Zachry Leadership Program Project Final Report

Zhiren Zhou

April 2022

1   Team Roles

No.	Role	Name	Email
1	Productor Owner	Ruicheng Ni	ruichenn98@tamu.edu
2	Scrum Master	Stone Chen	stonechen@tamu.edu
3	Backend developer	Huancheng Zhou	hczhou@tamu.edu
4	Backend developer	Haotian Xu	hx105@tamu.edu
5	Backend developer	Zhiren Zhou	zzr997good@tamu.edu

2   Summary of the Project

This project, with legacy codes, is about developing a web application that helps a professor to find an appropriate 2-hour time slot to fit all students' schedules. Initially, the customer requested to change the way how students input their schedules. On the web app, there are two user roles: admin and student. Originally, students are allowed to enter their schedules by choosing courses that were scraped from TAMU's course catalog API. Once all students finish adding their schedules, the system would then run algorithms to determine which time slots are available for all students. It also provides some backup choices in which few students need to change their added schedules to find a good time slot. Upon customer's request, the way students input their schedules should be changed. Instead of having students choose courses from course catalog that were scraped from TAMU's website, students need to manually input their busy time slots. Plus, the algorithms to calculate each conflicts with any time slots should be corrected.

Moreover, we discussed several ways that can potentially improve the overall project and implemented new user stories discussed during meetings with the customer. For examples, an admin can enable or disable student entries whenever they want. There is also a new page that shows history of students' actions, and we also added a feature that enables students' permission to edit their own schedules. Besides the bug fixes and new features, we simplified the new term selection mechanism which now requires only one selection.

3 Getting Started

3.1 Clone Repo to Local

To begin the project, the legacy code and project structure needs to be understood. So, you should clone the repository to your local first.

3.1.1 Install Git

First, make sure Git is installed on your computer. To know how to install Git, please visit https://git-scm.com/book/en/v2/Getting-Started-Installing-Git

3.1.2 Fork and Clone the Repo

Login in your GitHub account and visit the link of our repository. Fork our repo to your own account by clicking the 'Fork' botton on the left-top of the page. The link of our GitHub repository is given in **LINKS** part.

On your local PC, connect to GitHub with SSH. For more information, please visit https://docs.github.com/en/authentication/connecting-to-github-with-ssh

Clone the repository to your local by
> git clone #{*your_repo_ssh*}

3.2 Create Local DataBase

All app needs the corresponding data otherwise they can not run successfully and the data is saved in the database. To let app run on your local successfully, you need to create the local database for your app.

3.2.1 Install PostgreSQL

Our project uses PostgreSQL as database server, so make sure it is installed on yoru computer. To know how to install PostgreSQL, please visit https://www.postgresql.org/download/

3.2.2 Create Database for app

Login to the default postgreSQL admin account. You can log in admin account 'postgres' without any password and has all the operation rights to other accounts.
> sudo su postgres

Enter the PostgreSQL service.
> psql

Create a new superuser account whose name is same with your current PC

username (you can check your current PC username by command `>whoami`). My PC username is 'ubuntu', so I will take 'ubuntu' as an example username for the following tutorial. (NOTICE : DO NOT FORGET THE SEMICOLON AT THE END!!!)

```
> CREATE USER ubuntu SUPERUSER;
```

We have to change the password of 'ubuntu' because PostgreSQL does not allow accounts other than 'postgres' to login with an empty password, so you must use 'postgres' account to modify the password of 'ubuntu' account. Here I set 'root' as the new password of 'ubuntu'. You can set any other password as you want.

```
> ALTER ROLE ubuntu WITH PASSWORD 'root';
```

Use 'postgres' account to create a database named 'ubuntu' for account 'ubuntu'.

```
> CREATE DATABASE ubuntu WITH OWNER ubuntu;
```

Log out of 'postgres' account.

```
> \q
```

```
> exit
```

Log in to 'ubuntu' account and specify that the server is running on local. Here you need to enter the password you just set for account 'ubuntu'.

```
> psql -h localhost
```

Create another three databases needed by the app. (NOTICE : DO NOT FORGET THE SEMICOLON AT THE END!!!)

```
> create database zlp_scheduler_dev;
```

```
> create database zlp_scheduler_test;
```

```
> create database zlp_scheduler_prod;
```

Make sure all needed database are created.

```
> \l;
```

Log out of 'ubuntu' account.

```
> \q
```

3.3   Run App on Local

When all preparations are done, you can run the app now.

