Table of Contents

Project Description **1**

Our Process **2**

Requirements & Specifications **3**

Architecture & Design **5**

Backend 5

Website 6

Android 11

Reflections & Lessons Learned **17**

Project Description

When going home from school, many students drive back with empty seats, and many students without cars are forced to pay expensive fares on slow and crowded buses. Drivers looking to split costs and riders looking for a way home have no convenient way of connecting with each other. Enter Chartr: a new way to ride. Chartr is a platform aiming to engage drivers and riders to save money and time.

From our experience at UIUC, we know that traveling home can be a long and expensive process. For students who take the bus, round-trip tickets start at $30. For those with cars, the travel may be faster and more comfortable, but can often cost $30 or more for gas depending on fuel efficiency. The students taking the bus have to deal with crowded spaces and a longer, inflexible journey; the students driving their cars are often hit with the full cost of driving, despite having plenty of extra room.

Chartr aims to eliminate this inefficiency. By allowing drivers and passengers to connect on our platform, we enable both parties to save money and have a better time travelling. Part of the excitement of this project is being able to help the student community in a tangible way, especially in times where many students are looking for alternative options to bus companies with questionable ethics.

In its current iteration, Chartr supports both Web and Android access to the platform, with nearly identical functionality existing on both platforms. Users can sign up for the service using an email, name, phone number, birthdate, and password. After authenticating their account by clicking on an activation link sent to their email, users can view lists of trips that other users have posted. If they wish to be a passenger in an existing trip, they can click on a trip to request to be a rider. The creator of the trip can subsequently accept their request and add them to the trip. Similarly, if the user wishes to post a trip, they can do so by simply filling out information such as the starting location, ending location, price, and other miscellaneous fields to create a trip in our database.

Chartr is also exciting due to its future expansion possibilities. While the initial rollout would be focused on student travel to and from school, the concept is extremely generalizable to any situation where someone wishes to drive somewhere while splitting the costs. Additionally, there is a social component that could allow us to tailor the driver/passenger pairing based on the preferences of each (i.e., if they would prefer music, chatting, or silence in the car; if they're interested in sports, movies, or food; etc.).

Our Process

Our goal for the team process for Chartr is to take the best parts of XP such as iterative development and testing that worked well last semester while eliminating the development bottlenecks and tedium that does not contribute to the overall health of codebase or project. We continued with the agile methodology of iterative development, with the goal of always having working prototypes for all platforms at the end of each iteration. This helped ensure progress on the project and give a foundation on which to build new features each week. We hope by constraining development and setting deadlines in such a manner our team can prevent the project from falling behind schedule and failing. We continued employing comprehensive test coverage as part of our new process. Having a regression testing suite is extremely valuable, as it encourages maintainable code. To ensure thorough testing of all implemented features, we required a minimum code coverage of 80% on the website code, and 40% total coverage on Android.

We did, however, eliminate the processes in XP that we collectively felt were ill-fitting for an academic project like Chartr. We eliminated Pair Programming, as scheduling concerns would make such a process inefficient and tedious. Instead we will move to a more general buddy system where we split the larger groups into smaller groups, and each member in the group used their teammates as a resource for completing the assigned tasks. We hoped this would encourage collaboration and code review while maintaining independence and accountability for every member of the group. We also did not enforce Test Driven Development. We recognized the importance of thoroughly testing specifications for all code that is written, but we understood that we could accomplish the same goal as TDD by enforcing code coverage standards and code reviews. While in theory writing tests before the code would have been a great exercise, the integrated development environments that we use expect method calls to already exist before calling them, which makes writing test for non-existent code more challenging. Additionally, without knowing a complete specification of every behavior, method signatures and failure modes often changed as the team wrote and refactored the code, which would have required additional work if the teammates have already written the tests. To prevent this inefficiency, we allowed team members to write comprehensive tests after the feature has already been implemented.

