SW Engineering CSC648/848

WhereWeEatinMeow

Restaurant Choice Solution

Section 04, Team 03

Raul Cardenas – *Team Lead / Infrastructure Lead*
Andy Nguyen – *Github Master / Back-end*
David Ye Luo – *Database Admin / Front-end*
Dean De Leon – *Scrum Master / Front-end*
Fred Gunther – *Back-End Master / Front-end*
Jonathan Ko – *Back-End Master / Front-end*

“Milestone 1”

March 8th, 2023

Revision History Table

Revision ID	Revision Date	Revised By
1.0	March 8, 2023	Raul Cardenas

Content and structure for Milestone 1 document for review

Executive Summary

WhereWeEatinMeow is a mobile/website application that solves the common problem of group indecision when it comes to choosing a restaurant. The app provides personalized recommendations based on user preferences, location, and availability. With a simple user interface, WhereWeEatinMeow streamlines the decision-making process by suggesting a single restaurant that meets the group's criteria, eliminating the need for lengthy discussions and debates.

Unlike other food recommendation apps, WhereWeEatinMeow focuses on providing a single option based on user input, saving time and reducing decision fatigue. The app is unique in its ability to provide personalized suggestions based on group preferences, making it an ideal tool for coordinating group outings.

Our team of experienced developers and designers are committed to creating a seamless and user-friendly experience for our customers. We are confident that WhereWeEatinMeow will

revolutionize the way groups make decisions about where to eat, and we welcome the opportunity to bring our product to market.

Personas and User Stories

Name: Calvin

- Age: 25
- Occupation: Boyfriend
- Information:
    - Has a girlfriend, Sarah(24).
    - Girlfriend has a busy schedule but when they meet, they always get in an endless debate about what to eat and where to eat
    - Both are college students and rarely have time together.
    - Calvin wished there was an easier way of deciding somewhere for the two so it would be adventurous, or so Sarah claims.
- Priority: 2

Name: Ben

- Age: 20
- College Student
- Occasionally want to make plans with friends to catch up
- Ben has a busy schedule juggling homework, work, and chores
- He wants to hang out with his friends but also don't want to spend too much time and effort in planning
- Everyone pitch in what they like (Korean BBQ, Sushi, Pho, Burrito, …)
- However, the group is waiting that someone to make a decision
- Ben wants to plan the meeting quickly but wants to respect other people's pitch

Name: Bobby

- Age: 31
- Occupation: architect
- Behaviors: Bobby just moved into the city with his wife, and wants to try new food places within the area of their new home. Bobby and his wife often do not make a decision and end up eating at the few spots near them. Bobby and his wife love western European foods, such as French and Spanish cuisine.
- Priority: 1
- Pain points:

    - Indecisiveness bothers Bobby because it conflicts with his personality as an architect, diligent in conveying his thoughts to others and in decision-making.
    - left with too little time to choose at the end of a workday, especially with a busy working wife
    - wants to try new foods

Name: Ryan

- Age: 16
- Occupation: Cashier
- Behaviors: Ryan lives with his single-mom, who works two jobs to support the two. Ryan often cooks, but many times has no ingredients, little time, or is too tired to cook. His mother often leaves him some money for takeout. Ryan opts for pizza most times.
- Priority: 4
- Pain points:
    - Ryan's mother does not want him to keep ordering foods that are too unhealthy.
    - Ryan has not tried many foods in his life, so he does not realize the variety of food he can potentially enjoy.
    - Ryan is indecisive.