Make sure that you are in the ../TAMU-ZachryLeadershipScheduler/zlp-scheduler directory.

Install the bundles.

```
> bundle install
```

Create the tables in database.
```
> rake db:migrate
```

Seed the database.
```
> rake db:seed
```

This will create totally five accounts as follows:

role	*username*	*password*
admin	laurenrhaylock@tamu.edu	Temp
student	kyliebrown@tamu.edu	Temp
student	gabihernandez@tamu.edu	Temp
student	tonystark@tamu.edu	Temp
student	scoobydoo@tamu.edu	Temp

Run the server.
```
> rails s
```

If you are using Cloud9, to view the running application, you should click the folloing botton by order: Preview-Preview Running Application-Pop Out Into New Window(which is near the Browser botton)

3.4   Deploy to Heroku

To let others use your app through their browser, you can deploy your app on Heroku.

3.4.1   Install Heroku CLI

To deploy your app on Heroku, make sure Heroku CLI is installed. To know how to install Heroku CLI, please visit https://devcenter.heroku.com/articles/heroku-cli#install-the-heroku-cli

3.4.2   Deploy App onto Heroku

Sign up for a free Heroku account on https://signup.heroku.com/.

Make sure that you are in the ../TAMU-ZachryLeadershipScheduler/zlp-scheduler directory.

Set up ssh keys to securely communicate with Heroku for app deployments.
```
> ssh-keygen -t rsa
```
```
> heroku login -i
```
```
> heroku keys:add
```

Once your keys are set up, you should be able to create an "app container" on Heroku into which you'll deploy your app. Here I set 'example-app' as the name of "app container". You can set any name as you want.

`> heroku create -a example-app`

Add 'example-app' as a remote to your local repository.

`> heroku git:remote -a example-app`

Go back to ../TAMU-ZachryLeadershipScheduler directory.

`> cd ..`

Deploy the app to Heroku.

`> git subtree push --prefix zlp-scheduler heroku main`

Create the database tables and seed the database for app on Heroku

`> heroku run rake db:migrate`

`> heroku run rake db:seed`

Now you can login your heroku account on https://id.heroku.com/login and open your app.

4 User Stories

4.1 Feature : Student Schedule Input

As a student, so that I can input my busy time to form my schedule, I want to have four input fields: day, start time, end time, mandatory to generate my schedule.

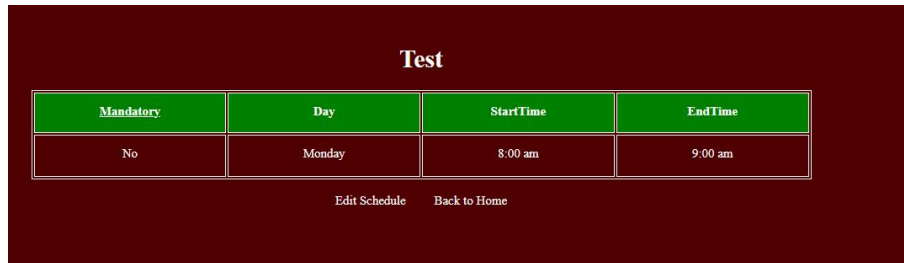**Add Schedule for Test Term**

Schedule Name: _____

Mandatory	Day	StartTime	EndTime
■	Monday ∨	xx:xx	xx:xx
■	Monday ∨	xx:xx	xx:xx
■	Monday ∨	xx:xx	xx:xx
■	Monday ∨	xx:xx	xx:xx
■	Monday ∨	xx:xx	xx:xx
■	Monday ∨	xx:xx	xx:xx
■	Monday ∨	xx:xx	xx:xx

