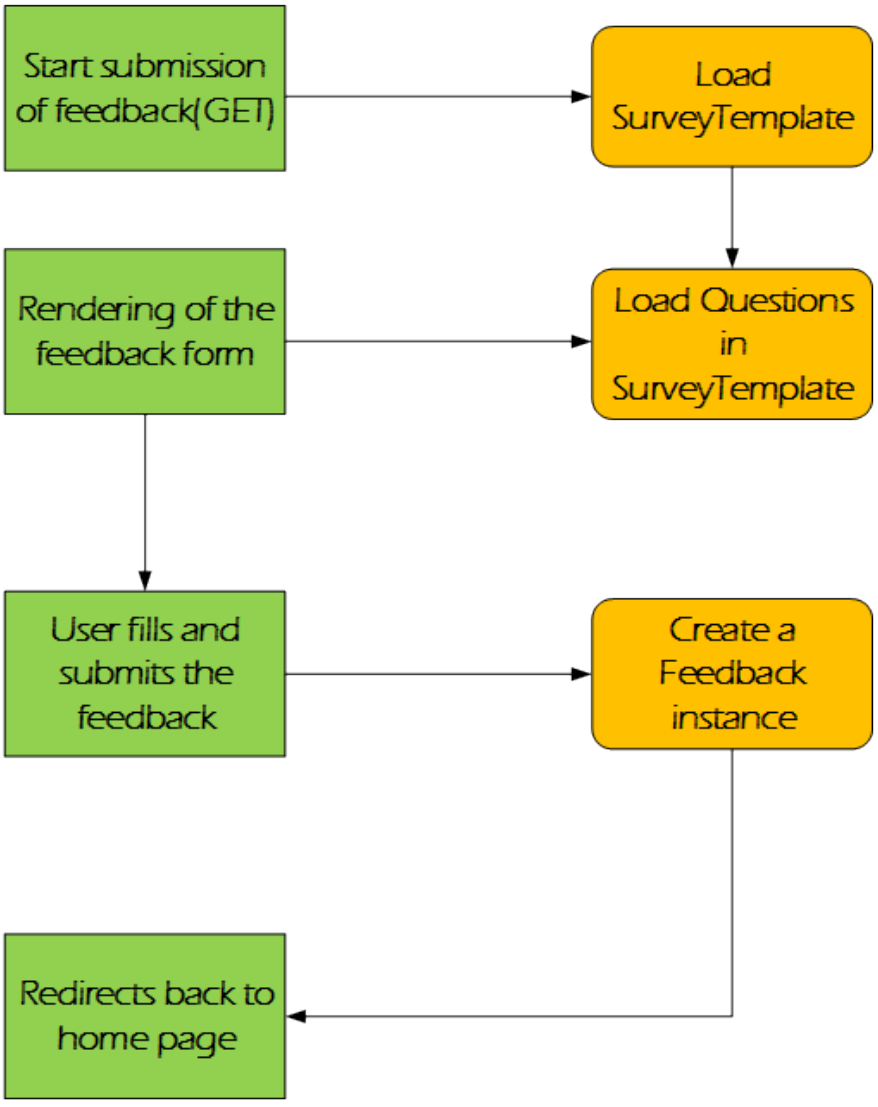

FIGURE 3.2: Flow of a feedback creation process

- **MultipleSelectQuestion**: a multiple choice question which will be rendered as a select element with "multiple" enabled with each option as one choice of selection.

All types of question will have *title*, *instruction*, *content*, *extras* and *question_type* and each type of question will have its own way of rendering and validation. In this way, adding new types of questions can simply be done via creating a new question model with its own rendering and validation methods.

