ParkItPlace

COMPSCI 4ZP6 - Development Plan Assignment Description

Ivan Ebos
Avin Lanson Tharakan
Ben Lee
Muhammad Salim
John Wu
Ritvik Deshpande

Version history

Version	Date
Version 0.0	October 25, 2024

Table of Contents

Team Meeting and Communication Plan

Document Collaboration:
- We will share and collaborate on documents together using the google drive suite. including services as google docs, google slides, google excel

Code Collaboration:
- We will be using GitHub to collaborate on code.

Program management tools:
- We will use Jira for program managements (the TA will be added)

Team Member Roles

Functional Requirements

Feature	Priority	Effort (x2 weeks)	Role
User Registration and Authentication	P0	2	Ben, John
Driveway Listing Creation and Management	P0	0.5	**Maaz**, Avin
Search and Filtering	P0	2	Ritvik, **Ben**, Maaz, John
In-App Communication	P0	0.5	**Ritvik**
Booking and Payment Management	P0	2	Ivan **Avin**, Ritvik
Ratings and Reviews for Credibility	P0	0.5	**John**
Driveway Listing Creation and Management	P1	2	Maaz,BenB
Search and Filtering	P1	0.5	Ritvik, Maaz, Ben
In-App Communication	P1	0.5	Ritvik
Booking and Payment Management	P1	0.5	Ivan, Avin, Ritvik
Admin and Moderation Panel	P1	1	Ivan, Avin
User Credibility Score	P1	1	John
User Authentication	P2	0.5	Ivan, Maaz

Profile Management	P2	1	John,Ben
Booking and Payment Management	P2	0.5	Ritvik, Avin
User Registration and Authentication	P3	1.5	Ivan, Avin
Driveway Listing Creation and Management	P3	1	Maaz
Search and Filtering	P3	0.5	Ritvik,Ben
Cancellation Policy	P3	2	John
Report Misuse	P3	1	Ritvik
Driveway Listing Creation and Management	P4	1	Maaz, Ritvik
Time Enforcement and Penalties	P4	0.5	Ivan
User Credibility	P4	1	Avin,Ben
Parking Timer and Overstay Alerts	P4	0.5	John,Ben

Non-Functional Requirements:

Colour Scheme	P4	0.5	All
Organisation	P1	0.5	All
Uniformity	P1	0.5	All
Necessity	p2	0.5	All
Clarity	P2	0.5	All
User FeedBack	P3	0.5	Ben
Tutorial	P2	0.5	Ritvik
Concise	P1	0.5	All
Accessibility	P0	0.5	Ivan

Performance Requirements	P0	0.5	John, Avin

Project manager:

- We will have a rotational system where a new pm will be appointed in biweekly Primary PM leaders will be John, Ivan, Avin.

Workflow Plan

Branching Strategy:

- **Main Branch**: The main branch will contain stable, production-ready code. No direct commits will be made to this branch.
- **Development Branch**: The develop branch will contain code that is in active development but has passed initial testing. It serves as an integration branch for features.
- **Feature Branches:** Each new feature will have its own branch (feature/feature-name) branched off from develop. Once the feature is completed and tested, it will be merged back into the develop branch via a pull request (PR).
- **Hotfix/Release Branches**: If a critical bug needs to be fixed in the production environment, a hotfix/hotfix-name branch will be created from main and merged back into both main and develop. For releases, a release/release-version branch will be created and merged into main once finalized.

Pull Requests (PRs):

- All changes must be submitted through pull requests to ensure code reviews and prevent unauthorized direct pushes to the develop or main branches.
- **Code Review Process:** At least 2 other team member must review each PR before it can be merged. This ensures code quality and functionality.
- **Continuous Integration (CI):** Automated tests will run on each code change to ensure new updates do not break existing functionality. Code will only be integrated once all tests successfully pass.

We will adopt the **Scrumban approach** for project management, using **Jira** to manage tasks and workflow. Jira will handle:

- **Backlog Management**: User stories and tasks will be prioritized in the Jira backlog and broken into smaller tasks as needed.
- **Kanban Board**: The board will track tasks through the following states: Backlog, In Progress, Complete, Merged, and Analysis.
- **Sprint Planning**: While not strict, we'll plan short sprints (two weeks) to manage deliverables and track progress.

This setup balances the structured planning of Scrum with Kanban's flexibility, allowing us to manage tasks efficiently while adapting to our university schedules

Data Storage

- **Cloud Firestore**: A NoSQL database for storing and syncing data in real time across users and devices.
- **Cloud Storage**: For storing and serving user-generated content such as images, videos, and other large files.
- **Realtime Database**: For real-time data syncing, providing instant updates across all connected devices

Compute Heavy Tasks

Since there's no need for model training, we'll use Firebase's free server to handle backend tasks. Firebase's Cloud Functions will manage server-side logic, while Firestore will store real-time data. This serverless setup scales automatically, providing a cost-effective and easy-to-manage infrastructure for your app's needs.

