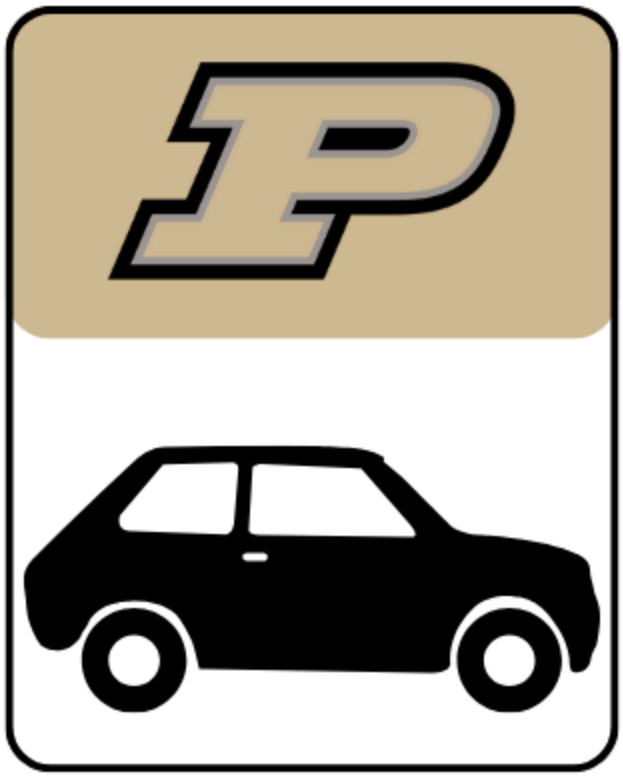

CS 307 Sprint 1 Retrospective - Group 6

BoilerPark

Pranay Nandkeolyar
Utkarsh Majithia
Anthony McCrovitz
Logan Portscheller
George Samra
Neel Vachhani

What went well during Sprint 1?

We worked well as a team this sprint. When members of our team were unable to complete their given stories, we stepped up together to ensure that the majority of the work got done. People were not beholden to their chosen user stories and stepped up to ensure that the work got done. Additionally, we were able to work well independently and everyone mostly got the work that they said they would do. People were also good at communicating on meetings about what was required of them and what they needed help on or were blocked on.

User Story #1 - As a developer I want to set up a Redis cache and a PostgreSQL database that stores the data and sends it to the frontend**.**

Task #	Task Description	Estimated Time	Owner
1	Create a Redis Cache to store the parking lot availability counters	1 hour	Neel
2	Create a PostgreSQL database to store historical data and login information	2 hour	Neel
3	Configure the Redis Cache to store key-value pairs tracking remaining parking spots	2 hours	Neel
4	Configure the Redis Cache to store key-value pairs for tracking last updated time	2 hours	Neel
5	Configure the PostgreSQL database with tables for login information and each parking lot	2 hours	Neel
6	Configure a Redis Pub/Sub channel to send key-value pair updates to clients	2 hours	Neel
7	Test the robustness and functionality of the Redis Cache and PostgreSQL databases	2 hours	Neel

Completed

When the Redis Cache is updated, the user is able to see the updated count onscreen in the client. Every 15 minutes, the values of the Redis Cache are sent to the PostgreSQL server for future data analytics. Finally, when the user starts a new client instance on their device, the backend subscribes to the Redis Pub/Sub channel and awaits updates from key-value pairs.

User Story #2 - As a developer I want to set up a python backend using django and implement basic routes and server functionalities.

Task #	Task Description	Estimated Time	Owner
1	Create Python backend and initialize Django framework	3 hours	George
2	Implement basic routing to the Redis cache and to the PostgreSQL database	4 hours	George
3	Test the connection by incrementing and decrementing Redis and checking backend	2 hours	George
4	Test the connection to the PostgreSQL database	2 hours	George

Completed

The Python backend was able to be set up correctly. There was some extra research required to make it available as a REST API using the Django Rest Framework, but once that was set up it went smoother.