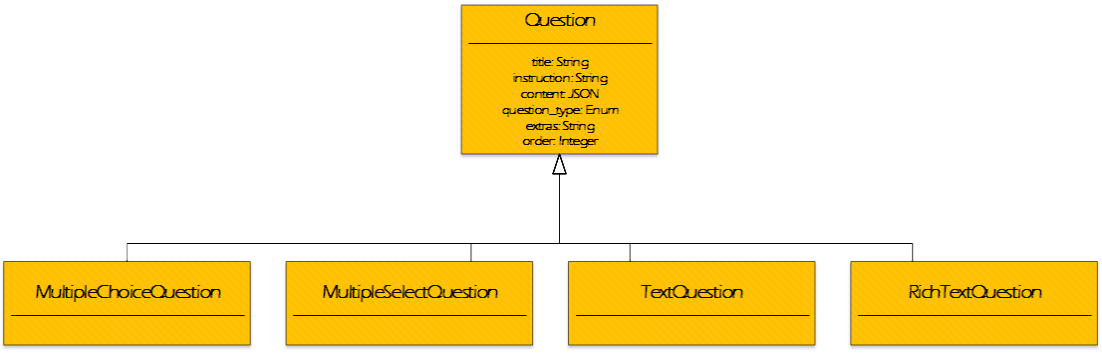

FIGURE 3.3: All types of questions in Skylab

Chapter 4

Administrative Portal

Administrative users are super users who monitor and control Skylab to monitor progress of students and performance of different teams and advisers. Therefore they have the highest privileges in Skylab. And due to the fact that admin users have so much to do in Skylab, a lot of effort has been put into the development and improvement of functionalities related to administration.

4.1 Overview and export of data

To oversee and manage different perspectives of Skylab, admin users need to have access to all information in Skylab in a easy way. In Skylab, the basic steps for an admin user to have an overview of something is simply to navigate through navigation header and after that a table consisting of brief information about each record in the collection will be presented in a table's form. For example, if the admin user wants to view all the students in current cohort, he can simply press the link in the navigation header and students in the current cohort will be listed with basic information such as name, email, team, adviser, is_pending. Also for each student, the admin user can get more information by clicking view to go to the student's page or edit the student's information.

Sorting of records in the table is provided for admin users to quickly rank students by simple criteria. However, filtering is not currently implemented, as admin users are considered to be skillful and familiar with the data, and mainly use the "Find" functionality provided by the browser to locate records.

To enable admin users to do more work, we implemented an export function for student and team data. After getting the data exported as a comma separated values (CSV), admin users can use text editors or Excel to manipulate the data further. Although ideally some features could be added to Skylab so that admin users can just do it in the system without extra steps, due to the time limit and other tasks, exporting data is a quick way to get admin users to carry out desired tasks. Such a facility also enables future use cases beyond what is currently envisioned with Skylab.

4.2 Preview as other users

Admin users can view any user's information for review. Additionally, there is the added functionality for admin to "(simulate) fake login as" any other user; such that the admin session continues with the credentials of the user being simulated as. During the development of Skylab, checking student's view and adviser's view was needed in development to more easily verify problems and proofcheck their fixes. What is more, admin users can also use this feature to actually go through the user's workflows and to assess whether they need adjustment or not. Through this functionality, good suggestions about the user experience enhancement have been vetted by admin users. Besides these, "fake login as" can help admin see the issues reported by students and advisers in production, such that the code fix can developed, tested and deployed faster.

4.3 Mailing

Mailing is a very important feature for administrative users, as many reminders and announcements need to be delivered to users' emails for them to check. While Orbital also relies on other external methods for reminders (i.e., Facebook, Slack and Wordpress website), in practice multiple methods need to be used. There are then basically two types of emails sent in Skylab:

- **System-generated reminder emails**: these emails do not require admin users' actions to be sent to users; e.g., forget-password emails to reset password, welcome email when admin users create a user, reminder emails when deadlines are coming for submission, peer evaluation and feedback.
- **User-initiated emails**: this type of emails are written by users and sent at users specified timings and would serve as the channel for general reminders and announcements. And these emails will be recored as history so that user can look at past records to compose a similar email. This feature is not limited to admin users and in fact advisers are also allowed to use this to remind students in his/her evaluation group.

Admin users can make use of mailing to remind students and this can save administrative time and effort in manually sending emails. It can also serve as an alternative to existing communication tools (i.e., Slack) when students ignore those discussions.