Much of the rest of the Extreme Programming process remained relatively unchanged. We continued to write user stories, although we did augment some stories with notes regarding administrative tasks that are required to begin work on the story. We used a system for continuous integration and encourage team members to commit and integrate code frequently. We also followed an iteration schedule, as prescribed by XP and the CS428 course staff. In the end, the following processes were enforced throughout the project:

1. Master and Dev branches on Github were protected and required reviews before merging Pull Requests
2. Travis-CI provided our Continuous Integration, running tests on every branch and every commit and informing our Slack channel upon any failure
3. Web test coverage needed to be maintained at 80%, Android at 40%
4. Iterations lasted 2 weeks
5. All features must have a user story with a priority, acceptance test, and estimated number of story points
6. Teams were split into Backend, Android, and Web, with each team delegating subtasks and collaborating in development
7. Refactors were encouraged during each iteration to ensure code maintainability

Requirements & Specifications

The following is a comprehensive list of the user stories that were implemented for Chartr. Note that this list only includes the functional user stories, not the maintenance, testing, or tutorial user stories that we included as part of our weekly iteration plans.

1. Log in to mobile app
    a. User can go to a login screen to log in to the app based on the username and password entered when originally setting up their account. This login will then be authenticated, afterwhich the user will be able to access all content saved to their account.
2. Sign up in mobile app
    a. User can sign up for an account in the app on a “sign-up” page by providing first name, last name, email address, username, and password. This will add them to the system and send a confirmation email.
3. View posted trips in mobile app
    a. User can look through a list of trips that drivers have posted. The list can be scrolled and shows the start and end destinations for each trip, the driver’s name, and the price. There may be a picture in the card for visual aesthetic.
4. Search trips in mobile app
    a. User can filter the trips that are displayed in their list by start and end region, drivers, and/or cost.
5. View past trips in mobile app
    a. User can go to a page to view past trips that they have taken. They can view information about the driver, price, and destinations of the trip.
6. Post a trip in mobile app
    a. User can post a trip on a “post trip” page by providing the starting point, the destination, the time and date of travel, the price, and the number seats available.
7. View trip requests in mobile app
    a. User can look through a list of trip requests that potential passengers have posted. The list can be scrolled and shows the starting point, destination, passenger’s name, and proposed price.
8. Post trip request in mobile app
    a. User can post a trip request on a “post request” page by providing the starting point, the destination, the time and date of travel, the proposed price, and the number of seats available.
9. Log in to web app
    a. User can go to a login screen to log in to the app based on the username and password entered when originally setting up their account. This login will then be authenticated, afterwhich the user will be able to access all content saved to their account.
10. Sign up on web app
    a. User can sign up for an account in the app on a “sign-up” page by providing first name, last name, email address, username, and password. This will add them to the system and send a confirmation email.
11. View posted trips on web app
    a. User can look through a list of trips that drivers have posted. The list can be scrolled and shows the start and end destinations for each trip, the driver’s name, and the price. There may be a picture in the card for visual aesthetic.
12. Search trips on web app

    a. User can filter the trips that are displayed in their list by start and end region, drivers, and/or cost.
13. View past trips on web app
    a. User can go to a page to view past trips that they have taken. They can view information about the driver, price, and destinations of the trip.
14. Post a trip on web app
    a. User can post a trip on a “post trip” page by providing the starting point, the destination, the time and date of travel, the price, and the number seats available.
15. View trip requests on web app
    a. User can look through a list of trip requests that potential passengers have posted. The list can be scrolled and shows the starting point, destination, passenger’s name, and proposed price.
16. Post trip request on web app
    a. User can post a trip request on a “post request” page by providing the starting point, the destination, the time and date of travel, the proposed price, and the number of seats available.
17. UI Design for Mobile
    a. Create and iterate on a design for the mobile application as a guide.
18. Delete/Update User on API
    a. Expose endpoints and logic to delete and update user information in the DynamoDB table
19. Post/Get Trips using API
    a. Expose endpoints and logic to create and get trip information from the DynamoDB table
20. UI Design for Web
    a. Create and iterate on a design for the web application as a guide.
21. Search Trip Backend Support
    a. The API should support requests to find and filter trips based on criteria such as the user that posted the trip, the starting location, and the ending location
22. Create Review Lambda Infrastructure
    a. Create a new Review Lambda and expose an API endpoint allowing users to post reviews of other users.
23. View my trips on web app
    a. User can view their posted trips on the web interface
24. View interested riders on web app
    a. User can view interested riders for a trip on the web interface and accept/reject them
25. Request to join trip on web app
    a. User can request to join a trip on the web interface
26. Request to join trip in mobile app
    a. User can request to join a trip in the Android mobile interface
27. View interested riders in mobile app
    a. User can view interested riders for a trip on the Android interface and accept/reject them
28. Trip Information Emails
    a. A driver should receive an email when they accept a rider, and a rider should receive an email when they are accepted by a driver. The email shall contain the other person’s name and phone number, along with the trip date

