WayPoint

CSIS 4495

Section 1

Russell Han Josef - 300369073

Simone Lue (Team Lead) - 300276605

Introduction

Travel planning is often a complex and time-consuming process, especially for travelers seeking personalized experiences. Many existing travel applications offer generic recommendations that fail to align with individual preferences, making itinerary creation an overwhelming task. WayPoint aims to bridge this gap by leveraging AI-powered recommendations, geolocation services, and real-time interactivity to enhance travel planning efficiency.

Existing research highlights the increasing role of AI in the travel sector, with applications like Google Travel and TripIt streamlining planning. However, these platforms lack in-depth personalization, leaving a gap for solutions that cater to specific traveler styles. Our research addresses this gap by developing a personalized travel planning app that dynamically tailors recommendations to user preferences.

Our initial hypotheses include:

- Users will engage with a travel style quiz to receive tailored recommendations.
- Integration of APIs such as Google Places, Eventbrite, and OpenWeatherMap will enrich the travel experience.
- AI-driven recommendations will streamline trip planning and increase user satisfaction.
- Real-time updates and offline mode will enhance usability.

Potential Benefits:

- For travelers: Enhanced personalization, reduced research time, and enriched travel experiences.
- For BC's tourism industry: Increased user engagement with local attractions and businesses.
- For research: A scalable model for personalized travel applications applicable to other regions.

Summary of Research Project

The proposed research project focuses on the development of WayPoint, a mobile application that simplifies travel planning through AI-powered recommendations and interactive features. WayPoint is tailored specifically for visitors to British Columbia, focusing on simplifying the process of itinerary creation and enhancing user experience. The primary objectives of this project are to deliver a personalized and efficient solution that caters to individual preferences and travel styles, integrate diverse data sources into a cohesive platform, and leverage advanced technologies such as AI-driven recommendations and real-time geolocation services.

The final form of WayPoint includes the following:

- User Profiles: Users can set up their own profile with detailed sections such as About, Fun Facts, Travel Behaviours, and Planning Habits
- Travel Style Quiz: Users set preferences through a quiz that determines their travel style
- Personalized Recommendations: Suggests made are based on user preferences and real-time geolocations
- Interactive Map: A visual representation of recommendations using Google Maps API
- Gamification: Users earn badges for exploring and logging visits at category-specific locations
- Chatbot Assistance: AI-driven travel tips
- Friends and Public Profiles: Users can add friends by email and view their friend's public profile page
- Personal and Collaborative Itinerary Building: Building itineraries for personal use or in collaboration with friends. Users have the ability to use a budgeting tool and add places to itineraries directly from the Interactive maps.

The tech stack includes:

- Frontend: React Native (CLI) for UI and user interactions.
- Backend: FastAPI (Python) for database and API integrations.
- Databases: PostgreSQL (Heroku) and Firebase for real-time updates.
- APIs: Google Places, Google Maps, OpenAI, OpenWeatherMap.
- Storage: AWS

Paid Services:

- OpenAI

Free Credit Services:

- Heroku - Student Program

Changes to the Proposal

1. Shift from React Native Expo to React Native CLI
   - Reason: Expo had limitations with native modules required for Firebase and API integrations.
   - Justification: React Native CLI provided more flexibility for dependency management and native module integration.
2. Modification of Database Architecture
   - Initial Approach: Firebase-only for real-time data.
   - New Approach: Hybrid approach using PostgreSQL for structured data and Firebase for real-time updates.
   - Justification: Ensures structured data integrity while allowing real-time updates for collaborative trip planning.

3. Revised MVP Priorities Based on User Surveys

- Initial MVP: All features developed without priority. Development based on what team members deemed fit.
- Revised Approach: Prioritized features based on user feedback. There were certain features that users wanted more than others and those are now the priority features to work on. We sorted the remaining features into “medium priorities” and “nice-to-haves".

4. API Integration Adjustments

- Initial Plan: Use Google Places API without caching.
- New Plan: Store Google Places data in PostgreSQL to reduce API calls and improve performance.
- Justification: Avoid exceeding API request limits and speed up app performance.

5. Task Allocation

- Initial Plan: Simone takes care of frontend majority; Russell takes care of backend majority.
- New Plan: Simone and Russell assigned one MVP component each week during the meetings which involves backend, frontend, and integration.
- Justification: Both Simone and Russell can have equal learning opportunities for frontend and backend. In addition, it makes developing MVP components smoother as there is no handoff on an incomplete feature. This also reduces the time it takes to develop the feature.

6. Event Search Feature

- Initial Plan: Feature will show local events from EventBrite API, displayed in the homescreen.
- New Plan: Feature will no longer be developed.

- Justification: The EventBrite Search API has been discontinued. Explored other event-related APIs but no other suitable event searching API could be found.

7. Google Places API
   - Initial Plan: Use of legacy Google Places API. Able to use google images for places retrieved.
   - New Plan: Migration to Google Places (New) API. Places retrieved no longer displays associated image.
   - Justification: Google Places required legacy users to migrate to the New API. Usage of API incurred unexpected high costs. Unable to retrieve places photos with new Places API to reduce API calls and usage.

8. AWS S3 Image Storage
   - Initial Plan: Use Firebase Storage for storing images
   - New Plan: Use of AWS S3 for image storage.
   - Justification: Lots of unexpected issues and errors with implementation and integration of Firebase Storage. Difficulty in handling file uploads and permission settings. Tried AWS as the alternative path and had success with implementation. Decided to proceed with AWS as a lot of time was spent trying to implement Firebase Storage without success.

<u>Project Completion Timeline</u>

Updated Timeline (Beginning - End of Term)

Jan 24 – 29: Design Phase

- Deliverables:
    - Wireframes showcasing the app's user interface and flow.

    - A well-documented database schema outlining the structure for user data, itineraries, and recommendations.

Jan 30 – Feb 5: Project Setup

- Deliverables:
    - A ready-to-use centralized GitHub repository for version control.
    - Fully configured frontend and backend environments ready for development.

Feb 6 – Apr 3: Development of Minimal Viable Product

- Responsibilities:
    - Each team member assigned a MVP feature every week during the weekly meetings.
    - Each MVP feature taken from beginning to the end by the assigned team member (frontend + backend)
    - Simone: Lead on frontend
    - Russell: Lead on backend
- Milestone Features:
    - Travel Style Quiz (Feb 6-11)
    - Personalized Recommendations (Feb 11-25)
    - Interactive Map (Feb 11-25)
    - User Profile Management (Feb 25-Mar 11)
    - Itinerary Planning (Feb 25-Mar 11)
    - Chatbot Assistance (Mar 11-25)
    - Gamified Exploration (Mar 11-25)
    - Friends Feature (Mar 25 – Apr 1)
    - UI Clean up (March 31 – Apr 3)

- Weekly Deliverables:
    - (Feb 6-11): Users can take a quiz to determine their travel style (e.g., Relaxation, Culture, Adventure). Results are saved to the database to be used in other features.
    - (Feb 11-25): Tailored suggestions for destinations, activities, and events based on user preferences and location, using Google Places API.
    - (Feb 11-25): Visualize recommended places on an interactive map using Google Maps API.
    - (Feb 25-Mar 11): Users can create an account, manage profiles, and save preferences (e.g., travel style, favorite destinations).
    - (Feb 25-Mar 11): Users can create and edit trip itineraries in real-time, with live updates and syncing. Collaborative itineraries with friends.
    - (Mar 11-25) A simple chatbot powered by OpenAI to assist users with travel-related FAQs and personalized advice.
    - (Mar 11-25): Users earn badges and collectibles by logging visits at category-specific destinations, with real-time updates.
    - (Mar 25 – Apr 1): Users can find and friends by email and view their friend's public profiles.
    - (March 31 – Apr 3): Clean-up resolved inconsistencies in UI. Adjusted layout to a favourable outcome.

Mar 31 - Apr 3: Integration

- Responsibilities:
    - Simone and Russell have confirmed all features are well integrated together.
    - Feature integrations have been divided between the team members during the weekly meeting.

    - Both team members have ensured the application works on both their own separate devices.
- Milestones:
    - Frontend and backend integration completed
    - Backend deployed to Heroku
- Deliverables:
    - A seamlessly integrated application working on iOS.

Apr 3-6: User Testing Phase

- Responsibilities:
    - Both team members have reached out to a total of 6 users for user testing
    - Both team members contributed to developing a structured user testing interview
    - Both team members have created an analysis of the user testing results with a summary of the next steps for adjustments.
- Milestones:
    - User testing for Waypoint
    - Noting of any problems/bugs/issues with user flow
- Deliverables:
    - User testing analysis
    - Documentation of app issues

Apr 6-11: Tweaks and Adjustments

- Responsibilities:
    - Both team members have been assigned features to adjust based on user testing results.
- Milestones:

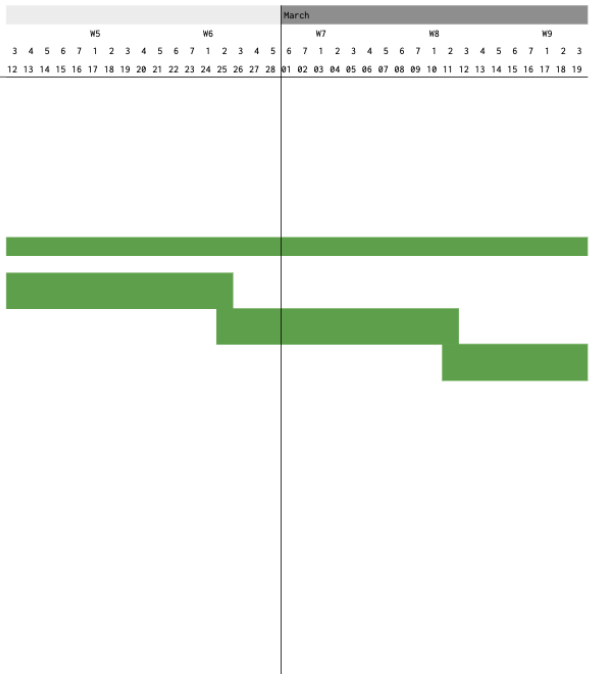
    - Fixed any issues that arose from user testing
    - Tweaked any additional features if necessary or if time permitted
- Deliverables:
    - Fully functional and user tested WayPoint deployed on iOS.

April 9-12: Documentation and Finalization

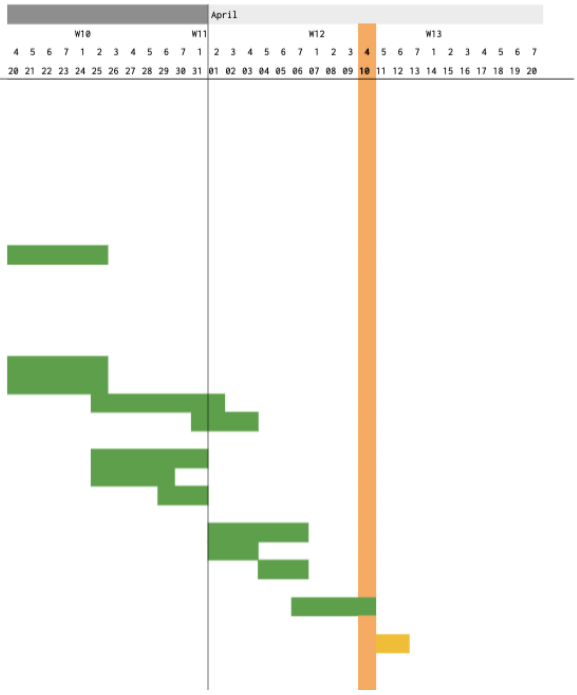
- Responsibilities:
    - Simone will handle the project submission report.
    - Russell will handle the README and user guides.
- Milestones:
    - README and user guide completed
    - Final project review and polish
- Deliverables:
    - Comprehensive user guide, project README, and developer notes.
    - A polished and fully functional version of WayPoint ready for submission.

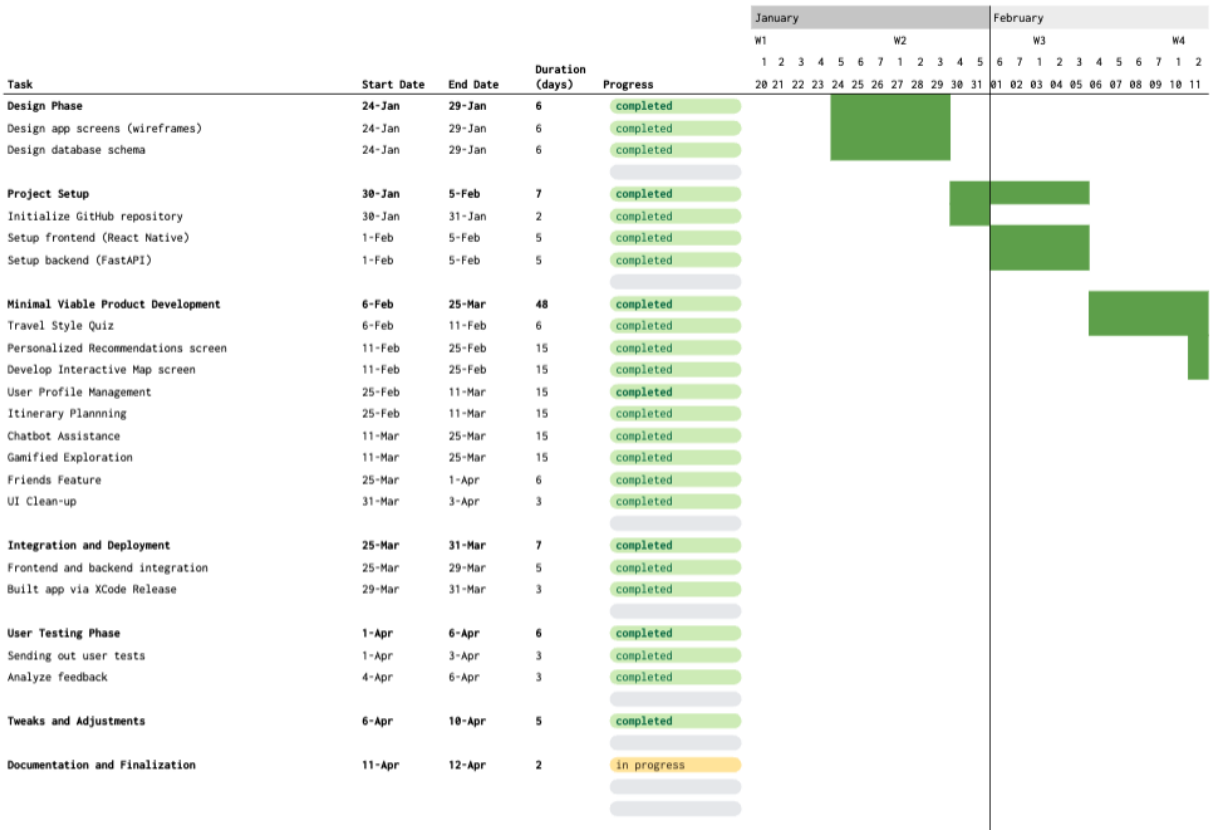
Gantt Chart:

Task	Start Date	End Date	Duration (days)	Progress
Design Phase	**24-Jan**	**29-Jan**	**6**	completed
Design app screens (wireframes)	24-Jan	29-Jan	6	completed
Design database schema	24-Jan	29-Jan	6	completed
Project Setup	**30-Jan**	**5-Feb**	**7**	completed
Initialize GitHub repository	30-Jan	31-Jan	2	completed
Setup frontend (React Native)	1-Feb	5-Feb	5	completed
Setup backend (FastAPI)	1-Feb	5-Feb	5	completed
Minimal Viable Product Development	**6-Feb**	**25-Mar**	**48**	completed
Travel Style Quiz	6-Feb	11-Feb	6	completed
Personalized Recommendations screen	11-Feb	25-Feb	15	completed
Develop Interactive Map screen	11-Feb	25-Feb	15	completed
User Profile Management	25-Feb	11-Mar	15	completed
Itinerary Plannning	25-Feb	11-Mar	15	completed
Chatbot Assistance	11-Mar	25-Mar	15	completed
Gamified Exploration	11-Mar	25-Mar	15	completed
Friends Feature	25-Mar	1-Apr	6	completed
UI Clean-up	31-Mar	3-Apr	3	completed
Integration and Deployment	**25-Mar**	**31-Mar**	**7**	completed
Frontend and backend integration	25-Mar	29-Mar	5	completed
Built app via XCode Release	29-Mar	31-Mar	3	completed
User Testing Phase	**1-Apr**	**6-Apr**	**6**	completed
Sending out user tests	1-Apr	3-Apr	3	completed
Analyze feedback	4-Apr	6-Apr	3	completed
Tweaks and Adjustments	**6-Apr**	**10-Apr**	**5**	completed
Documentation and Finalization	**11-Apr**	**12-Apr**	**2**	in progress



Task	Start Date	End Date	Duration (days)	Progress
Design Phase	**24-Jan**	**29-Jan**	**6**	completed
Design app screens (wireframes)	24-Jan	29-Jan	6	completed
Design database schema	24-Jan	29-Jan	6	completed
Project Setup	**30-Jan**	**5-Feb**	**7**	completed
Initialize GitHub repository	30-Jan	31-Jan	2	completed
Setup frontend (React Native)	1-Feb	5-Feb	5	completed
Setup backend (FastAPI)	1-Feb	5-Feb	5	completed
Minimal Viable Product Development	**6-Feb**	**25-Mar**	**48**	completed
Travel Style Quiz	6-Feb	11-Feb	6	completed
Personalized Recommendations screen	11-Feb	25-Feb	15	completed
Develop Interactive Map screen	11-Feb	25-Feb	15	completed
User Profile Management	25-Feb	11-Mar	15	completed
Itinerary Plannning	25-Feb	11-Mar	15	completed
Chatbot Assistance	11-Mar	25-Mar	15	completed
Gamified Exploration	11-Mar	25-Mar	15	completed
Friends Feature	25-Mar	1-Apr	6	completed
UI Clean-up	31-Mar	3-Apr	3	completed
Integration and Deployment	**25-Mar**	**31-Mar**	**7**	completed
Frontend and backend integration	25-Mar	29-Mar	5	completed
Built app via XCode Release	29-Mar	31-Mar	3	completed
User Testing Phase	**1-Apr**	**6-Apr**	**6**	completed
Sending out user tests	1-Apr	3-Apr	3	completed
Analyze feedback	4-Apr	6-Apr	3	completed
Tweaks and Adjustments	**6-Apr**	**10-Apr**	**5**	completed
Documentation and Finalization	**11-Apr**	**12-Apr**	**2**	in progress





Task	Start Date	End Date	Duration (days)	Progress
Design Phase	**24-Jan**	**29-Jan**	**6**	completed
Design app screens (wireframes)	24-Jan	29-Jan	6	completed
Design database schema	24-Jan	29-Jan	6	completed
Project Setup	**30-Jan**	**5-Feb**	**7**	completed
Initialize GitHub repository	30-Jan	31-Jan	2	completed
Setup frontend (React Native)	1-Feb	5-Feb	5	completed
Setup backend (FastAPI)	1-Feb	5-Feb	5	completed
Minimal Viable Product Development	**6-Feb**	**25-Mar**	**48**	completed
Travel Style Quiz	6-Feb	11-Feb	6	completed
Personalized Recommendations screen	11-Feb	25-Feb	15	completed
Develop Interactive Map screen	11-Feb	25-Feb	15	completed
User Profile Management	25-Feb	11-Mar	15	completed
Itinerary Plannning	25-Feb	11-Mar	15	completed
Chatbot Assistance	11-Mar	25-Mar	15	completed
Gamified Exploration	11-Mar	25-Mar	15	completed
Friends Feature	25-Mar	1-Apr	6	completed
UI Clean-up	31-Mar	3-Apr	3	completed
Integration and Deployment	**25-Mar**	**31-Mar**	**7**	completed
Frontend and backend integration	25-Mar	29-Mar	5	completed
Built app via XCode Release	29-Mar	31-Mar	3	completed
User Testing Phase	**1-Apr**	**6-Apr**	**6**	completed
Sending out user tests	1-Apr	3-Apr	3	completed
Analyze feedback	4-Apr	6-Apr	3	completed
Tweaks and Adjustments	**6-Apr**	**10-Apr**	**5**	completed
Documentation and Finalization	**11-Apr**	**12-Apr**	**2**	in progress

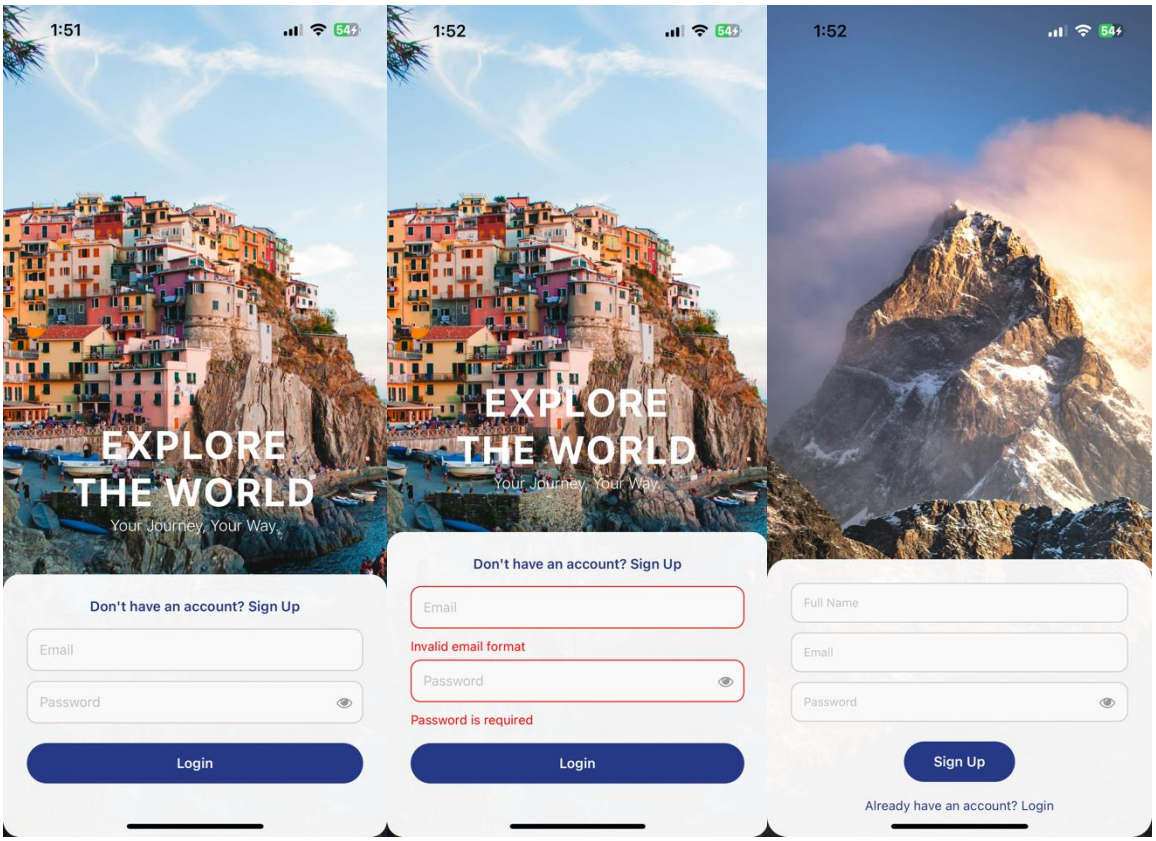
Implemented Feature: User Profile Management

- Designed and implemented the user authentication system.
- Login Process:
    - Users enter their email and password in the login form.
    - Form validation ensures the email format is correct and the password field is not empty.
    - Upon submission, credentials are sent to the FastAPI backend (/users/auth/login) for verification.
    - If authentication succeeds:
        - The user's details are stored in AsyncStorage for session persistence.
        - A login event is recorded in Firebase Realtime Database under /logins/{user_id}.

        - The last login timestamp is updated in Firebase.
        - The user is navigated to the Main Screen.
    - If authentication fails, an error message is displayed.
- Navigation Behaviour**:**
    - Successful login redirects users to the main app screen.
    - Users can navigate to the signup screen if they don't have an account.
- Error Handling and Edge Cases:
    - Displays appropriate error messages for incorrect credentials or missing fields.
    - Prevents duplicate login attempts while the request is processing.
    - Uses Alert to notify users about errors.
- Logout Functionality:
    - Users can log out via the modal “More” menu
    - Clears stored session data in AsyncStorage and navigates users back to the login screen.
- Integration with Firebase:
    - Logs login events for tracking user activity.
    - Updates the last login timestamp in Firebase Realtime Database.
- Security Considerations:
    - Ensures passwords are not stored locally.
    - Uses AsyncStorage only for non-sensitive user data.
- Connected Screens:
    - Signup Screen: Allows users to create an account.
    - Profile Screen: Displays user details and provides a logout option.

- Settings Screen: Allows users to manage preferences, such as travel style and language.

- Screenshots:



```
// ✅ Handle Login (modified to ensure `travel_style_id` is included)
const handleLogin = async () => {
  if (!validateInputs()) return;
  setLoading(true);

  try {
      const response = await axios.post(
          `${API_BASE_URL}/users/auth/login`,
          { email: email.toLowerCase(), password },
          { headers: { 'Content-Type': 'application/json' } }
      );

      if (response.status === 200) {
          const user = response.data.user;

          if (!user.travel_style_id) {
              console.warn("⚠ travel_style_id is missing from backend response");
              user.travel_style_id = 4;  // ✅ Default to Undefined if not present
          }

          // ✅ Store user details in AsyncStorage
          await storeUserSession(user);

          // ✅ Fetch and store recent itineraries (only if any exist)
          await fetchAndStoreRecentItineraries(user.id);

          // ✅ Log login event to Firebase
          await logLoginToFirebase(user.id);

          // ✅ Fetch and store profile image
          await fetchAndStoreProfileImage(user.id);

          // ✅ Navigate to HomeScreen with user details
          navigation.replace('Main', { user });
      }
  } catch (error) {
    const detail = error.response?.data?.detail;
    const newErrors = { email: null, password: null };

    if (detail === "Email not found") {
      newErrors.email = "Email not found. Try another email or sign up.";
    } else if (detail === "Incorrect password") {
      newErrors.password = "Incorrect password. Try again.";
    } else {
      if (!newErrors.email && !newErrors.password) {
        Alert.alert("Login Failed", detail || "Invalid credentials");
      }
    }
    setErrors(newErrors);

  } finally {
      setLoading(false);
  }
};
```

Implemented Feature: Travel Style Quiz

- Quiz Process:
    - Users are presented with a series of travel-related questions
    - Each question offers multiple choice answers representing Relaxation, Cultural, Adventure, or None.