User Story #3 - As a user, I want sign in with Apple to make creating a user account easier

Task #	Task Description	Estimated Time	Owner
1	Create and register app with Apple	1 hour	Utkarsh
2	Install Apple SDKs and libraries	1 hour	Utkarsh
3	Create a button on the frontend to sign in with Apple	2 hours	Utkarsh
4	Connect to the backend and implement endpoints for OAuth, tokens, and verification	5 hours	Utkarsh
5	Test integration to the PostgreSQL database, backend, and frontend	2 hours	Utkarsh

Completed: Utkarsh

The Apple authentication was properly integrated with a complete end to end setup where a user can login with their Apple ID and it creates an account in the backend of our app. The same apple ID can also be reused to login with your preferences being saved.

User Story #4 - As a user I want sign in with Google to make creating a user account easier

Task #	Task Description	Estimated Time	Owner
1	Register the app in Google Cloud Console and obtain OAuth 2.0 client credentials (Client ID & Secret).	1 hour	Utkarsh

2	Install Google OAuth SDKs and libraries	1 hour	Utkarsh
3	Create a button on the frontend to sign in with Google	2 hours	Utkarsh
4	Connect to the backend and implement endpoints for Google OAuth callback, token exchange, and Google ID token.	5 hours	Utkarsh
5	Test integration with PostgreSQL database, backend, and frontend	2 hours	Utkarsh

Completed: Utkarsh
The Google authentication was properly integrated with a complete end to end setup where a user can login with their Google ID and it creates an account in the backend of our app. The same Google ID can also be reused to login with your preferences being saved.

User Story #5 - As a user I want sign in with email and password to allow me to make an account and save my preferences

Task #	Task Description	Estimated Time	Owner
1	Create a user data type in the backend and database	1 hour	George
2	Create REST API routes for basic login and signup	2 hours	Utkarsh
3	Create some generic preferences that can be used to test persistence	3 hours	Logan
4	Create unit tests for the login and signup	2 hours	Logan
5	Test with admin users and run end to end tests	2 hours	Utkarsh

Completed: Utkarsh
We created a complete end to end login process that allows the user to submit a unique email and strong password that is then stored and hashed by the backend. The same email and password can then be reused by the user to login with their notification preferences being saved across sessions.

User Story #6 - As a developer I want to use computer vision to detect cars using online training data

Task #	Task Description	Estimated Time	Owner

1	Research pre-labeled car detection models from Open CV (COCO, Pascal VOC, Open Images, Roboflow)	3 hours	Pranay
2	Download pre-trained model and verify environment set up	3 hours	Pranay
3	Download parking lot and non parking lot video footage	2 hour	Pranay
4	Verify and test car image is detected by model	1 hour	Pranay
5	Verify and test that the test negative image is not detected by the model.	1 hour	Pranay

Completed

The model was downloaded with the correct weights from YOLO and Open CV. This was added to the backend cv_model folder. It was then tested on static images “2.png” inside of the cv_model folder to correctly identify the cars. Additionally, it was able to identify that “1.png” and “3.png” did not have cars in the picture. It also initially did the same for the first “cars.mp4” video and the “bikes.mp4” videos for correctly identifying and not identifying cars.

User Story #7 - As a developer I want the Redis Cache and the backend to be linked and accurately show counts on the frontend

Task #	Task Description	Estimated Time	Owner
1	Link Redis Cache to PostgreSQL DB through Python HTTP request on edge computer	2 hours	Pranay
2	Develop backend Django route to query the Redis Cache	2 hours	Pranay
3	Develop backend route to query the PostgreSQL DB as a fallback	2 hours	Pranay
4	Develop display to show the count for the parking lot	2 hours	Pranay
5	Test backend routes and user display	2 hours	Pranay

Completed

As the client opens up the app, the client can see the latest values on the frontend which are accurate as of the Redis Cache. We decided to not develop the backend route for PostgreSQL DB. The client can also see updated values when the Redis cache value changes as well. The displays show for each of the parking lots on the Garage List screen as well, showing that it keeps the values updated per garage. The client gets these updated values by subbing to a channel where the Redis values update.