Chapter 5

Public Views

Before Skylab was implemented, the display of past Orbital staff and projects was done by means of a manually curated WordPress post. Project links for teams and team members were all manually created, which was a tedious and error-prone process. Since Skylab already captures all information on teams and students, publicly-accessible (i.e., without needing to login) pages are a logical form of data export that has been created to supersede these past facilities.

5.1 User Avatars

Skylab does not currently handle image uploading, as it requires a lot more storage and more efforts in security as well. However, we need to support user avatar pictures as this enhances the user experience of Skylab. To implement this, we have integrated with an external third party service. We have decided to use Gravatar, a globally recognized avatar for any user[20]. Gravatar is popularized by Wordpress and in use by many sites. It is easy to integrate and a number of users already use gravatar for creating a universal user avatar for themselves. The steps to get an avatar are as follows:

- Trim the user's email address, convert it to lower case, and calculate MD5 hash of the transformed email address.

- Use the hash computed in previous step and append it to http://www.gravatar.com/avatar/ to obtain the gravatar's image source.

Gravatar serves our purpose of serving users' avatars while bypassing the complication of supporting image uploading and storage, as this is handled by the service. And the most important part in arriving at this solution is that it is a relatively quick, easy yet clean to serve users' profile images. Compromises like this happen as we need to weigh advantages and disadvantages of different approaches and take the time taken and priority of issues in implementation into consideration. In the future, if we find too many inconveniences from using Gravatar, we may decide to host avatar images on our own server, and handle problems arising from the support of file uploading and storage.

Chapter 6

Security

During the development of Skylab, security has always been a priority. Common attacks such as SQL injection, XSS attacks and CSRF attacks are all addressed by the Rails framework itself. Skylab's main security focus is to make sure authentication process and access control are well-defined and can withstand various attacks.

6.1 User authentication

There are basically 2 ways to login to Skylab: email and password combination and NUS OpenID. For email and password combination, we employ Devise to manage authentication related information in *User* model. Devise has been used widely in Rails community and currently there are more than 13,000 stars of Devise repository on GitHub[21]. What is more, according to security scanning of Devise by Hakiri there is no security warning, which further proves the trust-ability of Devise[22]. As for NUS OpenID, it is using OAuth 2.0 as authentication protocol, which is believed to be safe and currently is adopted by many OpenID providers as well[23].

6.2 Access control

Skylab is adopting role based access control strategy to grant different permissions for different users. There are basically 4 levels of checking for each incoming requests:

- **login_required:** a boolean value. If a action is login_required then a user must sign to be granted for the access.
- **admin_only:** a boolean value. If a action is admin_only then only users with role of *Admin* can access; but if current user is admin, the next check will be skipped and return true directly as admin users can access any resources in Skylab.
- **allowed_users:** a boolean value. If a action requires login and is not only accessed by admins, then the current logged-in user will be checked against the list of allowed users who have the permissions.
- **strategy:** a lambda function which can be executed and is expected to return a boolean value. If the lambda is provided by the caller, it will be executed and whether users in allowed_users can actually access the resource or not depends on the returned result.

As this checking procedure is a common to nearly all actions, it is defined in *ApplicationController* which will be effectively inherited by all controllers in Skylab. So all methods in those controllers can invoke this method to check user's permission first.

6.3 Security warnings

Hakiri is used as the security analysis tool in Skylab. Currently there are some warnings in cross-site scripting, code injection, file access, input validation, resource management according to Hakiri scanning result. However, all of these

security issues are actually vulnerabilities in current version of Ruby on Rails framework used by Skylab — so this means that by upgrading the Rails framework we can simply remove these issues.