Tool/Method to Achieve Requirements and Performance metrics

- **Quick User Interaction Responses (e.g., map navigation, parking search)**
  - **Frontend Optimization**: Use **React** for fast rendering of user interactions.
  - **Firebase Firestore Database**: Use **Firestore** for real-time synchronisation of parking spot data because it allows for low-latency updates and can scale as your app grows.
- **Instantaneous Transaction Processing: Microservices Architecture**: Split the app's functionalities (e.g., booking, payment processing, user management) into individual microservices to prevent one process from slowing down others.
- **Fast API Responses (e.g., retrieving available spots)**
  - **Firestore Queries**: Optimise **Firestore** queries by using indexes to retrieve parking spot availability efficiently
  -
- **Encryption of Personal Data**
  - **Data Encryption in Firebase**: Firebase provides built-in **AES-256 encryption** for all data at rest. Data in transit is automatically encrypted using **TLS**.
- **Uptime for Critical Features (Booking & Payments)**
  - **Firebase Cloud Functions & Hosting**: Firebase's serverless architecture (with **Cloud Functions** and **Firebase Hosting**) provides auto-scaling and high availability, ensuring that critical features like bookings and payments remain operational.

        - **Monitoring & Alerting:** Implement monitoring tools to continuously monitor system health and alert for potential issues.

Proof of Concept Demonstration Plan

For our proof-of-concept demonstration, we plan to have some simplified versions of our P0 requirement features implemented in a web-application to showcase a low fidelity but functional prototype. This along with our slideshow will explain our main problem statement and how the aforementioned features will tackle this need. With our presentation we aim to show evidence of working code, and visualize our P0 features to show our implementation process. In our demonstration, we plan to quickly cover our low level implementations in a step-by-step progression, beginning with a simple user login system, listing and posting capabilities, and a simple search bar to filter these listings.

Additionally, if time permits we will implement a basic communication protocol and a functioning map component as well. The focus of our proof of concept is to show our problem statement and to exhibit some meaningful progress has been made to implement our solutions.The development focus will be aligned with these priorities to establish confidence in our project's feasibility.

Technology

Front End Technology

- **React Native**: Used for building cross-platform mobile applications for iOS and Android.
- **React**: Used for developing the web application's interactive user interface.

Backend Technology

- **Firebase**: Serves as the cloud provider for backend functionality, enabling real-time data handling and user management.
    - **Cloud Functions:** Executes backend code in response to events or HTTP requests.
    - **Authentication:** Manages user sign-up, login, and authentication processes.
    - **Firebase Hosting:** Hosts the web application securely and efficiently.
    - **Performance Monitoring:** Tracks app performance metrics to identify bottlenecks.
    - **Firebase Cloud Messaging:** Sends notifications to users for enhanced engagement.
    - **Crashlytics:** Reports app crashes and stability issues for timely fixes.
    - **Cloud Firestore:** NoSQL database for real-time data storage and synchronisation.

    - **Cloud Storage:** Stores user-generated content like images and videos.
    - **Realtime Database**: Provides real-time data syncing for instant updates across devices.

DevOps Technology

- **GitHub Actions:** Automates code deployment and infrastructure management to streamline development.

Third-Party Integrations

- **Stripe**: Integrates payment processing to facilitate secure transactions within the app.
- **Leaflet**: Implements interactive maps for location-based features.
- **Twilio**: Sends automated SMS notifications to enhance user communication.
- **SendGrid**: Manages automated email notifications for user updates and marketing outreach.

Coding Environment

- **VS Code**: Code editor for developing and debugging the application efficiently.
- **Git**: Version control system for tracking code changes and collaboration.
- **GitHub**: Hosts repositories and facilitates issue management for team collaboration.
- **Node.js**: JavaScript runtime for executing server-side code and development tools.
- **npm**: Package manager for managing project dependencies and libraries.

Testing

- **Jest**: Used for unit testing React components to ensure functionality.
- **React Testing Library:** Tests user interactions and component behaviour for a user-centric approach.
- **Firebase Emulator Suite**: Enables local testing of Firebase services without impacting production.

Project Scheduling

Task	Nov 1-15	Nov 16-30	Dec 1-15	Dec 16-31	Jan 1-15	Jan 16-31	Feb 1-15	Feb 16-29	Mar 1-15	Mar 16-31
P0 Features										
User Registration and Authentication	🟦									
Driveway Listing Creation and Management		🟦								
Search and Filtering		🟦	🟦							
In-App Communication		🟦								
Booking and Payment Management			🟦	🟦						
Ratings and Reviews for Credibility			🟦							
Development & Feature Enhancements										
Additional Feature Development (P1-P3)			🟦	🟦	🟦					
Feature Refinement & Minor Enhancements							🟦	🟦		
Testing										
Unit & Integration Testing (P0 Features)			🟦	🟦						
Functional Testing (P1-P2 Features)					🟦	🟦				
User Acceptance Testing (All Features)									🟦	🟦
Final Adjustments & Launch Prep										
Bug Fixes & Optimization								🟪	🟪	
Final Testing & Pre-Launch Review									🟪	🟪

```
project-root/
│
├── frontend/
│   ├── web/                # Web app code
│   └── mobile/             # Mobile app code
│
├── backend/                # Backend code
│
├── shared/                 # Shared resources (like utility functions or libraries)
│
├── ci-cd/                  # CI/CD configuration (if not managed elsewhere)
│
└── docs/                   # Documentation for team and project
```