User Story #8 - As a developer, I want to sync the camera to the CV model

Task #	Task Description	Estimated Time	Owner
1	Configure the camera to expose a video/data stream and verify local connection over HTTP	2 hours	Pranay
2	Set up a lightweight server (Flask, FastAPI, or MQTT broker) on the edge device	1 hour	Pranay
3	Connect the camera stream to the server and ensure data flow	1 hour	Pranay
4	Define event schema for edge to backend messages	1 hr	Pranay
5	Develop edge computing code to detect cars coming in and cars coming out	2 hours	Pranay
6	Connect the edge computing resource to the	3 hours	Neel

	Redis Cache and test		

Completed

The front end list view was able to display the accurate counts of the cars based on the traffic videos by accurately determining which way the car was moving across the line. Additionally, the value of the frontend was connected to the Redis Cache directly. Additionally, all garages were displayed in one place so the count is easy to determine. We used traffic feed footage instead as it was easier to obtain and then used that for the model. We will have a meeting with Purdue Parking soon as we have got confirmation we can set up a camera.

User Story #9 - As a user I want an attractive app icon that follows the new liquid glass design language for iOS users

Task #	Task Description	Estimated Time	Owner
1	Design App Icon	3 hours	George
2	Create Apple version of the App Icon with dark and clear versions	2 hours	George
3	Upload it to the codebase and check the correctness	1 hour	George
4	Design a React components that utilises the icon and displays it on the home screen	3 hours	Logan
5	Add the app icon to a splash screen that appears on load	1 hour	Logan

Completed

The App Icon was designed to appear like a parking sign. We had an initial drawing that was on our design doc, etc. that we decided not to use as we wanted to use the Motion P. This meant that we needed a car that also appeared to be moving/in motion. Transferring it to Liquid Glass was also harder than expected, but was able to be completed.

User Story #10 - As a developer I want to setup a react native frontend with a proper file structure and basic API routes to connect to the frontend and redis

Task #	Task Description	Estimated Time	Owner
1	Design React Native frontend architecture and format	1 hour	Anthony
2	Create an easy-to-understand file structure to store and organize frontend files	1 hour	Anthony
3	Develop basic theme (colors and format) that will be consistent across pages	2 hours	Anthony

4	Connect the frontend and redis cache	3 hours	Neel
5	Develop working log-in page	1 hour	Logan
6	Develop working home page (list of Purdue garages)	2 hours	Logan
7	Develop buttons on the bottom of pages to change between pages	1 hour	Anthony
8	Test integration of pages	1 hour	Anthony

Completed

The foundation for the React Native frontend is now complete. We started by designing the core architecture and creating an easy-to-understand file structure, which will be crucial for organizing our code as the app grows. A basic theme was also developed to keep the colors and formatting consistent across all pages. With the structure in place, the working log-in page and the main home page (which displays the list of Purdue garages) were built out. We also implemented the bottom navigation buttons to allow users to move between pages. A critical step was successfully connecting the frontend to the Redis cache. Finally, all pages were tested to ensure proper integration and functionality.

User Story #11 - As a user, I want to link my calendar to the app

Task #	Task Description	Estimated Time	Owner
1	Set up the frontend element to accept the uploading of an iCS file	3 hours	Anthony
2	Send the iCS file to the backend	2 hours	George
3	Ingest and clean data	1 hour	George
4	Create a basic display for uploaded calendar events	2 hours	Pranay
5	Test accuracy of calendar upload	2 hours	George

Completed

Uploading an ics calendar to python is something that has a decent amount of support online, so through a couple of libraries we were able to make it happen. Figuring out how to take care of recurring events was one of the harder parts, but one that was very applicable to our project. We're expecting people to use this feature at least in part for class schedules or club meetings, things that are very repetitive. Getting the backend hooked up to the frontend took a little more work, but we were able to spin up an API route.

User Story #12 - As a user, I want to see a "Last updated <timestamp>" label on each lot so I can trust freshness.

Task #	Task Description	Estimated Time	Owner
1	Set up the PostgreSQL table to have a last updated column	1 hour	Neel
2	Setup the frontend client to subscribe to the Redis Pub/Sub channel	1 hour	Neel
3	Develop a Python script to update the PostgreSQL table with the updated count for parking lots every 15 minutes	2 hours	Neel
4	Develop a Django route to fetch the count	1 hour	George
5	Test the connection between the database, the Redis cache, and the backend	1 hour	Logan

Completed

When the user starts a new client on their devices, the backend subscribes to the Pub/Sub channel and awaits updates using Django. Using crontab, the Redis cache values are sent and stored in the Postgres database every 15 minutes.