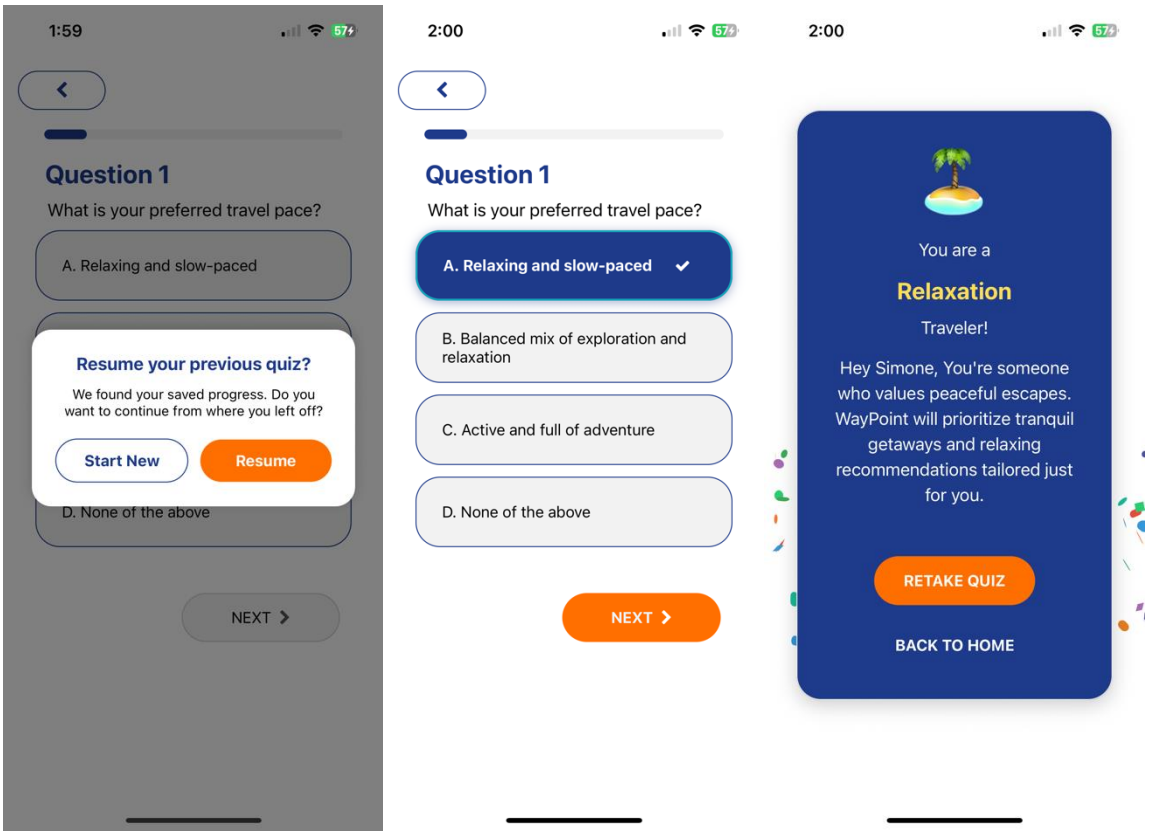
    - Validation ensures a selection is made before proceeding.
- State Management and Progress Persistence:
    - Quiz progress, including current question, selected answers, and scores, is maintained with React state.
    - Progress is saved in Firebase Realtime Database and AsyncStorage for session resumption.
    - Users are prompted to resume a saved quiz if progress exists.
- Result Determination and Presentation:
    - Upon quiz completion, scores are aggregated to determine the dominant travel style.
    - A personalized travel style, with a corresponding emoji and description, is generated.
    - A confetti animation celebrates the result, and users are offered options to retake or return home.
    - The result is sent to the FastAPI backend to create or update the user's quiz record.
- Backend Integration:
    - FastAPI endpoints handle creating, updating, and retrieving quiz results.
    - The backend assigns a travel style ID based on the quiz score and updates user data accordingly.
    - Travel style details are fetched from the backend and displayed in the UI.
- Navigation Behaviour:
    - Successful quiz completion navigates users to the results screen with options to retake the quiz or go back to home.

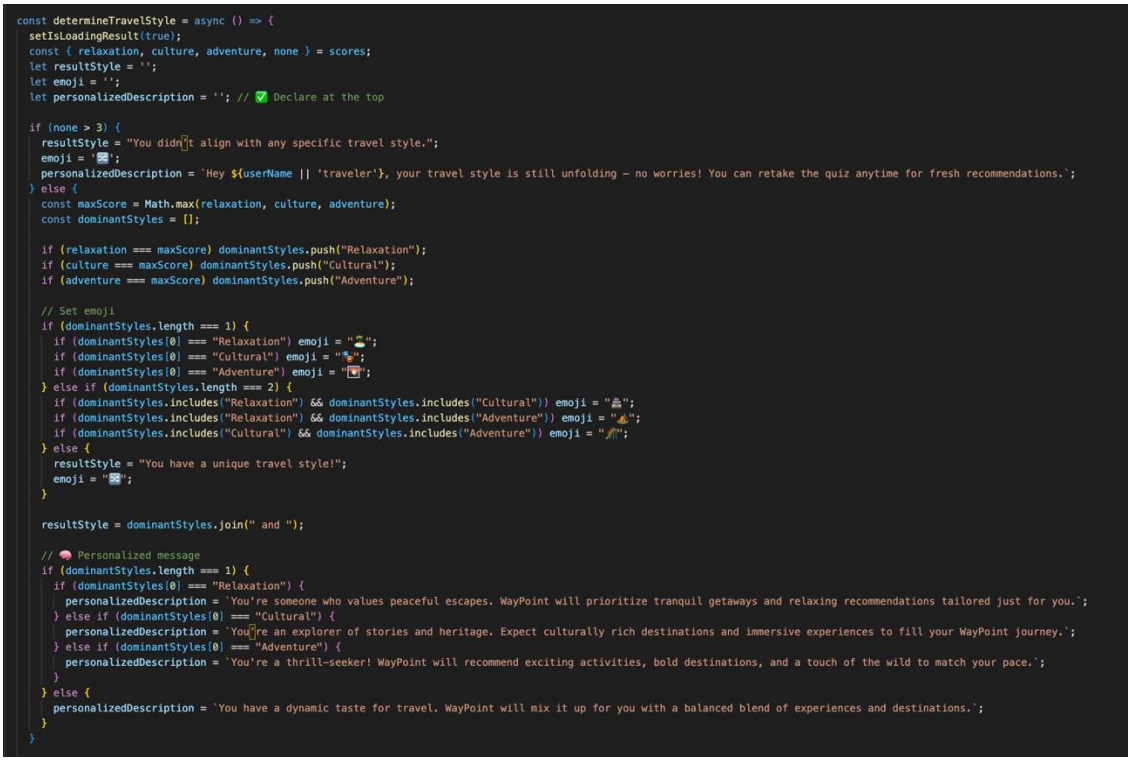
- Smooth UI transitions and interactive elements guide users throughout the quiz process.

- Error Handling and Data Consistency:
  - Prevents advancing without selecting an answer.
  - Ensures data consistency using Firebase and AsyncStorage for real-time progress tracking.

- Connected Screens:
  - Quiz Screen: Interactive interface for answering questions.
  - Home Screen: Accessible after completion to view personalized travel recommendations.

- Screenshots:



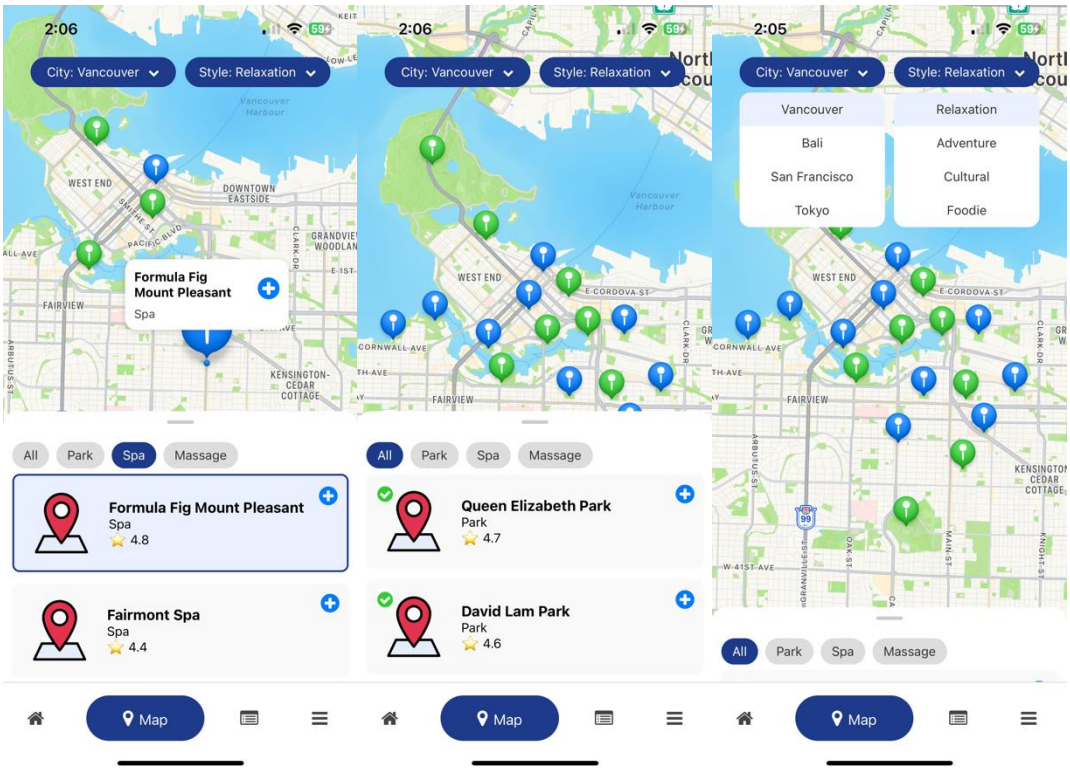


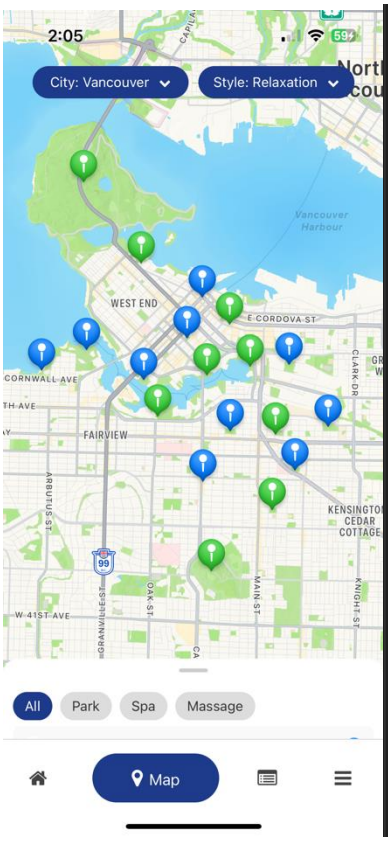
```
const determineTravelStyle = async () => {
  setIsLoadingResult(true);
  const { relaxation, culture, adventure, none } = scores;
  let resultStyle = '';
  let emoji = '';
  let personalizedDescription = ''; // ✅ Declare at the top

  if (none > 3) {
    resultStyle = "You didn't align with any specific travel style.";
    emoji = '❓';
    personalizedDescription = `Hey ${userName || 'traveler'}, your travel style is still unfolding — no worries! You can retake the quiz anytime for fresh recommendations.`;
  } else {
    const maxScore = Math.max(relaxation, culture, adventure);
    const dominantStyles = [];

    if (relaxation === maxScore) dominantStyles.push("Relaxation");
    if (culture === maxScore) dominantStyles.push("Cultural");
    if (adventure === maxScore) dominantStyles.push("Adventure");

    // Set emoji
    if (dominantStyles.length === 1) {
      if (dominantStyles[0] === "Relaxation") emoji = "🧘";
      if (dominantStyles[0] === "Cultural") emoji = "🏛️";
      if (dominantStyles[0] === "Adventure") emoji = "🌄";
    } else if (dominantStyles.length === 2) {
      if (dominantStyles.includes("Relaxation") && dominantStyles.includes("Cultural")) emoji = "🏯";
      if (dominantStyles.includes("Relaxation") && dominantStyles.includes("Adventure")) emoji = "🏕️";
      if (dominantStyles.includes("Cultural") && dominantStyles.includes("Adventure")) emoji = "🧗";
    } else {
      resultStyle = "You have a unique travel style!";
      emoji = "🌟";
    }

    resultStyle = dominantStyles.join(" and ");

    // 🧠 Personalized message
    if (dominantStyles.length === 1) {
      if (dominantStyles[0] === "Relaxation") {
        personalizedDescription = `You're someone who values peaceful escapes. WayPoint will prioritize tranquil getaways and relaxing recommendations tailored just for you.`;
      } else if (dominantStyles[0] === "Cultural") {
        personalizedDescription = `You're an explorer of stories and heritage. Expect culturally rich destinations and immersive experiences to fill your WayPoint journey.`;
      } else if (dominantStyles[0] === "Adventure") {
        personalizedDescription = `You're a thrill-seeker! WayPoint will recommend exciting activities, bold destinations, and a touch of the wild to match your pace.`;
      }
    } else {
      personalizedDescription = `You have a dynamic taste for travel. WayPoint will mix it up for you with a balanced blend of experiences and destinations.`;
    }
  }
```

Implemented Feature: Interactive Recommendations

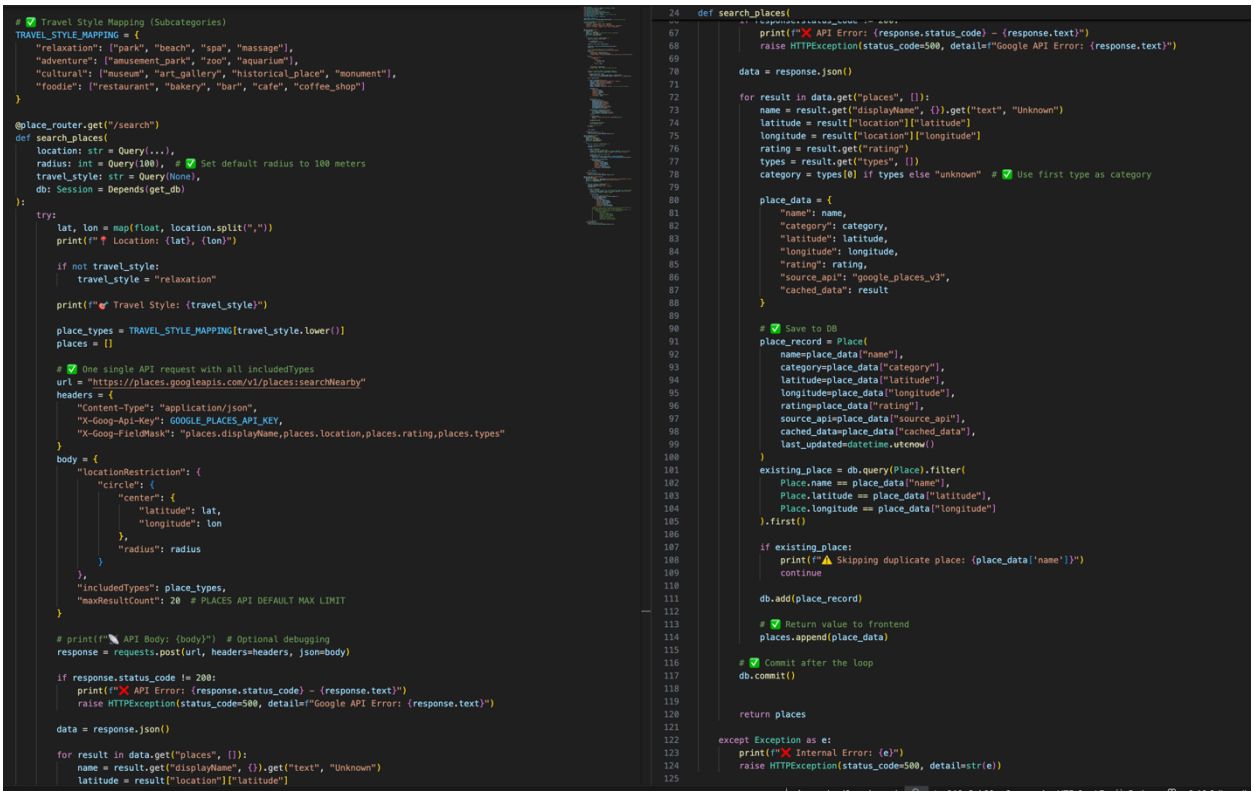
- Designed and implemented a combined interactive map and personalized recommendations screen using the new Google Places API.
- Fetching and Displaying Recommendations:
    - Users receive location-based travel recommendations based on their travel style (e.g., Relaxation, Adventure, Cultural, Foodie) and current location (default: Vancouver, BC).
    - The FastAPI backend retrieves makes requests to the new Google Places API and stored in the cached database.
    - Image previews from Google Places are no longer supported; a default placeholder image is used instead.
- Interactive Map and Filtering:

    - An interactive map displays recommended places as markers with dynamic callouts.
    - Tapping a marker focuses the map and scrolls the recommendations list to the corresponding place.
    - Filters allow users to narrow recommendations by travel style and city.
    - A bottom sheet provides a scrollable list view with place details (name, category, rating).
    - Different coloured markers are used for places that have already been added to an existing itinerary
- Performance and UX Enhancements:
    - Utilizes AsyncStorage to manage user preferences and session data for personalized recommendations.
    - Optimized API calls through caching to reduce redundant requests.
    - Loading indicators enhance the user experience during data fetches.
    - Smooth UI transitions and interactive elements guide users throughout the experience.
- Integration and Navigation:
    - Combines map and list view within a single screen for seamless exploration.
    - Users can add places to their itinerary through an integrated modal interface.
- Error Handling and Data Consistency:
    - Displays appropriate error messages if fetching fails.
    - Ensures efficient use of API resources by validating region boundaries before requesting data.
- Screenshots:






```
const fetchPlaces = async () => {
  setLoading(true);
  try {
    const response = await axios.get(`${API_BASE_URL}/places/cached`, {
      params: {
        location: `${region.latitude},${region.longitude}`,
        radius: 5000,
        travel_style: travelStyle
      }
    });
    setMapPlaces(response.data);
  } catch (error) {
    console.error("Error fetching places:", error);
  }
  setLoading(false);
};

const filteredPlaces = selectedCategory
  ? mapPlaces.filter(place => place.category === selectedCategory)
  : mapPlaces;

const capitalize = (s) => s.charAt(0).toUpperCase() + s.slice(1);
const isPlaceAlreadySaved = (placeName) => {
  return Object.values(savedPlacesByItinerary).some(placeList =>
    Array.isArray(placeList) && placeList.includes(placeName)
  );
};

const focusMapOnPlace = (place, index) => {
  setActiveIndex(index);

  // Do NOT animate or change region — just show callout and scroll
  if (markerRefs.current[index]?.current) {
    markerRefs.current[index].current.showCallout();
  }

  if (cardRefs.current[index] && scrollViewRef.current) {
    cardRefs.current[index].measureLayout(
      scrollViewRef.current,
      (x, y) => {
        scrollViewRef.current.scrollTo({ y, animated: true });
      },
      (error) => console.error("❌ Scroll error:", error)
    );
  }
};
```



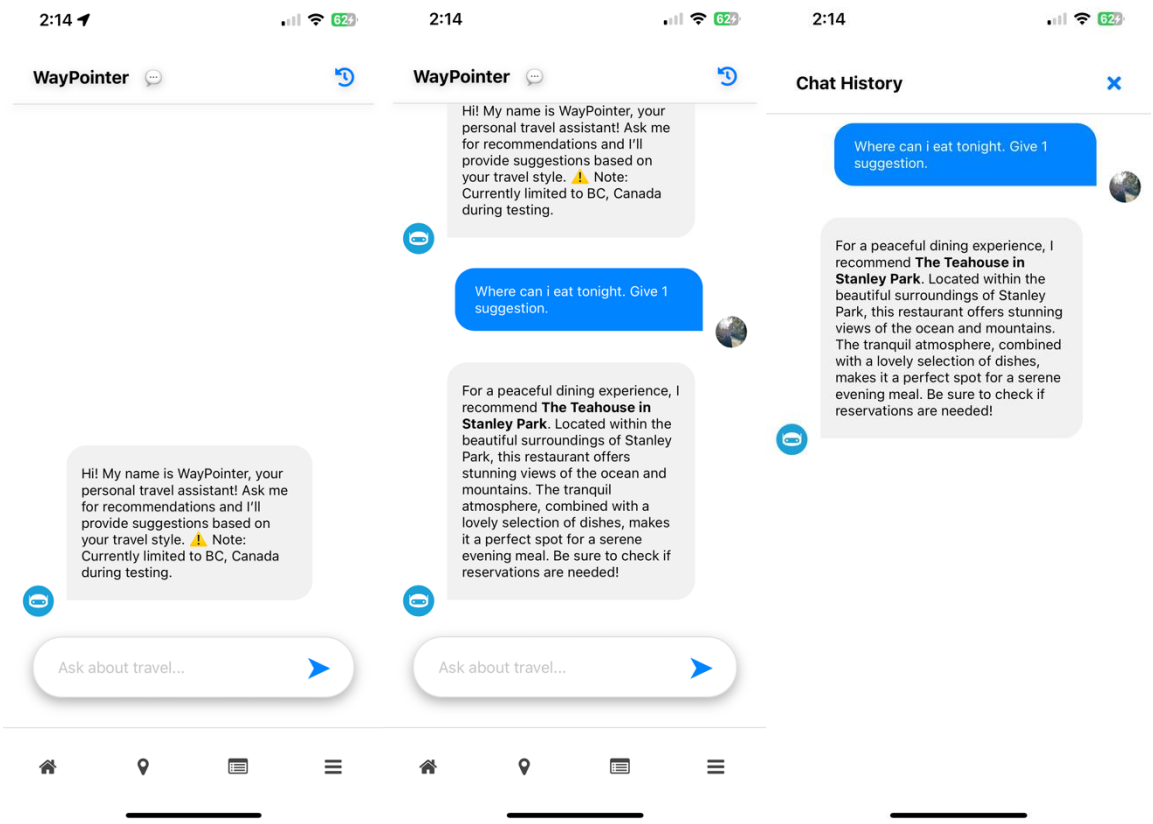
```python
# ✅ Travel Style Mapping (Subcategories)
TRAVEL_STYLE_MAPPING = {
    "relaxation": ["park", "beach", "spa", "massage"],
    "adventure": ["amusement_park", "zoo", "aquarium"],
    "cultural": ["museum", "art_gallery", "historical_place", "monument"],
    "foodie": ["restaurant", "bakery", "bar", "cafe", "coffee_shop"]
}

@place_router.get("/search")
def search_places(
    location: str = Query(...),
    radius: int = Query(100),  # ✅ Set default radius to 100 meters
    travel_style: str = Query(None),
    db: Session = Depends(get_db)
):
    try:
        lat, lon = map(float, location.split(","))
        print(f"📍 Location: {lat}, {lon}")

        if not travel_style:
            travel_style = "relaxation"

        print(f"🎨 Travel Style: {travel_style}")

        place_types = TRAVEL_STYLE_MAPPING[travel_style.lower()]
        places = []

        # ✅ One single API request with all includedTypes
        url = "https://places.googleapis.com/v1/places:searchNearby"
        headers = {
            "Content-Type": "application/json",
            "X-Goog-Api-Key": GOOGLE_PLACES_API_KEY,
            "X-Goog-FieldMask": "places.displayName,places.location,places.rating,places.types"
        }
        body = {
            "locationRestriction": {
                "circle": {
                    "center": {
                        "latitude": lat,
                        "longitude": lon
                    },
                    "radius": radius
                }
            },
            "includedTypes": place_types,
            "maxResultCount": 20  # PLACES API DEFAULT MAX LIMIT
        }

        # print(f"🔪 API Body: {body}")  # Optional debugging
        response = requests.post(url, headers=headers, json=body)

        if response.status_code != 200:
            print(f"❌ API Error: {response.status_code} - {response.text}")
            raise HTTPException(status_code=500, detail=f"Google API Error: {response.text}")

        data = response.json()

        for result in data.get("places", []):
            name = result.get("displayName", {}).get("text", "Unknown")
            latitude = result["location"]["latitude"]
```

```python
def search_places(
            print(f"❌ API Error: {response.status_code} - {response.text}")
            raise HTTPException(status_code=500, detail=f"Google API Error: {response.text}")

        data = response.json()

        for result in data.get("places", []):
            name = result.get("displayName", {}).get("text", "Unknown")
            latitude = result["location"]["latitude"]
            longitude = result["location"]["longitude"]
            rating = result.get("rating")
            types = result.get("types", [])
            category = types[0] if types else "unknown"  # ✅ Use first type as category

            place_data = {
                "name": name,
                "category": category,
                "latitude": latitude,
                "longitude": longitude,
                "rating": rating,
                "source_api": "google_places_v3",
                "cached_data": result
            }

            # ✅ Save to DB
            place_record = Place(
                name=place_data["name"],
                category=place_data["category"],
                latitude=place_data["latitude"],
                longitude=place_data["longitude"],
                rating=place_data["rating"],
                source_api=place_data["source_api"],
                cached_data=place_data["cached_data"],
                last_updated=datetime.utcnow()
            )
            existing_place = db.query(Place).filter(
                Place.name == place_data["name"],
                Place.latitude == place_data["latitude"],
                Place.longitude == place_data["longitude"]
            ).first()

            if existing_place:
                print(f"⚠️ Skipping duplicate place: {place_data['name']}")
                continue

            db.add(place_record)

            # ✅ Return value to frontend
            places.append(place_data)

        # ✅ Commit after the loop
        db.commit()


        return places

    except Exception as e:
        print(f"❌ Internal Error: {e}")
        raise HTTPException(status_code=500, detail=str(e))
```

Implemented Feature: Chatbot

- Chatbot Interaction:
    - Users converse with WayPointer via a dedicated chat screen where they ask travel-related questions.
    - A welcome message introduces WayPointer as a personal travel assistant focused on BC, Canada.
    - Chat history is persisted locally using AsyncStorage and can be reviewed in a modal.
- Processing and Backend Integration:
    - Chatbot requests are sent to FastAPI endpoints that interface with OpenAI's API.