Save Schedule   Cancel

Figure 1: Student Schedule Input

4.2   Feature : Student Schedule Display

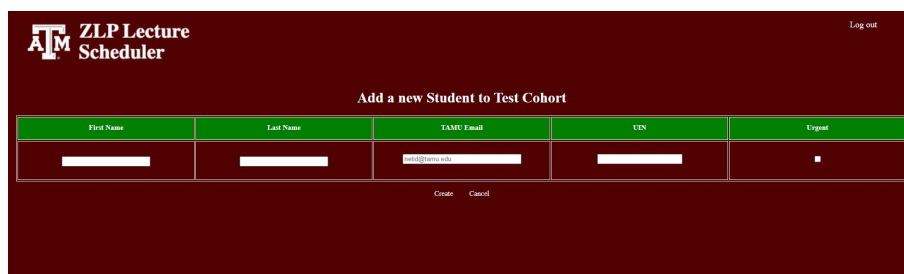As a student, I hope this app can display the schedule I just added in a simple and correct way.

Figure 2: Student Schedule Display

4.3   Feature : Admin adds urgent students

As an admin, so that I can force on the urgent students who need this course for graduation, I want to have the button to mark the student as urgent.

Figure 3: Admin adds urgent students

4.4   Feature : Type in reasons for schedule

As a student, so that I can explain why I am not available for the zlp course during this time slot, I want to explain it to the admin when I add my schedule table.

Figure 4: Type in reasons for schedule

4.5 Feature : Reasons in schedule display

As a student, I hope this app can display the schedule I just added in a simple and correct way.

Test 1				
Mandatory	**Day**	**StartTime**	**EndTime**	**Reason**
Yes	Tuesday	9:35	10:50	ECEN-602-600

Figure 5: Reasons in schedule display

4.6 Feature : Student Input Tooltip

As a student, so that I can get direction about how to input data correctly, I want to have tooltips above each column to generate my schedule. I also hope this tip can restrict the format of some input.

Figure 6: Student Input Tooltip

4.7 Feature : Keep history when editing schedule

As a student, I hope that I can see previously filled information when I try to edit an existing schedule so that I can know which course I have added to this schedule. As a result, duplicate records in a same schedule will be prevented.

Figure 7: Keep history when editing schedule

4.8 Feature : Edit student's urgent status

As an admin, I want to be able to change student's urgent status manually and save the update the the database.

Alli	M	alli@tamu.edu	403828492	false	Edit	Delete

Edit Alli M

First Name	Last Name	TAMU Email	UIN	Urgent
Alli	M	alli@tamu.edu	403828492	☑

Update   Cancel

Alli	M	alli@tamu.edu	403828492	true	Edit	Delete

Figure 8: Edit student's urgent status

4.9   Feature : 30 rows in schedule

As a student, I need enough lines to fill in my weekly schedule.

Figure 9: Edit student's urgent status

4.10   Feature : Add the Save button near schedule name

As a student, if I just need to fill about 5 lines, I don't want to scroll the page until the end so that I can click the 'Save' button. I need a Save button which is available no matter I am at the top or bottom of the page.

Figure 10: Add the Save button near schedule name

4.11 Feature : Readable schedule name when editing an existing schedule

As an admin, I don't want to allow students to change their schedule name arbitrarily when they just want to edit their existing schedule because I don't want to get confused between the new schedule and the edited schedule.

Figure 11: Readable schedule name when editing an existing schedule

4.12 Feature : View student action

As an admin, I can view the actions of each student when they create, edit or delete a schedule which is helpful for me to find out the progress.

Figure 12: View student action

4.13 Feature : Straightforward time slot conflict

As an admin, I hope I can differentiate between the common time slots which just conflict with freshmen's schedules and those special time slots which conflict with urgent senior students' schedules. Because even if freshmen fail to choose the zlp course they have more chances to choose in the next few semesters. But if a senior student can not choose the zlp course, they can not graduate successfully.

Conflicted Time Slots			
Day	**Time**	**Cost**	**Option**
Thursday	12:00 - 14:00	210	Details
Thursday	12:15 - 14:15	210	Details
Monday	14:45 - 16:45	490 ▲	Details
Monday	15:00 - 17:00	490 ▲	Details
Monday	14:15 - 16:15	658 ▲	Details
Monday	14:30 - 16:30	658 ▲	Details

Figure 13: Straightforward time slot conflict

4.14 Feature : Straightforward time slot conflict detail

As an admin, I hope I can easily see into the time slot details, including who is an urgent student, what schedule this conflict comes from. An urgent student is noted with a triangle symbol. Which schedule is marked in the "Reason" column as [1], [2], or [3].

