CS 130: Software Engineering

Professor: Miryung Kim

TA: Sneha Shankar

ZOW

Part A Report

Team:

Naim Ayat, Christopher Aziz, Austin Berke,
Dmitri Brereton, Reinaldo Daniswara, Aidan Wolk

October 25, 2018

Project URL: https://github.com/NaimAyat/UCLA_CS130_Project

Motivation

Student organizations on campus handle recruiting on a regular basis. Currently, they use a mix of Google Forms, Docs and Spreadsheets to collect data and notes about different candidates. From our first-hand experience, this process is very painful and time consuming. Having to coordinate between these different documents can lead to much confusion. Additionally, scoring candidates becomes very difficult. Evaluating candidates to come up with a score involves working with so many different documents that the process becomes a hassle. Furthermore, organizations are limited by Google Sheets and have to manually calculate certain statistics for their scoring.

We propose Zow to solve all of these issues. Zow allows recruiters to create postings for their open positions. They will be able to make custom forms to collect all of the data they need to screen candidates. In addition to that, student organizations will be able to score each applicant and see the overall score ranking from the lowest to highest or vice versa. This way, student organizations can easily distinguish between strong and weak candidates. Therefore, the sorting process can be done in much less time. Recruiters will be able to automatically schedule interviews, and go through multiple rounds of scoring, so that they can easily get to a final decision. Lastly, there will be a summary view that lets recruiters see the overall statistics for this applicant pool, allowing them to identify trends in candidates.

Features and Requirements

Application Postings

Recruiter accounts can have more than one posting. A posting for a specific job will have its own associated forms, scoring and interview scheduling. Applicants can only see public forms associated with a posting.

Form Building

Recruiters can create an application form for their posting. They will do this through a simple and intuitive form building interface. They will be able to add the following types of fields to their form:

- Short Text
- Long Text
- Phone Number (verified)
- Email (verified)
- Dropdown
- Checkboxes (multiple selection)
- Radio Buttons (single selection)
- File upload

These forms can be private while created, and then posted publicly on completion. After publication, the form can be disabled from accepting future applicants.

Applicant Scoring

Recruiters will be able to open specific candidate responses and assign scores to them. In this view, they'll see all the relevant information about this candidate that they need to score them. The scores will be out of five stars. They will also be able to add textual notes about users.

Multiple recruiters will be able to give scores on a single candidate. The average of all their scores will be calculated and used later when selecting the final candidates.

There can be multiple scoring rounds. For example, the first scoring round can be an application screen while the second scoring round is based on the interview. The recruiter will be able to end a scoring round and create a new one at any time.

At any point, an applicant can be explicitly accepted or denied. If accepted or denied, they will not be shown to any recruiters in the scoring interface.

Summary View

After recruiters receive all the responses and scores from candidates, recruiters can browse through candidates' responses and their scores. Based on scores that were given by recruiters themself or other candidates, our platform will generate some basic statistics such as average score, highest score, lowest score, and many more. In addition to some statistics, it will also provide some data visualizations such as charts and graphs to help recruiters have a broad summary view among all candidates. Recruiters will be able to filters candidates using various statistics such as average store, etc. to accept or reject applicants at any point. They will also be able to gain insights about who is a strong accept, strong decline, and borderline, based on rankings and the target number of candidates they want to accept.

Interview Scheduling

After screening out the first round of candidates, recruiters will be able to easily schedule interviews with their desired candidates. A recruiter can select the candidates that they want to interview, and send them link to them via email. When candidates go to a unique link included in the email, they will be able to sign up for an interviewing slot. Recruiters can list available interview slots for the candidates to select from.

Our system will be keep track of which interview slots are taken, and it will adjust the options on the candidate side based on what slots are still open.

Automatic Emails

The recruiting process usually involves sending many different emails which can become confusing and error prone. Our system will automate email sending based on different events. We will automatically send a scheduling email when a candidate is selected for an interview, and a reminder email nearer to the interview. We will also automatically send them an email if there is a new form they need to fill out, or an approaching deadline.