Name: Gabe

- Age: 40
- Occupation: High School Teacher
- Behaviors
    - A well liked teacher by his students and easy to approach
    - Provides an open and well nurturing environment for his student to learn
    - If his class (or classes) are reward deserving, he provides the reward
- Priority: 5
- Pain Points
    - Unaware of suitable, affordable, and close to the area bulk meals to order to go
    - Needs to respect all, or as many as possible, food wants from his students

Name: Sarah

- Age: 24
- Occupation: College student

- Behaviors: Sarah is always on the go, juggling her classes and part-time job. She prefers quick, easy and affordable meals. She often eats out with friends but struggles to make decisions on where to go due to indecisiveness and having a hard time finding a restaurant that meets everyone's preferences. She values healthy options and enjoys exploring new food places.
- Priority: 2
- Pain Points:
    - Indecisiveness, lack of time, too many restaurant choices.
    - Not knowing which ones serve the type of food she wants.

Name: Rose

- Age: 40
- Occupation: Saleswoman
- Behaviors:
    - Frequently working a 9 to 5 job from home with her husband working a different job outside the house.
    - Rose and her husband don't have any kids so they can be adventurous with their dining decisions.
    - Rose is working so much that when Friday hits she is finally ready to get out of the house with her husband for a nice night out and neither of them want to cook. The two decide to go out to eat together.
    - Rose and her husband don't care how much the dinner costs because they are just excited to get out of the house together after a long work week. Rose and her husband pick the "four dollar sign" and up option ($$$$, $$$$$) on the application and one that has 4 stars or more. This decision will send them to a restaurant that is more expensive and on the higher end of the list for restaurant recommendations.
    - They leave the food type up to the application because they are both open to trying something different that is in the more exclusive range.
      -Rose and her husband don't want to be out too late so they set the distance to a restaurant within a 20 mile radius.
      -Rose is trying to watch what she eats because she is stuck in her house all day working she needs to choose healthier options.
    - Priority: 3

Data Definitions

User Table

Primary Data Name	Definition	Usage
Registered_user	A person who can sign up with an account on the application. Ex:	The necessary information of the registered users is collected and stored. This will

	Name: Andy user_id: 1 Username: Andy101 Home_address: Andy Ave, San Andy, CA, 10101 Preferences: Mexican food	be used to have default settings so that the user doesn't need to input every time.
History	The last 5 restaurants the app showed to the user	The history of restaurant information. The user can pay for more search history if they want at $10 per extra history list. Ex: 10 search history, costs $50/mo
Reroll	Amount of times a user can reroll. Reset every 24 hours. Ex: Reroll: 3	The user can use reroll to change restaurants to 3 times per day. Can pay for more per day usage at $100.
Search	Keeps track of search cooldown for each user. Ex: SearchTimes: 2 Cooldown: 10mins	Used to keep track how many searches a user has done. Used to stop bypassing reroll. Every 10 mins, increments search time up to 2

Users

Primary Data Name	Sub-Data
user	● user_id UNIQUE ● username ● home_address
restaurants_preference	● restaurant_pref_id UNIQUE ● user_id FOREIGN KEY ● restaurant_name ● preset_number
type_of_restaurant_preference	● restaurant_pref_id UNIQUE ● user_id FOREIGN KEY ● type ● preset_number

area_preference	• area_pref_id UNIQUE • user_id FOREIGN KEY • radius • preset_number
time_preference	• time_time_pref_id UNIQUE • user_id FOREIGN KEY • Time • preset _number
rating_preference	• rating_pref_id UNIQUE • user_id FOREIGN KEY • rating • preset_number
price_preference	• price_pref_id UNIQUE • user_id FOREIGN KEY • price • preset_number
food_preference	• food_pref_id UNIQUE • user_id FOREIGN KEY • food_name • preset_number
attributes_preference	• attributes_pref_id UNIQUE • user_id FOREIGN KEY • attributes • preset_number
history	• history_id UNIQUE • user_id FOREIGN KEY • preference • date • preset_number
reroll	• reroll_id UNIQUE • user_id FOREIGN KEY • count • time_since_last_used
search	• search_id UNIQUE • user_id FOREIGN KEY • count • time_since_last_used