    - The backend generates tailored system prompts based on the user's travel style (e.g., relaxation, adventure, cultural) to provide specific travel recommendations.
    - API keys are securely managed with environment variables, ensuring safe access to the OpenAI service.
- User Experience and Error Handling:
    - The chat UI dynamically displays messages with markdown formatting and animated typing indicators.
    - User input is handled efficiently, with real-time feedback and session persistence.
    - Error handling in both the UI and backend ensures that informative messages are shown if a request fails or the travel style is not yet loaded.
- Integration with Other Features:
    - The chatbot works in tandem with user profile and travel style features to deliver personalized travel recommendations.
    - Firebase is used to track if the user has engaged with the chatbot, providing insights on usage within the app.
- Screenshots:



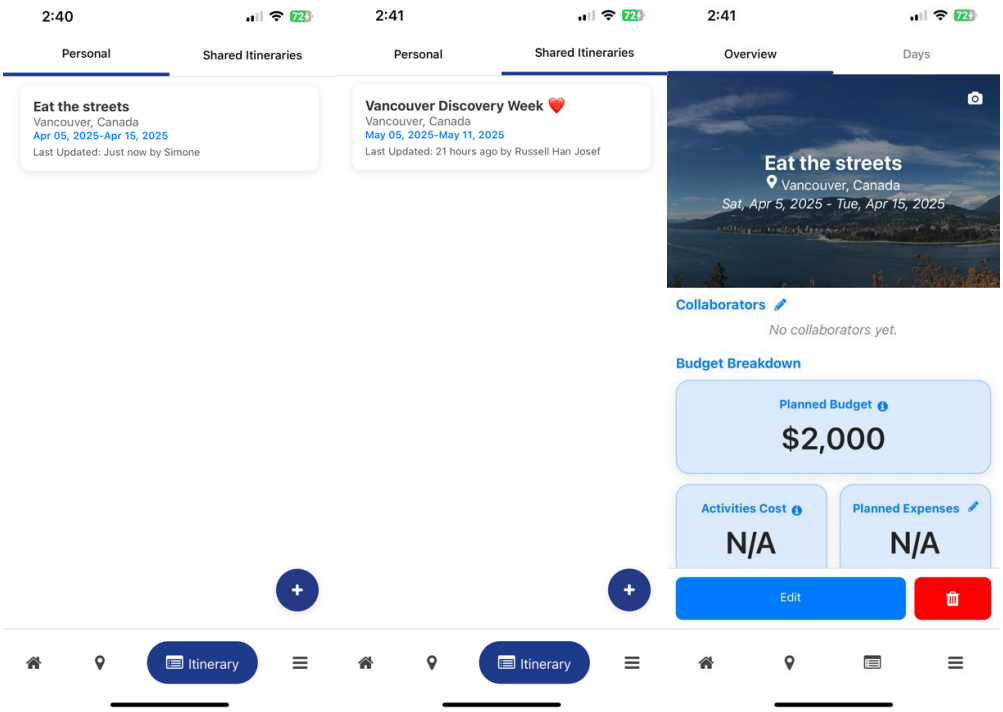

```python
def get_system_prompt(travel_style):
    base_prompt = "Your name is Waypointer, a travel assistant for Vancouver, British Columbia, Canada."

    style_prompts = {
        "relaxation": "Focus on suggesting quiet, peaceful, and scenic locations like spas, beaches, and tranquil parks. Provide specific place suggestions.",
        "adventure": "Recommend thrilling activities such as hiking, kayaking, zip-lining, and outdoor exploration. Provide specific place suggestions.",
        "cultural": "Suggest historical sites, museums, art galleries, and local cultural experiences. Provide specific place suggestions.",
    }

    additional_prompt = "Please limit to 3 suggestions unless specified in my request. Provide specific place suggestions."

    style_message = style_prompts.get(travel_style.lower(), "Provide general travel recommendations.")

    return f"{base_prompt} {style_message} {additional_prompt}"

@chatbot_router.post("/")
async def chatbot_interaction(request: ChatbotRequest):
    try:
        system_prompt = get_system_prompt(request.travel_style)

        completion = client.chat.completions.create(
            model="gpt-4o-mini",
            messages=[
                {"role": "system", "content": system_prompt},
                {"role": "user", "content": request.user_message}
                ]
        )
        # Debugging: Add travel style in response to confirm user travel style retrieval
        #response_text = completion.choices[0].message.content
        #formatted_response = f"For {request.travel_style} lovers: {response_text}"
        #return {"response": formatted_response}

        return {"response": completion.choices[0].message.content}

    except Exception as e:
        raise HTTPException(status_code=500, detail=str(e))
```

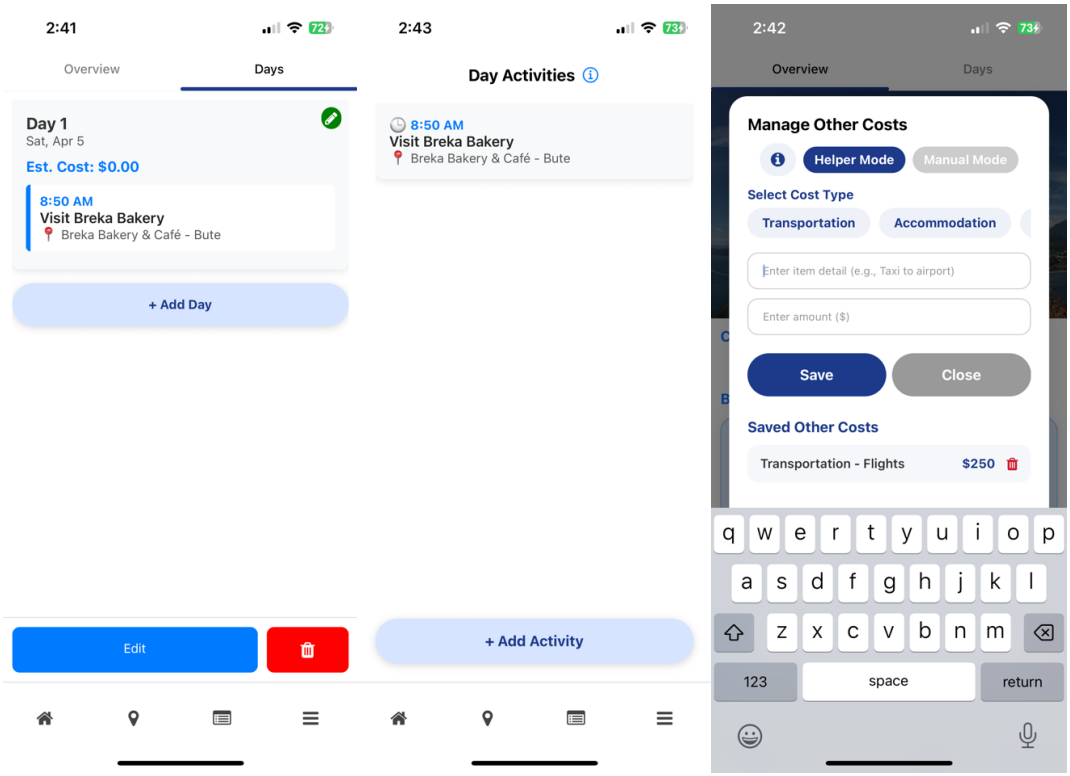
Implemented Feature: Itineraries

- Itinerary Management Process:
    - Users create new itineraries by providing key details—name, destination, travel dates, travel style, budget, and additional data (e.g., image URLs).
    - Each itinerary is assigned a unique UUID, with created_by and last_updated_by fields tracking user contributions.
- Itinerary Day and Activity Handling:
    - Users add itinerary days in a specified order, which can later be reordered via dedicated endpoints.
    - Within each day, activities (with details like time, name, location, notes, and estimated cost) can be added, updated, or deleted.
    - Time values are uniformly formatted to ensure consistent display and accurate sorting.
- Image Management and S3 Integration:
    - When extra_data includes an image URL, the backend extracts the S3 key and generates a presigned URL, ensuring secure and temporary image access.
- Collaboration and Membership:
    - Users invite collaborators to join itineraries via endpoints that add members and track pending invites in Firebase.
    - The system prevents duplicate invitations and allows collaborators to be removed as needed.
- Data Consistency and Update Operations:
    - Robust endpoints update itinerary details, reorder days, and handle deletions (of itineraries, days, or activities).

    - Metadata such as updated_at and last_updated_by is maintained with every change to ensure data integrity.
- Integration with Other App Features:
    - The itinerary feature seamlessly integrates with list screens, detail screens, day screens, and form screens.
    - Real-time updates using Firebase enhance collaborative planning and invitation management.
    - API endpoints support both personal and shared itineraries, providing a comprehensive planning experience.
- Additional Enhancements:
    - Destination Search:
        - A dedicated Destination Search Modal enables users to search for destination regions via autocomplete powered by the Google Places API.
        - Debouncing is implemented to optimize search performance as users type, and selected destinations are parsed to extract city and country details for the itinerary.
    - Other Costs Management:
        - An Other Costs Modal allows users to manage additional expenses by entering cost items.
        - It supports both helper mode (selecting from predefined cost categories and subtypes) and manual mode (custom category input).
        - New cost entries are saved to Firebase with visual confirmation through animations.
    - Overview and Places Integration:

        - The Overview Tab provides a comprehensive itinerary summary displaying the trip's image (with editable camera option), name, destination, dates, and collaborator details.
        - It features interactive budget breakdowns (planned budget, activities cost, and planned expenses), with info modals to explain each section.
        - A dedicated Places Modal allows users to add, delete, and save a list of places to visit, which integrates into the itinerary for further planning.
- Error Handling and Robustness:
    - Backend endpoints validate input data (e.g., ensuring required fields are provided and that referenced IDs exist) and return appropriate HTTP status codes and error messages when errors occur.
    - Exceptions in create, update, and delete operations are caught, with error details logged and informative responses delivered to the client.
    - Collaboration and Firebase update operations include error handling to manage failed requests, preventing data inconsistencies.
    - On the client side, missing or invalid fields trigger alerts and user-friendly error messages, guiding users to correct input errors.
- Screenshots:


2:40
Personal
Shared Itineraries
Eat the streets
Vancouver, Canada
Apr 05, 2025-Apr 15, 2025
Last Updated: Just now by Simone
Itinerary
2:41
Personal
Shared Itineraries
Vancouver Discovery Week ❤️
Vancouver, Canada
May 05, 2025-May 11, 2025
Last Updated: 21 hours ago by Russell Han Josef
Itinerary
2:41
Overview
Days
Eat the streets
Vancouver, Canada
Sat, Apr 5, 2025 - Tue, Apr 15, 2025
Collaborators
No collaborators yet.
Budget Breakdown
Planned Budget
$2,000
Activities Cost
N/A
Planned Expenses
N/A
Edit


2:41
Overview
Days
Day 1
Sat, Apr 5
Est. Cost: $0.00
8:50 AM
Visit Breka Bakery
Breka Bakery & Café - Bute
+ Add Day
Edit
2:43
Day Activities
8:50 AM
Visit Breka Bakery
Breka Bakery & Café - Bute
+ Add Activity
2:42
Overview
Days
Manage Other Costs
Helper Mode
Manual Mode
Select Cost Type
Transportation
Accommodation
Enter item detail (e.g., Taxi to airport)
Enter amount ($)
Save
Close
Saved Other Costs
Transportation - Flights
$250

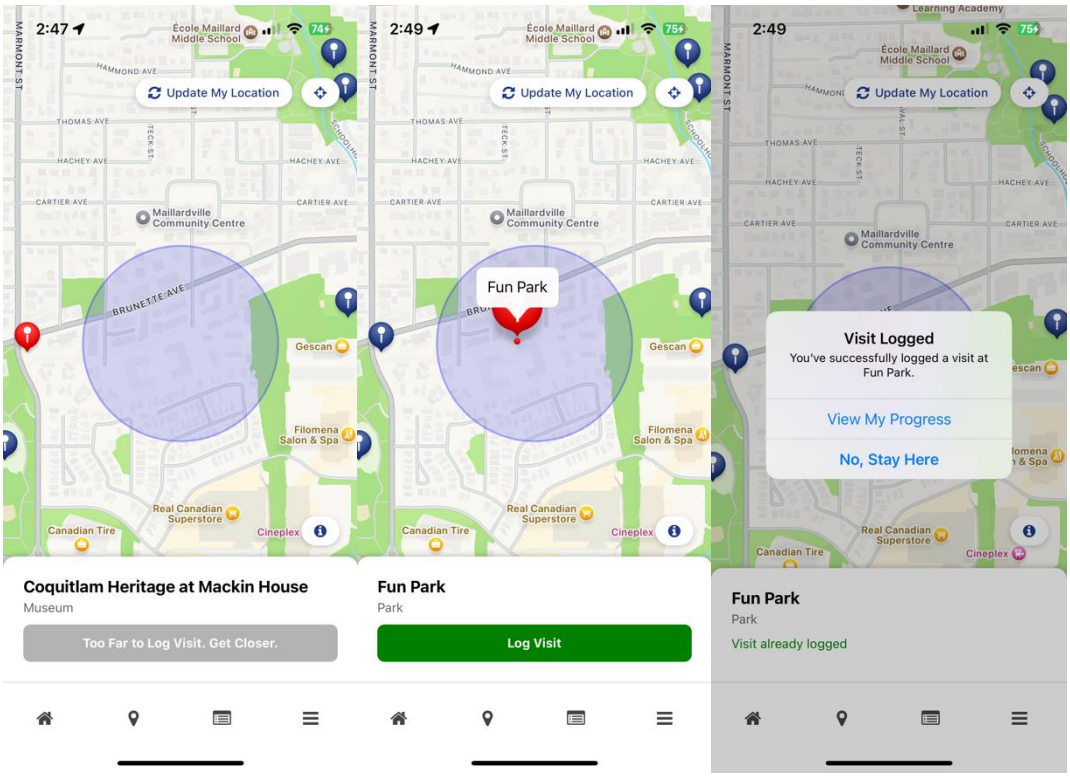
```
// ✅ Fetch Owned Itineraries from PostgreSQL
const fetchOwnedItineraries = async (userId) => {
    try {
        const response = await axios.get(`${API_BASE_URL}/itineraries/users/${userId}/itineraries`);
        if (response.status === 200) {
            setOwnedItineraries(response.data);
        }
    } catch (error) {
        console.error("❌ Error fetching owned itineraries:", error.response?.data || error.message);
    } finally {
        setLoading(false);
    }
};

// ✅ Fetch Shared Itineraries from Firebase Realtime Database
const fetchSharedItineraries = async (userId) => {
    try {
        const snapshot = await database().ref('/live_itineraries').once('value');

        if (!snapshot.exists()) {
            setSharedItineraries([]);
            return;
        }

        const data = snapshot.val();
        const itineraryIds = Object.keys(data).filter(itineraryId =>
            data[itineraryId].collaborators && data[itineraryId].collaborators[userId]
        );

        if (itineraryIds.length === 0) {
            setSharedItineraries([]);
            return;
        }

        // ✅ Fetch each itinerary from FastAPI
        const itineraryPromises = itineraryIds.map(async (itineraryId) => {
            try {
                const response = await axios.get(`${API_BASE_URL}/itineraries/${itineraryId}`);
                return response.status === 200 ? response.data : null;
            } catch (error) {
                console.error(`❌ Error fetching itinerary ${itineraryId}:`, error.response?.data || error.message);
                return null;
            }
        });

        const fullItineraries = (await Promise.all(itineraryPromises)).filter(Boolean);
        setSharedItineraries(fullItineraries);
        console.log("✅ Updated shared itineraries:", fullItineraries);
    } catch (error) {
        console.error("❌ Error fetching shared itineraries:", error.response?.data || error.message);
    }
};
```

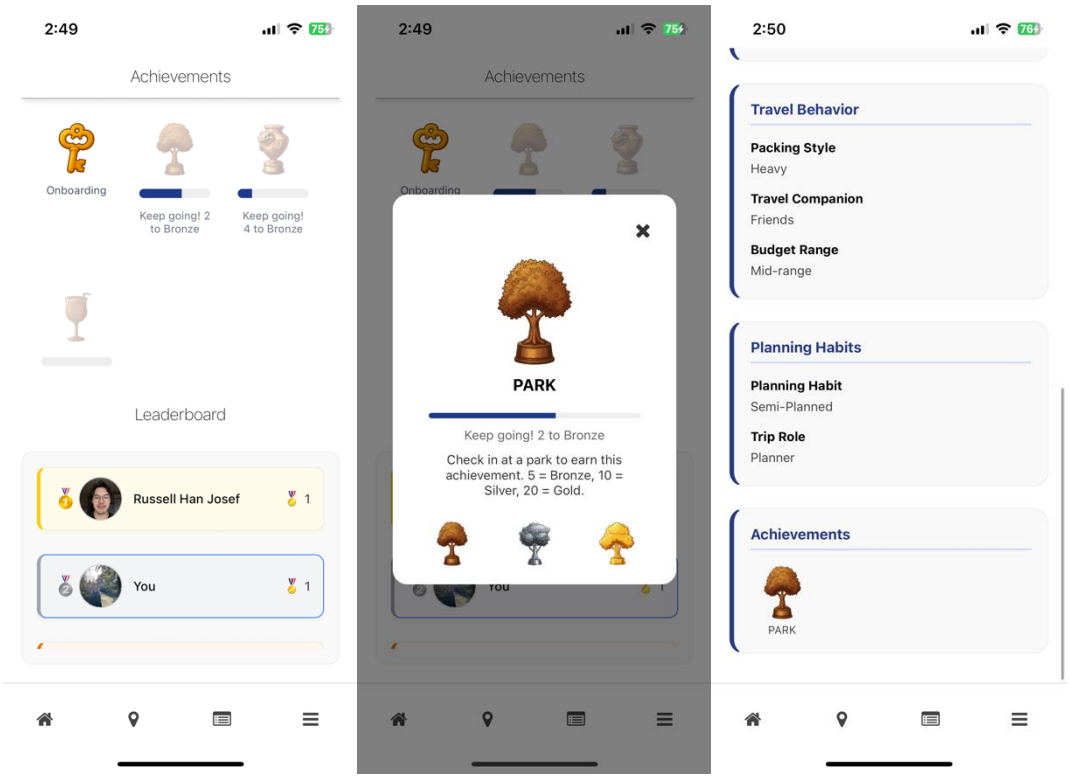
Implemented Feature: Gamified Exploration

- Overview:
    - Integrates location-based check-ins where users log visits at real-world locations and earn rewards.
    - Designed to encourage exploration through progressive badge rewards for repeated visits.
- Check-In Process:

    - Users access a map interface that displays nearby locations filtered by category (e.g., parks, bars, museums).
    - Geolocation is used to obtain the user's current position. When within a defined proximity (e.g., within 200 meters) of a location, a "Log Visit" option becomes available.
    - The system verifies that the user hasn't already checked into the selected place.
- Reward Mechanism:
    - Each check-in is recorded in Firebase under the user's gamification data.
    - Visit counts are tallied by category. Based on thresholds (5 visits for Bronze, 10 for Silver, and 20 for Gold), users earn badges.
    - Newly unlocked badges trigger an animated modal with trophy images, celebrating the achievement.
    - Badge information (tier and category) is also utilized in an Achievements screen displaying progress, leaderboards, and friends' badges.
- Backend Integration:
    - Check-in data is submitted using a unique UUID, timestamp, and place identifier, then stored in Firebase.
    - A dedicated API endpoint (from the places routes) provides filtered nearby locations, ensuring only allowed categories appear on the map.
- Error Handling and Robustness:
    - If geolocation fails or is unavailable, error messages prompt the user to update their location.
    - Failures in fetching nearby places or writing check-in data trigger alerts, ensuring users are informed of issues immediately.

    - All operations involving user check-ins and badge awards include try/catch blocks to gracefully handle and log errors without disrupting the user experience.
    - Validation is in place to prevent duplicate check-ins for the same location.
- UX and Visual Feedback:
    - The map displays real-time markers and proximity circles, offering clear visual cues.
    - Animated transitions and modals celebrate new badge achievements, reinforcing the gamified experience.
    - Refresh and centering controls on the map ensure users can update their location and explore new areas seamlessly.
    - A Friends Achievements leaderboard sorts friends based on badge count and visually displays rankings with medals or emojis.
- Integration with Other App Features:
    - Check-in data contributes to the overall achievements system, which aggregates progress and displays badges on a dedicated Achievements screen.
    - The feature complements other aspects of the app by providing a playful, interactive element that motivates exploration and repeat engagement.
- Screenshots:


2:47
Update My Location
Maillardville Community Centre
Coquitlam Heritage at Mackin House
Museum
Too Far to Log Visit. Get Closer.
2:49
Update My Location
Fun Park
Fun Park
Park
Log Visit
2:49
Update My Location
Visit Logged
You've successfully logged a visit at Fun Park.
View My Progress
No, Stay Here
Fun Park
Park
Visit already logged


2:49
Achievements
Onboarding
Keep going! 2 to Bronze
Keep going! 4 to Bronze
Leaderboard
Russell Han Josef
1
You
1
2:49
Achievements
PARK
Keep going! 2 to Bronze
Check in at a park to earn this achievement. 5 = Bronze, 10 = Silver, 20 = Gold.
2:50
Travel Behavior
Packing Style
Heavy
Travel Companion
Friends
Budget Range
Mid-range
Planning Habits
Planning Habit
Semi-Planned
Trip Role
Planner
Achievements
PARK

```
const confirmCheckIn = useCallback(async (place) => {

  if (!place) return;
  setLoading(true);
  let badgeShown = false;
  try {
    const checkinId = uuidv4();
    const createdAt = new Date().toISOString();

    const storedUser = await AsyncStorage.getItem('user');
    if (!storedUser) {
      Alert.alert("Error", "User not logged in.");
      setLoading(false);
      return;
    }
    const userData = JSON.parse(storedUser);
    const userId = userData.id;

    const checkInData = {
      coordinates: {
        latitude: place.latitude,
        longitude: place.longitude,
      },
      place_id: place.cached_data?.place_id || place.id || place.name,
      created_at: createdAt,
    };
    const categoryKey = place.category.toLowerCase();

    // Get previous check-in count for this category
    const prevSnapshot = await database().ref(`/game/${userId}/${categoryKey}`).once('value');
    const prevData = prevSnapshot.val() || {};
    const prevCount = Object.keys(prevData).length;
    const prevBadge = getBadge(prevCount);

    await database().ref(`/game/${userId}/${place.category.toLowerCase()}/${checkinId}`).set(checkInData);
    await database().ref(`/users/${userId}/onboarding/checked_in`).set(true);
    // Get new check-in count
    const updatedSnapshot = await database().ref(`/game/${userId}/${categoryKey}`).once('value');
    const updatedData = updatedSnapshot.val() || {};
    const newCount = Object.keys(updatedData).length;
    const newBadge = getBadge(newCount);

    // Check if badge tier increased
    if (newBadge && newBadge !== prevBadge) {
      setNewBadgeInfo({
        badgeTier: newBadge,
        category: capitalize(categoryKey),
      });
      setShowBadgeModal(true);
      badgeShown = true;
    }
```

```
    // Now sort the results array:
    results.sort((a, b) => {
        // Always show the onboarding badge first.
        if (a.category === 'onboarding' && b.category !== 'onboarding') return -1;
        if (b.category === 'onboarding' && a.category !== 'onboarding') return 1;

        // For non-onboarding achievements, consider complete if count >= 5.
        const aComplete = a.category === 'onboarding' || a.count >= 5;
        const bComplete = b.category === 'onboarding' || b.count >= 5;
        if (aComplete && !bComplete) return -1;
        if (!aComplete && bComplete) return 1;

        // Prioritize badge levels (Completed > Gold > Silver > Bronze)
        const badgePriority = { 'Completed': 4, 'Gold': 3, 'Silver': 2, 'Bronze': 1 };
        const aPriority = badgePriority[a.badge] || 0;
        const bPriority = badgePriority[b.badge] || 0;
        if (aPriority !== bPriority) {
            return bPriority - aPriority;
        }

        // If same level, sort by check-in count (higher count first)
        return b.count - a.count;
    });

    setAchievements(results);
```

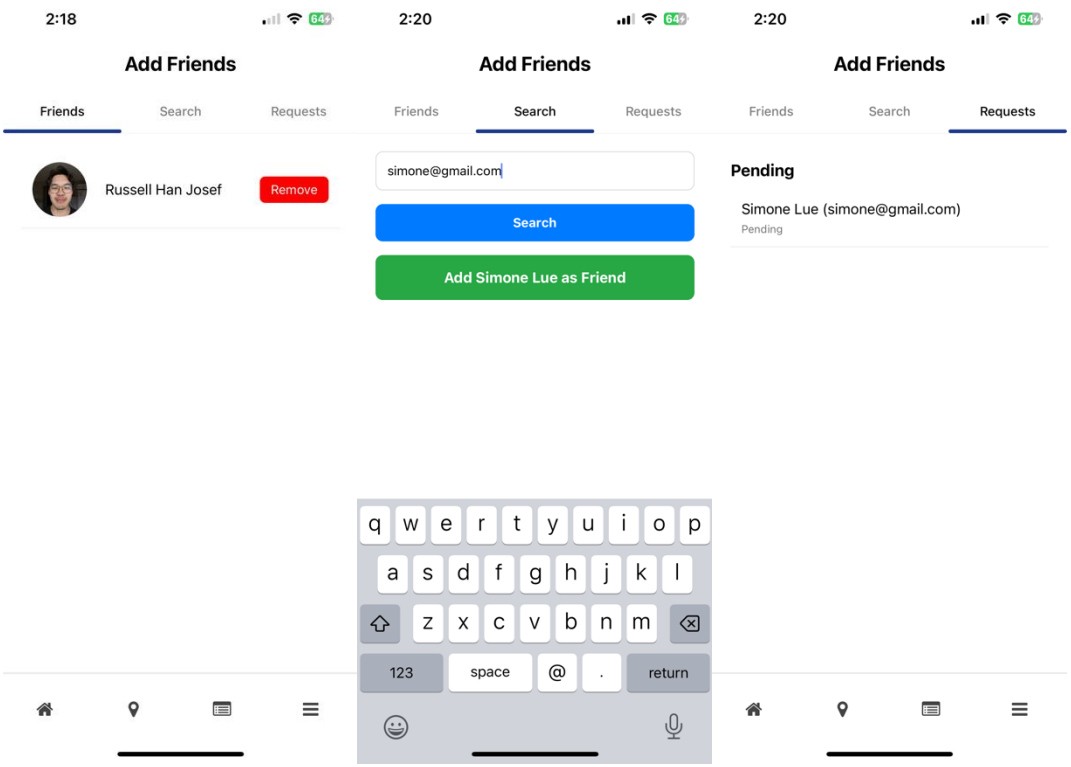
Implemented Feature: Friends

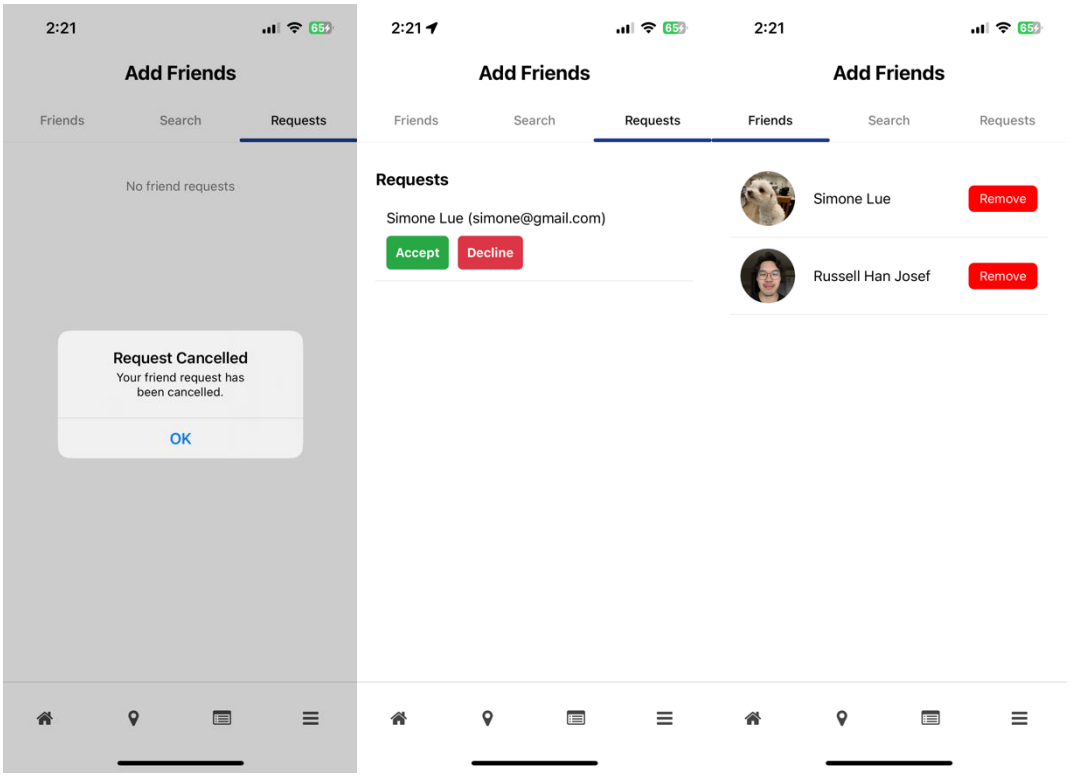
- Overview and Purpose:
    - Enables users to connect with others, manage friend requests, and build a social network within the app.
    - Provides multiple tabs to view current friends, search for new connections, and review pending friend requests.
- Friend Management Process:
    - Users can view their friend list, which displays profile photos and names.
    - Search functionality lets users search for potential friends by email, ensuring they do not add themselves or duplicate existing friends.
    - Incoming friend requests are received via Firebase; users can accept or decline these requests.
    - On acceptance, both parties are added to each other's friend lists, establishing a bidirectional connection.
    - Outgoing friend requests can be tracked and cancelled if needed.
    - Friends can be removed from the friend list through clear action prompts.
- Public Profile Integration:
    - The Public Profile screen displays detailed friend information, including profile photo, username, email, and comprehensive "About" sections.
    - Profile details cover bio/travel philosophy, home location, languages spoken, favorite destinations, fun facts, travel behavior, and planning habits.
    - Achievements are showcased as a grid of badges (e.g., Bronze, Silver, Gold, or special onboarding badge), with the ability to tap for more details.
    - Users can navigate from friend lists directly to a friend's public profile for an in-depth view.