User Story #13 - As a user, I want to manage notification preferences (which alerts I get and when).

Task #	Task Description	Estimated Time	Owner
1	Create a list of notifications that we want to offer users	2 hours	Neel
2	Create a list view in the frontend that contains a list of all the notifications and allows a user to select which ones they want to receive	4 hours	Utkarsh
3	Set up route to the backend that accurately conveys the information	4 hours	Neel
4	Test backend user notification settings with a fake user	2 hours	Logan

Completed
When the user starts the client and navigates to the settings page, they can toggle notification preferences in a list of preferences and the preferences are saved and associated with the user on Postgres. When the user restarts the client at a later time, their previous notification preferences are reflected on the app.

User Story #14

Task #	Task Description	Estimated Time	Owner
1	Add push notification permissions and token registration in the mobile app	2 hours	Logan
2	Create API endpoints to enable user to enable push notifications	2 hours	Logan
3	Store and manage user device tokens in the PostgreSQL database	2 hours	Logan
4	Add backend logic to notify active users when parking passes go on sale	3 hours	George
5	Test that devices receive push notifications when a sale is active	2 hours	Logan

Completed: The first time the user installs and logs into the app, the user is prompted with a pop-up that asks if the user would like to enable notifications. If the user enables notifications, a push notification token is created and sent to the Django backend using the appropriate API call. The backend receives the token and stores it in the PostgreSQL database.

User Story #15 - As a user I want to have a map with all of the garages/lots listed on them

Task #	Task Description	Estimated Time	Owner
1	Find/create Purdue map	2 hours	Anthony
2	Add pindrop of tracked parking lots on the map	2 hours	Logan
3	Create map page on UI	2 hours	Logan
4	Add number on pindrops that display how full the lot is	2 hours	Logan
5	Connect availability data from database to frontend map	3 hours	Neel
6	Test availability updates from database	2 hours	George

Completed: A map page was successfully implemented with all the garages/lots on them. We found an appropriate Purdue map and then successfully integrated it in the app. On top of this map are five pin drops over the locations of the five garages/lots, with automatically-updating statistics.

User Story #16 - As a user, I want to toggle between a list view and a map view depending on my preference.

Task #	Task Description	Estimated Time	Owner
1	Add a toggle button (e.g., “List / Map”) to the UI.	2 hours	Utkarsh
2	Implement a list view that displays lots/garages with availability info.	3 hours	Utkarsh
3	Connect toggle logic to switch between the list view and the map view.	2 hours	Utkarsh
4	Ensure data (lots and availability) is consistent across both views.	2 hours	Utkarsh
5	Test switching between views to confirm state persists and no errors occur.	1 hour	Pranay

Completed

We created a beautiful listview in react with Purdue colours and tab buttons to navigate between the list and map view. All the garages are available with an integrated live count and a button to favorite them.

User Story #17 - As a user, I want to choose dark mode/light mode so the app fits my phone theme

Task #	Task Description	Estimated Time	Owner
1	Create a toggle on the frontend that effectively switches between light and dark mode	3 hours	Anthony
2	Send those changes to the backend through the proper channels	1 hour	Anthony
3	Create UI elements in both light and dark modes	4 hours	Anthony
4	Test the accessibility of the UI elements in the light and dark modes	1 hour	Anthony

Completed

We successfully implemented the light and dark mode feature to allow users to match the app to their phone's theme. A toggle was created on the frontend that allows the user to switch between modes, and this preference is sent to the backend to be saved. The main task was building out all the UI elements in both light and dark variations. Once completed, we tested all elements for accessibility to ensure they are readable and usable in both modes.

User Story #18 - As a user, I want the color scheme of the app to follow the Purdue color scheme as a Purdue-associated app.