The following is a list of some of the key use cases for the app. Due to space constraints, we will present the “casual” use cases here:

- Accept/Reject potential rider
    - The driver opens up his or her list of posted trips in either the web or mobile app. The driver then selects the trip they wish to see the riders for, and a list of interested riders for the selected trip is shown. The driver can scroll through the list and accept/reject potential riders. If the driver accepts a rider, that rider is added to the trip. If the driver rejects a rider, that rider is removed from the interested rider list.
- Post a trip
    - The driver decides to go on a trip from point A to point B. The driver opens up the post trip form in the web or mobile app. The driver then inputs the essential information such at starting point, ending point, date and time. The posted event will appear in the driver's list of planned trips and will be visible to all riders who browse through the list of posted trips.
- Search trip posts
    - A rider can enter a destination into the search bar of the application to show a list of available rides going to that destination with some trip details (time, date, and driver rating). The rider can then choose to view more specific details or request to ride the trip.

Architecture & Design

Backend

Each request to the backend will execute in a similar way, as expressed in this general sequence diagram outline:

The backend is hosted on Amazon Web Services and uses the services Cognito for user authentication, DynamoDB for saving the application data, Lambda for the business logic, and

API Gateway to create a serverless API for the web and mobile application to use. We setup the API Gateway to have a single /{proxy} endpoint that delegates the endpoint parsing to a single Lambda function. We wrote the Lambda function in Node.js (Javascript) and utilized its body-parser library to parse the requests. Then, we used Express, a web application framework that allowed us to easily create new endpoints. The Lamba connects to the DynamoDB, which is a NoSQL database that uses tables to organize the data. We had a Users table and a Trips table, and group the endpoints based on which table was primarily used. The endpoints for the backend are specified below:

Users:

- POST /user
  - Registers a new user
- GET /user/{email}
  - Gets a user by their email
- GET /user/id/{uid}
  - Gets a user by their uid
- DELETE /user/{uid}

Trips:

- POST /user/{uid}/trip
  - Creates a new trip with the user set as the driver
- GET /trip/{tid}
  - Gets a trip by its tid
- GET /trip/current
  - Gets all current trips (i.e. all trips are start after the current date and time)
- GET /user/{uid}/trip
  - Gets all trips for a user
- GET /user/{uid}/trip/{status}
  - Gets all trips for a user filtered by status (i.e. “driving”, “pending”, “riding”)
- PUT /user/{uid}/trip/{tid}/{status}
  - Updates a user’s status (i.e. “pending”, “riding”, “declined”) for a specified trip
- DELETE /trip/{tid}
  - Deletes a trip by its tid

Testing is contained within the `/chartr-lambda/test` directory of the Chartr repository. Tests will invoke each endpoint with a mocked AWS service that allows us to do unit testing for each of the endpoints. We used `aws-sdk-mock` and `mocha` in order to do the testing, as they were the most common frameworks used by people writing AWS Lambdas.

Clearly the choice of cloud computing provider (AWS in this case) heavily influences the design of a serverless architecture like the Chartr backend. Once a provider is selected, the developers must use that provider’s available services to achieve the desired goal. In our case, we had to use API Gateway, Lambda, DynamoDB, and Cognito. Each of these services must be configured to work with the other following AWS practices. If we had used another cloud computing provider, the design and setup of the backend would completely change.