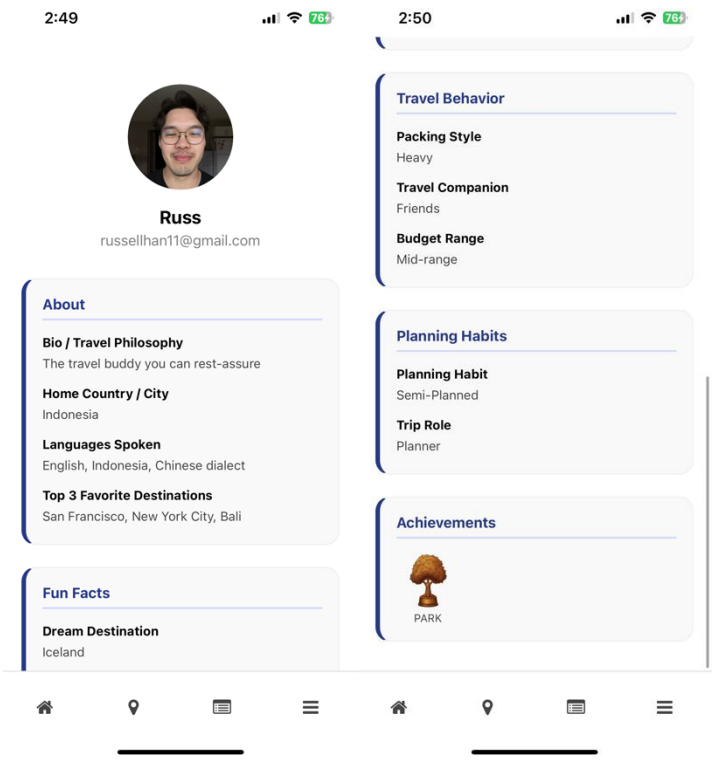
- Backend Integration:
    - Leverages Firebase Realtime Database to manage friend data:
        - Uses nodes such as /friends, /friend_requests, and /outgoing_friend_requests to track relationships and requests.
        - Real-time listeners in Firebase update the UI immediately when friend lists or request statuses change.
        - Integration with PublicProfile screens allows navigation to detailed friend profiles.
- Error Handling and Robustness:
    - Validates email input to prevent self-addition and duplicate friend requests.
    - Provides user-friendly alerts when no matching user is found or when a friend request has already been sent.
    - Operations (sending, accepting, declining, canceling requests) are wrapped in error handling blocks to catch exceptions and notify users with descriptive messages.
    - Ensures that Firebase transactions are robust, logging errors and maintaining data consistency.
- UX and Visual Feedback:
    - Uses a TabView with distinct tabs for Friends, Search, and Requests to facilitate easy navigation.
    - Friend items in the list display profile images, names, and options to view profiles or remove a friend.
    - Interactive elements (buttons, icons, swipe gestures) create a smooth and responsive experience for managing connections.

- Real-time updates via Firebase keep the friend network current and engaging.

- Screenshots:







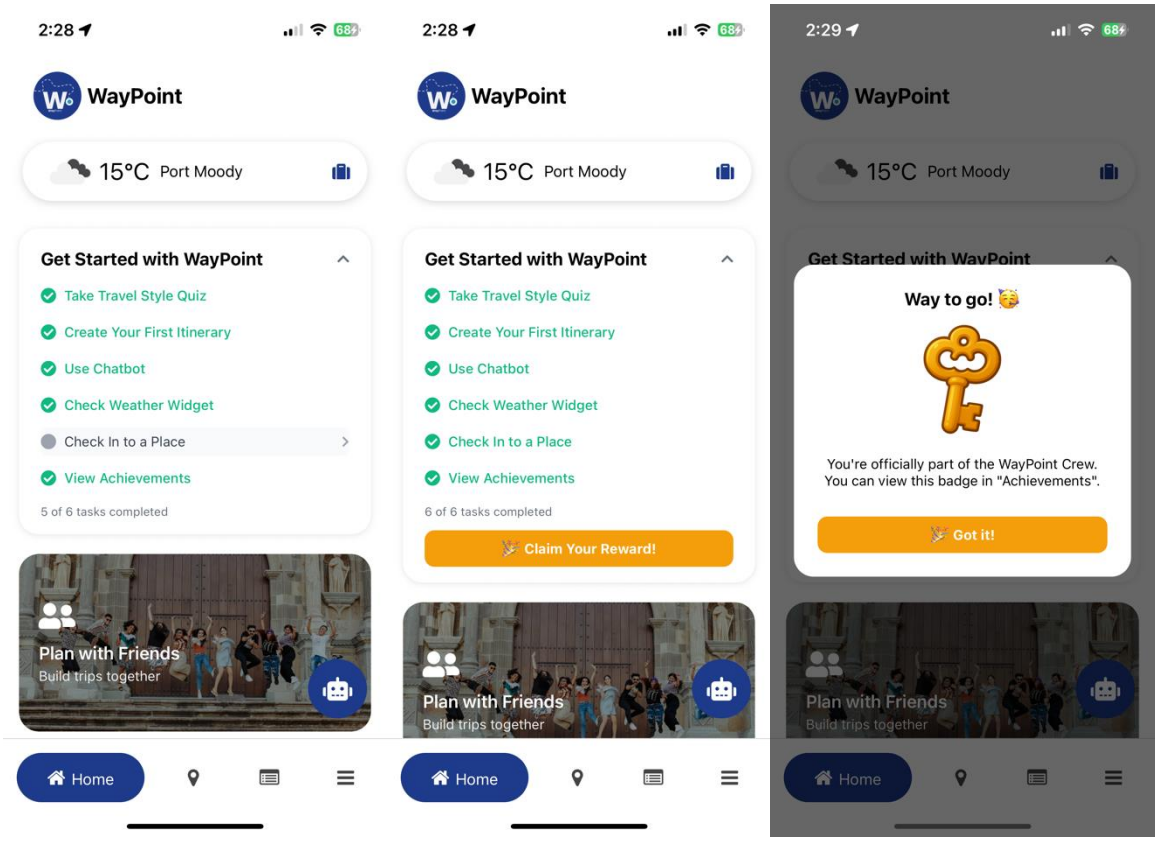

```
// Listen for changes in current friend list
useEffect(() => {
    if (currentUser) {
        const friendsRef = database().ref(`/friends/${currentUser.id}`);
        const handleFriends = (snapshot) => {
            if (snapshot.exists()) {
                const data = snapshot.val();
                const friendsArray = Object.entries(data).map(([key, value]) => ({
                    friendId: key, // from the Firebase path
                    friendName: value.friendName,
                    friendEmail: value.friendEmail,
                    addedAt: value.addedAt,
                }));
                setFriends(friendsArray);
            } else {
                setFriends([]);
            }
        };
        friendsRef.on('value', handleFriends);
        return () => friendsRef.off('value', handleFriends);
    }
}, [currentUser]);

// Listen for incoming friend requests
useEffect(() => {
    if (currentUser) {
        const requestsRef = database().ref(`/friend_requests/${currentUser.id}`);
        const handleRequests = (snapshot) => {
            if (snapshot.exists()) {
                const data = snapshot.val();
                const requestsArray = Object.keys(data).map(key => ({ id: key, ...data[key] }));
                setPendingRequests(requestsArray);
            } else {
                setPendingRequests([]);
            }
        };
        requestsRef.on('value', handleRequests);
        return () => requestsRef.off('value', handleRequests);
    }
}, [currentUser]);

// Listen for outgoing friend requests
useEffect(() => {
    if (currentUser) {
        const outgoingRef = database().ref(`/outgoing_friend_requests/${currentUser.id}`);
        const handleOutgoing = (snapshot) => {
            if (snapshot.exists()) {
                const data = snapshot.val();
                const outgoingArray = Object.keys(data).map(key => ({ id: key, ...data[key] }));
                setOutgoingRequests(outgoingArray);
            } else {
                setOutgoingRequests([]);
            }
        };
        outgoingRef.on('value', handleOutgoing);
        return () => outgoingRef.off('value', handleOutgoing);
    }
}, [currentUser]);
```

```
// Search for a user by email
const searchUserByEmail = async () => {
    if (!email.trim()) return;
    setSearchLoading(true);
    try {
        const snapshot = await database().ref('/users').once('value');
        const usersData = snapshot.val();
        if (!usersData) {
            setFoundUser(null);
            Alert.alert("User Not Found", "No user found with this email.");
            setSearchLoading(false);
            return;
        }
        const matchedUser = Object.entries(usersData).find(([userId, user]) =>
            user.email && user.email.toLowerCase() === email.toLowerCase()
        );
        if (matchedUser) {
            const searchedUser = { userId: matchedUser[0], ...matchedUser[1] };
            // Prevent adding yourself
            if (searchedUser.email.toLowerCase() === currentUser.email.toLowerCase()) {
                Alert.alert("Invalid Request", "You cannot add yourself.");
                setFoundUser(null);
            } else {
                if (friends.some(friend => friend.friendId === searchedUser.userId)) {
                    Alert.alert("Already Friends", "This user is already in your friends list.");
                    setFoundUser(null);
                } else {
                    setFoundUser(searchedUser);
                }
            }
        } else {
            setFoundUser(null);
            Alert.alert("User Not Found", "No user found with this email.");
        }
    } catch (error) {
        console.error("Error searching user:", error);
        Alert.alert("Error", "Could not search for user.");
    }
    setSearchLoading(false);
};
```

Implemented Feature: Onboarding

- Overview and Purpose:
    - Guides new users through a structured introduction to key app features with an interactive checklist.
    - Ensures users engage with essential actions (e.g., taking the quiz, creating an itinerary, using the chatbot, checking in, and viewing achievements) to quickly become familiar with the app.
- Onboarding Process:
    - Displays a collapsible checklist with tasks such as “Take Travel Style Quiz,” “Create Your First Itinerary,” “Use Chatbot,” “Check Weather Widget,” “Check In to a Place,” and “View Achievements.”
    - Each task is marked with a check icon upon completion and the current progress is displayed as a count of tasks completed.
    - When all tasks are completed, a reward modal is triggered, showcasing an animated badge that celebrates the user’s onboarding success.
- Backend Integration:
    - Retrieves onboarding status by fetching multiple values from Firebase (e.g., travel style, used chatbot flag, weather widget check, check-in confirmation, achievements viewed) in a consolidated asynchronous call.
    - Updates Firebase once all checklist items are completed, marking the onboarding process as complete.
    - Facilitates navigation to relevant screens (QuizScreen, Itinerary, Chatbot, CheckIn, and Badges) based on each checklist item.
- Error Handling and Robustness:

    - Employs try/catch blocks during Firebase data fetches and updates to gracefully handle errors and log issues without disrupting the user experience.
    - Provides fallback behavior to ensure the checklist renders correctly even if some data retrieval operations fail.
    - Prevents crashes by validating user IDs and checking for the availability of required data before proceeding with actions.
- UX and Visual Feedback:
    - Features an expandable/collapsible checklist with smooth animated transitions and real-time remeasurement to accommodate dynamic content.
    - Clearly indicates task completion through color-coded icons and text changes.
    - On completion, an animated reward modal with badge scaling, fading, and rotating effects delivers celebratory feedback.
    - Displays a progress summary indicating the number of tasks completed out of the total, reinforcing user engagement.
- Integration with Other App Features:
    - Each checklist task is interactive, linking users directly to corresponding screens (QuizScreen, Itinerary creation, Chatbot, CheckIn, Achievements) for a comprehensive app introduction.
    - The onboarding process sets a completion flag in Firebase that unlocks further functionality and rewards within the app.
    - Seamlessly ties together key social and gamification elements, ensuring a smooth transition from onboarding to regular app usage.
- Screenshots




```
useFocusEffect(
  useCallback(() => {
    const fetchChecklistStatus = async () => {
      if (!userId) return;

      try {
        const [travelStyleSnap, itinerariesRes, chatbotSnap, weatherChangedSnap, packingTipSnap, checkedInSnap, achievementsSnap] = await Promise.all([
          database().ref(`/users/${userId}/travel_style_id`).once('value'),
          fetch(`${API_BASE_URL}/itineraries/users/${userId}/itineraries`),
          database().ref(`/users/${userId}/onboarding/used_chat`).once('value'),
          database().ref(`/users/${userId}/onboarding/weather_changed`).once('value'),
          database().ref(`/users/${userId}/onboarding/packing_tip_viewed`).once('value'),
          database().ref(`/users/${userId}/onboarding/checked_in`).once('value'),
          database().ref(`/users/${userId}/onboarding/viewed_achievements`).once('value'),
        ]);

        const travelStyleId = travelStyleSnap.val();
        const quizDone = travelStyleId && travelStyleId !== 4;
        const itineraries = await itinerariesRes.json();
        const hasItinerary = itineraries.length > 0;
        const chatbotUsed = chatbotSnap.val() === true;
        const weatherChecked = weatherChangedSnap.val() === true && packingTipSnap.val() === true;
        const checkedIn = checkedInSnap.val() === true;
        const achievementsViewed = achievementsSnap.val() === true;

        setProgress({
          quiz: quizDone,
          itinerary: hasItinerary,
          chatbot: chatbotUsed,
          weatherChecked,
          checkedIn,
          achievementsViewed,
        });

        // Trigger layout remeasure after a frame
        setTimeout(() => {
          if (expanded) updateHeight();
        }, 100);

      } catch (err) {
        console.error('Checklist fetch error:', err);
      }
    };

    fetchChecklistStatus();
  }, [userId, refreshTrigger])
);
```

Implemented Feature: Packing/Weather Modal

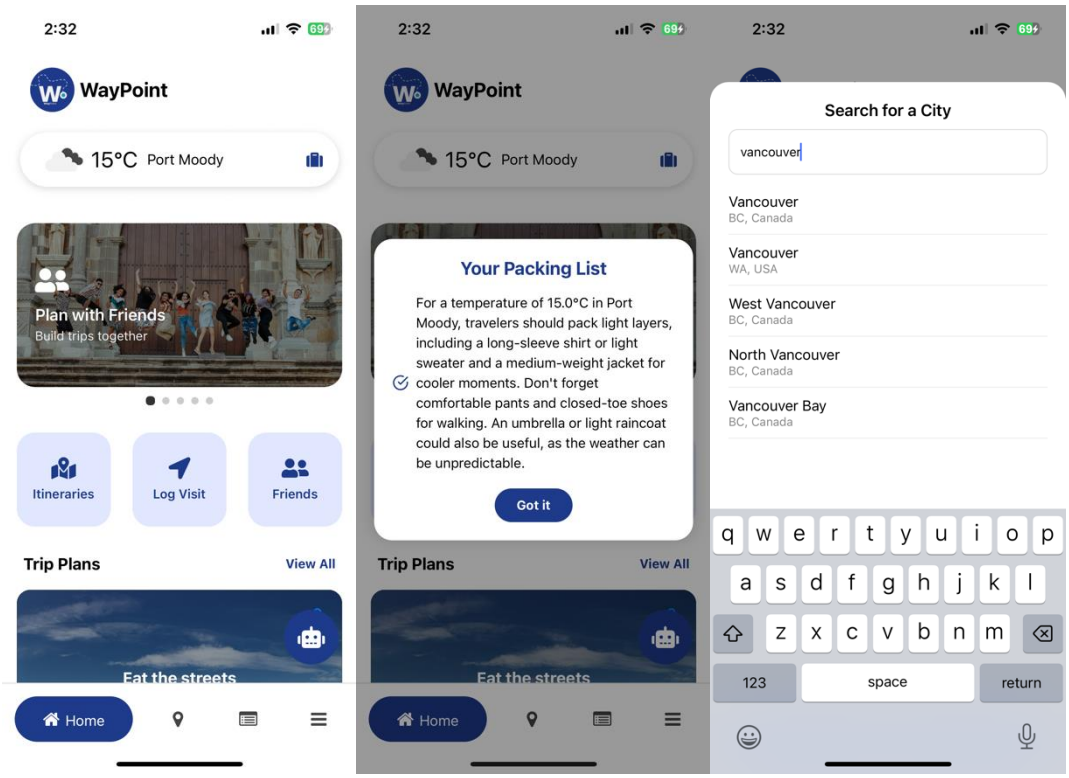
- Overview and Purpose:
    - Enhances travel planning by providing smart packing suggestions tailored to current weather conditions.
    - Combines real-time weather data from OpenWeatherMap with dynamic packing tips generated via a chatbot endpoint.
    - Empowers users to prepare effectively for their trips by linking environmental data with actionable packing advice.
- Weather Modal Process:
    - A WeatherSearchModal component enables users to search for a city using Google Places autocomplete.
    - Upon selecting a city, the modal passes the city name and its coordinates, triggering an API call to a dedicated weather endpoint.
    - The weather endpoint retrieves current weather details (temperature, weather condition, city name) using the OpenWeatherMap API.
    - The HomeScreen displays a weather widget showing an icon, temperature, and city name, with an interactive element (tapping the suitcase icon) that guides users to receive packing suggestions.
    - A tooltip on the widget instructs users, “Tap the suitcase to get smart packing suggestions!”
- Packing Suggestion Process:

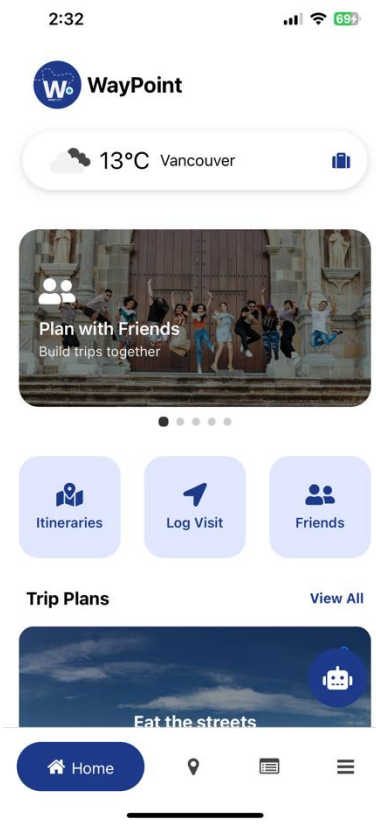
    - When the user taps the suitcase icon, a function is triggered to fetch packing tips based on the current weather data.
    - The packing suggestion is generated via a chatbot endpoint which takes the city, temperature, and weather condition as inputs.
    - A PackingSuggestionModal component displays the packing list, formatted as a scrollable list with check icons and clear text, while showing an ActivityIndicator during loading.
- Backend Integration:
    - The weather router calls OpenWeatherMap to fetch current weather data, ensuring that temperature and condition details are current.
    - The packing suggestion endpoint leverages weather data to generate concise, actionable packing advice.
    - Firebase and AsyncStorage are used to mark and track onboarding actions (such as packing tip viewed), which contributes to the overall user progress.
- Error Handling and Robustness:
    - API calls for weather data and packing suggestions are enclosed in try/catch blocks, ensuring any errors (such as network issues or API failures) are gracefully handled.
    - Informative alerts notify users if weather data cannot be fetched or if packing suggestions fail to generate, prompting users to try again.
    - Fallback defaults and error logs maintain data consistency and prevent crashes during unexpected failures.
- UX and Visual Feedback:

- The HomeScreen integrates the weather widget and packing modal seamlessly, offering an intuitive, interactive experience.
- Clear visual cues (weather icons, temperature display, tooltips) and smooth modal transitions guide users through searching for a city and obtaining packing tips.
- The PackingSuggestionModal's scrollable list, friendly icons, and loader ensure users receive clear and actionable packing advice.

- Integration with Other App Features:
  - The feature is tightly coupled with the onboarding and HomeScreen flows, updating the user's onboarding status upon viewing packing tips.
  - Real-time updates via Firebase and AsyncStorage integration help maintain consistent user progress tracking.

- Screenshots:






```python
@chatbot_router.post("/packing")
async def get_packing_tip(req: PackingRequest):
    prompt = (
        f"You are a helpful travel assistant. Based on this weather: "
        f"{req.temperature}°C, {req.condition.lower()} in {req.city}, "
        f"suggest what a traveler should pack. Keep it short and under 3 sentences."
    )

    try:
        response = client.chat.completions.create(
            model="gpt-4o-mini",
            messages=[{"role": "user", "content": prompt}],
            temperature=0.7,
            max_tokens=80,
        )

        message = response.choices[0].message.content
        return {"status": "success", "packing_tip": message}

    except Exception as e:
        return {"status": "error", "detail": str(e)}
```

```python
@weather_router.get("")
def get_weather(latitude: float, longitude: float, db: Session = Depends(get_db)):

    weather_url = f"https://api.openweathermap.org/data/2.5/weather?lat={latitude}&lon={longitude}&appid={OPENWEATHERMAP_API_KEY}&units=metric"

    try:
        response = requests.get(weather_url)
        data = response.json()

        if response.status_code != 200:
            raise HTTPException(status_code=response.status_code, detail=data.get("message", "Error fetching weather data"))

        weather_info = {
            "temperature": data["main"]["temp"],
            "weather_main": data["weather"][0]["main"],
            "weather_icon": data["weather"][0]["icon"],
            "weather_name": data["name"],
        }

        return {"status": "success", "data": weather_info}

    except requests.RequestException as e:
        raise HTTPException(status_code=500, detail=str(e))
```

Evaluation Techniques

For Waypoint, we adopted user testing evaluation approach using a combined think aloud protocol during structured task sessions. We recruited 6 participants who represented a varied user base to interact with our app. The testing environment was designed to capture qualitative feedback (e.g., verbalized frustrations, comments on usability) to comprehensively assess the user experience across multiple features.

Key components of our evaluation methodology included:

- Structured Tasks: We designed a set of specific, goal-oriented tasks that covered essential features of the app. Each task was crafted to uncover usability issues, gauge intuitiveness, and verify that core functions (from sign up to free exploration of the app) worked seamlessly.
- Think Aloud Protocol: Participants were encouraged to verbalize their thoughts, challenges, and expectations while interacting with the product. This real-time commentary provided valuable insights into how users interpreted the interface and navigated through tasks.
- Behavioural Monitoring: In addition to direct feedback, we observed and recorded important behaviours such as where users paused, expressions of confusion, navigation patterns, and moments when tasks were abandoned or completed incorrectly.

Based on our detailed user evaluations, the following design changes were prioritized and implemented:

- **Across Signup/Login and Profile:**

  Modifications were focused on improving button prominence, easing input interactions,

and clarifying field behavior. For example, adjustments were made to input focus transitions and the visual hierarchy of key actions.

- **For the Travel Style Quiz:**

  The assessment led to rethinking the visibility and guidance for the quiz. Enhancements now ensure that the quiz is easily locatable, that the answer choices are legible, and that results come with contextual explanations and clear navigation back to the home screen.

- **In the Weather Widget Area:**

  User feedback drove a redesign so that the entire widget is interactive, with more intuitive icon usage and visual feedback such as activity indicators to confirm that a location update is in progress.

- **Within Itinerary Management:**

  Adjustments included clarifying how users add and edit activities, addressing feedback related to oversights in budgeting and saving actions, and ensuring that transitions and navigational cues (such as clickable itinerary cards) are smoother and more intuitive.

- **On the Interactive Map and Chatbot Interfaces:**

  We focused on enhancing the user experience by providing more detailed marker cards, clearer options for day assignments, and ensuring that chatbot responses are actionable with clear call-to-actions. Although some advanced features (like richer visual details or dynamic filters) were noted as future enhancements, the initial round of changes improved overall usability.

- **For Check-In, Achievements, and Friends:**

  The evaluation highlighted the need for better discoverability, more precise terminology, and stronger visual feedback (e.g., notifications, progress tracking, and social comparisons). These insights informed changes such as renaming check-ins to “Log

Visit," adding visual badges for new friend requests, and incorporating animated feedback for earned achievements.

By linking each evaluation finding with design modifications, our iterative process has ensured that Waypoint evolves in line with user expectations—addressing current pain points while also creating a roadmap for future refinements.

Discussions

The development journey for Waypoint was both challenging and enlightening. One of the primary technical challenges we faced was related to integrating multiple systems and third-party APIs. Additionally, synchronizing data across structured systems like PostgreSQL and unstructured databases like Firebase proved complex, particularly when handling multiple APIs such as Google Places, OpenAI, and OpenWeatherMap. We had to implement caching logic for Google Places to avoid redundant API calls and remain within quota limits, which underscored the importance of efficient API integration. Balancing these technical demands with a seamless user experience required careful attention to detail and continuous troubleshooting.

On the design front, our team encountered significant challenges in ensuring a consistent and intuitive UI/UX. Achieving an optimal balance between engaging map marker interactivity and maintaining smooth navigation within scrollable bottom sheets was not straightforward. Maintaining a uniform design across screens was crucial to delivering a cohesive experience, yet it demanded iterative refinements and constant design revisions. We learned that a visual-first approach greatly enhances discoverability, though certain interactions—such as swipe gestures—were not universally intuitive, leading us to question and refine our gesture-based controls.

Data handling further complicated the development process. The coexistence of Firebase, PostgreSQL, and AsyncStorage introduced redundancy and potential conflicts, especially when real-time collaboration features were at stake. Ensuring that live itinerary edits were accurately synchronized between multiple users became an essential focus, and the experience taught us the importance of a reliable data management strategy where local-first logic plays a critical role in reducing API load. Through this process, we gained valuable insights into the strengths of each data storage solution: PostgreSQL proved to be ideal for structured, persistent data, while Firebase excelled in managing real-time, collaborative interactions.

Our experiences also imparted numerous lessons regarding workflow and project management. Adopting a feature-first MVP approach allowed us to focus on core functionalities, such as user profiles, quizzes, and recommendation systems, which provided a solid foundation for further expansion. The iterative cycle of coding, testing locally, and deploying through Heroku minimized bugs and streamlined our development process. Effective project organization on GitHub, reinforced by clear commit summaries and disciplined folder naming conventions, greatly enhanced team collaboration and task division. This systematic approach not only accelerated development but also enabled us to remain agile and responsive to user feedback.

The most satisfying aspect of the project was witnessing Waypoint come to life on a real iOS device and observing the positive reactions from our users during testing. Users particularly appreciated the AI-powered tools, the real-time collaborative features, and the map-focused design, which validated our design choices and reinforced our commitment to creating an intuitive user experience. The successful integration of multiple external APIs, despite the technical hurdles, was a testament to our team's resilience and technical capabilities. Overall, these experiences not only enhanced our technical and design skills but also reinforced the value

of iterative testing, user feedback, and continuous improvement, establishing a strong foundation for future developments.