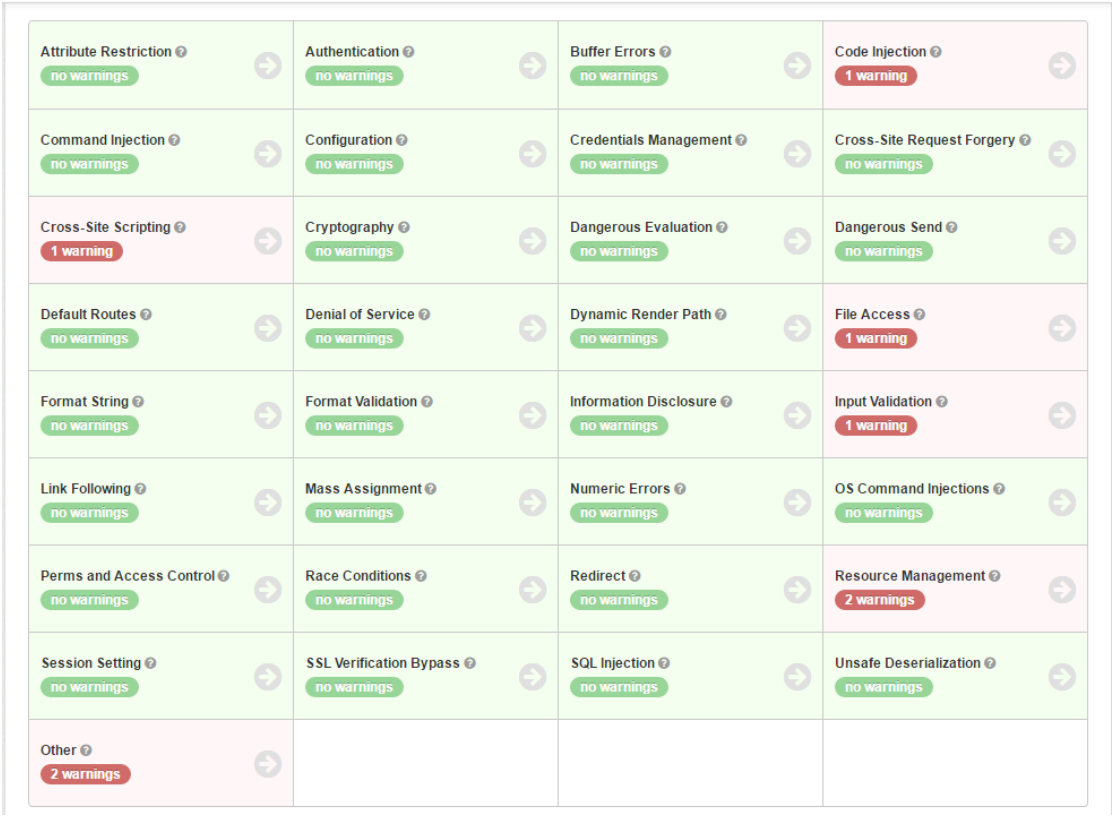

FIGURE 6.1: Hakiri security warnings

Due the importance of security in the development of a web application, security will remain as an prioritized task in Skylab and more future work will be done regarding this as well.

Chapter 7

Testing and user evaluations

Testing is very important in development as it serves as the validation and verification of the program[24]. In the process of continuous development, having tests helps us to catch errors, especially regression errors. Skylab as a software engineering project is covered by different levels of tests to ensure the delivered features are as promised.

Testing in GitHub flow – the development process adopted in Skylab – has a special place as well. Through continuous integration tools like Travis, each pull requests and commits on master branch will be tested and the status will be used as the indicator whether merging to master brunch should be done and the health of current master brunch.

RSpec is the testing library used in Skylab and it is widely used the testing tool for many Rails programmers as well[25]. And for tests involving database, FactoryGirl is used for creating records in databases and DataCleaner to ensure testing database is cleaned before and after tests. The syntax of FactoryGirl and RSpec is pretty expressive and concise, which aligns well with design of Rails, improving the readability of tests.

7.1 Unit testing

Models are the most fundamental components in Rails' MVC pattern and they are tested and fully covered in Skylab. In Skylab, models' testing are executed on testing database instead of faking models to test logic and validation in models only. Although this would slow down execution of test suite overall as database access is usually slow, we want to make the integration of models' logic and constraints in database work as expected. And since the test suite of Skylab is quick to complete (less than 2 minutes to run), we favor better tested code over faster test execution.