List of Functional Requirements

- Users should…
    - Sign-up/login: User can sign up with an username, email, and password. User can then login with said username/email and password.
        - uniqueID: user1
        - Priority: userPriority4
    - Display/set preferences: User can choose different food types to be considered in the selection (display criteria they want to use). User can also choose maximum distance from their set location. User can set minimum star rating of restaurant to be considered.
        - uniqueID: user2
        - Priority: userPriority1
    - Confirm to be given a restaurant: User can accept their given restaurant or choose to reroll.
        - uniqueID: user3
        - Priority: userPriority2
    - Reroll: User can be given another random restaurant based on preferences or criteria. User has a maximum of three rerolls before a cooldown.
        - uniqueID: user4
        - Priority: userPriority3
    - Search for past restaurant given: User can search past rerolls or randomly selected restaurants.
        - uniqueID: user5
        - Priority: userPriority5

- App should
    - Connect to internet: this app requires internet usages
        - uniqueID: app1
        - Priority: appPriority1
    - Be able to order food from a restaurant: ???
        - uniqueID: app2
        - Priority: ????
    - Display preferences: App shows user's selected food, location preferences before selecting a restaurant.
        - uniqueID: app3
        - Priority: appPriority2
    - Display randomly chosen restaurant based on preferences: After preferences are chosen and user is ready to be given a restaurant, the app will provide info on one restaurant based on preferences
        - uniqueID: app4
        - Priority: appPriority5

    - Promote another restaurant in promoted section: ???
        - uniqueID: app5
        - Priority: ????
    - Pull from other/similar restaurant provider app (Yelp/grubHub/Doordash api): App pulls number of restaurants from other websites/app based on preferences.
        - uniqueID: app6
        - Priority: appPriority4
    - Calculate/find correct restaurant within distance from set location if set: App would find restaurant(s) within preferred distance from set location
        - uniqueID: app7
        - Priority: appPriority3

List of Non Functional Requirements

- Performance:
    - The app must be responsive and fast even when it has a large number of users.
    - The maximum time for the app to respond to user requests should be less than 1 second.
    - The app should be able to handle a load of 100 concurrent users.
- Storage space:
    - The app must have enough storage space to store user data, and other necessary files.
    - The app should be able to store data for at least 10,000 users and 500 restaurants.
    - The maximum size of any file stored in the app should be less than 5MB.
- Security:
    - User passwords should be hashed and stored securely.
    - The app should prevent unauthorized access to user data and protect against common attacks such as SQL injection, cross-site scripting, and cross-site request forgery.
    - Security groups should be locked down to the specific AWS components:
        - Only the EC2 instance will have permissions to access the database
        - Only the application load balancer should be able to connect to EC2 instance
        - Team members must use the console into the private subnet to access the EC2 instance
- Scalability:
    - The app should be designed to scale horizontally by adding more servers to handle additional load up to 4 servers limit to 400 concurrent users .
- Maintainability:
    - Code is in the main branch
    - Code will be tested on each PR for basic runnability and unit tests.

Competitive Analysis

Ideas

Main ideas to generate the matrix:

- Checking if restaurant is open or not for the user.
- Selecting restaurants within the user's budget.
- Providing star ratings of restaurants.
- Providing a walkability score (difficult).
- Adding a promoted section.

Matrix

Competitive Analysis						
WhereWeEatingAtMeow	**Roulette**	**Dinner Decider**	**Dining Decider**	**Grub Scrambler**	**Uber Eats**	**Yelp**
Check to see if the restaurant is open so users don't go to a closed restaurant.	Has a feature that will only show restaurants that are currently open.	Doesn't have a filter to only show restaurants that are open but it does have a way to schedule your meal to see if it is open at the time you want to eat.	Shows time the restaurant closes on that day.	Doesn't show times for the restaurant but clicking on the rating will take you to the restaurants yelp site.	Doesn't show closed restaurants	X
Selecting a restaurant within the user's budget.	User can set a price range based on Yelps $ ratings.	Can set a min and max budget based on Yelps $ ratings.	Can set a budget preference based on Yelps $ ratings.	No budget asked for.	Price range is available.	X