Work Logs (Simone)

Date	Number of Hours	Description of Work Done
1/17/25 6:00 PM	1	Meeting. Distribution of work. Choosing team lead. Decide on the app. -> GitHub Repo -> Misc -> Applied Research_ Logo and Name Research.pdf
1/20/25 9:00 PM	1	Project Proposal Draft writing - Started the introduction section.
1/21/25 6:00 PM	2	Project Proposal writing - Finalizing draft writing -> Proposed Research Project, Project Planning and Timeline, Project Contract - Proposal in Github Repo -> ReportsAndDocuments -> SimoneL_Proposal.pdf
1/22/25 10:00 AM	2	Figma initialization, developing wireframes, general idea of screens
1/22/25 2:00 PM	1	Wireframe for homepage
1/24/25 7:30 PM	1	Wireframing for quiz screens
1/25/25 12:00 PM	0.5	Finalized proposal, ready for submission
1/27/25 7:00 PM	2	Wireframing screens: interactive maps, chatbot, my trips -> Github Repo -> Misc -> Figma Wireframes.png
1/27/25 10:00 PM	0.5	Regroup to discuss about the progress and knowledge sharing: Wireframe, APIs.
1/31/25 3:12 PM	3	Research on Reactive Native Expo vs CLI, environment setup and ways to start a project + gluestack v2 ui. Decided on React Native CLI as it was more suitable for the project scope

2/1/25 9:00 AM	2	Started and added frontend project shell to github. Encountered issues with the folder structuring when adding to github: Empty folder was being pushed to github instead of with the code. Had to restructure multiple times before successful
2/3/25 11:00 PM	1	Regroup to discuss about Database Schema of PostgreSQL and NoSQL. Update on the frontend shell. Update on the backend shell. Troubleshoot how to run ios on the machine on the first time pulling. Ensure frontend shell can work on both machines. Tasks assignment for the upcoming week.
2/6/25 10:00 PM	1.5	Started on the Quiz Screen component: Adding pressable buttons and sorting out the general layout of the text and buttons
2/7/25 6:00 PM	2.5	Completed QuizScreen and the styling file for it. Added: - Back/Next button for navigation between questions. - Back button not visible on the first question - Progress bar to visually represent questions done/left - Color change when button is selected - Have a couple of example questions set as placeholders
2/8/25 3:00 PM	2	Started on the layout of the HomeScreen as well as its corresponding styling file. Added: - Search bar (text input) - Horizontal scroll for 'My Trips' (with placeholder trips set + default card for when there are no current trips added) Problems encountered: - Had trouble with using Carousel (react-native-snap-carousel) - The dependencies had conflicts and version compatibility issues, specific issues with propTypes - Nothing was working Alternate solution: - Used FlatList with horizontal scrolling instead

2/8/25 9:00 PM	2	Started on the bottom navigation menu -> Problem: Tried using external library for icons but icons would not show up properly (Question mark in the middle of a box shows up) - Issues with linking fonts with Xcode Alternate plan: For now I am using emojis as a placeholder icon, will revisit adding external library of icons at a later time Bottom Navigation now has seamless navigation between Home and Quiz screens -> Quiz screen to be replaced at a later time, currently acting as a placeholder for other screens
2/10/25 8:00 PM	2	Regroup / Knowledge Sharing session. Russell: - Explained how the backend works for PSQL part. - Explained how to use pqAdmin tool as database management tool. - Explained the workflow with backend: when writing new code for backend, test it on local machine before pushing to Github. - Showed how deployment works with Github - Heroku setup. General: - Discussed the next workflow in tackling MVPs. - Discussed to re-order MVP priority based on survey. Task Assignments:
2/11/25 11:30 PM	2.5	Started on the backend for the quiz MVP -> quiz model, schema, api and routing Problem: Had trouble with the database setup/running it on my local machine and connecting to postgresql Solution: After consulting with Russell on the setup, server is able to run on my local device. Next step is to make sure that front end is connecting properly to the back end and able to pass information -> update to local database and make sure it is functioning as intended

2/13/25 8:00 PM	3.5	- Built the look for the quiz results on Figma - Adding the code and styling for the results display to QuizScreen.js and QuizScreenStyles.js - Will dynamically display the type of traveler depending on the quiz scoring logic from the quiz - Added an “x” button that will take user back to the home page after seeing the results Problems Encountered: - Tried to do a share button functionality where user will be able to save the results as an image to their camera roll. This didn’t work out as there were Xcode dependencies that conflicted with my system. Had to remove this functionality. Can revisit once the MVP is complete - Dependencies that were added to do this share implementation were not completely erased and repeatedly threw errors as it was still somewhere in the files. Solutions and Alternatives: - Had to remove node modules and reinstall dependencies and pod files multiple times to get rid of the errors. Next Steps: - Send quiz results to the backend and save it to the database - Have the option to retake quiz and erase results previous from the backend - (Optional once MVP is complete) Retry the share functionality

2/15/25 1:30 PM	2	Trying to send data to the backend and save the travel style pertaining to the user - Using AsyncStorage to save and retrieve the user_id upon successful logins Problem: - There were AsyncStorage runtime errors due to it not being properly linked - This was due to Cocoapods gem error -> broken or outdated Ruby gem Solution: - Reinstalled and updated ruby gem - Reinstalled cocoa pods - Reinstalled iOS pods - No runtime errors for AsyncStorage now Trying to retrieve userId from the user's login session - Created a method in QuizScreen.js to retrieve userId - Successfully able to retrieve userid Trying to send quiz results to the backend - Modified api for posting quiz results - Created method in QuizScreen.js to send results to the backend Problem: - 404 Error: api endpoint does not exist or incorrect Next step: Fix the connection to API endpoint

2/16/25 2:00 PM	1.5	Fixing the connection to API endpoint - Need to send the quiz results to the back end Problem: - 404 Error: api endpoint does not exist or incorrect Solution: - main.py had “/quiz_results” as the router prefix - quiz_routes.py had “/quiz_results” as the route - To send the results to the quiz_results table, the quiz results endpoint should be `${API_BASE_URL}/quiz_results/quiz_results` instead of ‘${API_BASE_URL}/quiz_results` - Changed the endpoint to “/“ --> proper full endpoint is now `${API_BASE_URL}/quiz_results/` - The results are now stored in the quiz_results table with the correctly retrieved userId and travelStyle When the user redoes the quiz, the results save as a new row in the table rather than updating the existing record - Need to update the record instead of making a new row - Modified POST “/“ in quiz_routes.py to check if a result already exists for the user, update it if it exists - Now successfully modifies the record instead of creating a new one Next Steps: - Add frontend option for user to retake quiz. - Fix scoring logic for when users navigate backwards during the quiz. - Users should not be able to move on to the next question without selecting an answer

2/17/25 12:00 AM	1	Added frontend option for user to retake quiz: - Added retake quiz button code and styles - Added a method to trigger onPress to handle resetting all scores quiz UI - New quiz results are being sent to backend, updating the user's travel style Next/Submit buttons are 'disabled' without selecting an answer - Added disabled button styling - Modified handleNextQuestion() to prevent moving forward when selectedAnswer is null Next Steps: - Fix scoring logic for when users navigate backwards during the quiz - When nagivating backwards through questions, retain previously selected answer
2/17/25 4:15 PM	0.5	Analyzed and summarized survey results to determing high/medium/low priority features for the application.

2/17/25 4:45 PM	1.25	Navigating backwards through questions, UI retains the previously selected answer - Used an array to track selected answers instead of a single selectedAnswer state - Updated handleAnswerSelection to store answers in the new array (which the index corresponds to the question index) - Set the selected answer when navigating to the previous question - Modify handlePreviousQuestion to retrieve and display the stored selection. Problem: - Navigating forwards AFTER navigating backwards retains the selected answer previously selected Solution: - When moving forward after going backwards, clear future selections to ensure past choices aren't remembered. Fix scoring logic for when users navigate backwards during the quiz. The current scoring logic does not automatically adjust the previous selection's score when changing an answer. Instead, it only adds points when selecting an option but does not remove points when an answer is changed - Modified handleAnswerSelection to first try to subtract points from the answers that are being changed Problem: - This method of scoring retained the scores from the questions that remained unchanged and added on an additional point for the new selection (whether from the same category or not) Solution: - Modify handleAnswerSelection to build scores from the remaining selections instead of subtracting points one by one - Modify handlePreviousQuestion and handleNextQuestion to reset scores of future selections when moving forward after going back.

2/17/25 10:30 PM	1.25	Meeting Notes Summary (February 17, 2025) 1. Upcoming 1-Week Tasks Focus on the next two MVPs: Personalized Recommendations (Google Places API integration for recommendations) Interactive Map (Google Maps API integration for visualization) 2. Past Week Progress Updates Team members shared knowledge and updates on completed tasks. 3. Heroku Backend Server Documentation Discussion on CRUD operations for backend endpoints 4. Connecting Simulator to Heroku Server Setting up the React Native simulator to interact with the backend hosted on Heroku. 5. Planning for Video Workflow Outlining the video workflow for the mid-term report. Deciding on tools and steps for video creation. 6. Firebase Realtime Database & App Distribution Revisiting Firebase Realtime Database setup. Setting up Firebase App Distribution for testing. 7. Google Places API on /search Path Integrating Google Places API for search functionality. Ensuring that the API can return filtered results based on user preferences.

2/20/25 7:00 PM	2	Implementing personalized recommendations feature - Built a FastAPI route (/places/recommendations) to fetch places based on user preferences. - Integrated Google Places API to get places based on user travel style (quiz results). - Stored cached places in PostgreSQL to reduce API calls. Problem 1: 422 Error (Unprocessable Entity) - FastAPI expected user_id as an integer, but it was sent as a string. Solution: - Ensured user_id is an integer Problem 2: 404 Error - FastAPI couldn't find user_id in quiz_results, even though it existed in PostgreSQL - The backend was using the local database instead of Heroku. - FastAPI was querying the wrong database (local waypoint_db instead of Heroku). Solution: - Connected FastAPI to Heroku Postgres by updating .env database_url Problem 3: API Route Conflict - FastAPI treated /recommendations as {place_id}, causing a 400 Bad Request. - The dynamic route @place_router.get("/{place_id}") was above /recommendations, so FastAPI assumed "recommendations" was a place_id. Solution: - Reordered routes in place_routes.py !!Confirmed API is working!! - FastAPI backend is using Heroku Postgres instead of local PostgreSQL. - Recommendations API (/places/recommendations) now works as expected.

2/20/25 9:00 PM	2	Built Recommended Places screen and styling Connecting frontend UI to make API requests to display the recommendations: - Developed methods to fetch recommendations - Developed filter options - Switched back to local to work on the app Problem: app UI was not displaying results - The frontend was now receiving an empty array upon making requests Solution: - Debugging logs indicated that google places was denying the requests due to invalid api key - Realized Heroku was configured with google places api but not locally - Added API key in .env and was able to retrieve the recommendations which reflected in the UI Next Steps: - Images for the recommended places are not found- need to fix - Refine filters - Make sure recommendations work for all travel styles
2/21/25 3:00 PM	2	Fixing the photo display on the recommendations screen - Logs show that the image can not be found - At first, thought it was the API key not being used in the frontend so it was restricting access to images, but upon correcting that, images were still not loading - Adjusted the method for image rendering a couple times and one finally worked (not too sure why) - Images now loading with the recommendations

2/21/25 5:00 PM	1	Making sure recommendations work for all travel styles: - Matching the different travel styles correctly to TRAVEL_STYLE_MAPPING - Had to rename the some of the keys to match the recognized travel styles Adjusting the displayed emoji for each travel style - Tried to do multiple emojis for combined travel styles but it offset the styling - Alternative: Selected one emoji best fit for different combined styles instead - Fixed the missing emoji for “No travel style” - Fixed the awkward wording for some of the travel styles such as "You are a You didn't align with any specific travel style Traveler" - Modify the return inside QuizScreen.js so that the formatting dynamically adjusts Next Steps: - Recommendations displaying for mixed travel styles - Recommendations fetch new results after travel style changes - Make sure the filters work
2/23/25 5:00 PM	4	Midterm Report: Title page (complete), Introduction (complete), summary of initial proposed project (complete), changes to the proposal (complete), UPDATED project planning and timeline (IN PROGRESS: individual responsibilities; COMPLETED: Gantt chart, new proposed timelines, milestones, deiverables), Implemented Feature (IN PROGRESS: Login, Recommendations, Interactive Maps; COMPLETED: Quiz), Work Logs (IN PROGRESS), Closing and References (IN PROGRESS) Proposal changes: - Gantt Chart - Responbilities - Timelines - MVPs Priority
2/24/25 5:00 PM	1.75	Video Recording for Mid Term Report Checkpoint
2/24/25 9:30 PM	2.5	Finishing and finalizing midterm report. Preparing documents for submission. Push log to the github.

2/26/25 9:30 AM	2	Fixing filters Problem 1: Clicking a filter does not change the UI - Passing 'places' instead of 'filteredPlaces' in the FlatList filtering logic Solution: Passed the correct name Problem 2: Clicking a filter does not show anything (blank page) - Added debugging logs to check - Debugging showed that theres a mismatch between frontend filters and backend categories/frontend filtering is based on unmapped categories - Tried remapping google places categories -> didn't work Solution: - Realized the retrieved google places for the current location were not related to any of the filters. - Changed one of the retrieved places to map from the raw API response category to one of the filter names and confirmed that it works. - For now, will have random google places categories map to the current applicable filters due to default location being set for recommendations retrieval and none matching the filters (ex. Mapping "museum" -> "spa") - This filtering is to be integrated with interactive maps and will be changed.
2/26/25 12:00 PM	0.25	Midterm Feedback by Priya
3/3/25 5:00 PM	1.5	Combined the Interactive maps screen and the Recommendations screen - Both components working as intended - Both filters working separately Problem: Recommendations don't change with the travel style filter change for the maps Next Steps: - Make sure the recommendations pull from the travel style selected - Have both filters working together

3/3/25 8:40 PM	3	Added OpenAI dependencies - Added to the backend for security reasons - Since calling from the backend, added chatbot_routes, and updated main.py to include the new router Problem: openai version and syntax incompatibility - New syntax with the latest openai version Solution: - Fixed the syntax to match the latest version of openai - cURL response confirms api is working Added Chatbot Screen and simple styling - Able to send request to backend and generate AI response Next steps: - Style the Chatbot - Chat messaging needs to be shown in order - Loading visual to indicate the answer is being generated - Make refinements to provide tailored information
3/5/25 10:00 AM	1	Updated Buttons for Zoom and Fullscreen Updated icon for list to load from Places API fixed the list spacing (scrollFilterContainer)

3/5/25 2:30 PM	5.5	Focus: UI Components Fixing the Bottom Navigation Icons: - Using Ion Icons - Previously tried to use depreciated version of IonIcons and had trouble displaying the icon properly - Installed latest version of react native vector icons and re-installed pods Problem 1: - Having the same issue displaying icons: square with '?' In the middle Alternative Solution: - Switching to FontAwesome6 for icons - Following readme for fontawesome6 to carefully install packages Problem 2: runtime error, vector icons module not found - Lots of errors with dependencies - Square icon with '?' - Tried to try with fontawesome5 - Nothing would work - Broke the node_modules package Solution: - Had to revert to original code Redoing the TabBar - Elevated look to the TabBar Problem: - Icons were getting cut off - Icons and label were shifted upwards and not centered Solution: - Played with padding a lot and finally got a good match

3/5/25 11:30 PM	4.5	Focus: UI Components Adding “More” menu - Small popup animated window when “More” tab is clicked - Used Animation for the window - Unable to navigate yet Blurring the background when the “more” menu is toggled on - Using react-native-community/blur Problem: - After careful and correct installation of dependencies and pods, it showed that the component <BlurView> was not working Solution: - Manually add blur library to Xcode and rebuilt app Adjusting the “More” menu - Able to toggle blur off and close popup when tapping outside the menu popup - Moved Settings and Profile to the popup navigation - Added InteractiveRecommendations and Chatbot to Bottom Tab Bar Problem 1: Tried to make the tab bar collapsable; screen background became empty Solution: - Using props to pass the hideTabBar and showTabBar methods to every screen Problem 2: Not every screen is scrollable Solution: - Using timeout option instead - Bottom tab times out after 5 seconds of inactivity to hide the screen Problem 3: InteractiveRecommendations flat list does not scroll behind the bottom tab bar Problem 4: Project build broke - Have to reclone Lost progress: Toggling blur, Adjusting popup menu, timeout option
3/6/25 4:00 AM	1	Retrieved lost progress: Toggling blur, adjusting popup menu, timeout option

3/7/25 12:00 AM	1.5	Focus: Chatbot Making the chatbot and user messages send in a normal texting format - Most recent messages on the bottom instead of on top of older messages - Order of the messages are preserved inside setMessages() without overwriting previous updates - Latest messages always appearing at the bottom by using 'inverted' on flatlist Next Steps: - Style the Chatbot - Loading visual to indicate the answer is being generated - Make refinements to provide tailored information
3/7/25 11:30 PM	2.5	Configuring the AI to provide tailored information - Added content body to the role of the system (AI) - Ai is defined as a travel assistant with a default location in Vancouver BC - Chatbot screen retrieves user's travel style and tailors recommendations to the user's style Problem: Had trouble sending the travel style to the backend Solution: - Retrieved user travel style from from async storage and stored in state variable - Passed state variable to the backend Next Steps: - Style the Chatbot - Loading visual to indicate the answer is being generated - (If time permits) Tailor responses with weather information

3/9/25 12:00 AM	2.5	Added Chatbot and user avatar - Indicate Chatbot/user message - Styled and positioned avatars in desired position - Adjusted styling for message responses Styled the text input box - Changed the styling to a more stylish look - Send icon instead of text button Added a customer Header for the Chatbot Added animated "..." to indicate to users when chatbot is fetching a response - useState isTyping to track when the bot is processing a response - Before sending in a request, insert "..." as a placeholder message from the bot - "Animate" the "..." by cycling through "." ".." "..." every 500ms while isTyping is true - Remove "..." when the response arrives and replaces with the real message Tried to retain bottom navigation on profile, settings, and chatbot screens Problem: - Navigation is separate from Tab Navigator - Tried to use separate AppNavigator - Tried to use hidden Tab Screens - Could not resolve; will revisit Next Steps: - (If time permits) Tailor responses with weather information

3/9/25 1:00 PM	1.5	Problems: - No consistency in navigating between screens (some using Tab Navigation, some using Stack Navigation) - Back swipe is not working on Tab Screens - Bottom Navigation Bar is not showing up on screens in the “More” menu Solution: - Discarded Tab Navigator method - Strictly using Navigation and routes as custom bottom navigation - Changed main stack and app navigator stacks Results: - Consistent navigation - Back swipe working on all screens - Bottom navigation shows up on all screens unless specified - Flat list in InteractiveRecommendations is scrolling due to these changes

3/9/25 1:00 PM	1.5	Focus: Navigation No active state on the Bottom Navigation - Adding detection for active screen Problem 1: Incorrect Active Screen Detection - activeRouteName.name === item.name; was incorrect because activeRouteName was already a string, so .name was invalid. - This caused isActive to always be false, meaning no tab was ever marked as active. Solution 1: - Changed const isActive = activeRouteName === item.name; to directly compare the string values Problem 2: Navigation Structure Issue - By using navigation.navigate("Main", { screen: item.name }), the active screen was inside a nested stack. - route.name only gave the top-level screen, so it wasn't detecting the focused tab correctly. Solution 2: - Used navigation.getState() to extract the focused screen inside the "Main" stack Results: - Active state added - Activate state shows label and changes background colour

3/9/25 6:30 PM	1	Focus: Navigation Ensure HomeScreen is active on initial load - Default activeRouteName to "Home" on first load - Ensure "Home" is highlighted on first load before navigation updates. - After navigation changes, activeRouteName updates normally. Problem 1: - The first useEffect runs before navigation state is fully loaded, so "Home" is not recognized as active. - Setting active state inside useEffect([navigation, route]) does not immediately reflect the correct screen. Solution: - Use useFocusEffect to detect the correct active screen on initial load. Problem 2: - Home is recognized as active on initial load but navigating to other screens do not update the active state Solution: - use setTimeout() to delay updating active state until navigation completes because navigation and state updates are happening out of sync

3/10/25 12:00 AM	2	The filterScrollContainer disappeared once the bottom nav got added back to the screen Problem: - Through debugging, found that the Flatlist in InteractiveRecommendations got pushed up and covered the filters Solution: - Reduced Flatlist sizing - reduced minimum height of the scroll container so that it stops scrolling vertically Changed the splash screen title to WayPoint Tried adding splash image logo - Adding and editing in xcode Storyboards Problem: - Image will not load - Image was in the wrong place Solution: - Moved image to images.xcassets INSIDE xcode.workspace
3/10/25 7:00 PM	0.5	Limiting Chatbot response - OpenAI generates too many recommendations and suggestions: took too much time to process and text is too long - Limiting chatbot recommendations to 3 suggestions - Added additional prompt to the full system prompt that is sent to openai Added an initial load in message from chatbot - Use setTimeout() inside useEffect() - Introduction message displays after 1 second

3/10/25 8:30 PM	3.5	Researching - OpenWeatherMap API calls: Need geographical coordinates (lat, lon) - Direct Geocoding API to get geographical coordinates (lat, lon)- Requires city name, state code, and country code for accurate results of coordinates - react-native-geolocation-service library - Need to request phone's geolocation feature Location Permissions - Creating a screen to allow user to enable location services - Created styling for the LocationPermissions screen - Modify Podfile to set up script for permissions - Add permissions to Podfile Problem: - Podfile was not recognizing RNPermissions/LocationWhenInUse - Throw an error when trying to access permissions Solution: - Modified podfiles and removed RNPermissions/LocationWhenInUse and RNPermissions/LocationAlways to only use RNPermissions - The core RNPermissions pod dynamically manages location permissions Next Steps: - Use retrieved latitude and longitude coordinates to call OpenWeatherMap API

3/12/25
10:30 AM
	1	Using geographical coordinates retrieved to access OpenWeatherMap API
		- Changed simulator's location to New West.
		- Want:
		  - weather.icon
		  - weather.main
		  - main.temp : Unit Default: Kelvin, Metric: Celsius, Imperial: Fahrenheit
		- Verified API call works
		- Created weather_routes.py and linked to main.py
		
		Next Steps:
		- Complete weather router
		- Fetch coordinates for every session and pass to router
		- Retrieve openweathermaps api data
		- Show details on screen to confirm

3/13/25 12:00 AM	1	OpenWeatherMap API - Completed weather_routes to retrieve temperature, icon, and weather condition - Set up method in the frontend to pass location coordinates to the backend - Showing the retrieved information on LocationPermissionsScreen (temporarily) to confirm working API call and to confirm the passing for information from frontend to backend Restructuring LocationPermissions screen - Passing location permissions to homescreen - if location permissions are granted, fetch location to pass to homescreen

3/13/25 1:00 AM	1.25	Restructuring HomeScreen - Accept getLocation() as a prop to fetch location - Displaying weather information on homscreen instead of in LocationPermissions Problem 1: - Rendering LocationPermissions screen behind the Homescreen - Both screens rendering simultaneously Solution: - Instead of rendering as a child inside HomeScreen's main view, use conditional full-screen rendering Problem 2: Weather information not displaying - axios and API_BASE_URL imports missing from HomeScreen Solution: - Added necessary imports Problem 3: LocationPermissions screen loads up on subsequent app launches even when permissions have been granted - permissions start off as false and only sets to true when user presses "Allow" on the LocationPermissions screen. - The component doesn't know permissions were granted in a previous session until button has been pressed Solution: - Added useEffect to check permission status of the device on component mount Next Steps: - LocationPermissions screen won't go away if permissions not granted

3/13/25 7:30 PM	2.5	Adjusting the weather display in home screen - Rounding temperature to display no decimals - Adjusted the layout of the weather box Adjusting the MoreMenu - Converted MoreMenu from a separate screen into a Modal Overlay - Modal overlay is toggled when the more button is pressed which displays over the current view Styling Modal menu - Fixing up the UI - Implementing BlurView to blur background when modal is toggled Problem: - BlurView is pushing my modal menu out of position Solution: - Tried to put modal menu inside BlurView -> did not work; blurred everything - Use absolute positioning to place modal menu on top of blurview Next Steps: - LocationPermissions screen won't go away if permissions not granted - Handle denied permissions - Format chatbot responses - start Events screen
3/14/25 11:00 PM	0.5	Research EventBrite API Problem: Event search API is no longer available - Unable to search general events without specific event ID Alternative Solution: - Research for different events API - Found PredictHQ which allowed searching of local events - Set up token and tested api call on PredictHQ's website Next Steps: - Set up Events component (Screens, styles, routes)

3/15/25 1:00 PM	3.5	Chatbot Text Input not clearing - Added useRef to manually clear text input after sending message Set up Events Screen - Set up placeholders for how the events screen should look - Added temporary styling Set up Events router - Set events router and connected to main.py - Set GET method for the API - Set params for the API call Set up methods in Events Screen - Using Geolocation to retrieve device's coordinates - Passing coordinates to the router to use the PredictHQ API - Fetching response data Problem: API 401 Unauthorized error - Token refuses to work - Also discovered PredictHQ is a paid subscription after a free trial -> will not be able to use Decision to remove Events MVP following failure to search for a Events search API
3/17/25 9:15 PM	0.25	Format chatbot responses: - Added Markdown text formatting for chatbot - Styled user message text to allow more contrast for visibility
3/19/25 6:30 PM	1	Planning and identified the structure for badges MVP: - Need to set up Firebase - Retrieve user id from the session - Category of places (Park, bar, museum etc.) - Check in completion ID (UUID) - **DONE** - Coordinates of the location check in - Place_id of the location check in - Time the check in was at - Firebase Realtime Database set up with intended structure

3/19/25 11:30 PM	2.5	Check In Screen - Uses geolocation to get current coordinates - Send GET request to google places search API with a small radius for precise verification - Return list of the GET request is filtered for categories with “park”, “bar” or “museum” - Once check in verification is successful, a UUID is generated + time stamp recorded + user UUID is retrieved from AsyncStorage and sent to Firebase - (Placeholder) Alert to notify check in successful or error matching location - Verified using Stanley park coordinates Problem 1: Using Stanley Park coordinates retrieved Shakespeare Garden INSIDE Stanley Park Next Steps 1/ Solution 1: - Display a list for users to check nearby places and select the correct one Problem 2: Multiple check ins at the same location should not be allowed Next Steps 2/Solution 2: - Same location name/coordinates will not be allowed a second check in

3/20/25 11:00 PM	1.5	Displaying a list for users to select a check in location - Fetch location using geolocation on mount and queries google search API to fetch nearby places - Display retrieved places in flatlist - Selecting a place confirms the check in Problem 1: Flatlist showing incorrect name - Flatlist showing the “name” property to be location name (ex. Central Vancouver) instead of the name of the place (ex. Shakespeare Garden) Solution 1: The Flatlist’s radius retrieval is too small - Increase radius to 300 Problem 2: Flatlist showing repetitive items with different categories - Shakespeare Garden appears 4 times with categories: park, beach, botanical_garden, hotel Solution 2: Filter data by place_id - One location has one place_id - Filter duplicates in response data so only one instance of each place_id appears Problem 3: Flatlist displaying diverse categories Solution 3: Further filtering - Defined a list of allowed categories (park, museum, bar- the current categories for achievements) - Filter out any places that do not match these categories
3/21/25 12:30 AM	0.75	Prevent multiple check ins at the same location - Fetch previously checked-in place_ids for the current user and store in a state variable - If the place has been checked in before, touchable is disabled with a “Checked In” label Problem: Duplicate check ins were still allowed - Realized place_id was not being properly sent to be stored in firebase Solution: Added place_id in firebase properly - confirmed place_id was being added - Previously checked in locations are now disabled
3/21/25 1:15 AM	0.25	Separated styling sheet Next Steps: - Fixing the UI of Check In screen

3/21/25 7:00 PM	3	Styling Check In screen - Rendered a MapView to show 2 types of markers - 1 Marker for the user's current location - Another set of markers to show nearby locations that match the allowed place categories - Details card on the bottom to show the name of the place and the category (defaulted to nearest fetched place) - Tapping on a place marker sets the details card to show that selected place Problem 1: Marker not showing for nearby places - Asynchronous timing between the user location marker and the fetch locations marker - Tried normalizing data -> did not work - Tried useRef -> did not work - Tried setting initialRegion -> did not work Solution 1: - Render map only after user location and places markers are available - Allows for everything to be loaded in at once
3/21/25 10:00 PM	1.25	Default place (closest place) does not have a marker - Tried to introduce a defaultPlace state to store nearest place and set as initial selectedPlace -> did not work - Tried using unique marker keys and fallbacks -> did not work Solution: Removed user location marker and discovered the default place was hiding behind it - Removed user marker as its not important - Added Circle from react-native-maps to show circle centered on user's coordinates Cleanup - Added refresh button to update current user location and nearby places - Extracted logic from useEffect into its own function so it can be called on mount and via manual refresh button

3/21/25 11:30 PM	2	Starting the Achievements system - Retrieve checkins by the user from Firebase - Count check ins for each category node - Badges set up with milestones (5/10/20 check ins) Troubles encountered: - incorrect queries to Firebase Next steps/problems: - Categories with no previous check ins are not displayed - Adding badges (images/icons) - Styling
3/21/25 1:30 PM	1.75	Categories with no previous check ins are not displayed - Instead of iterating over categories that currently exist in the database, iterate over all categories to make it all show - Do data[category] || {}, which means if data[category] is missing in firebase, we use an empty object instead Separate style sheet - Removed in line styles - Moved all styles into a separate sheet Generate badges - Generate badges for the achievement system that suit the style of the app - Use gemini AI to generate badges Problem: Difficult to generate image exactly how we want it - Have to generate a lot of times to get it close to desired image

3/21/25 10:30 PM	1.5	Adding badges - uploaded badge images to the app - return correct badge for each category according to the number of check ins Problem: no badge displayed when the user has not completed the minimum for achievement Solution: need to display something - Showing bronze badge with lower opacity Add progress bar - To show progress to next achievement - Using progress bar - Used getProgress function to calculate a fraction to the next threshold - Displayed under trophy badge Problem: There is an empty gap at the start of the progress bar - Fixing 0 margin -> did not work Solution: - added width={null} to inline styles to let it fill the container
3/22/25 12:00 AM	0.75	Generating more trophies - More trophies for categories bar and museum - Added all generated trophies for each achievement level and category to the project assets Displaying new badges - Defined a new trophy mapping object based on achievement category and badge level - Calls getBadgeImage to display the correct trophy image for that category

3/22/25 12:45 AM	2.25	Styling Achievements - Styled in grid layout - Allow flexibility for future category addons -> any addition will wrap automatically - Removed card layout -> replaced with grid cell layout - Some badge images have a checkered background -> removed Adding a modal - Add a modal to display the category, current progress, and description of the selected badge item - Also displays available tiers of the badges - When user taps a badge, the selectedAchievement state is updated and modalVisible is set to true - Clicking “X” closes modal and sets modalVisible to false + clears selectedAchievement state Problem 1: Progress bar not showing in modal Solution 1: removed null width from the progress bar Problem 2: Progress is set centered in the bar, leaving unfilled progress at the start and end of the bar - React native expects a numeric value for the width of the bar Solution 2: - Imported Dimensions from react-native library to get screen width - Set progress bar width to a % of the screen instead of having a set width (i.e. 200) to ensure consistency across screen sizes Problem 3: Progress bar margin is not setting; touching edges of the modal Solution 3: - Wrapped progress bar in a View - Adjusted width based on screen size - aligned it in the center Next Steps: - Confirm all achievements display correctly for each tier - Adjust styling

3/24/25 12:30 AM	0.5	Adjusting the UI - Removed title and progress text on main achievements screen for cleaner look - adjusted height of progress bar - adding border shadow to achievements title container Problem 1: the fill height of the progress bar didnt change Solution 1: - Adding height property to inline text instead of in style sheet Problem 2: shadow applying to container border and text Solution 2: - separate border into its own View below the text
3/24/25 1:00 AM	2.75	Confirm achievements - Checked at 20 parks to verify all badges displaying correctly at every tier Problem: - Museum categories confirmed to not be fetched - No museums are showing up on the marker even in the correct coordinates - The response data has “category” and “types” - “museum” is not under category, but under “types” Tried: - adjusting the filter to check if the types array contains the allowed categories -> did not work Problems found during debugging + priority fixing: - Filtering should show museums and bars - Screen is stuck infinitely on loading if there are no fetched places from the allowed_categories Non-priority next steps: - Want to indicate in modal which tier is selected in the panel at the bottom
3/24/25 10:00 PM	1.5	Walk through the workaround for new API route. We are not pulling API using Places API all the time anymore since we put a limitation. Introduced places/search/cached as the route we are using for pulling data from our own database.