Task #	Task Description	Estimated Time	Owner
1	Find and decide what actual hex colors to use in our app	1 hour	Anthony
2	Create UI elements with our specified colors in both light and dark modes	4 hours	Anthony
3	Test the legibility of the UI elements in both light and dark mode	2 hours	Anthony

Completed

To ensure the app feels like an official Purdue application, we have updated the color scheme. We began by researching and deciding on the official Purdue hex colors to use for our theme. Following that, we created the UI elements using these specified colors, making sure to support both the light and dark modes from the previous story. The final step was to test the legibility of all components in both modes to ensure the app is readable and the colors work well together.

What did not go well during Sprint 1?

Our communication left something to be desired during this sprint. We struggled to have meetings that people could attend in person due to exams, fall break, and general business. This led to problems where people who were working on the frontend did not know how it connected to the backend and vice versa. We siloed our development and it did not pay off in the end.

We also need to improve our assignment of tasks. There were multiple times where we had bottlenecks due to user stories that were assigned to 2 or 3 people, where 1 needed to do their work before 2 and 3 had the chance to. Assigning complete user stories to individual people will allow them to complete the required tasks in order without backups lasting until the end of the sprint.

We also need to implement better testing for Sprint 2. We tested our things for sprint 1, but often the testing was done by the person that wrote the code. This testing caught some bugs, but was not as comprehensive as we'd have liked. Luckily, this did not lead to any major issues in our first sprint, but definitely something that we need to improve on this sprint.

Additionally, we had integration issues frequently as we waited to merge code at the end. This led to people doing work on their own and testing the system at their own level. However, this caused issues when merging stuff together as it took a lot of time to ensure that data was sent in a consistent and expected format from the frontend to the backend. This led to a lot of time wasted merging. Additionally, the merged code often had errors and refactoring was required.

User Stories/Tasks Not Completed:

User Story #1

4	Configure the Redis Cache to store key-value pairs for tracking last updated time	2 hours	Neel

Not Completed:

During development, we realized that instead of tracking the last updated time in Redis, it would make more sense to use the local clock to show the last updated time. This means that as users refresh the list, the time shown will reflect when they last refreshed the list, not when the value in Redis itself was actually updated. This realization made this task obsolete.

User Story #7

3	Develop backend route to query the PostgreSQL DB as a fallback	2 hours	Pranay

Not Completed:

During development, we realized that instead of handling requests to multiple different databases for a single request to query the count, we can simply use 1 as long as we are sure Redidis is consistent, which we did with the CV model.

User Story #8

1	Configure the camera to expose a video/data stream and verify local connection over HTTP	2 hours	Pranay
2	Set up a lightweight server (Flask, FastAPI, or MQTT broker) on the edge device	1 hour	Pranay
3	Connect the camera stream to the server and ensure data flow	1 hour	Pranay

Not Completed:

The meeting with Purdue Parking got delayed so we replaced the camera work with setting up webcam capabilities for the script as well as testing the application on traffic based videos. This ensures we will both have webcam capabilities and car detecting capabilities instead.

User Story #12

1	Set up the PostgreSQL table to have a last updated column	1 hour	Neel

Not completed:
From the same logic as the uncompleted task in User Story #1, we realized that it's not needed to keep a last updated timestamp column in PostgreSQL. The database already has an entry timestamp for when the data was entered into the system, so a last updated timestamp would be redundant. Additionally, the availability data in the database is only used for insights, so the entry timestamp would be sufficient for our needs in that case.

User Story #14

4	Add backend logic to notify active users when parking passes go on sale	3 hours	George

Not completed:
This was not completed because we struggled to get notifications working until the very end, so the routes for the notifications were left unfinished.

How should we improve?

We need to improve our communication, and plan on doing that in 2 main ways. First off, we plan on utilizing the class time that will be vacated when lectures end this week. Second, we plan on having dedicated knowledge transfer time where each member gets up and explains what they've done, how it relates to the project as a whole, and how it will be used by members of the front/backends. We also plan on improving our communication outside of meetings, sending each other quick texts when we have problems versus waiting for the meetings to talk about them. Additionally, we plan to merge our code early and often whenever a feature is ready to be implemented. These meetings will be a great time to do so as we can discuss whatever miscommunication there may be as well as ensure the code works. Finally, we plan to discuss what each person working on a feature expects as input and output. We will do this through meetings, text messages, and good comments. We'd like to get our work done sooner and we also think it would be a good idea if we all showed up for the demo next sprint.