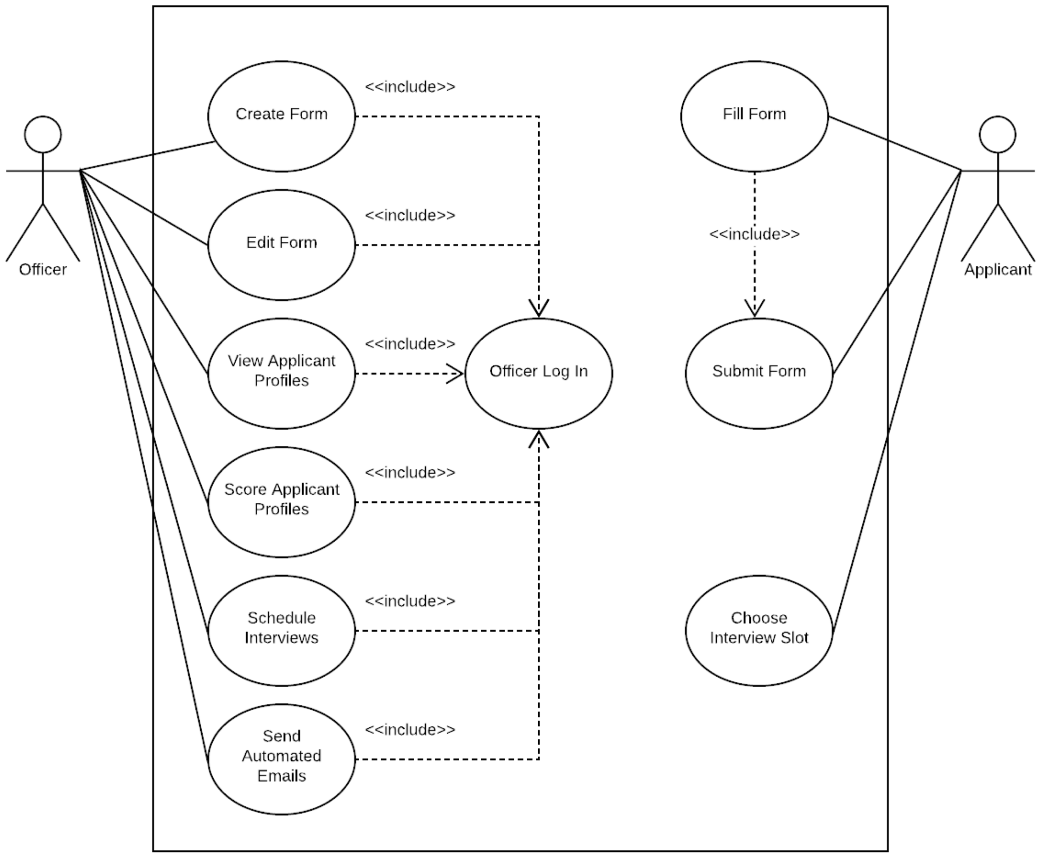


Figure 1: Zow use case diagram. The two types of users, Recruiters and Applicants, differ in that only Recruiters have a login portal. Applicants simply access forms and interview schedulers via unique URLs sent by Recruiters.

Usage Scenarios

User Story 1: Recruiting Officer

I am the head of recruiting for a student organization at UCLA. In the past, our applicant vetting process has been decentralized and disorganized. I've had to share candidate information with other club officers via several messaging apps, and each officer's feedback came back to me through a different channel. Beyond the fact that manually aggregating this feedback was time-consuming, it was difficult to come to a quantitative, data-driven consensus decision on each candidate. In the future, Zow will enable me and each of our officers to log into a centralized portal wherein each candidate's application may be viewed and scored. Based on each candidate's combined score, Zow will generate an automatic ranking of candidates that reflects our consensus.

User Story 2: Secretary

As the secretary of my student organization, I typically create Google Forms to collect applicant information. I have to export the data and then condense it into a readable spreadsheet so my club officers can easily review each applicant. Zow will eliminate this step entirely: I will simply be able to create a Zow form, and easy-to-read profiles will automatically be generated for each applicant. These profiles can then be viewed and edited by my club officers when they log into their officer accounts.

User Story 3: Applicant

I'm hoping to earn a spot on UCLA's Pre-Med Club board. I received an email from a current officer containing a link to a Zow form. After submitting my contact information, essay responses, and resume, I receive confirmation that my Zow form was received. A few days later, I get another email with a Zow Interview Scheduling link. I reserve the 3:30pm Thursday time slot and receive a confirmation message. A few days after my interview, I receive an email notifying me that I got the position.

User Story 4: Interviewer

I am interviewing ten candidates for a student organization board position. Since my secretary used Zow to schedule the interviews, I can log into my Zow account to view the profile of each candidate as I interview them. We also asked each applicant to attach a resume in the initial Zow Form, so I already have the interviewee's resume open on my laptop screen before we begin speaking. As the interviewee answers my questions, I type notes in a textbox on his profile. These notes will be saved and made accessible to all other club officers when they view the candidate's profile.