3/24/25 8:55 PM	1	UI - Added profile picture to more menu modal - Added user name to the more menu modal - Combined both in a profile card - Added an indent style (tried react-native-shadow-2 library but it didn't give desired effect) - Fixed width of the profile card
3/25/25 12:20 PM	1	Removing nav bar when on LocationPermissions screen - tried adding it to hiddenScreens array in CustomBottomNavigation -> did not work - tried navigating to location permissions screen separately -> did not work - "lifting the screen"; having a separate stack navigator for MainBottomNav and a separate one for AppNavigator -> did not work - Reverted to old for now
3/25/25 9:40 AM	2	Removing nav bar from LocationPermissions screen - conditionally render either the Location Permissions screen or the Main stack (AppNavigator) based on whether locations permissions has been handled Problem: once the location permission is granted, does not notify the parent navigator that it should transition away from this screen - Found and tried out a solution that worked but changed too many components of the app backbone - Russell was working on a wide range of features that depended on some code that was being changed - Reverted back to old code

3/26/25 4:00 PM	2	Problem from before: - category “park” was being displayed but not “bar” or “museum” - only the closest park being retrieved was shown - was stuck on the activity indicator every time no nearby places were being fetched Solution: - Instead of filtering for categories in the front end, added new place route (/cached/filtered) to get places from cached Places table and returns only categories from allowed categories (meant to use for achievements) - Started over MapCheckInScreen and corresponding styles from scratch to ensure: no repetitive and looping logic, new route is used, no unnecessary code from before - Screen fetches from database for nearby places (2000m radius) to ensure there will be at least 1 place pulled New Check In feature: - Screen fetches from database for nearby places (2000m radius) from the user’s location - if the place is outside of user’s 300m radius, check in is disabled
3/26/25 6:00 PM	0.75	Daily 20 pulls from google places API to add to database Problem : Progress bar in achievements modal not displaying the right progress - Set width of the progress bar is cutting off the progress fill Solution - Removed bounds for the width

3/26/25 8:00 PM	1.75	UI clean up - More menu clean up - Shifting everything in Homescreen so it looks better on an actual device - Capitalizing category names for display in InteractiveRecommendations and CheckInScreen InteractiveRecommendations map zoom control - Zooming in with gesture makes the map pop back to original view Solution: - Added gesture handling -> did not work - found a restriction for zooming view -> removed it Researching how to retrieve places.photos from google places api - Within free useage - Share findings with russell to discuss
3/26/25 10:45 PM	2.5	HomeScreen x Itineraries - Replacing hardcoded trips list with a dynamic API call to retrieve based on user - Store saved itineraries as a state variable - Display itineraries in card view - Itinerary cards display: trip title, trip image, and start date - Placeholder image is used if the itinerary does not have an image associated - Added dark overlay to have enough contrast for white text Problem/Next Steps: - Image from trips not pulling from the trips - Limit to display only 3 trips in carousel - Check if no trips - Option to select “View All” to navigate to Itinerary List
3/28/25 12:00 AM	0.25	Pulled from google places search api to add to database

3/28/25 10:45 AM	0.5	Limit ScrollView to only display 3 items - Instead of mapping over all itineraries, use slice to show only first 3 Added “View All” - To navigate to ItineraryListScreen to view all trips Problem: Navigation not working Solution: - Screen navigation is wrapped in “Main” have to do navigation.navigate("Main", { screen: "Itinerary" })
3/28/25 1:00 PM	3.1	No trips - render a card with the same style as trips card to display “No itineraries” - replaces the default basic text Image from trips not pulling from the trips - Tried: Calling single-itinerary endpoint for each itinerary to get the presigned image URL -> did not work - Tried: Creating separate hook to use in HomeScreen to fetch Itinerary details -> did not work - trip data from aws is undefined in the homescreen - Tried: fetching itineraries directly in homescreen -> did not work
3/31/25 1:00 AM	1.25	Displaying both owned itineraries and shared itineraries in homescreen - Fetches and merges itineraries from both shared and owned trips into a single array stored in state - Limits 3 carousel items Problem: Trip image not displaying for Owned trips - Fetching from our API instead of Async - Structure for retrieving image is changed Solution: - Separated fetching - Fetch Owned trips from Async - Fetch Shared trips from backend API

3/31/25 2:15 AM	0.25	Displaying shared/personal icon in trips carousel - Added type property to owned and shared trips - show distinctive icons for personal/shared trips based on type
3/31/25 1:00 PM	0.75	Achievements sorting logic: - Custom sorting to allow for onboarding trophy first - Completed achievements (>5 check ins) prioritized - Rank and display achievement based on tier (Gold > silver > bronze)
3/31/25 1:45 PM	0.15	Small UI adjustments - Found areas where texts had 0 margin - Adding margins to improve UI Map Check in refresh button - Dynamic bottom card kept moving the button out of place - Moved it to absolute position at the top of the screen instead of the bottom so it doesnt get disrupted - Removed “refresh” text and replaced with icon to improve ui experience

3/31/25 1:55 PM	2	Friends - New Screen with 3 tabs: search, friends, requests - Search Tab: Uses a TextInput for entering an email address and a button to trigger the search. The search function reads from the /users node in Firebase. If a valid user is found (and it isn't the current user), an "Add Friend" button is displayed which sends a friend request - A realtime listener fetches the current friend list from the /friends/${currentUser.id} node and displays it in a FlatList with an option to remove a friend - A listener is attached to /friend_requests/${currentUser.id} to show pending requests. Accepting adds both users as friends while declining removes the request Problem: Repeat requests allowed Solution: - fetches existing requests by reading the data from the /friend_requests/${foundUser.userId} node using once('value'). This node contains all the pending friend requests for the user - if a request is found, function returns early/stops new request from being sent Display pending - New tab to display pending (outgoing) requests - new state variable outgoingRequests is created, + useEffect hook that listens on /outgoing_friend_requests/${currentUser.id}. This keeps track of all friend requests the current user has sent - In the handleAddFriend function, after sending the friend request to the target user's /friend_requests node, an entry is also created in /outgoing_friend_requests/${currentUser.id} -> ensures the outgoing request appears in the Pending tab

3/31/25 3:55 PM	0.5	Problem: Once friend request is accepted, the pending (outgoing from original sender) is not removed Solution: - When a friend request is accepted, add both users to each other's friend lists and then removes the incoming request from the current user's /friend_requests node - Queries for firebase records where the receiverId matches the current user's ID -> If records exist, matching records are removed from the sender's outgoing node Problem: Can add emails that are already in friends list - New check in searchUserByEmail function - checks if the friends state already contains an entry where the friendId matches target user's id - prevents the friend request from being made
3/31/25 4:25 PM	1	New PublicProfileScreen - Navigate to this new profile screen when pressing the profile from the friend tab in Friends screen - Uses friendId to fetch that user's public profile from Firebase - Displays public information like profile picture, about section, fun facts, travel behavior and planning habits Adjusting layout for existing friends - Remove display of email - Show Profile picture and name Problem: Friend profile picture not showing in friends list Solution: - (working) public profile screen directly retrieve's the profile picture from /users node using their friendId - Adjust the retrieval in friends list to do the same instead of using a snapshot of the friend's data

3/31/25 5:25 PM	0.5	Problem: (previously working) friend's public profile not found - Previous code passed an object to handleViewProfile in order to extract the friend id - Altered code required a string being passed (no extraction needed, just the friendid) Solution: - Updated the handleViewProfile function to accept a friendId string directly
4/1/25 2:00 AM	0.75	Consistency in Profile and Public Profile UI - Match styles of both screens Public Profile Information - Extract available data to display - Do not display profile information that has not been added by user - EX. If user's Bio is not filled, do not display in public profile screen

4/2/25 2:45 PM	0.8	Problem: Onboarding- “Claim Rewards” button was taking up space even when hidden - Button was conditionally rendered but height was a fixed number, so there was an awkward empty space where the button is supposed to be rendered when checklist is complete Solution: - Wrapped checklist items (and reward button) in a View that uses the onLayout prop + in the callback, update shared value so the Animated.View expands just enough to wrap its children -> did not work - Added a new state variable measuredHeight to store the content’s full height - animatedContainerStyle to use measuredHeight instead of a fixed shared value - Added a hidden view (with absolute positioning and opacity 0) that renders the checklist content. Its onLayout callback sets measuredHeight Problem 2: The checklist content is properly wrapped but now the Claim Rewards button does not show when onboarding is complete Solution: - Always render the hidden measurement view (even when measuredHeight is nonzero) - Whenever the checklist content changes (when the reward button appears), the updated height is captured and the animated container expands accordingly

4/3/25 12:30 AM	2	Modifying Add Friends screen - Removed the "Pending" route (used for outgoing friend requests) from the TabView - Created one new “Requests” tab that displays both incoming requests and outgoing pending - Switched from FlatLists to a single SectionList- one section for incoming requests and one for outgoing requests - Created an array called sections that only pushes a section into it only if there is data for it (Pending, and requests) - The SectionList uses the dynamically built sections array so that if only one type of request exists, only that section appears - Wrapped the pending request items in a Swipeable to cancel requests UI Clean up - Moved personal/shared trip icon in home screen trip component to the top corner for better visibility - Removed Name display from the more menu - Conditional rendering in public profile to prevent displaying empty About sections - Feature Carousel navigations to corresponding screens Problem: Trouble navigating to itineraries list screen with “Shared” set as active tab Solution - Access the route parameters with useRoute - Checked if route.params.activeTab was equal to 'shared'. If yes, initialized the TabView's index to 1 (selecting the Shared Itineraries tab), otherwise set it to 0
4/4/25 9:30 PM	0.5	First user testing: Task based questions
4/5/25 7:30 PM	0.5	Started Final Report: Title page + Introduction complete
4/8/25 12:15 AM	0.75	Chart user feedback and plan fixes for issues. Adjusted user check in radius and height of modal

4/9/25 9:45 AM	2	Chatbot: - Configured Chatbot to provide specific place recommendations (less vague responses) - Added a useEffect to load the chat history from AsyncStorage when the ChatbotScreen mounts - Added a useEffect that listens for changes in the messages state and saves the updated chat history back to AsyncStorage Problem 1: - There is an initial welcome from chat bot - The separate delayed effect for setting the welcome message was overriding the loaded chat history Solution: - If chat history exists in AsyncStorage, it is loaded - if not, the initial welcome message is set right away Problem 2: - Retrieving chat history from async will always show a chat history - Want a refresh restart if user leaves app or logs out but async is tied to the user profile Solution 2: - Add toggle for a new chat history modal to view chat history instead of displaying on mount Problem 3: Initial message resetting the chat history to the welcome message when user navigates away from the screen Solution 3: - A new state variable (persistedHistory) is used to load and store the full session history from Async - A helper function appendToPersistedHistory is used to update this history whenever new messages are sent or received - History modal displays the persistedHistory (loaded once on mount and updated with new messages)
4/10/25 8:30 PM	2.5	Working on Final Report: Title Page (Complete), Introduction (Complete), Summary of Research Project (Complete), Changes to Proposal (Complete), Project Completion Timeline (Complete),
4/11/25 4:45 PM	2.5	Working on Final Report: Implemented Features (Done except missing screenshots)
4/12/25 11:00 PM	1	Working on Final Report: Evaluation Techniques (Completed)
4/13/25 12:00 PM	4	Final Report: Implemented Features (Completed- Added screenshots), Discussion (Complete), Concluding Remarks (Completed), References (Completed)

Work Log (Russell)

Date	Number of Hours	Description of Work Done
1/17/25 6:00 PM	1	Meeting. Distribution of work. Choosing team lead. Decide on the app. -> GitHub Repo -> Misc -> Applied Research_ Logo and Name Research.pdf
1/18/25 4:00 PM	1	Planning on the project scope and role distribution.
1/20/25 8:30 PM	1.5	Research on screens to have. Finding screens inspiration. App name and branding.
1/22/25 10:00 AM	2	Research on Heroku Dynos and Postgres workflow

1/22/25 4:30 PM	1	Researched about conducting surveys and user testing.
1/22/25 5:30 PM	1	Created survey questionnaires and shared using Microsoft Forms. -> Github Repo -> Misc -> Form_Exploration App Survey_ Help Us Build Your Dream Travel Planner.pdf
1/25/25 8:00 PM	2	Commit Git for Proposal.pdf. Researched for APIs: OpenAI gpt model to use, Google Places, Google Maps, Eventbrite, OpenWeatherMap. Get all the API Keys needed.
1/27/25 10:00 PM	0.5	Regroup to discuss about the progress and knowledge sharing: Wireframe, APIs.
1/28/25 3:00 PM	0.5	Get Free credits from Heroku account using GitHub for Student Developer Pack
1/28/25 3:30 PM	2	Researched on Database Schema. Learned more about PostgreSQL vs Microsoft SQL vs MySQL. Then Firebase NoSQL. Learned about Hybrid Architecture Approach with both PostgreSQL and NoSQL. Exported 2 diagrams. -> GitHub Repo -> Misc -> WayPoint-SQL-Schema.png -> GitHub Repo -> Misc -> WayPoint-NoSQL-Schema.png
1/28/25 7:30 PM	1	Created Video to share surveys on Instagram. Link: https://www.instagram.com/reel/DFXOyODRLOl/?igsh=ZnF4OWhqbTk0NnVx
1/29/25 9:30 AM	1	Get more insights from Prof. Priya in how do we approach surveys collected and user testing if user doesn't reside in a same country.
2/3/25 9:30 PM	1.5	Created backend shell. Tested the shell and it's running. Pushed to GitHub repo.
2/3/25 11:00 PM	1	Regroup to discuss about Database Schema of PostgreSQL and NoSQL. Update on the frontend shell. Update on the backend shell. Troubleshoot how to run ios on the machine on the first time pulling. Ensure frontend shell can work on both machines. Tasks assignment for the upcoming week.

2/4/25 12:00 AM	1	Frontend shell wasn't working on Russell's machine. Troubleshooting. Cocoapods installed but pod wasn't installed successfully. Finding: XCode wasn't install properly. When running xcode-select -p on terminal. It showed other thing than "/Applications/Xcode.app/Contents/Developer" Details: The problem was that the path to the Xcode command-line tools was not correctly set, causing the xcrun command to be unable to locate the iOS SDK (iphoneos). This resulted in the error message: SDK "iphoneos" cannot be located. Code to run: sudo xcode-select -s /Applications/Xcode.app/Contents/Developer Solved!
2/4/25 9:30 AM	2.5	I encountered an issue while deploying my FastAPI app to Heroku. The deployment failed with a ModuleNotFoundError for the backend module. After investigating, I realized that the folder containing the app was named Implentation with a capital "I", but the Procfile was referencing it as implementation with a lowercase "i". Instead of renaming the folder, I updated the Procfile to correctly reference the folder name with the capital "I" as it appeared in the project. After this update, the deployment was successful, and the app was properly hosted on Heroku.
2/4/25 1:30 PM	1	Problem: Setting up PostgreSQL was challenging, especially connecting pgAdmin to both my local database and Heroku's remote database. I struggled with authentication issues, missing roles, and ensuring my tables were correctly created in both environments. Solution: I configured my local PostgreSQL by setting the correct roles and connected pgAdmin to Heroku using the provided DATABASE_URL. I ensured the database schema was consistent across both environments and created tables using SQLAlchemy. Explanation: This helped me understand how PostgreSQL differs locally and on Heroku, how to manage database credentials, and how to properly set up pgAdmin for database administration.

2/4/25 2:30 PM	1	Problem: Deploying to Heroku failed due to an incorrectly placed Procfile, missing dependencies in requirements.txt, and misconfigured environment variables like DATABASE_URL and SECRET_KEY. Solution: I moved Procfile to the root directory, updated requirements.txt, and set DATABASE_URL correctly in Heroku's environment variables. Restarting the Heroku dyno applied these fixes. Explanation: This taught me the importance of directory structure and configuration files in deployment and how to debug deployment failures using heroku logs --tail.
2/4/25 3:30 PM	2	Problem: After fixing deployment, my app still crashed on Heroku (H10 App Crashed) due to SQLAlchemy not recognizing Heroku's DATABASE_URL format and FastAPI failing to bind to the correct port. Solution: I modified db.py to convert postgres:// to postgresql://, ensuring SQLAlchemy could connect. I also updated Procfile to bind FastAPI to Heroku's $PORT. Explanation: This reinforced the differences between local and production environments, the need for dynamic configurations, and how Heroku manages deployments and environment variables.
2/9/25 7:30 PM	1	Created travel style quiz. Total 7 questions. Scoring system using point-based system.

2/10/25 8:00 PM	2	Regroup / Knowledge Sharing session. Russell: - Explained how the backend works for PSQL part. - Explained how to use pqAdmin tool as database management tool. - Explained the workflow with backend: when writing new code for backend, test it on local machine before pushing to Github. - Showed how deployment works with Github - Heroku setup. General: - Discussed the next workflow in tackling MVPs. - Discussed to re-order MVP priority based on survey. Task Assignments:
2/11/25 1:30 PM	3.5	1. Foreign Key Dependency Issues Problem: Models had incorrect import order, causing foreign key errors. Solution: Adjusted import order in __init__.py to ensure dependencies load correctly. 2. Circular Import Issue Problem: Importing Base from db.py led to circular dependencies. Solution: Moved Base to base.py and updated model imports. 3. Missing email-validator Error Problem: FastAPI required email-validator, despite being in requirements.txt. Solution: Reinstalled dependencies manually on Heroku. 4. uvicorn: command not found on Heroku Problem: uvicorn was missing in the runtime environment. Solution: Updated Procfile to use python -m uvicorn and verified installation. 5. Heroku App Not Restarting Properly Problem: Deployment changes weren't reflecting. Solution: Restarted the app and purged Heroku build cache. 6. App Not Binding to $PORT Problem: FastAPI wasn't binding correctly to the environment port. Solution: Ensured uvicorn runs with --port=${PORT} in Procfile. 7. Database Connection Test Failed on Heroku Problem: Remote database connection wasn't verifying. Solution: Created /test-db endpoint and confirmed it works.

2/11/25 5:00 PM	0.5	Running db.py to Create Tables on Heroku: Problems & Solutions 1. Running db.py on Heroku caused ModuleNotFoundError: No module named 'app' Fix: Used PYTHONPATH=. python app/db/db.py to ensure the correct module path. 2. Tables were not appearing in Heroku Postgres after running db.py Fix: Explicitly set Base.metadata.schema = "public" in db.py to ensure tables are placed in the correct schema. 3. Needed a way to manually trigger db.py on Heroku Fix: Opened a Heroku shell with heroku run bash -a waypoint-travel, then executed: PYTHONPATH=. python app/db/db.py 4. Wanted to verify if tables were created in Heroku Postgres Fix: Used Heroku Postgres CLI to check tables: heroku pg:psql -a waypoint-travel SELECT tablename FROM pg_tables WHERE schemaname = 'public'; Final Outcome: Successfully ran db.py on Heroku, ensuring tables were created in the correct schema.

2/12/25 10:00 AM	1.5	Summary of Fixes & Progress 1. CRUD Implementation for Users Created POST /users → Create User (with password hashing). Created GET /users/{user_id} → Retrieve User by ID. Created PUT /users/{user_id} → Update User (name, email, password). Created DELETE /users/{user_id} → Delete User. 2. Fixed Errors InvalidRequestError → Added ForeignKey("users.id") in quiz_model.py. TypeError: 'password' is an invalid keyword argument for User → Ensured password_hash is used in user_model.py. NameError: name 'user_schema' is not defined → Fixed incorrect import in user_routes.py. zsh: no matches found: passlib[bcrypt] → Installed using pip install "passlib[bcrypt]". 3. Fixed Duplicate URL Path Issue Issue: "/users/users/{user_id}" in FastAPI /docs. Fix: Removed redundant /users prefix from routes in user_routes.py. 4. Tested Locally Verified all CRUD operations using FastAPI /docs. Confirmed correct URL paths after fixing duplication. Next Steps Test CRUD operations for Itineraries, Places, Badges, and Quiz Results. Once confirmed, deploy to Heroku and re-test on live API.

2/12/25 11:30 AM	1	Log Summary for Places CRUD Implementation 1. Implemented CRUD for Places Created POST /places → Add a new place. Created GET /places/{id} → Retrieve a place by ID. Created PUT /places/{id} → Update place details. Created DELETE /places/{id} → Remove a place. 2. Fixed Issues Fixed timezone inconsistency → Ensured last_updated is stored in UTC. Resolved datetime.utcnow() deprecation warning → Used datetime.now(timezone.utc).replace(tzinfo=None). Verified timestamps consistency → Matched last_updated with created_at format. 3. Successfully Tested POST /places → Verified place creation with manual data. GET /places/{id} → Retrieved created places correctly. PUT /places/{id} → Updated place details without timezone mismatch. DELETE /places/{id} → Successfully removed places from the database. Next Steps Implement User Favorites (user_favorite_routes.py). Ensure Users ↔ Places relationship works correctly. Prepare for Google Places API integration.

2/12/25 3:30 PM	0.5	Log Summary for User Favorites Implementation 1. Implemented CRUD for User Favorites Created POST /user_favorites → Add a place to favorites. Created GET /user_favorites/{user_id} → Retrieve a user's favorite places. Created DELETE /user_favorites/{favorite_id} → Remove a favorite place. 2. Fixed Issues Validated user and place existence before adding a favorite. Prevented duplicate favorites by checking existing records. Ensured added_at timestamp is stored in UTC for consistency. 3. Successfully Tested POST /user_favorites → Added places to favorites successfully. GET /user_favorites/{user_id} → Retrieved correct favorites for users. DELETE /user_favorites/{favorite_id} → Removed favorites as expected. 4. Updated main.py Included user_favorite_routes in FastAPI router. Next Steps Implement Badges (badge_routes.py). Ensure User ↔ Badges relationship works correctly.

2/12/25 4:00 PM	1.5	Set Up Google Places API Integration Chose Google Places API (Old Version) for simpler API key authentication. Tested API manually using Postman & cURL. Implemented FastAPI Route for Places Search Created /places/search endpoint to fetch nearby places. Integrated Google Places API (maps.googleapis.com). Cached results in PostgreSQL to reduce API calls. Restricted API to British Columbia (BC), Canada Implemented latitude/longitude boundary check to block requests outside BC. Verified restriction by testing New York (Successfully blocked). Error Handling & Optimizations Improved handling for invalid locations and API failures. Implemented database caching to avoid redundant API requests. Added X-Goog-FieldMask to optimize API responses. Tested & Debugged API Responses Verified working results for Vancouver, BC. Ensured API key security using environment variables (.env, Heroku Config Vars).

2/13/25 10:00 AM	1.5	Tasks Completed: - Configured React Navigation with StackNavigator & BottomTabNavigator. - Created Login & Signup screens with placeholder values. - Implemented Profile screen with user details, Edit Profile (future), and Log Out. - Built Settings screen with Travel Style, Notifications, Language, and Account Management. - Used SafeAreaView & ScrollView to fix UI layout issues. - Replaced deprecated Picker with @react-native-picker/picker. - Ensured dynamic spacing to prevent overlap with iPhone Dynamic Island. Next Steps: - Implement Edit Profile feature. - Add form validation for Login & Signup. - Prepare backend integration for authentication and profile updates. - Enhance UI with better styling.

2/13/25 11:30 AM	0.75	Tasks Completed: - Backend Integration for Authentication: - Reviewed backend schemas, models, and routes. - Confirmed API endpoint for user registration (POST /users/). - Identified and fixed login API endpoint (POST /users/auth/login). - Updated SignupScreen.js: - Connected to backend (POST /users/) for user registration. - Handled form submission, API request, and error handling. - Added navigation to Login screen upon successful signup. - Updated LoginScreen.js: - Integrated POST /users/auth/login using query parameters. - Ensured login request matches the correct FastAPI route. - Redirects users to Main app upon successful login. - Displays alerts for errors and invalid credentials. - Backend API Testing & Debugging: - Successfully tested user registration and login via FastAPI. - Ensured POST /users/auth/login worked with query parameters. - Verified API response handling in React Native app. Next Steps: - Implement persistent authentication (store session/token). eg. JWT - Add form validation for signup & login fields. - Enhance UI styling & error messages for better user experience. - Implement Edit Profile feature in ProfileScreen.js.