Website

Below is a UML class diagram outlining the major classes of the Chartr web interface:

Below are UML sequence diagrams for the main actions of the Chartr web interface:

Login

Sign Up

View Current Trips

Post Trips

Join Trips

Accept Riders

Reject Riders

Logout

The TypeScript files located in the `src/app` directory drive the web interfaces' functionality. The other files in the repository consist of various documentation and configuration files that most developers won't need to update. Below is an overview of the `src/app` files:

`component/`

The component directory contains the Angular components making up the web interface. Each component consists of four files: the TypeScript file, the CSS file, the HTML file, and the test file.

- `home/`
  - The home component is displayed to the user after he or she logs in. It displays a welcome message and shows trips to the user
- `index/`
  - The index component is the main homepage of the Chartr website. From here a user can sign up or log in
- `login/`
  - The login component displays the login form to the user and attempts authentication when the form is submitted
- `logout/`
  - The logout component exists solely to log a user out
- `my-trips/`
  - The my trips component displays a user's posted/pending/confirmed trips
- `past-trips/`
  - The past trips component displays a user's past trips
- `post-trip/`
  - The post trip component displays the post trip form to the user and creates a trip in the database when the form is submitted
- `search/`
  - The search component allows a user to search current trips. It displays a form to the user with various filters and displays matching trips when the form is submitted
- `signup/`
  - The signup component displays the sign up form to the user and creates a new user in the database when the form is submitted
- `trips/`
  - The trips component displays current trips not associated with the user
- `ui/`
  - The `ui/` directory consists of various UI components making up Chartr's web interface

`model/`

The model directory contains the user and trip models used in the web interface. These files are simple and self-explanatory.

`service/`

The service directory contains all the services responsible for interfacing with the Chartr backend and other external systems. The components employ these services to interact with the backend. Each service consists of a TypeScript file and a test file.

- `AuthenticationService`
  - The authentication service is responsible for retrieving and storing the current authenticated Cognito user
- `CognitoService`
  - The cognito service enables the web interface to retrieve the Cognito user pool
- `EmailService`
  - The email service interfaces with the backend API to send notification emails
- `GeoService`
  - The geo service interfaces with the Google Maps API to reverse geocode latitude and longitude coordinates

- `LoginService`
    - The login service provides the functions to authenticate a user and log a user out
- `SignupService`
    - The signup service provides the functions to register a new user
- `TripService`
    - The trip service deals with all trip related backend requests
- `UserService`
    - The user service deals with all user related backend requests

`testing/`

The testing directory contains all the stub classes used in testing. Similar to the `model/` directory, these files are simple and self-explanatory.

The other files inside the `src/app` directory define the Angular application and are rarely modified by developers. Instead, the Angular CLI modifies these files automatically when components/services are generated.

Clearly, the choice of framework (Angular in this case) heavily influences the design of the system. Most frameworks have specific ways they want files laid out and employ various concepts to provide a clean development experience to end users. In order to successfully use a framework, developers must adopt the framework's methodology and design their system accordingly. In the case of Chartr Web, the system consists of Angular services and components that all work together to create the final product. If Chartr Web had been built on a different framework, the design of the system would be much different.

Android

Below we present the key class diagrams. Due to the extensive number of components in any Android project, we cannot present the entirety of project's class diagram all in one. Instead, we will present the key packages as individual UML class diagrams

Activity Package Class Diagram

Fragment Package Class Diagram

Adapter Package Class Diagram

Below, we present the sequence diagrams for key sections of the code, with each image captioned with the function of the sequence:

The main Java files reside in the `app/src/main/java/com/example/mac/chartr` directory, while the tests mainly reside in the `app/src/test/java/com/example/mac/chartr` directory. Both directories have the same structure described below, but the “test” directory has all of the “`ClassNameTest`” files. We will give an overview of the purpose of each directory, and the primary function of each file.

`activities/`

The activities directory contains all the services responsible for interfacing with the Chartr backend and other external systems. They contain the various fragments and should be focused on one “activity” that the user might want to do.