User Story 5: Decision-Maker

I am the president of a student organization. Ten candidates were interviewed by my vice president, and he made notes on each of their Zow profiles. The other club officers also scored each candidate in Zow based the vice president's notes. Now, when I log into my Zow account, I see a ranking of the ten candidates based on my fellow officers' scores. I review each candidate's profile and add my own scores. When I'm done, I notice that the ranking changed slightly after my scores were incorporated into each candidate's average score. I click on the top applicant's profile and send him an acceptance email. With a single click, Zow also allows me to send an automated rejection email to the bottom nine candidates.

Diagrams

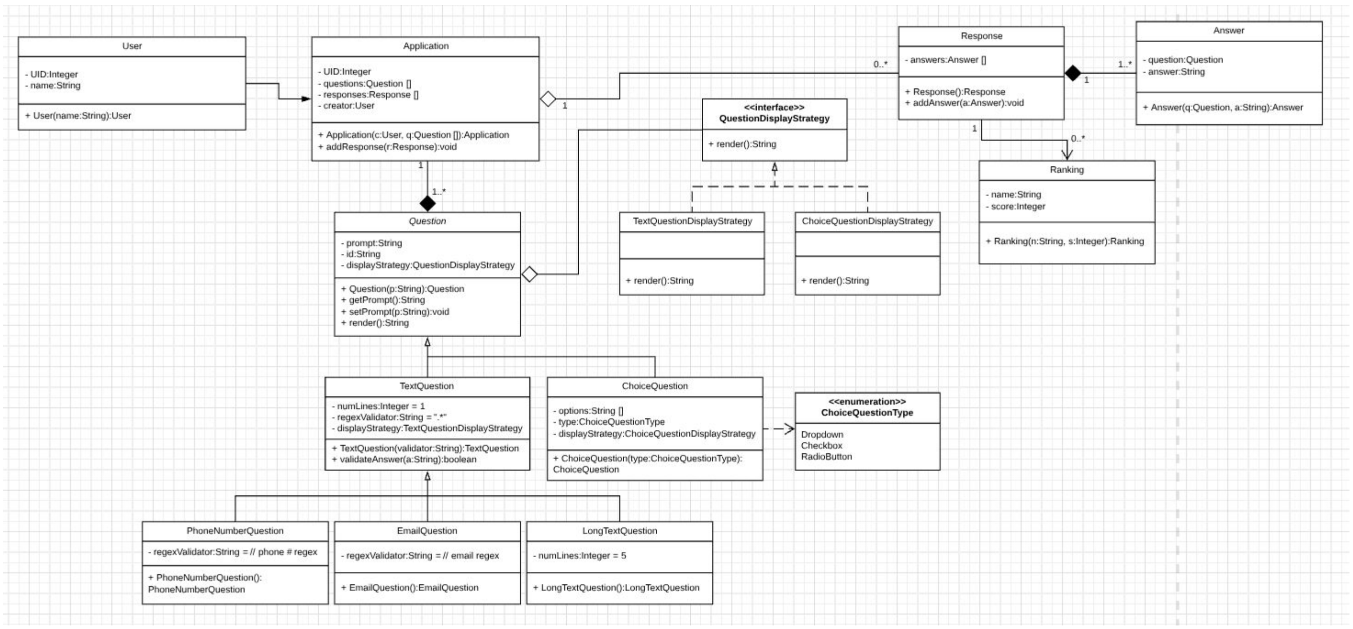
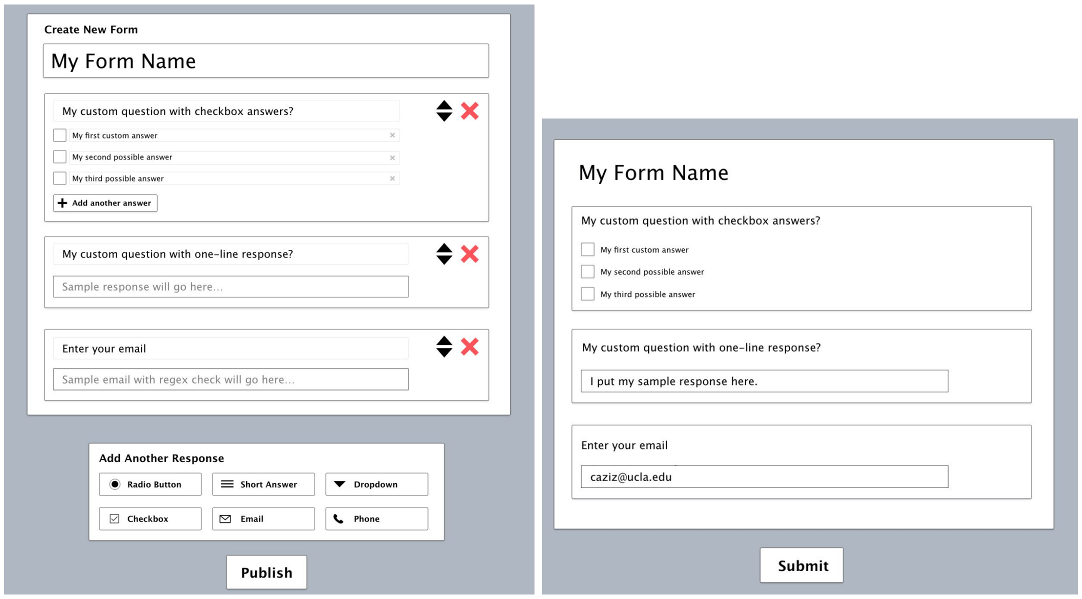