Thursday : 14:30 - 16:30			
Student Name	**Reason**	**Time**	**Mandatory**
Valentina Alarcon	[1] CSCE-625-600	Thursday: 15:55 - 17:10	True
Gabi Hernandez	[1] ECEN-619-600	Thursday: 14:20 - 15:35	True
Kylie Brown ▲	[2] ECEN-619-600	Thursday: 14:20 - 15:35	True
Kylie Brown ▲	[1] ECEN-601-600	Thursday: 14:20 - 15:35	True

Figure 14: Straightforward time slot conflict detail

5 Modification on Legacy Code

Our modification on legacy code is mainly on their UI feature and on their algorithm part. The UI feature modification is listed above. So I will illustrate our modification on their algorithm part in detail.

As an admin, I hope I click the button "run algorithm" and the result can be displayed efficiently and correctly. The old algorithm is incorrect and somewhat counter-intuitive. So we make a new one and the detailed idea is described as follows.

(1) Cost of **non-mandatory** course for one **urgent** student in different schedule:
First schedule cost for one conflict$=5 \cdot 7^2 = 245$
Second schedule cost for one conflict $= 5 \cdot 7^1 = 35$
Third schedule cost for one conflict $= 5 \cdot 7^0 = 5$

(2) Cost of **non-mandatory** course for one **non-urgent** student in different schedule:
First schedule cost for one conflict$= 7^2 = 49$

Second schedule cost for one conflict $= 7^1 = 7$
Third schedule cost for one conflict $= 7^0 = 1$

(3) Cost of **mandatory** course for one **urgent** student in different schedule:
First schedule cost for one conflict$= 2 \cdot 5 \cdot 7^2 = 490$
Second schedule cost for one conflict $= 2 \cdot 5 \cdot 7^1 = 70$
Third schedule cost for one conflict $= 2 \cdot 5 \cdot 7^0 = 10$

(4) Cost of **mandatory** course for one **non-urgent** student in different schedule:
First schedule cost for one conflict$= 2 \cdot 7^2 = 98$
Second schedule cost for one conflict $= 2 \cdot 7^1 = 14$
Third schedule cost for one conflict $= 2 \cdot 7^0 = 2$

Generally speaking,

(a) For any urgent students, the cost is 5 times higher than non-urgent students.

(b) For any mandatory courses, the cost is 2 times higher than non-mandatory courses.

(c) The cost of the time slot after 3 o'clock will increase exponentially until it reaches 1500.

(d) Students are allowed to input three schedules in total. The first one they input is the one they deem the most important; the third one, the least. Hence, the cost for the first schedule is 7^2, the second 7^1, the third 7^0. The reason is that we need to make sure that the cost of three courses from the second or third schedule will not exceed that of any courses from the first schedule.

(e) If the first schedule encounters no conflicts at all, there is no need to display the conflicts due to the second or third schedule.

6   Iterations

Total Points Completed:200.

6.1   Iteration 0

We met with our client for the first time and we talk about his needs on this app. We demo the legacy project and find some bugs. We listed preliminary user stories. We spent most of the time to understand the legacy code.

Points Completed In Iteration0: 0.

6.2 Iteration 1

In this iteration we changed the UI feature of the project to satisfy the needs of our customer. In more detail, we worked on the features as following **Student Schedule Input**, **Student Schedule Display**, **Admin adds urgent students** and **Student Input Tooltip**.

Points Completed In Iteration1: 50.

6.3 Iteration 2

In this iteration we changed the UI feature of the project to satisfy the needs of our customer. In more detail, we worked on the features as following **Type in reasons for schedule**, **Reasons in schedule display**.

Points Completed In Iteration2: 40.