- `ConfirmRegisterActivity`
    - Confirms that the user has successfully created an account or disallows registration
- `LoginActivity`
    - Allows the user to sign up for an account and to login to the app
- `MainActivity`

    - Sets up most of the visible features of the application, including the tabbed view for the search, trips, requests, and profile features.
- `PostTripActivity`
    - Post trip allows the user to post trips that can be viewed by riders
- `RegisterActivity`
    - Verifies validity of user inputs and attempts to create an account if valid
- `TripDetailActivity`
    - Displays a page with all the information related to a current trip.

`adapters/`

The adapters are responsible for displaying trips in fragment/activities by inflating the right layouts. They allow for reuse of key features of the app, such as displaying trip cards in various locations throughout the app, specifically interfacing with the `RecyclerView` components.

- `RequestAdapter`
    - Controller that allows displaying request trip cards in the request fragment
- `TripAdapter`
    - Controller that allows displaying trip cards in different fragments

`fragments/`

The fragments represent pieces of activities and allows more modular design. They enable different functionalities that exist inside of an Activity, such as the different tabs on the `MainActivity`.

- `trips/`
    - `ListTripsFragment`
        - Page that displays filtered trips in terms of status
    - `TripsFragment`
        - Page that displays the trips and their various states such as posted and requested
- `ProfileFragment`
    - Page that displays the users information
- `RequestsFragment`
    - Page that allows the user to request a ride
- `SearchFragment`
    - The user can search specific trips based off of the trip information

`objects/`

Objects represent the main entities of our application. They are POJOs (Plain Old Java Objects) with instance fields and “getter”/”setter” methods that allow for JSON libraries to easily serialize and deserialize these classes. They are used specifically when making requests to and processing responses from the backend.

- `Review`
    - Class to allow the user to review a request and accept/ reject it
- `Trip`

    - Class that stores the trip information
- `User`
    - Class that stores the user information

As stated above, tests are stored in the `app/src/test/java/com/example/mac/chartr` directory and contain both standard JUnit unit tests, as well as tests that use Robolectric, which allows us to do what would typically be called “instrumented” tests on a JVM instead of an Android emulator. We choose Robolectric for its speed and powerful simplicity that allowed us to write unit tests involving fragments and activities much more easily. We also used Mockito and on occasion PowerMock to do standard mocking and static mocking respectively inside our tests.

The largest framework we used, of course, was Android itself. We chose to develop in Android over iOS since we all had experience in Java and believed that an Android app would have the potential to reach more users. We used Retrofit as our HTTP client to interface with the backend, as well as Amazon’s Cognito Provider package to allow us to interface with the AWS Cognito service for user authentication. We choose Retrofit because it was widely used and allowed us to easily write code that interfaced with the backend. Cognito also allowed us to offload the handling of sensitive user data to a trusted third party so we would not have to deal with issues like saving user’s passwords.

Clearly, the choice of frameworks heavily influences the design of a mobile app. Android apps typically follow a certain design pattern that users expect to see, yet at the same time, we did not want to entrench the design in Android too heavily to prevent being able to have the same look on other devices. We chose to use native Date and Time Pickers to keep the experience familiar with Android users, and used a tab navigation that would be familiar to both Android and iOS users. The rest of the structure of the code was dictated by what the Android framework expects from your code, such as organizing classes into Activities and Fragments. Other frameworks such as Retrofit and Robolectric did not strongly influence the design, but definitely influenced how the individual classes that interact with the frameworks were structured.

Reflections & Lessons Learned

Christian Cygnus:

Working on Chartr made me appreciate the importance of team communication and coordination and become better at estimating the amount of time implementing a certain feature will take. Many of the obstacles we had to overcome during development simply had to do with getting everyone on the same page on what was going on. Similarly, during the iterations, some failure of communication was at fault for nearly every issue we encountered. I’ve learned to take the coordination more seriously in a team, and have improved in my team communication and delegation skills. Additionally, in the past I used to struggle with being able to accurately estimate the amount of time I would need to implement a user story on a project. However, through our work with Chartr, I’ve learned to be more cognisant of the amount of time it takes to plan, write, test, debug a user story, not to mention the additional time it takes to properly coordinate the work. By the end of our project, I learned to adjust expectations and account more for development time.