User
- UID:Integer
- name:String
+ User(name:String):User
Application
- UID:Integer
- questions:Question []
- responses:Response []
- creator:User
+ Application(c:User, q:Question []):Application
+ addResponse(r:Response):void
1
1..*
Question
- prompt:String
- id:String
- displayStrategy:QuestionDisplayStrategy
+ Question(p:String):Question
+ getPrompt():String
+ setPrompt(p:String):void
+ render():String
<<interface>>
QuestionDisplayStrategy
+ render():String
TextQuestionDisplayStrategy
+ render():String
ChoiceQuestionDisplayStrategy
+ render():String
Response
- answers:Answer []
+ Response():Response
+ addAnswer(a:Answer):void
0..*
1
Answer
- question:Question
- answer:String
+ Answer(q:Question, a:String):Answer
1..*
1
1
0..*
Ranking
- name:String
- score:Integer
+ Ranking(n:String, s:Integer):Ranking
TextQuestion
- numLines:Integer = 1
- regexValidator:String = ".*"
- displayStrategy:TextQuestionDisplayStrategy
+ TextQuestion(validator:String):TextQuestion
+ validateAnswer(a:String):boolean
ChoiceQuestion
- options:String []
- type:ChoiceQuestionType
- displayStrategy:ChoiceQuestionDisplayStrategy
+ ChoiceQuestion(type:ChoiceQuestionType):
ChoiceQuestion
<<enumeration>>
ChoiceQuestionType
Dropdown
Checkbox
RadioButton
PhoneNumberQuestion
- regexValidator:String = // phone # regex
+ PhoneNumberQuestion():
PhoneNumberQuestion
EmailQuestion
- regexValidator:String = // email regex
+ EmailQuestion():EmailQuestion
LongTextQuestion
- numLines:Integer = 5
+ LongTextQuestion():LongTextQuestion

Figure 2: Zow class diagram. The Application class will be the focus of our implementation. Subject to change.


Create New Form
My Form Name
My custom question with checkbox answers?
My first custom answer
My second possible answer
My third possible answer
Add another answer
My custom question with one-line response?
Sample response will go here...
Enter your email
Sample email with regex check will go here...
Add Another Response
Radio Button
Short Answer
Dropdown
Checkbox
Email
Phone
Publish
My Form Name
My custom question with checkbox answers?
My first custom answer
My second possible answer
My third possible answer
My custom question with one-line response?
Enter your email
Submit

I put my sample response here.

caziz@ucla.edu

Figure 3: Zow form creation for recruiters (left) features customizable and recordable responses. For applicants, Zow offers a clean form submission interface (right) that displays the custom questions.