2/13/25 12:45 PM	2.25	Attempted to rename React Native app from "frontend" to "WayPoint" → Encountered issues, reverted to "frontend" Updated package.json and app.json to reflect the correct app name → Reverted due to build errors Checked and updated Xcode Signing & Capabilities → Used free Apple ID for provisioning Attempted to set correct Bundle Identifier for Firebase setup → Reverted due to build failures Installed Firebase dependencies (@react-native-firebase/app) → Successfully installed Fixed CocoaPods issues with modular headers → Modified Podfile and ran pod install --repo-update Configured Firebase in AppDelegate.swift → Updated to FirebaseApp.configure() Encountered xcodebuild error code 65 while running iOS build → Attempted multiple fixes Deleted and reinstalled CocoaPods, node_modules, and Xcode DerivedData → No success Manually deleted ios/build/ and cleaned Xcode project → Issue persisted Ran xcodebuild clean and pod install --repo-update → Did not resolve the issue Tried running the app via Metro Bundler (npx react-native run-ios) → Still failed Decided to fully reset the project by deleting and reinstalling all dependencies → Still encountering build issues

2/14/25 2:00 PM	4	Summary Log: Firebase Realtime Database Setup & Next Steps Problems: Multiple React-Core dependencies causing conflicts React-RCTAppDelegate not linking correctly FirebaseAuth/FirebaseAuth-Swift.h file not found (even though not needed) Xcode build error: "unable to initiate PIF transfer session" ReactCommon module redefinition error CocoaPods installation issues Solutions Attempted: Refactored Podfile to use use_modular_headers! and fixed React-Core conflicts Updated AppDelegate.swift with FirebaseApp.configure() Removed and reinstalled dependencies (node_modules, Pods, Podfile.lock) Cleared Xcode cache (DerivedData, xcodebuild clean) Ensured only needed Firebase modules were installed What's Next: Start fresh to ensure a clean build Get React Native running first before adding Firebase Verify Podfile with default settings, then add Firebase Test a basic build (npx react-native run-ios) before integrating Firebase features Implement Firebase Realtime Database CRUD to confirm it works Next attempt: Clean setup from the beginning

2/14/25 7:00 PM	1.5	Work Log: Firebase Integration in React Native (iOS) Problems & Solutions Firebase Not Initializing (No Firebase App '[DEFAULT]' has been created) Firebase was not auto-detecting GoogleService-Info.plist. Manually initialized Firebase in firebase.js. Firebase connected successfully using manual config. Missing or Invalid FirebaseOptions property 'apiKey' Error Firebase could not find apiKey from GoogleService-Info.plist. Verified plist format and corrected key names. Ensured plist was inside Implementation/frontend/ios/. Linked plist in Xcode under Build Phases → Copy Bundle Resources. Still using manual config; plist auto-detection needs verification. Firebase Data Not Appearing in Realtime Database Firebase connection worked, but no data appeared. Updated Firebase database rules to allow reads/writes. Created a test function in LoginScreen.js to write data. Confirmed successful data write to Firebase Console. Next Steps Remove manual Firebase config and verify plist auto-detection. Fetch and display a list of data from Firebase. Secure Firebase Database rules based on authentication.

2/14/25 8:30 PM	0.5	Work Log: Firebase Auto-Detection Fix in React Native (iOS) Problems & Solutions **Firebase Not Initializing Automatically** Firebase was not detecting GoogleService-Info.plist. Manually initialized Firebase in AppDelegate.swift. Confirmed Firebase auto-detection now works. **Missing Firebase Setup in AppDelegate.swift** React Native Firebase requires Firebase to be initialized in AppDelegate.swift. Added FirebaseApp.configure() inside didFinishLaunchingWithOptions. Restarted the app and confirmed successful Firebase initialization. **Plist File Not Being Read by Xcode** GoogleService-Info.plist was not linked in Build Phases → Copy Bundle Resources. Manually added the plist file in Xcode. Verified correct plist location in Implementation/frontend/ios/. Next Steps Fetch and display data from Firebase in the app. Secure Firebase database rules based on authentication.

2/17/25 10:30 PM	1.25	Meeting Notes Summary (February 17, 2025) 1. Upcoming 1-Week Tasks Focus on the next two MVPs: Personalized Recommendations (Google Places API integration for recommendations) Interactive Map (Google Maps API integration for visualization) 2. Past Week Progress Updates Team members shared knowledge and updates on completed tasks. 3. Heroku Backend Server Documentation Discussion on CRUD operations for backend endpoints 4. Connecting Simulator to Heroku Server Setting up the React Native simulator to interact with the backend hosted on Heroku. 5. Planning for Video Workflow Outlining the video workflow for the mid-term report. Deciding on tools and steps for video creation. 6. Firebase Realtime Database & App Distribution Revisiting Firebase Realtime Database setup. Setting up Firebase App Distribution for testing. 7. Google Places API on /search Path Integrating Google Places API for search functionality. Ensuring that the API can return filtered results based on user preferences.

2/22/25 10:30 PM	2	Fixes & Improvements in SettingsScreen.js and Backend Initial Issues & Fixes Login Issues (422 Unprocessable Content) Issue: FastAPI rejected login requests due to incorrect request body formatting. Fix: Ensured email and password were correctly passed in the axios.post request in LoginScreen.js. User Data Not Persisting After Login Issue: Logged-in user details were not being saved for profile and settings. Fix: Stored user data in AsyncStorage after a successful login. Navigating to Home Screen After Login (REPLACE Error) Issue: navigation.replace('HomeScreen') failed due to missing screen. Fix: Updated App.js to correctly route users to Main after login. Backend Issues & Fixes Travel Style Not Saving (422 Unprocessable Content) Issue: FastAPI expected user_id in the request body for PUT requests. Fix: Created QuizResultUpdate schema to accept only travel_style. Modified PUT /quiz_results/user/{user_id} to update travel style correctly. Fetching Travel Style for User Settings Issue: Travel style was not being retrieved from PostgreSQL. Fix: Created GET /quiz_results/user/{user_id} to fetch travel style. Updated SettingsScreen.js to call this API and store the result in AsyncStorage. Frontend UI Issues & Fixes VirtualizedLists Error (Nested inside ScrollView) Issue: DropDownPicker (using FlatList) conflicted with ScrollView. Fix: Replaced ScrollView with KeyboardAvoidingView, set removeClippedSubviews={false}, and adjusted zIndex. Dropdown Overlapping UI Issue: Travel Style dropdown was overlaying the Language dropdown. Fix: Wrapped dropdowns in View with zIndex, used modalProps={{ animationType: 'fade' }} to prevent overlap. Language Change Incorrectly Updating Travel Style Issue: Changing language was overwriting travel_style in AsyncStorage. Fix: Updated updateLanguage function to modify only language, preserving travel_style.
2/22/25 2:00 PM	1	Drafted video recording flow.

2/22/25 3:00 PM	0.5	Fixed QuizScreen and LoginScreen to ensure first time user is able to take the quiz and save it correctly.
2/23/25 1:00 PM	3	Google Maps Integration on iOS Problem: Needed to integrate Google Maps SDK for iOS. Solution: Installed and configured Google Maps SDK (v7.0.0). Problem: API key was stored in Info.plist, causing security issues. Solution: Moved API key to .env and dynamically loaded it in AppDelegate.swift. Problem: "Tried to register two views with the same name AIRMap" error. Solution: Ensured only one instance of react-native-maps to fix duplicate registration. Problem: API key needed to be dynamically passed to Swift. Solution: Updated Podfile to load .env variables and inject API key into the build. Problem: Map was not displaying if the API key was missing. Solution: Added error handling to InteractiveMapScreen.js to show a message when the API key is missing. Problem: Needed to verify Google Maps displayed properly. Solution: Successfully displayed Google Maps centered on Vancouver, BC in the iOS simulator. Next Steps Implement user location tracking. Add custom markers for points of interest. Optimize map rendering and interactions. Test on a real iOS device for stability.
2/23/25 10:00 PM	2	Added markers with Google Places API.
2/24/25 10:00 AM	1	Check on Firebase app Distribution. Upon checking, need to enrol in Apple Developer Program. Completed: Register for enrolment under educational institution. Request to waive the fee. What's next: Waiting for reply.
2/24/25 5:00 PM	1.75	Video Recording for Mid Term Report Checkpoint
2/24/25 9:30 PM	2.5	Finishing and finalizing midterm report. Preparing documents for submission. Push log to the github.

2/26/25 12:00 PM	0.25	Midterm Feedback by Priya
3/4/25 1:00 PM	3	Interactive Recommendations Screen Ensured the filter buttons update dynamically based on the selected travel style. Adjusted the zoom in, zoom out, and fullscreen buttons, positioning them correctly in the bottom-right corner of the map (35% of the screen). Refactored the category filter logic to dynamically pull unique categories based on available places. Places Router Updates Updated the API request to fetch places dynamically based on the selected travel style. Extracted unique categories from the API response to dynamically populate the category filter. Ensured error handling for cases where a travel style is missing or places cannot be retrieved. Next Steps Fix the extra white space issue in the scrollable category filter ➡ Investigate margin/padding issues in filterScrollContainer and listContainer Implement API caching for places data ➡ Cache the API response to prevent unnecessary requests when the same location and travel style are selected again

3/4/25 4:00 PM	2	Updated user_routes.py Removed quiz_results dependency. Now stores travel_style_id directly in the users table. Defaults travel_style_id = 4 (Undefined) on signup. Allows updating travel_style_id after quiz completion. Created Stack Navigator (AppNavigator.js) Wrapped BottomNavigation inside a Stack.Navigator. Added QuizScreen as a separate screen for navigation. Ensures navigation.navigate('QuizScreen') works from HomeScreen. Updated App.js Now uses AppNavigator instead of BottomNavigation. Ensures proper screen navigation for non-tab screens. Updated HomeScreen.js Fetches travel_style_id from AsyncStorage & Firebase. Shows "Take Quiz" banner if travel_style_id = 4. Logs user activity in Firebase when viewing the home screen. Updated LoginScreen.js & SignupScreen.js Integrated Firebase Realtime Database. Stores user details in Firebase on signup. Retrieves and syncs user data on login. 🛠 Next Steps: Update QuizScreen.js Ensure quiz tracking and progress-saving work correctly. Properly update travel_style_id in Firebase & Backend.
3/4/25 10:30 PM	2	Update QuizScreen Added Firebase to track Quiz Progression
3/5/25 10:00 AM	1	Updated Buttons for Zoom and Fullscreen Updated icon for list to load from Places API fixed the list spacing (scrollFilterContainer)

3/6/25 10:30 AM	2	Fix device node modules issue. Unable to launch iOS device simulator. Checked the existing Bundler, Ruby, and CocoaPods versions. Verified the project's Gemfile.lock for required dependencies. Removed old Bundler 1.17.2 references from .bundle and vendor/bundle. Updated Bundler to the latest version using gem update bundler. Ran rbenv rehash to ensure the system recognized the new Bundler version. Installed missing gems bigdecimal and logger, which were removed from Ruby 3.4.2 defaults. Removed and reinstalled CocoaPods using gem uninstall cocoapods and gem install cocoapods -v 1.15.2. Updated the project dependencies using bundle install and bundle update --bundler. Verified the installed CocoaPods version with pod --version. Ran bundle exec pod install in the ios directory to set up iOS dependencies. Started Metro Bundler without issues using npx react-native start --reset-cache. Successfully ran the app using npx react-native run-ios.
3/6/25 12:30 PM	3.5	◇ Backend Changes (FastAPI & PostgreSQL) Migrated user_id to UUID Format Replaced Integer user IDs with UUID for security, scalability, and consistency across PostgreSQL, Firebase, and AsyncStorage. Added Email Availability Check Before Signup Implemented /users/check_email/ API to prevent duplicate signups. Integrated email validation in SignupScreen.js. Created itinerary_models.py for Collaborative Planning Consolidated Itinerary, ItineraryDay, ItineraryMember, and Activity models. Ensured UUID support across all itinerary models. Added API Endpoint to Fetch Travel Style Details Created /travel_styles/{travel_style_id} API to return travel style name & description. Used in ProfileScreen.js to enhance UX.

3/6/25 12:30 PM		◇ Frontend Changes (React Native) Ensured user_id is Always Stored as UUID String Updated SignupScreen.js, LoginScreen.js, and HomeScreen.js to store user_id as a UUID string in AsyncStorage. Implemented Travel Style Update Across All Data Sources SettingsScreen.js now updates travel_style_id in AsyncStorage, PostgreSQL, and Firebase for real-time sync. Fixed Dropdown Issue in SettingsScreen.js Replaced onChangeValue with onSelectItem to prevent alerts from triggering when expanding the dropdown. Updated HomeScreen.js to Use Firebase Instead of PostgreSQL Fetches travel_style_id from Firebase + AsyncStorage instead of making API calls, improving performance. Enhanced ProfileScreen.js to Display Travel Style Name & Description Uses travel_style_id from AsyncStorage to fetch travel style details from PostgreSQL. Shows meaningful travel style descriptions instead of just an ID.
3/6/25 4:45 PM	1.5	Fix Homescreen and added some details. More descriptions to be added....
3/7/25 10:00 AM	1.5	Itinerary Updated Router Added Itinerary List Screen.
3/7/25 11:30 PM	2	Itinerary: Added Itinerary Details Screen. Added Swipeable feature to delete.
3/7/25 2:30 PM	1.5	Itinerary: Added new feature to re order the day. Using npm install react-native-draggable-flatlist Due to this feature, I have to update the model, schema, and router. Add "Add Day" button at the bottom of Swipable Flat List

3/7/25 4:00 PM	1.5	Itinerary: Added ItineraryDayScreen to show all the activities within a day. Added Function for time sorting. Using Model to add itinerary. Implement useFocusEffect after navigating back from ItineraryDay to ItineraryList to refresh the activities
3/10/25 4:00 PM	1	
3/10/25 4:00 PM	0.6	Simone shared about the Navigation progress (done) and Chatbot. Russell shared about 50% progression of the Itinerary.
3/10/25 10:00 PM	1	Done
3/13/25 2:30 PM	3	Backend: Fixed POST /itineraries/ by ensuring created_by matches an existing user ID. Implemented GET /itineraries/{itinerary_id} to return full itinerary details. Added POST /itineraries/{itinerary_id}/days/ to allow adding days to an itinerary. Added POST /itineraries/{itinerary_id}/days/{day_id}/activities/ to support adding activities to a day. Implemented DELETE /itineraries/{itinerary_id} to allow itinerary deletion. Implemented DELETE /itineraries/{itinerary_id}/days/{day_id} to allow itinerary day deletion. Implemented PUT /itineraries/{itinerary_id} to allow editing itinerary details. Frontend: Implemented Itinerary List Screen to fetch itineraries from PostgreSQL and display them. Improved Itinerary List Item UI for better readability. Implemented Itinerary Form Screen to allow creating and editing itineraries. Integrated react-native-calendars for selecting start and end dates in the itinerary form. Ensured that upon itinerary creation, the user is navigated to ItineraryDetailScreen with the new itinerary ID. Implemented Itinerary Detail Screen to display itinerary details and handle deletion. Added Itinerary Day Screen to list all activities for a given itinerary day. Implemented react-native-date-picker for activity time selection. Converted activity list in ItineraryDayScreen to be swipeable, with a placeholder delete action. Fixed swipeable delete button height issue to ensure alignment with activity cards. Improved UI consistency across itinerary-related screens. Next steps: Implement actual deletion for activities in ItineraryDayScreen.

		Allow users to edit itinerary days and activities instead of just deleting them. Integrate Firebase Realtime Database for tracking changes in itineraries.
3/13/25 10:00 PM	2	Added an Invite Collaborators button in the Itinerary Detail Screen Placed the button in the bottom fixed button container Designed the InviteCollaboratorsScreen with inline styling Created a search bar for filtering users Displayed a list of users with an Invite button Added a pending invites section to track invited users Used dummy data for now, preparing for Firebase integration Fixed Text strings must be inside a Text component warning Wrapped the trash icon inside a Text component Checked and ensured all text elements are properly inside Text components

3/14/25 10:30 PM	1.5	Updated InviteCollaboratorsScreen.js to store invitations under /invitations/invitee/{userId}. Tracked pending invites under /live_itineraries/{itineraryId}/pendingInvites/. Ensured invited users are displayed in InviteCollaboratorsScreen.js. Updated ItineraryListScreen.js to show pending invitations before shared itineraries. Fetched invitations from /invitations/invitee/{userId}. Added "Accept" and "Decline" buttons with placeholder alerts.
3/15/25 10:00 AM	3	Fixed fetchItineraryDetails to update days properly Ensured the invite button only shows if the user is the itinerary owner Updated fetchSharedItineraries to fetch owner details from FastAPI Modified renderItineraryItem to display owner name and email Prevented users from inviting themselves in InviteCollaboratorsScreen Ensured adding a new day updates the UI correctly
3/15/25 5:00 PM	1.5	Improved UI/UX for Day: Implemented PUT endpoint for editing itinerary days Added left swipe action to reveal the Edit button on day cards Updated the modal to pre-fill day details for editing Improved date display format to "Sat, Mar 15" using locale options Handled timezone issues by parsing date strings into local Date objects Fixed button rendering logic to show "Remove" for collaborators and "Delete" for the owner Ensured isCollaborator updates correctly before rendering buttons Updated fetchItineraryDetails to properly check if the user is a collaborator Confirmed navigation back to ItineraryListScreen after removal Verified Firebase updates when a user removes themselves from an itinerary
3/15/25 5:30 PM	1	ItineraryDayScreen: Fix the delete activity height to match the card. ItineraryListScreen: Show Last Updated for better UX. Itinerary Models, Routes, Schema: Add last_updated_by column to the itineraries table

3/15/25 9:00 PM	2	Fix Activity Update API - Convert UUIDs to Strings in Response Encountered an issue where updating an activity was failing Checked the frontend code for issues in the request payload Verified that the correct X-User-Id header was being sent Ensured the itineraryId, dayId, and activityId were correctly passed Reviewed the FastAPI update activity route for possible issues Debugged the request and response to find inconsistencies Identified that UUID fields in the response were causing validation errors Prepared to modify the FastAPI response to return UUIDs as strings
3/16/25 2:50 PM	3	InviteCollaboratorsScreen: Added function for remove collaborators as the owner. Added a dedicated edit icon on the day card to decouple editing from the swipe gesture Integrated a collaborators list in the Overview tab using a Firebase listener Improved the Overview UI with a styled card and clear section for collaborators Ensured add day, edit day, delete day, and drag-and-drop reorder functionalities remain intact Refactored and organized code for clarity and maintainability
3/16/25 2:50 PM	3	Firebase initialization is performed both natively and in JavaScript Source of API configuration is embedded in native files (google-services.json, GoogleService-Info.plist) Native initialization in AppDelegate.swift remains necessary for iOS support Implementation of Firebase Storage for images has been challenging and remains unresolved
3/16/25 10:30 PM	2	Re-try Firebase again - Part 1 Firebase Storage module initialization not clear within the React Native context Difficulty integrating native Firebase configurations with JavaScript code Uncertainty in setting up correct file paths and handling file uploads Potential permission issues when accessing device storage for images

3/17/25 12:00 PM	5	Re-try Firebase again - Part 2 Problem: Build errors due to warnings treated as errors and non-modular includes in Firebase modules Adjusted the Podfile's post_install hook to remove -Werror and disable pedantic warnings Allowed non-modular includes for RNFB targets by setting CLANG_ALLOW_NON_MODULAR_INCLUDES_IN_FRAMEWORK_MODULES Removed a stray reference ("x") causing a syntax error in the Podfile Cleaned Derived Data, deintegrated, updated and reinstalled CocoaPods Converted Objective-C import syntax to Swift import (import Firebase) for Swift usage Conclusion: 10 Hours spent for Firebase Storage service and still not working. Going for another alternative: AWS S3
3/17/25 9:00 PM	4	Attempting AWS S3 for image storage. Successful implementation for AWS S3. Image upload feature with react-native-image-picker is not working due to iOS limitation. Alternative: react-native-image-crop-picker Updated the S3 upload endpoint to generate a presigned URL and update the itinerary record's extra_data with the new image URL Modified the extra_data field in the Itinerary model to use MutableDict so in-place JSON updates are detected Adjusted the React Native ItineraryDetailScreen to fetch the image URL from extra_data and display it using an ImageBackground Troubleshot duplicate image rendering by removing an extra Image component Provided an alternative endpoint approach to stream the image from S3 if needed

3/19/25 9:30 AM	2.5	Fixed alignment issues for the "Add" button in PlacesModal.js Added spacing between text input and "Add" button in PlacesModal.js Made notes section scrollable while keeping the title fixed Allowed notes panel to expand dynamically but stop at a max height Integrated Firebase Realtime Database for storing itinerary notes Replaced AsyncStorage with Firebase for notes persistence Synced notes preview in ItineraryDetailScreen.js with Firebase updates Adjusted modal layout in NotesModal.js to provide more writing space Ensured "Tap to add notes" appears when notes are empty Fixed "Places to Visit" container height to be flexible Ensured "Places to Visit" list appears below the title Allowed places panel to grow dynamically instead of having a fixed height Adjusted Firebase integration for NotesModal.js by correctly passing itineraryId Fixed inconsistencies between notesPanel and placesPanel heights Ensured preview text in notes appears below the title Wrapped notes preview in a ScrollView to enable proper scrolling Adjusted modal size in NotesModal.js to allow more space for text input Ensured notes persist and update in real-time across devices Implemented better text wrapping and positioning for notes preview Updated useEffect hooks in both NotesModal.js and ItineraryDetailScreen.js for better state handling
3/19/25 12:00 PM	0.1	Fixed "Remove" button not showing for collaborators Properly set isCollaborator when fetching collaborators Ensured collaborators list updates correctly in Firebase
3/19/25 2:00 PM	0.5	Replaced AsyncStorage with Firebase for storing the places list Ensured places sync in real-time across devices Fixed itineraryId not being passed to PlacesModal Updated ItineraryDetailScreen to fetch places from Firebase Ensured PlacesModal saves and retrieves places from Firebase Removed all AsyncStorage references for places list
3/19/25 2:30 PM	1.25	Identified estimated_cost missing in API response Fixed ActivitySchema to include estimated_cost Ensured estimated_cost is explicitly cast as float in get_itinerary Verified API response using Swagger UI Updated fetchItineraryDetails to sum total estimated cost of all days Preserved sortActivitiesByTime while calculating total cost Updated UI to display the dynamic total estimated cost in ItineraryDetailScreen

3/19/25 3:45 PM	0.75	Added a horizontal scroll view for the collaborators list Updated the collaborators section to allow left to right scrolling Ensured the scroll view does not show horizontal scroll indicators Nodified styles to align items in a row and add spacing Tested the UI to verify smooth horizontal scrolling
3/19/25 4:30 PM	0.5	added a modal for entering and managing other costs allowed selection of cost type and subtype from a predefined list enabled users to input item name and amount displayed total other costs in the other costs panel listed saved other costs inside the modal added functionality to remove saved costs
3/19/25 8:30 PM	0.5	Updated CFBundleDisplayName in Info.plist Renamed project in Xcode under Identity and Type Updated Bundle Identifier in Xcode (if needed) Cleared cache and rebuilt the project Replaced app icons in Images.xcassets/AppIcon.appiconset/ Ensured new icons appear in Xcode Cleaned and rebuilt the project
3/19/25 9:00 PM	0.25	Identified duplicate weather API calls in HomeScreen.js Found API calls triggered in both handleLocationGranted and useEffect Removed one API call to prevent redundant requests Ensured weather data is fetched only once when location updates
3/19/25 9:15 PM	0.25	Added a centered "No Itineraries" message when the list is empty Updated Personal Itineraries and Shared Itineraries to display the empty state Replaced the Add Itinerary button with a FontAwesome "+" icon Styled the floating "+" button to be minimalist and positioned at the bottom-right

3/19/25 9:30 PM	1.75	Feature Update: UI Enhancements & Date Picker Fixes Implemented Google Places Autocomplete with a modal for destination selection Ensured modal expands fully and remains visible when clicked Added a clear ("⨯") button to easily remove the destination input Adjusted destination input size to match other fields (90% input, 10% clear button) Standardized font size (18px) and left padding across all input fields Fixed date picker behavior to ensure the first click highlights the start date in blue Ensured date range selection properly marks start, end, and in-between dates Updated calendar logic to automatically close after selecting an end date Improved UI consistency for input spacing, margin, and alignment
3/20/25 10:15 PM	0.5	Created reusable FeatureCarousel component Added horizontal FlatList with paging and snapping Used image backgrounds for each feature card Applied resizeMode="cover" to handle varying image sizes Made layout responsive to SafeAreaView using useWindowDimensions() Added dark overlay for text readability Implemented pagination dots under the carousel Made dots tappable to scroll to specific cards
3/20/25 10:45 PM	0.5	Added a feature carousel with image backgrounds and pagination Implemented a tappable pagination dot system for smooth navigation Created a "Start Your Journey" banner with animated button effects Conditionally displayed the banner based on quiz completion state Always displayed the logo in the top left corner Adjusted SafeAreaView and header styles to reduce extra space Fine-tuned padding and height for better logo positioning
3/20/25 11:15 PM	0.75	Replaced separate letter buttons with full-width answer buttons Ensured the entire button is tappable for better UX Added bounce animation when selecting an answer Displayed checkmark on the right side of the selected answer Used useRef for animatedScales to maintain hook order Moved useAnimatedStyle outside JSX to prevent render errors Fixed "Rendered Fewer Hooks Than Expected" issue

3/21/25 1:45 PM	0.25	Wrapped entire content inside ScrollView to allow vertical scrolling Moved the header outside ScrollView to keep it fixed at the top Applied position absolute to header to ensure it stays visible Added marginTop to ScrollView to prevent overlap with the fixed header Used zIndex and elevation to keep header above other content Added shadow for better visual separation
3/21/25 8:30 PM	0.5	Added isLoadingResult state to control when loading shows Updated determineTravelStyle to show loading before result Used Progress.CircleSnail as the animated loading spinner Added a delay using setTimeout before showing the result screen Created loadingContainer and loadingText styles for clean layout Installed react-native-svg to fix CircleSnail "Unimplemented" error
3/21/25 8:30 PM	0.25	Added Confetti Animation When Result Appears Added Fade-In Animation for the Result Card Using Reanimated Updated Result Card Layout for Better Visual Hierarchy Increased Emoji Size and Spacing Separated Result Text into Multiple Lines for Clarity Updated Result Card Background Color to Brand Navy #1E3A8A Adjusted Text Colors for Contrast on Dark Background Improved Card Elevation and Shadow for Depth
3/21/25 8:45 PM	0.5	Replaced animatedScales with a 2D animatedScalesRef using useRef Created animatedStylesRef outside render loop for safe hook usage Removed all useSharedValue and useAnimatedStyle from inside loops and conditionals Ensured only the selected answer animates when tapped Fixed runtime errors caused by missing or misplaced hook declarations Validated that quiz screen runs cleanly without crashing
3/21/25 9:15 PM	0.25	Replaced bottom navigation emoji icons with FontAwesome icons Updated CustomBottomNavigation to render FontAwesome icons with dynamic color Adjusted navIcon style for better alignment with FontAwesome Replaced More menu emoji items with FontAwesome icons Used sign-in-alt icon for Check In to avoid duplicating map icon Ensured layout and spacing remain consistent in both menus

3/21/25 9:30 PM	1.25	Created a modal component to display AI-generated packing tips Used OpenAI endpoint to generate suggestions based on weather and city Added tap interaction to the left side of the weather widget Handled modal visibility and loading states properly Fixed missing weather location name with fallback city logic Made sure the modal opens only when data is valid Confirmed that clicking the widget now triggers a smart packing tip
3/23/25 7:00 PM	3	Add AWS S3 bucket and created users/{userId}/profile.jpg folder structure Create presigned URL FastAPI route for profile photo upload Upload image from React Native using Image Crop Picker and XMLHttpRequest to S3 Save uploaded photo URL to Firebase Realtime Database Add image display logic in ProfileScreen with fallback to AsyncStorage Auto-refresh profile image using Firebase onValue listener Append ?ts=Date.now() to bust image cache after upload Show spinner (ActivityIndicator) while uploading image Display "Tap to change photo" hint below profile picture Handle image picker cancel without showing error Fix AWS S3 bucket permissions to allow public image access Ensure image loads with proper Firebase and S3 structure Prevent multiple uploads by disabling button while uploading
3/23/25 10:00 PM	1	added floating draggable button to HomeScreen button opens Chatbot screen when pressed error: upon clicking, the button stucks at the top left.
3/23/25 11:00 PM	1	Replaced PanResponder with Gesture.Pan from Gesture Handler Used useSharedValue and useAnimatedStyle for dynamic positioning Applied clamp to restrict dragging within screen bounds Set initial position to bottom right using screen dimensions Used Gesture.Simultaneous to handle both drag and tap Wrapped navigation inside runOnJS to avoid UI thread crash Ensured floating button is rendered after all other content Applied absolute positioning and high zIndex to stay above modals Removed TouchableOpacity and used View to avoid gesture conflicts
3/24/25 12:00 AM	0.8	Separated TouchableOpacity from direct string content Wrapped Tap to change photo inside its own <Text> component Moved image and text inside a parent <View> for layout safety Ensured no raw strings are rendered outside <Text> tags Prevented potential crash from unexpected text in TouchableOpacity block