Brian Kurek:

Working on Chartr throughout this semester taught me a great deal about successful teamwork and project management. While I had already taken two team project courses, CS 428 gave teams much more freedom with their projects. This freedom allowed me to learn new things about team projects that I did not learn through past, more structured team project courses. First, I learned how much communication affects project progress. A strong correlation existed between my team's communication and project progress. When we did not communicate well, progress slowed down and user stories did not get completed until right before the iteration deadline. However, when we communicated consistently, project progress continued at a steady pace and user stories were implemented in a timely manner. Second, I learned the importance of having a well-defined process. A well-defined process ensured consistency across the project even with eight different people contributing to it. The process enforced consistent coding style, code reviews, use of user stories, and many other concepts that kept the project in order. Finally, I learned how to write a solid test suite. Working on the Chartr web interface, I got a chance to work with Jasmine (a popular JavaScript testing library) and the Angular testing framework. I learned how to write both simple unit tests and complex tests using mocks and spies. In addition, I learned how to write end-to-end tests for the web interface using Protractor. Undoubtedly, working on Chartr has been an invaluable experience that will aid me in my future academic and professional endeavors.

Marco Valentino:

Prior to this project, I had no experience writing a server, but I set myself a personal goal of learning how to write and entire backend through this project. I started on the backend team where I wrote the entire backend, which allowed me to learn about how Amazon Web Services worked and how to use the different services (i.e. Cognito, DynamoDB, Lambda, API Gateway) to create an API for the other teams to connect to. After completing all the necessary endpoints for our user stories, I moved to the Android team to help them finish and polish the app. When moving to the Android team, there was already a substantially sized code base that I had to get familiar with before I could contribute. I spend a lot of time refactoring the Android codebase, and through this process I understood how each component connected with each other. After this, I was able to work on a few screens that other members could use as an example to finish the rest of the project. Furthermore, I haven't done much experience in Android testing, so working on this project exposed me to many tools to write more comprehensive tests, such as Roboclectic. Besides gaining knowledge about the technical aspects of our project, I learned a lot about how to work well in a team. This was my first time working in such a large team (8 members) compared to previous team projects (3-4 members). This required us to have a lot more coordination and communication between subgroups. For example, after writing the backend, the Web and Android team needed an extra endpoint to get users by their UID, they contacted me through Slack and I had to write the new endpoint as soon as I could so they could continue working on their parts. Furthermore, I learned that time management is very important for meeting project deadlines. Our team overall had a tendency to push off implementation to the end of each iteration, so when we had to merge code there were many problems. Furthermore, we didn't have sufficient time to do manual testing for bugs, so we would always have to push off bug fixing into the next iteration. Overall, I enjoyed working on this project because it gave me the chance to learn a lot about not only writing a backend and Android app, but also the skills necessary for creating and working in a good team environment.

Michael Rush:

This semester I worked on the project Chartr. This was an app for web and Android. I worked on the Android side of the project. I learned many different things throughout the process. One such thing was what is needed to get a project off the ground. One needs more than just an idea to create a project. you need to have documentation such as what the user stories are or even

more basic what platforms and software will be used. Another thing I learned through this project was about time management and how to judge how long a task will take. This was a recurring problem for our project. I learned that in order to plan properly you must be in constant communication with your team on where everything is at. As soon as this stops everything comes to a halt and you fall behind. The last major thing I learned was how android programming works especially on the testing side. Android uses intents in order to move between pages in the app, high end view. The testing for android is different than other platforms because it is run on a phone with a different OS that heavily depends on the interface. One way to get around them is to mock so you are in a way running the android OS. The other was to create and manual test plan that documents how the interfaces should be tested and the steps to follow in order to check that they work.