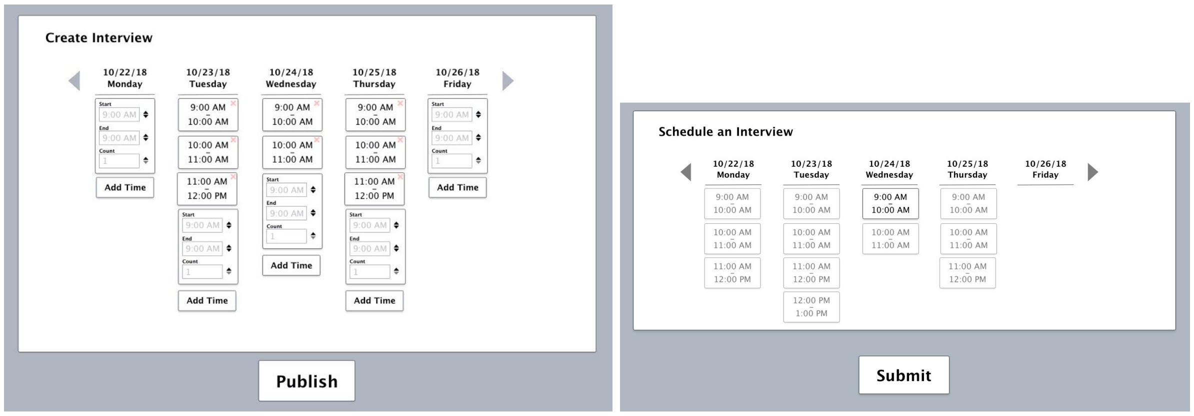


Figure 4: Zow recruiters can create custom interviews with modifiable lengths and times (left). The availability of each interview is displayed to the applicant in an automatic scheduler interface (right).

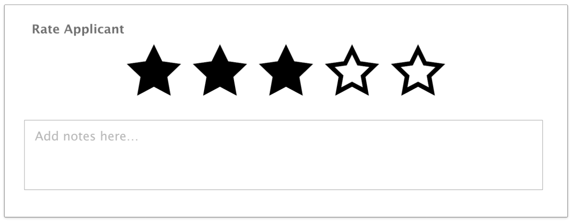
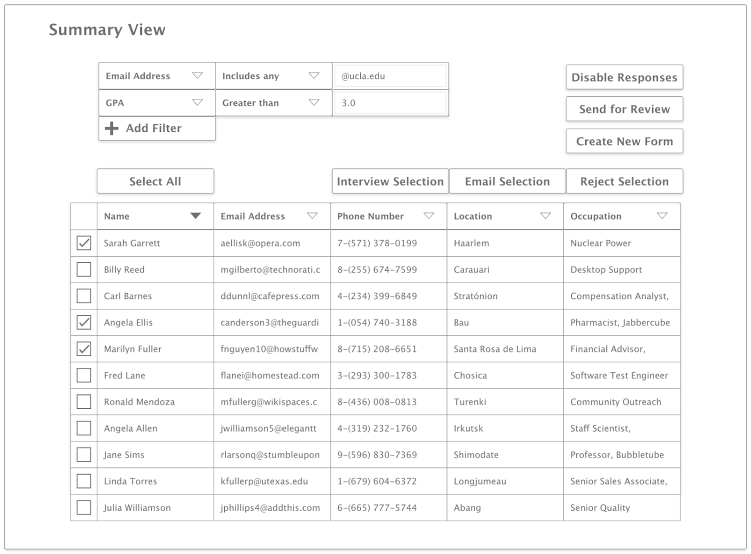


Figure 5: Zow scoring form with 0-5 rating and optional notes that are displayed when recruiters are requested to review an applicant.



Summary View

Email Address	Includes any	@ucla.edu
GPA	Greater than	3.0
+ Add Filter		

Disable Responses | Send for Review | Create New Form

Select All | Interview Selection | Email Selection | Reject Selection

	Name	Email Address	Phone Number	Location	Occupation
✓	Sarah Garrett	aellisk@opera.com	7-(571) 378-0199	Haarlem	Nuclear Power
☐	Billy Reed	mgilberto@technorati.c	8-(255) 674-7599	Carauari	Desktop Support
☐	Carl Barnes	ddunnl@cafepress.com	4-(234) 399-6849	Stratónion	Compensation Analyst,
✓	Angela Ellis	canderson3@theguardi	1-(054) 740-3188	Bau	Pharmacist, Jabbercube
✓	Marilyn Fuller	fnguyen10@howstuffw	8-(715) 208-6651	Santa Rosa de Lima	Financial Advisor,
☐	Fred Lane	flanei@homestead.com	3-(293) 300-1783	Chosica	Software Test Engineer
☐	Ronald Mendoza	mfullerg@wikispaces.c	8-(436) 008-0813	Turenki	Community Outreach
☐	Angela Allen	jwilliamson5@elegantt	4-(319) 232-1760	Irkutsk	Staff Scientist,
☐	Jane Sims	rlarsonq@stumbleupon	9-(596) 830-7369	Shimodate	Professor, Bubbletube
☐	Linda Torres	kfullerp@utexas.edu	1-(679) 604-6372	Longjumeau	Senior Sales Associate,
☐	Julia Williamson	jphillips4@addthis.com	6-(665) 777-5744	Abang	Senior Quality

Figure 6: Summary view which centralizes all information about applicants. All applicants that have submitted a form sent by the recruiter appear on this interface. Information regarding scheduled interviews, other forms, and scores received from recruiters also appear on this screen. Advanced filtering can be used to process and move applicants to additional stages of an admission process.