Provide the user the yelp star rating.	Once user searches, user can see star ratings/reviews for each restaurant	Might show the star ratings once restaurants come up, couldn't get restaurants to populate.	Gives a star rating from Yelp with the restaurant being recommended.	Gives star rating once restaurant is decided on by app.	Rating system based on other UberEats users.	X
Provide a walkability score for the user to decide how they will get to the restaurant.	No walkability scores.	No walkability scores.	No walkability scores.	No walkability scores.	No walkability scores.	No walkability scores.
Promote the restaurant	NO	NO	NO	NO	NO	NO

Advantages vs Competitor Applications

The advantages of our planned product compared to what is already available is that our application will actually decide on a restaurant for the user and take away any of the overwhelming possibilities. I believe people will come to use our product because they can't decide where to go to eat. Users will come to these competitor applications as well but the user will be able to see all other available restaurant options before the restaurant gets chosen for them (with the exception of Dining Decider) or after. Our product will not show other restaurants to the user once they have had a restaurant decided for them. This will help the user not be tempted to go to another restaurant or raise questions about other restaurants being better than the one that was chosen for them. This app is supposed to decide for the user, not complicate their problem even more. I also believe our application will be better because we have the ability to learn from our competitors. We can see what features we like and don't like. The competitor applications don't have any attractive features or an attractive look. They are all very bland and boring.

Quick breakdown of how the competitor applications work.

Roulette is powered by Yelp, it has no random search for a restaurant. It is very much like Yelp in the sense that you can set your distance/location, price range, and food type. Once you search for a restaurant you can click a link to take you directly to Yelp to see more about that specific restaurant. The application does not choose a restaurant for you, it gives you a list of restaurants that fit your preferences.

Dinner decider immediately makes you create an account with them, then sends an email to have you register your account. The app then asks you to create a party or join a party. Create a party will allow you to create a party and start with preferences, join a party will have you use your camera to link up with someone else's party. The filters offered are meal time(Any, Breakfast, Lunch and Dinner), Min budget & Max Budget, Distance radius, location, open at with date and time. There are also paid features, restaurants to rate and groups of 3-8, I did not pay for these so I do not know how they work. Once you put your preferences in it gives you a results page that gives 6 random restaurants to chose from, you can swipe left to delete and swipe right for more info on that restaurant. When I searched for my 6 random restaurants it stopped me with a QR code and I didn't know how to proceed further. I also noticed there were a bunch of "hints" staying on the screen.

Dining Decider immediately shows you a restaurant as soon as you open the app and behind that restaurant is a map to show user the location and give directions. . It then has Yelp features that show price, time its open 'til, Call, details, open maps and picture of a meal from that restaurant. If user clicks on details, it redirects them to yelp. You can change the distance, budget and decide if you want coffee, drinks, brunch or delivery. If you swipe up it will give you a new restaurant in that search radius and allow you to go back at previous restaurants that populated. Also has a place to donate to the app to get away from Ads. When I tried to see what the donate link would do it froze and wouldn't allow me to get out of the donate screen.

Grub scrambler will ask you to Find a "random restaurant" or "create or add to list." The "find random restaurant" feature will ask you for a distance, once distance is decided it will give you a list of random restaurants within the specified distance and then you hit scramble and it spins a wheel, whatever the wheel lands on, that is the random restaurant for you. The name of the random restaurant is in the middle of the screen with the address and the star rating below the address. You can find another random restaurant or hit scramble again to get it to pick another random restaurant from the original list. The user can choose the type of restaurant as well or select all. The create a list feature is a way to add restaurants that you want to try to a list and be able to spin that list until you have tried all of the restaurants on the list. Clicking on the address will take you to a map/give directions on how to get to the restaurant.