Besides models, controllers, mailers and helpers are tested as well. The unit testing can make sure each individual component in Skylab is working as expected and also contributes the most to overall test coverage.

7.2 Acceptance testing

Acceptance testing is to make sure the whole system has met the requirement specifications when users execute their various activities[26]. To simulate user actions such as filling the form and clicking buttons, Capybara is the used as the testing library for acceptance testing. It is a very popular testing library written in Ruby and works well with RSpec[27]. There are different drivers used by Capybara as well. For most use cases where JavaScript execution and external resources are not required, RackTest is used as the default driver as it is lightweight and fast to run[27]. Capybara-webkit is used for those cases in which JavaScript execution is required as it supports JavaScript and it is still faster than drivers such as selenium since it can save the process of loading of an entire browser[27].

Most use cases are covered in acceptance tests for Skylab, inclusive of: registration form submission, team invitation, admin's overview of different models in Skylab, students' submissions, peer evaluations and feedback. Although the testing coverage in Skylab is decent and that most use cases are included in acceptance testing

as well, there are more details to check that need more testing coverage, such as (currently): advisers' evaluations, management of evaluating relationships, among others. Trying to cover more of these workflows in Skylab is therefore one of the main future tasks.

7.3 Focus group meeting

A focus group is a form of qualitative research in which a group of people are asked about their perceptions, opinions, beliefs, and attitudes towards a product, service, concept, advertisement, idea, or packaging[28]. During the middle of the first semester, we convened a focus group meeting about Skylab with two advisers and one facilitator. The group was conducted to get advisers' suggestions and feedback on experience with Skylab. The feedback was generally positive, while there were useful suggestions and findings that helped to feed issues that were to be fixed and implemented at later rounds of development in Semester 2.

The focus group was conducted on 7th Oct 2015 and three questionnaires were designed beforehand to get advisers' opinions. During the meeting, discussions about the complete use case flow of Skylab were done and the whole meeting was recorded as well for later reference. Suggestions mentioned in the discussion and described in the responses to questionnaires are listed below, which are also documented in Skylab's repository[29]:

- Checking who had already evaluated / or sent feedback on an evaluation: we should have status columns (similar to "dropped" status) for submission status, peer evaluation status so that with just one glance adviser can figure out who have not submitted and remind them (this issue is still in the issue tracker);
- Change tab order for adviser homepage: move the current first tab to last as advisers rarely use them (this has been implemented);

- Forms too verbose Three radio fields take up more than one page. Suggested way of presenting options: Poor 1[?] 2[?] 3[?] 4[?] 5[?] Best(Question mark means only after the user hovers over the detailed explanation for the option will be given) (this issue is still in the issue tracker);
- Hosting of videos (this is not accepted as it seems to be too ambitious);
- Auto expand text boxes when close to full (this has been implemented);
- Dropdown for team selection not clear about whether the new team is being evaluated (this has been implemented);
- Autosave text and input / prompt for leaving page (this has been implemented);
- Summary composite page for the evaluation group with respect to each question for Likert scale and textboxes / Quick Fix: use anchors to jump to a particular form on a concatenated page (this issue is still in the issue tracker);
- "More info" tab in adviser homepage is misleading (this issue is still in the issue tracker);
- Enable users to view all past submissions, all past peer evaluations(by providing link to viewing them in some place) when you are editing/submitting a peer evaluation (this has been implemented);
- Evaluating relationship: explore use of D3(or similar stuff) to represent all relationships as graph (this issue is still in the issue tracker);

Chapter 8

Conclusion and future work

8.1 Conclusion

Skylab serves facilitators, advisers, mentors and students and potential students by easing the administrative experience in Orbital. Skylab supports the common workflows of the different roles of users through the lifecycle of their usage over the lifetime of an Orbital cohort. Skylab plays a helpful role in registration, match making after registration, submissions of project log and README, peer evaluations for evaluated teams' submissions and feedback to peer evaluations. We have also accomplished much additional work on the admininstrative portal for overseeing Skylab, inclusive of reminding users of deadlines, and creating suitable public views to display past projects and staff in Orbital program.