6.4 Iteration 3

In this iteration we realized the data transfer between the frontend and backend. In more detail, we **modified the database** and worked on the features as following **Edit student's urgent status**, **View student action**, **Readable schedule name when editing an existing schedule**,

Points Completed In Iteration3: 40.

6.5 Iteration 4

In this iteration we modified the original algorithm and changed some UI features. In more detail, we **improved the algorithm** and worked on the features as following **Keep history when editing schedule**, **30 rows in schedule**, **Add the Save button near schedule name**, **Readable schedule name when editing an existing schedulee**, **Straightforward time slot conflict**, **Straightforward time slot conflict detail**.

Points Completed In Iteration4: 70.

7 Customer Meetings

7.1 2/24/2022 4:30 - 5:00 pm

In our first meeting, we knew our customer's needs and asked him for the legacy code. We listed preliminary user stories.

7.2   3/3/2022 4:30pm - 5:30 pm

In this meeting, we informed the customer what we would do in iteration1 and asked him for his suggestion.

7.3   4/7/2022 4:30pm - 5:30 pm

We demoed the UI feature changes we made in iteration1 to customer and got a positive feedback. We informed the customer what we would do in iteration2.

7.4   4/14/2022 4:30pm - 5:30 pm

We demoed the work we made in iteration2 to customer and got a positive feedback. We informed the customer that we would modify the database in iteration3.

7.5   4/21/2022 4:30pm - 5:30 pm

We demoed the work we made in iteration3 to customer and got a positive feedback. We also showed the bugs in old algorithm to our customer. We informed the customer that we would improve the old algorithm iteration4.

7.6   4/29/2022 4:30pm - 5:30 pm

We demoed the final project version to customer and and got a positive feedback. He said he likes it.

Customer Meeting Record Link: https://drive.google.com/drive/folders/1JcwLaFLOKsFWCnyM9lc8SdKBnCPRUUNz

8   BDD+TDD

The legacy project had 84.6% test coverage. In order to satisfy the customer needs. We have to do huge modification on the origin UI feature. We also improved the old algorithm. With every feature we implemented we made sure it was covered with behavioral tests. And at the end, we refactored the legacy code and removed unnecessary parts that were not covered by any testing.
Since there were already written step definitions, we relied mostly on behavioral testing instead of unit testing. We implemented our cucumber tests without writing too many new step definitions.

9   Configuration Management

Our repository has totally 8 branches.
Three branches:**UIfeature**, **featureRemoveUI**, **feature_NewColumnForReason** are mainly about UI feature changes which are just about the frontend.

Two branches: **new**, **savenewfeaturetodb** are mainly about the modification on the database.

One branch: **bug_action_fixed** is mainly about the fixation on a bug which exists in iteration3.

algorithm is mainly about the modification we made on the legacy algorithm. It is also the main work we done in iteration4.

In each iteration, we built some new branches and worked on them. When all the jobs were done, we did test to these branches. If they passes the test, we merged them into the branch **main**.

The mainly job we done after the iteration4 was to fix some little bugs. So we did not add any new branches, instead we directly worked on the branch **main**. Through such configuration management, every one in this team starts being familiar with the GitHub team project cooperation process. This is a very important thing we learned through this project.

10   Tools and Gems

We inherited the project from the previous two groups, so we mostly use the same tools and gems. We use cucumber-rails, capybara, rspec-rails, and database_cleaner for common ruby-on-rails gems. We also used factory_bot_rails, faker, and selenium-webdriver in the testing environment.

We had just one change to the gems. We specify the ruby version to 2.7.2 to avoid many unnecessary problem.

For the tools, we followed the last group to use CodeClimate and SimpleCov to collect data about any smells we had in our code and the test coverage of our code.

11   Links

Github: https://github.com/RuichenNi/TAMU-ZachryLeadershipScheduler
Pivotal Tracker: https://www.pivotaltracker.com/reports/v2/projects/2555575/overview
Heroku: https://tamu-zlp.herokuapp.com/
Poster: https://github.com/RuichenNi/TAMU-ZachryLeadershipScheduler/blob/main/documentation/Spring2022/POSTER.pptx
Poster Video: https://youtu.be/fy3vdsKwNg4
Demo Video: https://youtu.be/fy3vdsKwNg4