Uber eats is more than just for restaurants, it also shows stores to get beverages and coffees, etc. this is not what our application is used for. This will complicate the users indecisiveness even more. This app also doesn't pick a restaurant for the user it just gives the user more options to chose from again not addressing the user of the user.

Yelp is a lot like ubereats but it doesn't provide as many delivery options as yelp and it also has rankings for just about any service offered in your area.

In conclusion, most of these apps have the same basic user preferences of budget, rating and type of food. All of the applications had a very user-friendly interface when they were working the way they were supposed to. The application that we can learn the most from in my opinion is "Grub Scrambler." Grub Scrambler takes a list of restaurants in the area and puts them in a wheel of fortune like wheel and allows the user to spin the wheel. We can do this but spin the wheel behind the scenes instead of showing the user the wheel spinning. We can also add more features to make this application more user friendly.

High Level System Requirements

Server-Side:

- Server host: AWS EC2 t2.micro 1vCPU 1GB RAM
- Operating System: Ubuntu 22.04 Server LTS
- Web framework: Django 4.1
- Server-Side Language: Python 3.11.1
- Remote database: AWS RDS MySQL v8.0.28, 1vCPU 1GB RAM

Client-Side:

- HTML/HTML5

Deployment:

- Application Load Balancer for load balancing
- AWS Elastic compute / auto scale for deployment and scaling

Testing and Quality Assurance:

- Unit tests and integration testing checks on Github Actions
- Github Actions for continuous integration

Monitoring and Logging:

- AWS CloudWatch for monitoring and logging

Supported Browsers:

- Google Chrome
- Mozilla Firefox
- Apple Safari
- Microsoft Edge

Team

Raul Cardenas – *Team Lead*

- Implement AWS Infrastructure (Connect to internet: this app requires internet usages)
    - Create EC2 Instance
    - Create RDS Database
    - Create Application Load Balancer
    - Create routes to subnets
    - Create Domain
    - Create SSL certs for hosted zones (domain)
-
- Study Plan: Project Management
    - Courses:
        - https://university.atlassian.com/student/activity/1300198-writing-complex-jql-in-jira-software-course-description
        - https://university.atlassian.com/student/activity/1300198-writing-complex-jql-in-jira-software-course-description
        - https://university.atlassian.com/student/activity/1255606-how-to-run-effective-meetings-course-description
        - https://university.atlassian.com/student/activity/1238419-managing-a-board-in-trello-course-description
        - https://university.atlassian.com/student/activity/1117976-the-beginner-s-guide-to-agile-in-jira-course-description
        - https://university.atlassian.com/student/path/815443-jira-fundamentals
- Study Plan: Django
    - Courses:
        - https://docs.djangoproject.com/en/4.1/intro/tutorial01/
        - https://docs.djangoproject.com/en/4.1/intro/tutorial02/
        - https://docs.djangoproject.com/en/4.1/intro/tutorial03/
        - https://docs.djangoproject.com/en/4.1/intro/tutorial04/
        - https://docs.djangoproject.com/en/4.1/intro/tutorial05/
        - https://docs.djangoproject.com/en/4.1/intro/tutorial06/
        - Read and understand:
            - https://docs.djangoproject.com/en/4.1/intro/reusable-apps/
            - https://docs.djangoproject.com/en/4.1/topics/performance/
        - https://docs.djangoproject.com/en/4.1/topics/db/
        - https://docs.djangoproject.com/en/4.1/topics/http/

        - https://docs.djangoproject.com/en/4.1/topics/forms/
        - https://docs.djangoproject.com/en/4.1/topics/class-based-views/
        - https://docs.djangoproject.com/en/4.1/topics/files/
        - Unit Testing: https://docs.djangoproject.com/en/4.1/topics/testing/
        - User Authentication: https://docs.djangoproject.com/en/4.1/topics/auth/
        - Pagination: https://docs.djangoproject.com/en/4.1/topics/pagination/
        - Security: https://docs.djangoproject.com/en/4.1/topics/security/