The development process of Skylab is certainly agile and iterative. Many features are developed as prototype first and after discussion adjustments are made to have a better user interaction experience. Besides, as Skylab is continuously being used, some requirements and suggestions are made by users as well. Many challenges to system design rise in the process of changing and refining. Various compromises have to be made to due to deadlines or other important issues. During implementation of Skylab, coping with changes and deadlines is the most important skill. Designing a system with enough features, with enough good features, with enough

good features and extensibility to add more good features is huge challenge and Skylab is trying to be a system with many good features and open to extensions in the future.

8.2 Achievements

8.2.1 Time and effort

Skylab has been developed for over an year. Since March 2015, various of issues and challenges have been conquered as achievements in the continuous implementation process. A recording of commit history of Skylab in GitHub is shown in Figure 8.1 which consists of over 500 commits and over 75K lines of code added and changed.

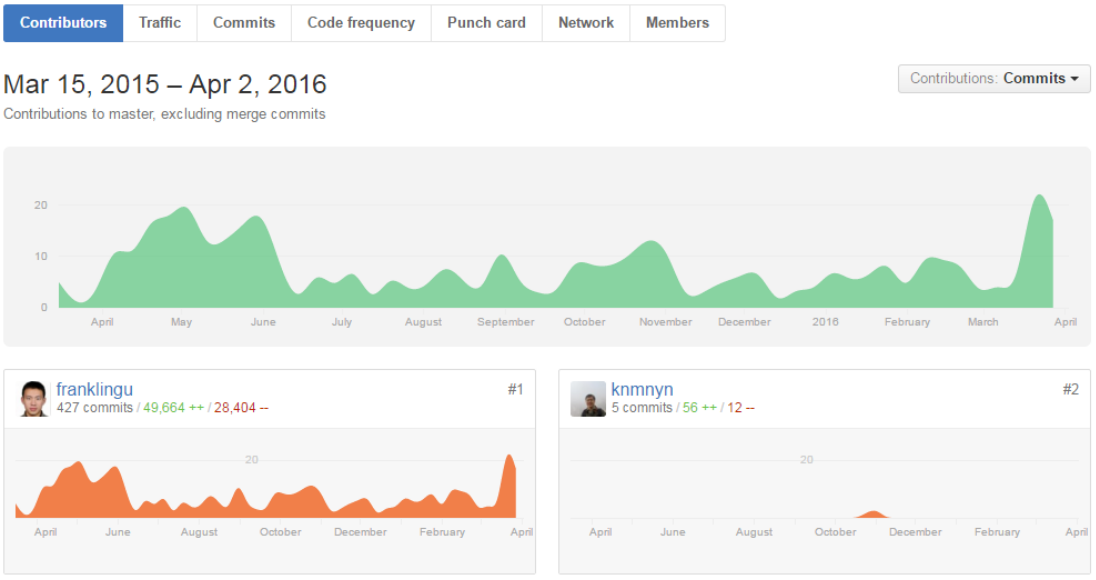

FIGURE 8.1: Contribution history of Skylab

8.2.2 Milestones, Deadlines and Changes

Just like any other software engineering projects, there are many deadlines during the process. And there will always be more requirements to complete, changes to expect along the way. Throughout the entire period of implementation of Skylab,

working with deadlines and changes along the way is common and this brought many practical challenges to system design as well. A timeline diagram with all important milestones is as Figure 8.2.