Alexander Chang:

I learned a lot about planning, communicating with my team, and developing with angular while working on Chartr. A challenge for me was the “openness” of the project which differed greatly from the experience in other classes. Instead of having a very structured project with very specific deliverables, we were (in a way) in control of the scope and the deliverables of the project. On top of that, we were in charge of managing every step of the project and how we prioritized features given input from the team and from the TAs. Through the project experience, I learned how to deal with and react to feedback. Furthermore, I was able to better communicate changes and work on my time management because we underestimated how long certain tasks would take over our iterations. Finally, working on the project taught me that a robust process was necessary to keep things working. During the course of the semester, I was given the opportunity to work on angular (technically, v5) and learned a lot about how to implement effective testing procedures in angular using compass.

Shin Taniguchi:

Working on Chartr has taught me two valuable lessons. First is the importance of deciding which part of the project to work on and sticking to it. I wasn’t sure if I wanted to work on Android or backend until the end, and so I was thinking of just working on whichever team needed help. As a result, I was never proactive and fell into the trap of not doing sufficient amounts of work since I thought the other team members would be able to handle the work. Nearing the end of the project, the Android team needed a lot of help to finish the application, and I found out I really enjoyed working on Android. Second is the importance of seeking help from other team members. In the beginning, I was in charge of making tests for the backend Javascript code. I spent days trying to set up the testing frameworks and make some tests working, but since I spent too much time trying to work it out myself, I was not able to get high enough test coverage. Fortunately, Christian was able to cover for me and we managed to get full coverage for our backend tests. If I reached out to him earlier, we could have worked together and I could have saved more time on the set up and focused on writing better and more comprehensive tests. After reflecting on this experience, when I moved to the Android team, I didn’t have any prior experience, so I decided to get help from Marco. Through this, I was able to get a good grasp of the Android basics and using Marco’s example code, I was able to contribute a substantial amount to finish and polish the Android app. Overall, this project gave me the opportunity to fix some of my personal problems when working in a team setting and also ignited my passion for developing Android applications, which I hope to continue to pursue in the future.

Gabriel Hayat:

Throughout the semester, working on the project Chartr allowed me to improve in various areas. First of all, the Chartr android application was my first experience in mobile development.

Although I had to quickly gain knowledge during the first iterations through many android tutorials, this experience was really rewarding as mobile development is a skill that I always wanted to acquire, and I am glad I was finally introduced in that field. Moreover, despite the fact that I have had quite a few projects in other classes, none of them left my group and I so much freedom about the final product as well as the process that leads to it. The absence of too restrictive guidelines taught me two major aspects that I would like to underline. The first aspect is communication. I often heard from previous experiences that communication is a key point in a project's success, but this was the first time I really understood the importance of it. Having a high cohesion within your group allows not only to have a more structured project, but also improves the quality of it as work is not done at the last minute. The second aspect is to keep your project as simple as possible. When eight different people are contributing to the same project, I feel simplicity is key in the project's success. Indeed, as more than one person work on the same code, duplication quickly becomes an important issue and refactoring is essential in order to have a coherent project. In addition, maintaining readability of your project (code standards, checkstyles...) allows any group member to easily proceed with the work you have started and thus be more efficient. Overall, working on Chartr not only allowed me to improve from a technical point of view but also introduced me to key areas of team working and software development, skills that will inevitably be very valuable in my future professional experiences.

Pawel Galusza:

Working on the Chartr, our group's course project, over the course of this semester has broadened my knowledge about of the challenges and benefits of working on such a project. A major demand of the project was communication. Since the provided project requirements were open ended, we had to make our own specifications and guide our own development. When making decisions concerning those specifications our group would need to frequently communicate with each other to refine our plans and share our progress. I found that times of little communication corresponded to little work being done and that, in order to complete our goals in a timely manner, we all needed to be aware of the division of labor and responsibility. Furthermore, as the project progressed, I could more accurately estimate the amount of time needed to complete various tasks. On a more technical note, I learned about Android programming and the usefulness of continuous integration, which were both relatively unknown to me prior to my work on Chartr. The project gave me a sought after opportunity to learn a bit about creating Android apps and testing them. Over the many iterations, I learned to better communicate with my group, better plan my work for each iteration, and write better Android code.