Andy Nguyen – *Github Master / Back-end*

- Study Plan: Unit Testing / Github Actions Automation
    - Courses:
        - https://www.youtube.com/watch?v=TLB5MY9BBa4 (90min, Basics)
        - https://www.youtube.com/watch?v=qJPLFDtEi1I (1hr, Django Actions)
        - https://www.codingforentrepreneurs.com/blog/django-github-actions/
- Study Plan: Django
    - Courses:
        - https://docs.djangoproject.com/en/4.1/intro/tutorial01/
        - https://docs.djangoproject.com/en/4.1/intro/tutorial02/
        - https://docs.djangoproject.com/en/4.1/intro/tutorial03/
        - https://docs.djangoproject.com/en/4.1/intro/tutorial04/
        - https://docs.djangoproject.com/en/4.1/intro/tutorial05/
        - https://docs.djangoproject.com/en/4.1/intro/tutorial06/
        - Read and understand:
            - https://docs.djangoproject.com/en/4.1/intro/reusable-apps/
            - https://docs.djangoproject.com/en/4.1/topics/performance/
        - https://docs.djangoproject.com/en/4.1/topics/db/
        - https://docs.djangoproject.com/en/4.1/topics/http/
        - https://docs.djangoproject.com/en/4.1/topics/forms/
        - https://docs.djangoproject.com/en/4.1/topics/class-based-views/
        - https://docs.djangoproject.com/en/4.1/topics/files/
        - Unit Testing: https://docs.djangoproject.com/en/4.1/topics/testing/
        - User Authentication: https://docs.djangoproject.com/en/4.1/topics/auth/
        - Pagination: https://docs.djangoproject.com/en/4.1/topics/pagination/
        - Security: https://docs.djangoproject.com/en/4.1/topics/security/

David Ye Luo – *Database Admin / Front-end*

- Implement DB scheme/model for main data items
- Study Plan: Django
    - Courses:
        - https://docs.djangoproject.com/en/4.1/intro/tutorial01/
        - https://docs.djangoproject.com/en/4.1/intro/tutorial02/
        - https://docs.djangoproject.com/en/4.1/intro/tutorial03/
        - https://docs.djangoproject.com/en/4.1/intro/tutorial04/
        - https://docs.djangoproject.com/en/4.1/intro/tutorial05/

        - https://docs.djangoproject.com/en/4.1/intro/tutorial06/
        - Read and understand:
            - https://docs.djangoproject.com/en/4.1/intro/reusable-apps/
            - https://docs.djangoproject.com/en/4.1/topics/performance/
        - https://docs.djangoproject.com/en/4.1/topics/db/
        - https://docs.djangoproject.com/en/4.1/topics/http/
        - https://docs.djangoproject.com/en/4.1/topics/forms/
        - https://docs.djangoproject.com/en/4.1/topics/class-based-views/
        - https://docs.djangoproject.com/en/4.1/topics/files/
        - Unit Testing: https://docs.djangoproject.com/en/4.1/topics/testing/
        - User Authentication: https://docs.djangoproject.com/en/4.1/topics/auth/
        - Pagination: https://docs.djangoproject.com/en/4.1/topics/pagination/
        - Security: https://docs.djangoproject.com/en/4.1/topics/security/