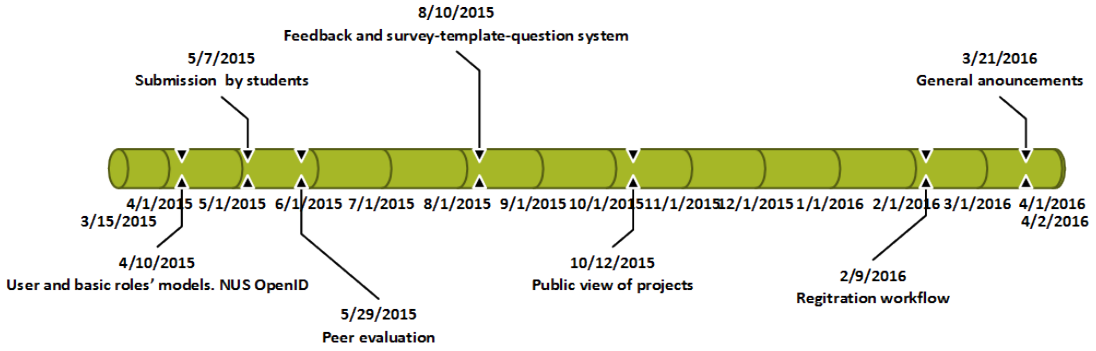

FIGURE 8.2: Timeline of Skylab

8.2.3 Live Server Setup and Maintenance

Skylab was first hosted on an Amazon EC2 instance and later migrated to an NUS SoC virtual machine. Currently, about 400 Skylab users perform their different roles. Maintenance work is needed on the production server, necessitating a good workflow to ensure data integrity and to minimize downtime to release new versions of Skylab.

8.2.4 Development Process

We adopted the GitHub workflow for Skylab's development to facilitate agile and iterative development. Also by adopting continuous integration tool like Travis, code quality and test coverage monitoring tool like CodeClimate, security analysis tool like Hakiri, the process of ensure testing, quality and security is greatly smoothened and different issues are easily monitored.

8.2.5 Major Use Cases

Skylab is designed to serve all kinds of people involved in Orbital and it has been working to support Orbital program. From registration phase of Orbital program to last feedback in Orbital, Skylab is critical infrastructure to the Orbital program, playing a major role in helping admins, advisers, mentors, tutors and students.

- Registration: students who are interested in Orbital can sign up via Skylab. They can do this as a team of two or as individuals first. For those individuals, a match making algorithm will be run to find potential teammates based on recorded interested topics of each student.
- Evaluation: students and advisers will go through evaluation process in Orbital which includes submissions, peer evaluations and feedback.
- Administration: admins can overview things in Skylab.
- Mailing: various emails as reminders can be sent through Skylab.
- Public facing of profiles: staff of Orbital, past projects of Orbital can be displayed in Skylab easily.

8.3 Future work

While Skylab serves its purpose well, there are still limitations and enhancemens that are planned. A proposed set of major features to be completed in the future for Skylab include:

- Questions/template system (involving migration of current data): currently *Feedback* and *Peer Evaluation* is utilizing the *SurveyTemplate* and *Question* system but *Submission* are still not. With a migration to the *Questions* system, we can further improve the system by adding more extensibility.

- Logging of user activities: by logging down activities carried out by different users, administrative staff can discover problematic events and acquire situational awareness of the context in Orbital.

- Unit testing needs to cover more classes and acceptance testing needs to cover more use cases in Skylab, especially by different roles.

- Upgrade of the Rails framework to address the warnings on Hakiri and continue working on improving security.

- Mailing could be improved by having mailing history and other customized reminders.

Bibliography

[1] *NUS School of Computing's Orbital Programme*, 2015. http://orbital.comp.nus.edu.sg/ [Accessed: 2015-11-03].

[2] Peter Argent. *Top 15 sites built with Ruby on Rails [@CoderFactory Blog]*, 2014. https://coderfactory.com/posts/top-15-sites-built-with-ruby-on-rails [Accessed: 2015-11-01].

[3] David Bolton. *Why I Choose PostgreSQL Over MySQL/MariaDB*, 2015. http://insights.dice.com/2015/03/19/why-i-choose-postgresql-over-mysqlmariadb/ [Accessed: 2015-11-01].

[4] Margaret Rouse. *ACID (atomicity, consistency, isolation, and durability) definition*, 2015. http://searchsqlserver.techtarget.com/definition/ACID [Accessed: 2015-11-05].

[5] Alan Moore. *FIVE REASONS WHY I CHOOSE POSTGRESQL*, 2013. http://www.alandmoore.com/blog/2013/02/28/five-reasons-why-i-choose-postgresql/ [Accessed: 2015-11-01].