Feasibility

While some features of Zow exist independently in other applications, such as the form builder in Google Forms, Zow will be the first web platform to centralize all features into a single unified interface. The interface for recruiters to collaboratively score and review applicants is new and novel and motivated by features we wished existed when previously applying to and recruiting for clubs at UCLA.

This project is feasible as all members of the group have experience in web development and JavaScript. We will use Node.js as the primary technology for the backend of the Zow website and the React framework for the frontend, and we have developers with strong experience in both. Using GraphQL will encourage standardization of the communication between frontend and backend. Furthermore, Zow has no third party API dependencies, and thus we control the entire system from end-to-end. Essentially, Zow follows the standard website model - frontend, backend, and database. It is also split cleanly into different features: form creation and rendering, applicant management, and interview scheduling.

Capability

Aidan interned in Summer 2018 doing backend web development for Uber using Go. He has designed and developed a website for the CHIPS lab at UCLA using server-side rendering with a backend written in JavaScript on top of the Node.js and Express.js frameworks and a PostgreSQL database. This website is hosted on AWS and includes secure user authentication, an event registration system, and an emailing service. He works on the UCLA-exclusive dating website BruinMeet as a full-stack developer, using the React framework on the frontend and Node/Express on backend. This site utilizes a MongoDB database. For this project Aidan will be working on both frontend and backend, bringing his web development experience. For part A of the project, Aidan helped write the “Features / Requirements” and “Feasibility” sections and worked on the presentation.

Naim Ayat is a 3rd year computer science student and product management intern at Western Digital, where he oversees the development of the SanDisk Memory Zone iPhone app. He also has professional software engineering experience at ZipRecruiter, where he worked with a Perl backend and created a JavaScript web integration linking the company’s bug tracking software to its IT alert system. Naim will contribute to the full stack engineering and overall product vision of Zow. For part A of the project, Naim wrote the usage scenarios and created the use case diagram.

Christopher Aziz is a 3rd year computer science student. He actively develops iOS applications using Swift and he has published apps to the App Store such as OneThreeFive, a news app for people on the go. In Summer 2018, Christopher interned at Adobe where he contributed to the Adobe XD, a UI/UX design application. At Adobe, he solely implemented a highly-requested text-transformation feature which required interfacing between the C++ backend, Node.js, and Cocoa/UWP frameworks. He currently works on BruinMeet as a front-end developer using the React framework. His experience in web development and working with large software engineering teams has prepared him to contribute to all layers of the Zow application. For part A of the project, Christopher designed mockups for the application and contributed to the “Features / Requirements” section.

Austin Berke is a 4th year computer science student. He spent Summer 2017 at Ticketmaster doing full-stack web development for an event discovery portal using Node.js, React, and several other infrastructure related technologies. In Summer 2018, Austin interned at Workday, where he built and modified North American Payroll applications using a proprietary Java framework. In his spare time, he has worked on various personal web projects, mostly using Node.js hosted on AWS instances. For part A, Austin designed the class model as detailed in the UML class diagram.

Dmitri Brereton is a 4th year computer science student. He has down two software engineering internships and one product management internship. In 2016, he worked on big data with Java, Hbase and Hadoop. In 2017, he was a full stack engineer working with Node.js, Closure, DynamoDB, Spark, AWS RDS and SQL. In 2018, he did product management and learned more about user research, product design and business strategy. He is well equipped to contribute to all aspects of Zow, from the technical architecture to the overall design direction. For part A of the project, he contributed to the “Features / Requirements” section.

Reinaldo Daniswara is a 4th year computer science student. He spent Summer 2017 interning as a Business Intelligence Analyst at Go-jek. He dealt with a lot of data warehousing and big data projects, as well as provided insight and analysis to various teams and division at Go-Jek. He also maximized the functionality of Go-Jek’s weather API system to be implemented in slack-bot. He earned a diploma in front end web development from Alison Online Course and this experience will be used to help Zow in building an interactive engaging platform. For part A, Reinaldo contributed in the “Motivation” and “Features / Requirements” section.