Dean De Leon – *Scrum Master / Front-end*

- Implement Jira User stories with Team Lead
    - For each item, an associated user story and story points created
    - Help manage the Jira project
- Study Plan: Scrum Master
    - Courses:
        - https://www.youtube.com/watch?v=FIseJjqdEnc
- Study Plan: Django
    - Courses:
        - https://docs.djangoproject.com/en/4.1/intro/tutorial01/
        - https://docs.djangoproject.com/en/4.1/intro/tutorial02/
        - https://docs.djangoproject.com/en/4.1/intro/tutorial03/
        - https://docs.djangoproject.com/en/4.1/intro/tutorial04/
        - https://docs.djangoproject.com/en/4.1/intro/tutorial05/
        - https://docs.djangoproject.com/en/4.1/intro/tutorial06/
        - Read and understand:
            - https://docs.djangoproject.com/en/4.1/intro/reusable-apps/
            - https://docs.djangoproject.com/en/4.1/topics/performance/
        - https://docs.djangoproject.com/en/4.1/topics/db/
        - https://docs.djangoproject.com/en/4.1/topics/http/
        - https://docs.djangoproject.com/en/4.1/topics/forms/
        - https://docs.djangoproject.com/en/4.1/topics/class-based-views/
        - https://docs.djangoproject.com/en/4.1/topics/files/
        - Unit Testing: https://docs.djangoproject.com/en/4.1/topics/testing/
        - User Authentication: https://docs.djangoproject.com/en/4.1/topics/auth/
        - Pagination: https://docs.djangoproject.com/en/4.1/topics/pagination/
        - Security: https://docs.djangoproject.com/en/4.1/topics/security/

Fred Gunther – *Back-End Master / Front-end*

- Understand and implement Pagination between requests
- Study Plan: Django
    - Courses:

        - https://docs.djangoproject.com/en/4.1/intro/tutorial01/
        - https://docs.djangoproject.com/en/4.1/intro/tutorial02/
        - https://docs.djangoproject.com/en/4.1/intro/tutorial03/
        - https://docs.djangoproject.com/en/4.1/intro/tutorial04/
        - https://docs.djangoproject.com/en/4.1/intro/tutorial05/
        - https://docs.djangoproject.com/en/4.1/intro/tutorial06/
        - Read and understand:
            - https://docs.djangoproject.com/en/4.1/intro/reusable-apps/
            - https://docs.djangoproject.com/en/4.1/topics/performance/
        - https://docs.djangoproject.com/en/4.1/topics/db/
        - https://docs.djangoproject.com/en/4.1/topics/http/
        - https://docs.djangoproject.com/en/4.1/topics/forms/
        - https://docs.djangoproject.com/en/4.1/topics/class-based-views/
        - https://docs.djangoproject.com/en/4.1/topics/files/
        - Unit Testing: https://docs.djangoproject.com/en/4.1/topics/testing/
        - User Authentication: https://docs.djangoproject.com/en/4.1/topics/auth/
        - Pagination: https://docs.djangoproject.com/en/4.1/topics/pagination/
        - Security: https://docs.djangoproject.com/en/4.1/topics/security/

Jonathan Ko – *Back-End Master / Front-end*

- Implement User authentication in Django
- Study Plan: Django
    - Courses:
        - https://docs.djangoproject.com/en/4.1/intro/tutorial01/
        - https://docs.djangoproject.com/en/4.1/intro/tutorial02/
        - https://docs.djangoproject.com/en/4.1/intro/tutorial03/
        - https://docs.djangoproject.com/en/4.1/intro/tutorial04/
        - https://docs.djangoproject.com/en/4.1/intro/tutorial05/
        - https://docs.djangoproject.com/en/4.1/intro/tutorial06/
        - Read and understand:
            - https://docs.djangoproject.com/en/4.1/intro/reusable-apps/
            - https://docs.djangoproject.com/en/4.1/topics/performance/
        - https://docs.djangoproject.com/en/4.1/topics/db/
        - https://docs.djangoproject.com/en/4.1/topics/http/
        - https://docs.djangoproject.com/en/4.1/topics/forms/
        - https://docs.djangoproject.com/en/4.1/topics/class-based-views/
        - https://docs.djangoproject.com/en/4.1/topics/files/
        - Unit Testing: https://docs.djangoproject.com/en/4.1/topics/testing/
        - User Authentication: https://docs.djangoproject.com/en/4.1/topics/auth/
        - Pagination: https://docs.djangoproject.com/en/4.1/topics/pagination/
        - Security: https://docs.djangoproject.com/en/4.1/topics/security/

General Study Plan Guide:

Because everyone needs to contribute to our Django project so that we can own different components but also peer review work, we are using a generated study plan from chatgpt:

Day 1-2: Getting Started with Django (4-5 hours)

- Read the official Django documentation to get an overview of the framework.
- Install Django and create your first project using the official Django tutorial.
- Familiarize yourself with the basic directory structure of a Django project.