[6] *A Modern, Concurrent Web Server for Ruby - Puma*, 2015. http://puma.io/ [Accessed: 2015-11-01].

[7] Engine Yard. *Rails Server Throwdown: Passenger, Unicorn or Puma?*, 2015. https://www.engineyard.com/articles/rails-server [Accessed: 2015-11-01].

[8] Justin Ellingwood. *Apache vs Nginx: Practical Considerations*, 2015. https://www.digitalocean.com/community/tutorials/apache-vs-nginx-practical-considerations [Accessed: 2015-11-01].

[9] *APACHE VS NGINX PERFORMANCE COMPARISON*, 2013. http://www.theorganicagency.com/apache-vs-nginx-performance-comparison/ [Accessed: 2015-11-01].

[10] *Ruby On Rails Tutorial: Model View Controller (MVC)*, 2015. http://www.bogotobogo.com/RubyOnRails/RubyOnRails_Model_View_Controller_MVC.php [Accessed: 2015-11-03].

[11] *nusskylab/Skylab_ER.png*, 2015. https://github.com/nusskylab/nusskylab/blob/master/docs/2015_2016_Sem1_Report/Images/Skylab_ER.png [Accessed: 2015-11-10].

[12] Nicolas Carlo. *Which git workflow for my project?*, 2013. http://nicoespeon.com/en/2013/08/which-git-workflow-for-my-project/ [Accessed: 2015-11-01].

[13] *Travis CI - Test and Deploy Your Code with Confidence*, 2011. https://travis-ci.org/ [Accessed: 2015-04-05].

[14] *Code Climate: Static analysis from your command line to the cloud.*, 2011. https://codeclimate.com/ [Accessed: 2015-04-05].

[15] Scott Chacon. *GitHub Flow*, 2011. http://scottchacon.com/2011/08/31/github-flow.html [Accessed: 2015-11-01].

[16] P. Raghavan C. D. Manning and H. Schtze. *Introduction to Information Retrieval*, 2008. http://nlp.stanford.edu/IR-book/html/htmledition/inverse-document-frequency-1.html [Accessed: 2016-03-25].

[17] P. Raghavan C. D. Manning and H. Schtze. *A Vector Space Model for Automatic Indexing*, 1975. http://elib.ict.nsc.ru/jspui/bitstream/ICT/1230/1/soltan_10.1.1.107.7453.pdf [Accessed: 2016-03-25].

[18] *TinyMCE 4.2*, 2015. http://www.tinymce.com/presentation/#/ [Accessed: 2015-11-02].

[19] *Ruby on Rails Security Guide*, 2015. http://guides.rubyonrails.org/security.html [Accessed: 2015-11-02].

[20] *What Is Gravatar?*, 2015. https://en.gravatar.com/support/what-is-gravatar/ [Accessed: 2016-06-25].

[21] *plataformatec/devise*, 2015. https://github.com/plataformatec/devise [Accessed: 2015-11-02].

[22] *plataformatec/devise master*, 2015. https://hakiri.io/github/plataformatec/devise/master [Accessed: 2015-11-02].

[23] *Developers' Resources*, 2015. https://openid.nus.edu.sg/about/developers [Accessed: 2015-11-02].

[24] *What is Software Testing?*, 2015. http://istqbexamcertification.com/what-is-a-software-testing/ [Accessed: 2016-03-25].

[25] *RSpec*, 2015. https://relishapp.com/rspec [Accessed: 2016-03-25].

[26] *Acceptance Testing*, 2015. http://www.tutorialspoint.com/software_testing_dictionary/acceptance_testing.htm [Accessed: 2016-04-02].

[27] *Capybara*, 2015. https://github.com/jnicklas/capybara [Accessed: 2016-04-02].

[28] Monica Patrick. *The Purpose of Focus Group Meetings*, 2015. http://smallbusiness.chron.com/purpose-focus-group-meetings-20703.html [Accessed: 2015-11-02].

[29] *Focus group meeting response: adviser*, 2015. https://github.com/nusskylab/nusskylab/blob/master/docs/adviser_focus_group_meeting_suggestions.md [Accessed: 2016-03-25].