Day 3-4: Models and Databases (4-5 hours)

- Learn about Django's Object-Relational Mapping (ORM) and how to create models to interact with databases.
- Practice creating models for your project and using Django's built-in admin interface to manage data.

Day 5-6: Views and Templates (4-5 hours)

- Learn about Django's views and how to use them to display data to users.
- Practice creating views and templates to render HTML pages.

Day 7-8: Forms and User Input (4-5 hours)

- Learn about Django's forms and how to use them to handle user input.
- Practice creating forms and handling form data in views.

Day 9-10: Static Files and Media (2-3 hours)

- Learn how to handle static files (CSS, JS, images) and media (user uploads) in Django.
- Practice serving static files and handling media uploads.

Day 11-12: Authentication and Authorization (4-5 hours)

- Learn about Django's built-in authentication and authorization system.
- Practice creating login and registration views and restricting access to certain views based on user permissions.

Day 13-14: REST APIs (4-5 hours)

- Learn how to create REST APIs using Django's built-in support for the Django REST Framework.
- Practice creating API views and testing them using tools like Postman.

Day 15-16: Deployment (2-3 hours)

- Learn about deploying Django projects to production servers.
- Practice deploying your project to a hosting service. Use the test instance on AWS.

Day 17-18: Testing and Debugging (2-3 hours)

- Learn about Django's testing framework and how to write unit and integration tests for your project.
- Practice writing and running tests and debugging issues in your code.

Day 19-20: Customizing Django (2-3 hours)

- Learn how to customize Django by creating custom template tags and filters, creating custom middleware, and extending Django's admin interface.
- Practice creating customizations for your project.

Day 21-22: Advanced Topics (4-5 hours)

- Learn about more advanced topics in Django, such as caching, signals, and handling email.
- Practice implementing these features in your project.

Day 23-24: Security and Performance (4-5 hours)

- Learn about best practices for securing your Django project and improving its performance.
- Practice implementing security measures like HTTPS and improving performance using caching and optimizing database queries.

Day 25-26: Additional Resources (2-3 hours)

- Explore additional resources for learning Django, such as blogs, podcasts, and online courses.
- Find a Django community to join, such as a local meetup or online forum.

Day 27-28: Recap and Review (2-3 hours)

- Go back over what you've learned in the past 26 days and review any areas where you feel you need more practice.
- Create a list of questions or topics to explore further.

Day 29-30: Project Work (4-5 hours)

- Spend the final two days of your study plan working on a Django project of your own, incorporating what you've learned over the

Checklist

- Team found a time slot to meet outside of the class: Every Sat 8am - 10am
    - DONE
- Scrum Master shares meeting minutes with everyone after each meeting: Dean De Leon is our Scrum Master
    - DONE
- Github master chosen: Andy Nguyen is our Github master
    - DONE
- Everyone sets up their local development environment from the team's git repo.
    - Everyone setup an ubuntu 22.04 LTS virtual machine
        - Install Django
        - Install Python
        - Install Pip
        - Get your hello world app working inside the virtual machine
    - DONE
- Team decided and agreed together on using the listed SW tools and deployment server
    - DONE
- Team ready and able to use the chosen back/front-end frameworks.
    - For each technology (front/back-end/DB/cloud) ,
        - team decides who will lead the study of each technology and what will be the specific goal of the study within one month from the M1 announcement.
        - If you list a detailed study plan for this, earn extra point!
    - DONE
- Team lead ensured that all team members read the final M1 and agree/understand it before submission
    - DONE