3/24/25 10:45 AM	0.5	Added input validation for email and password Displayed specific inline error messages instead of alerts Applied red border to inputs with errors Added password visibility toggle using FontAwesome icons Fixed padding/margin to align email and password fields Ensured errors persist visually after failed login attempts Removed double borders on password input field Used consistent styles between email and password fields Cleaned up state management for error updates on input change
3/24/25 12:00 PM	2.5	Reported unexpected charges for March billing Clarified that the project was for academic purposes and part of a student assignment Provided project details and confirmed usage intentions Google Cloud Support reviewed billing and usage activity Support escalated the case for further review Awaiting final resolution or credit approval from Google Cloud team
3/24/25 3:00 PM	2.5	Work on Places API (New) and update logic Updated to use latest Google Places API (v3) with includedTypes Saved API results to PostgreSQL with proper caching Added /cached router to fetch nearby places from DB Applied travel style filtering to cached results Refactored FlatList and MapView to share same filtered list Displayed placeholder image for each item Capitalized travel style and category labels for better UI Ensured category formatting handles underscore and multiple words Synced map marker press to scroll FlatList Synced FlatList item press to zoom and open map marker callout Fixed unique key warning in FlatList by adding index in keyExtractor
3/24/25 10:00 PM	1.5	Walk through the workaround for new API route. We are not pulling API using Places API all the time anymore since we put a limitation. Introduced places/search/cached as the route we are using for pulling data from our own database.
3/25/25 9:00 PM	0.5	Converted cached_data values to valid JSON strings (double-quoted) Ensured cached_data column is importable as JSONB into PostgreSQL Matched CSV columns with database table structure Filled empty rating values with 0.0 Saved final version of CSV for Heroku import via psql using \copy command

3/25/25 10:30 PM	1.5	Replaced GooglePlacesAutocomplete with custom modal using Places API v3 Created WeatherSearchModal component with city suggestions and selection Implemented debounce on city input to limit API requests Fetched place details including coordinates using placeId Saved selected city to AsyncStorage for persistence Called weather API using selected city coordinates Rendered modal using React Native Modal with 90 percent height Integrated modal into HomeScreen with proper state control Fixed hook usage and debounce-related bugs Verified and polished modal appearance and functionality
3/26/25 10:00 AM	0.5	Added custom modal to search destinations using Google Places API Updated modal to return both city and country Formatted selected destination to store as "City, Country" string Updated itinerary form to display and submit destination string properly Fixed render error from object being passed to Text Ensured compatibility with PostgreSQL text field requirement
3/26/25 3:00 PM	1	Added editable profile fields: username, bio, location, languages, and favorite destinations Implemented edit/save toggle with pencil and save icons in header Synced profile data with Firebase Realtime Database and AsyncStorage Fixed missing pencil icon by confirming header button setup in navigation Resolved dependency issue with objectWithoutProperties by cleaning and reinstalling node modules Verified compatibility of React Navigation and MaskedView packages Profile screen now switches between view and edit mode without error

3/26/25 4:00 PM	1.5	Added editable fields for username bio location languages favorite destinations dream destination travel app instagram Added editable dropdowns for packing style travel companion budget range planning habit trip role Connected all profile fields to Firebase Realtime Database with real-time updates and fallback from AsyncStorage Enabled S3 profile image upload with presigned URL from backend and automatic Firebase update Cached profile image locally for fallback display Displayed uploaded image with cache-busting timestamp Added loading spinner while uploading profile image Implemented “Edit Profile” mode toggle with "Save Profile" and FontAwesome "X" button Used useFocusEffect to sync data from Firebase and local storage Styled header to match navy blue color all the way to the notch Removed header shadow line using headerStyle with elevation and shadow properties Retained custom header buttons using useLayoutEffect when editing profile
3/26/25 10:00 PM	2	**Fix Destination Parsing for Itinerary Edit** Updated fetchItineraryDetails to parse destination string from PostgreSQL into city and country object Removed unnecessary setLocation call Confirmed destination modal sets destination object correctly Ensured destination is stored as a string when submitting the form Cleaned up state usage to avoid undefined display bugs **Update Login and Signup Screens with Background Image** Added ImageBackground to Login screen Styled Login screen with background image and overlay card Added SafeAreaView for proper layout on iOS Centered login form vertically when active Prepared styling for a smooth visual transition

3/28/25 9:00 PM	1	Updated /recent backend route to return full itinerary data including extra_data.image_url Fetched 3 most recent itineraries after login and saved them to AsyncStorage Loaded recent itineraries from AsyncStorage on HomeScreen mount Replaced horizontal ScrollView with FlatList for smoother trip card scrolling Matched FlatList styling to trip card design with snap behavior Fixed vertical gap between "Trip Plans" title and trip cards by adjusting styles
3/28/25 10:00 PM	0.5	Refactored ProfileScreen layout and logic for clarity and modularity Implemented Firebase data fetching using onValue and fallback with AsyncStorage Added calculateCompletion to compute profile completeness Resolved bug where calculateCompletion was undefined Displayed profile completion percentage with a styled progress bar Improved UX by turning progress bar green when completion reaches 100%
3/28/25 10:30 PM	0.5	Created AddToItineraryModal.js for selecting an itinerary. Fetched and combined personal and shared itineraries with type field. Sorted itineraries by updated_at. Added icon to distinguish between personal and shared itineraries. Moved icon from left to right of each itinerary item. Integrated modal into InteractiveRecommendations.js. Passed selected place and added it to Firebase under places. Prevented duplicate place entries and showed alerts for success or duplicate.
3/28/25 11:00 PM	2	Refactored AchievementsScreen to horizontally center achievement grid Adjusted columnWrapperStyle and gridItem width/margin for proper alignment Improved modal layout and progress bar width responsiveness Fixed AchievementsScreen container not being horizontally centered Resolved flicker of LocationPermissions by adding locationPermissionChecked Deferred rendering of HomeScreen until location check completes Ensured smoother transition from LocationPermissions to HomeScreen Moved itinerary fetching logic into useFocusEffect for consistency Re-enabled and integrated deleted useEffect logic for recent itinerary cards Fetched itinerary list from FastAPI using authenticated user ID Loaded recent itineraries from AsyncStorage fallback Updated recentTrips state to ensure trip cards render in Trip Plans section Cleaned up setUserId, onboardingComplete, and travel_style_id logic Improved overall HomeScreen onboarding and weather experience

3/29/25 12:00 PM	1.25	Adding Toggle mode for the Onboarding Checklist
3/29/25 1:15 PM	0.25	Added updateRecentTripsInStorage function to ItineraryDetailScreen Called updateRecentTripsInStorage after successful image upload in uploadImage Synced logic with ItineraryFormScreen to keep AsyncStorage updated with recent trips
3/29/25 1:30 PM	0.1	Added GestureHandlerRootView to wrap the entire app in App.js Identified crash in InteractiveRecommendations caused by gesture conflict with Modal Confirmed that AddToItineraryModal used FlatList and TouchableOpacity inside Modal Verified MapCheckInScreen worked fine because it did not use Modal Explained that wrapping GestureHandlerRootView globally prevents gesture-related crashes Confirmed nested GestureHandlerRootView is safe and does not cause issues
3/29/25 1:40 PM	2	Removed FlatList inside ScrollView to eliminate VirtualizedList warning Replaced FlatList with manually rendered list using .map() and BottomSheetScrollView Used cardRefs and scrollViewRef for smooth manual snapping to cards Refactored onPress logic so tapping a marker scrolls to and highlights the correct card Refactored onPress logic so tapping a card animates to the marker and shows callout Introduced activeIndex state to highlight active card with custom style Integrated @gorhom/bottom-sheet for swipeable modern layout with snapping positions Made travel style filter bar transparent to create a floating button appearance Preserved category filter inside bottom sheet just above the card list Preserved modal functionality to add a place to itinerary Ensured all components work on both local and Heroku environments Code is now cleaner, more performant, and designed for smoother mobile interaction
3/29/25 3:40 PM	0.1	Add Firebase profile image fetch in LoginScreen Save profile image to AsyncStorage after login Reuse saved image in MoreMenu for display Ensure profile image updates on ProfileScreen also update AsyncStorage Keep image synced and consistent between login, profile, and menu

3/29/25 3:50 PM	0.25	Passed itineraryId as prop to OtherCostsModal Fixed Firebase saving and deletion for other costs Improved modal styling with pill-shaped buttons and padding Styled cost type and subtype chips with horizontal scroll Enhanced text inputs with consistent padding and font size Updated saved cost list with cleaner layout and spacing Replaced "Remove" text with FontAwesome trash icon for deletion
3/29/25 4:45 PM	0.25	Added checklistRefreshTrigger state in HomeScreen Updated Firebase writes for weather_changed and packing_tip_viewed to increment the trigger Passed refreshTrigger prop to OnboardingChecklist Updated OnboardingChecklist to refetch checklist when refreshTrigger change
3/29/25 5:00 PM	0.25	Updated ItineraryFormScreen to use KeyboardAvoidingView, ScrollView, and TouchableWithoutFeedback for proper keyboard behavior Wrapped entire screen with keyboard-safe layout to prevent input fields from being blocked Refactored DestinationSearchModal to expand fully with flex: 1 Removed gray overlay issue by adjusting modalBox and backdrop styles Verified modal behaves correctly when keyboard is active and now appears clean without dimming artifact
3/29/25 5:15 PM	0.25	Fixed issue where plus button in callout was not clickable Changed Callout to use tooltip={true} Wrapped plus icon inside CalloutSubview Ensured styling keeps plus button on the right side Retained custom design with name, category, and plus icon layout
3/29/25 5:30 PM	0.5	Set selectedDate to itinerary's start date when adding a new day Used getNextAvailableDate() to ensure consistent default value Improved flow so users don't have to manually pick a starting date Reduced friction when creating the first itinerary day
3/30/25 1:30 PM	0.25	Wrapped place name text in a flex container Added placeTextWrapper style for layout control Enabled text wrapping with flexWrap: 'wrap' Added optional numberOfLines and ellipsizeMode for truncation Ensured buttons stay aligned with long text

3/30/25 1:45 PM	0.25	AddActivityModal extracted into its own file under components Refactored ItineraryDayScreen to use AddActivityModal Made display time default to current time when opening modal Preserved time picker selection and form behavior across both create and edit modes
3/30/25 2:00 PM	2	Added new feature to let users tap a place and assign it directly to a day Closed PlacesModal when a place is tapped If there are days, opened DaySelectionModal after short delay Once a day is selected, prefilled activity data and opened AddActivityModal Initialized time with current time and set place name as default activity If no days exist, switched to Days tab and opened Add Day modal automatically Pre-filled new day with next available date and cleared title Updated tab index using setIndex to show the Days screen Handled state transitions to avoid modal overlap Improved UX by streamlining the flow from selecting a place to planning it in a day
3/30/25 4:00 PM	0.25	Fixed crash in ItineraryDayScreen by removing strict UUID length check Ensured dayId presence check is simple and non-blocking for API call
3/30/25 4:15 PM	0.25	Retrieved profile image URL from AsyncStorage using profileImageUri key Stored S3 HTTPS image URL during profile setup Updated ChatbotScreen to load user avatar from S3 if available Added fallback to default image if no URL or invalid Added optional image prefetching for debug validation Ensured profile image renders only if it starts with http
3/30/25 4:30 PM	0.25	Updated card position to bottom of the screen with keyboard support Set card width to full screen Added tagline "Your Journey, Your Way." above the card Added main title "EXPLORE" and "THE WORLD" in uppercase above tagline Styled login button with full width and pill shape Matched signup button style to login button style for consistency

3/30/25 4:45 PM	0.25	Added AuthLoadingScreen to check AsyncStorage for user session Implemented logo zoom and fade animation using Animated API Adjusted logo scale for calm and smooth animation with easing Matched initial logo size to avoid starting too small Introduced 0.5s delay before animation for breathing room Updated transition to Main or Login screen after animation Investigated iOS white screen and found solution but did not apply Swift update
3/30/25 5:00 PM	0.25	Fixed crash caused by accessing user.id when user was null Added null check for user before making DELETE request Improved error message to prompt re-login if user not found Ensured day deletion only proceeds when user is properly loaded
3/30/25 5:15 PM	0.25	Fixed overlapping issue by adjusting top margin of filter bar Used useSafeAreaInsets to dynamically set spacing from top Replaced SafeAreaView with View and applied inline safe area style Ensured filter buttons stay clear of status bar and clock Added optional horizontal scroll for better layout on smaller screens
3/30/25 5:30 PM	0.25	Added isSheetExpanded state using useState Set isSheetExpanded to true when BottomSheet index is 2 using onChange Wrapped Travel Style Filter with conditional !isSheetExpanded Updated BottomSheet snapPoints to include 85 percent max height Preserved map interaction and filter logic during transition
3/30/25 9:30 PM	0.25	Wrapped DestinationSearchModal with React Native Modal Updated backdrop style to include semi-transparent background Set modalBox height to full screen using flex: 1 Ensured keyboard behavior using KeyboardAvoidingView Matched layout and behavior with WeatherSearchModal
3/30/25 9:45 PM	0.25	Wrapped modal in KeyboardAvoidingView for better keyboard handling Added ScrollView to allow scrolling when keyboard is open Implemented TouchableWithoutFeedback to dismiss keyboard on background tap Adjusted styles to support new layout behavior
3/30/25 10:00 PM	0.25	Updated cost type and subtype selectors with section labels Wrapped modal content with KeyboardAvoidingView for better keyboard handling Added ScrollView to allow scrolling when keyboard is open Improved overall user experience for input and selection in modal

3/30/25 10:15 PM	0.5	Moved parseToSortableTime function to the top of the file Updated DayCard to sort item.activities by time using parseToSortableTime Ensured consistent activity order matching ItineraryDayScreen Preserved original UI layout and logic in DayCard
3/30/25 10:45 PM	0.25	Fixed crash by adding optional chaining to itinerary date. Added budget, notes, and places panels to OverviewTab. Created and moved styles to ItineraryDetailScreenStyle.js. Updated OverviewTab to support external panels and style structure. Verified that screen renders correctly with tab view and modals.
3/30/25 11:00 PM	0.25	Modified addPlace function to immediately update Firebase after adding a place Removed the need for pressing the Save button Updated UI logic to optionally remove the Save button and keep only the Close button
3/31/25 12:00 AM	0.5	Added new city toggle with options for Vancouver and Bali Updated state and logic to change map region based on selected city Refactored layout to display both toggles side by side and horizontally centered Replaced ⌄ with FontAwesome chevron-down icon for better alignment Applied flexbox styling to align text and icon inside toggle buttons
3/31/25 12:30 AM	0.5	Updated <MapView /> to hide points of interest using showsPointsOfInterest={false} Kept street names visible on Apple Maps for iOS Ensured change only affects map visuals without altering map functionality
4/2/25 9:30 AM	0.25	Added currentIndex state to track active carousel item Adjusted card width and snapToInterval to subtract horizontal padding Used useWindowDimensions for responsive width Ensured card corners are visible inside HomeScreen scroll padding Updated onScroll logic to match adjusted card width Centered pagination dots correctly within padded layout
4/2/25 9:45 AM	0.25	Wrapped ActivityIndicator inside a centered View Added loadingContainer style with flex: 1, justifyContent: 'center', and alignItems: 'center' Ensured loading spinner appears centered both horizontally and vertically during data fetch

4/2/25 10:00 AM	0.5	Added flexGrow: 1 to ScrollView container to prevent background bleed when keyboard opens Ensured KeyboardAvoidingView has flex: 1 and correct keyboardVerticalOffset Verified outer wrapper (SafeAreaWrapper) has consistent background color Confirmed modal and screen background is set to #f8f9fa to prevent transparency issues during keyboard display Cleaned up container style to maintain consistent padding and alignment
4/7/25 5:00 PM	1	Plan for User Testing Priority
4/7/25 6:00 PM	0.5	Moved "Don't have an account? Sign Up" text to the top of the login card Added smooth card animation when keyboard appears and hides using Animated API Replaced static translateY value with dynamic keyboard height Used keyboardWillShow and keyboardWillHide for better timing on iOS Wrapped main layout with KeyboardAvoidingView to prevent keyboard overlap Adjusted card padding and layout to improve spacing Preserved animated sliding effect while ensuring inputs are fully visible above the keyboard Cleaned up styles and improved visual alignment for inputs and buttons
4/7/25 6:30 PM	0.5	Improved keyboard interaction using Animated.timing with smooth transition Prevented card from being pushed too far when keyboard appears Cleaned up flicker issue when switching from email to password input Ensured login button is not overlapped by autofill or keyboard bar Maintained smooth animation using useNativeDriver and easing function Validated input transitions without janky scrolling or layout shifts
4/7/25 7:00 PM	0.25	Added smooth keyboard animation handling to prevent layout jump Made card stay pinned to bottom like the login screen Prevented extra movement when switching between input fields Added eye icon to toggle password visibility Added clear (X) icon to Full Name and Email input fields Matched input layout and interaction style to the Login screen Updated App.js to register SignupScreen and ensure navigation works properly
4/7/25 9:30 PM	0.25	Added useRef for email and password fields Linked Full Name → Email → Password with onSubmitEditing Set returnKeyType on all fields for proper keyboard navigation

4/7/25 9:45 PM	0.5	Updates from user feedbacks: Visibility Quiz not prominent or prioritized Layout Answer choices hard to scan, fonts small Result Context Quiz result lacks explanation or follow-up Navigation No guidance post-quiz What's been done: Added “Back to Home” button on result screen Removed auto-navigation after timeout Displayed alert message on manual back navigation Removed “X” close button from result screen Kept confetti and result message display behavior Preserved quiz retake button for flexibility
4/7/25 10:15 PM	0.5	Redesigned packing suggestion modal with friendly layout using icons and checklist style Replaced basic alert-like modal with a rounded, styled card format Added onboarding tooltip bubble below weather widget for first-time users Created weather_tooltip_dismissed flag in Firebase to track tooltip visibility Made tooltip manually dismissible with a "Got it" button Wrapped weather widget and tooltip in a new container weatherSectionWrapper Reused weatherWidgetContainer styling to maintain consistent UI Styled tooltip bubble using weatherTooltipContainer with added tail Adjusted weatherTooltipTail to align with right-side suitcase icon Ensured tooltip logic runs based on Firebase and AsyncStorage on mount Integrated logic into existing HomeScreen component with clean conditional rendering
4/7/25 10:45 PM	0.75	Updated checklist to use sharedExpanded state with correct height control Ensured collapse animation works even before measuring height Removed unnecessary isExpanded and simplified toggle logic Refactored onLayout to set height only once and avoid double triggers Restored animatedBadgeStyle for reward modal animation Ensured smooth expand/collapse behavior without remeasurement glitch
4/9/25 9:45 AM	0.5	Added transparent black overlay with blur using BlurView Changed modal text color to white for better contrast Adjusted spacing between Lottie animations and text Added wrapper with overflow hidden to crop excess Lottie space Improved overall layout for cleaner and tighter visual spacing

4/9/25 10:15 AM	1	Added transparent black overlay with blur using BlurView Changed modal text color to white for better contrast Adjusted spacing between Lottie animations and text Added wrapper with overflow hidden to crop excess Lottie space Improved overall layout for cleaner and tighter visual spacing
4/9/25 1:45 PM	0.5	Changed tap behavior so entire row is tappable to add a place Removed lightning icon and added instructional tip above the list Limited modal height to 60% of screen using Dimensions API Made FlatList scrollable within fixed modal height Added numbering to each place item in the list for better clarity
4/9/25 2:15 PM	0.5	Changed modal animation type to fade for smoother close experience Wrapped modal in TouchableWithoutFeedback to dismiss keyboard on outside tap Adjusted modal vertical position to start 10 percent below the top Used Dimensions API for responsive modal height Ensured FlatList and layout remain scrollable inside modal container
4/9/25 2:45 PM	0.25	Dismissed keyboard when tapping date selection field Wrapped entire screen in TouchableWithoutFeedback to catch taps Used Keyboard.dismiss inside onPress of calendar opener Improved input-to-field navigation across the form
4/9/25 3:00 PM	0.25	Add TouchableWithoutFeedback to dismiss keyboard on outside tap inside modal Apply paddingTop using styles.modalContainer for better modal position Ensure date field tap also dismisses keyboard before opening calendar Cleaned up modal view logic and styling for consistency Used shared styles file for layout and spacing
4/9/25 3:20 PM	1	Added info icon with tooltip modal for "Planned Budget" Made the entire "Planned Budget" label row tappable Added info icon with tooltip modal for "Activities Cost" Made the entire "Activities Cost" label row tappable Tooltip modal explains what each budget item means Used state to control modal text and visibility Styled modal with backdrop, button, and centered text Switched budget section layout from horizontal scroll (carousel) to stacked vertical cards with side-by-side layout for sub-items Ensured consistent spacing and visual hierarchy in budget breakdown layout

4/9/25 3:20 PM	1	Changed selectedSubtype to selectedSubtypes as an array Allowed users to select multiple subtypes with toggleable pill buttons Updated save handler to store subtypes as an array Displayed saved subtypes with comma separation Fixed text overflow by wrapping type-subtype text with width constraint Reset item name and amount inputs after saving Cleaned up resetFields() to match updated state logic
4/9/25 4:20 PM	0.25	Used Animated API to fade in and out a toast message Positioned the message beside the "Saved Other Costs" heading Styled the message with soft green badge style Auto-dismissed the message after 1.5 seconds with smooth animation Improved user experience with visual confirmation without alerts
4/9/25 4:35 PM	0.25	Replaced default TabView tab bar with custom tab header layout Added red badge to show pending invite count on Shared Itineraries tab Used index state to toggle between Personal and Shared tab views Removed label customization and TabBar from TabView Kept SceneMap and swiping disabled to simplify layout Improved tab visibility and UX clarity using custom active tab styling
4/9/25 5:00 PM	1	Introduced two entry modes: Helper Mode and Manual Mode Helper Mode uses predefined cost types and subtypes with structured input Manual Mode allows free-form custom category entry without subtypes Added Firebase save logic to handle both modes with different structures Adjusted text wrapping in saved cost preview by limiting width to 70% Removed unnecessary hyphen when subtypes are not present Added info circle button with handleInfoPress() to explain mode differences Styled info button to match height of Helper and Manual buttons Centered the entire mode toggle row horizontally for better visual balance Replaced incorrect Icon import with proper FontAwesome to fix rendering issues
4/9/25 6:15 PM	0.25	Made the "Collaborators" text and pencil icon act as a single clickable element Wrapped both elements in one TouchableOpacity Navigated to InviteCollaborators screen on press Kept original styles and layout intact

4/9/25 10:45 PM	0.25	Moved chat history icon to the right side of the header Planned and selected a clear approach to show chatbot's location limitation Decided to add a banner at the top of the chat screen with the message: "Prototype: Currently limited to Vancouver, BC"
4/9/25 11:00 PM	0.25	Add friendly progress messages under each badge using encouraging phrases Updated getProgressText function to show motivational text instead of ratios Displayed the progress text below each progress bar on the grid items Centered the progress text using textAlign and alignSelf in gridProgressText style Updated modal to use the same motivational progress text without duplication
4/9/25 11:15 PM	0.25	Added toast alert for newly unlocked badge when badge tier increases Prevented duplicate alert if badge alert is already shown Defined capitalize function at top level to avoid undefined error Cleaned up confirmCheckIn flow for better UX and clarity
4/9/25 11:30 PM	0.25	added capitalize function back for title formatting added trophyImages map for category and tier-based badge images created getBadgeImage(category, tier) helper to dynamically select image updated unlock modal to use correct badge image based on both category and tier replaced static badge image in modal with dynamic image kept existing animation styling for badge image verified fallback behavior using optional chaining and null handling
4/11/25 7:30 PM	0.15	Replaced "X" text in modal with FontAwesome close icon Updated JSX to use TouchableOpacity with icon Adjusted modalCloseButton style for icon layout
4/11/25 7:45 PM	0.5	Added horizontal scroll for badge section in public profile Filtered out unearned badges (only show when count >= 5 or isSpecial) Replaced long press with tap interaction on badges Displayed alert with badge description on press Refactored badge section to be simpler and cleaner for read-only viewing

4/11/25 8:15 PM	1	Add new FriendsAchievements.js component with horizontal badge scroll Fetch each friend’s badge data and profile photo from Firebase Show friend name on long press of profile photo Restrict height of friends' achievements list with vertical scroll Remove duplicate "Friends' Achievements" header from component Ensure friend name uses friend.friendName field correctly from Firebase
4/11/25 9:15 PM	0.5	Updated ProfileScreen to use full image in modal instead of cropped circle Cleaned up previous wiggle animation logic from MoreMenu Ensured MoreMenu displays static profile card with optional photo Handled Firebase image upload and removal with AsyncStorage sync Ensured photo modal on ProfileScreen uses resizeMode: contain for full image display Reused consistent styling across cards and modals Simplified interaction logic across both screens for clarity and reliability
4/11/25 9:45 PM	0.15	Updated FAB buttons to stack vertically instead of horizontal row Adjusted animated styles to translate along Y-axis for vertical layout Aligned Save and Cancel buttons below the main Edit button visually Toggled isEditing state properly when pencil icon is tapped again Fixed button overlap and spacing in the FAB layout Added Alert.alert to notify user when profile is successfully saved Ensured editing is disabled after save or cancel from FAB buttons Updated saveProfile() to include success alert after saving to Firebase and AsyncStorage
4/11/25 10:00 PM	0.5	Added Firebase listener to load user's friend count Displayed number of friends in the profile header Replaced "Edit Profile" button with clickable friend count Navigated to AddFriends screen on friend count press Ensured friend count updates on data change via onValue listener
4/11/25 10:30 PM	0.5	Added Firebase fetch to load saved places by itinerary on mount Created helper function to check if a place is already saved Applied green pin color to markers for saved places Refreshed marker colors after adding a new place Displayed checkmark icon on cards for already saved places

4/11/25 11:00 PM	0.15	Created HomeActionTiles.js component with 3 main shortcuts: Itineraries, Check-In, Friends Used icon inside rounded rectangle layout inspired by Apple Shortcuts Added the component below FeatureCarousel in HomeScreen.js Handled navigation for each action using onNavigate callback Ensured consistent left-right alignment with the rest of the screen Updated tile layout to use flex: 1 and justifyContent: space-between instead of fixed padding
4/11/25 11:15 PM	0.5	Make collaborators clickable to view public profile Added navigation to PublicProfileScreen from each collaborator chip Wrapped each collaborator display in TouchableOpacity Passed friendId to the PublicProfileScreen via navigation Ensured PublicProfileScreen is registered in the app navigator
4/11/25 11:45 PM	0.5	Added leaderboard UI for FriendsAchievements with top 3 highlighting Ensured current user is included and shown as "You" Fixed issue where user's profile photo was not displaying Fetched full user profile from Firebase to extract profilePhotoUrl Cleaned up unused styles in FriendsAchievements (friendRow, badgesScroll, badgeIcon) Ensured proper sorting and row highlighting for gold, silver, bronze, and current user
4/11/25 11:45 PM	0.5	Added pull-to-refresh using RefreshControl on main ScrollView Created refreshHomeScreen function to reload itinerary and weather data Updated itinerary images and weather data on refresh Added showToast state to control toast visibility after refresh Displayed a toast message "☑ Refreshed" at bottom of screen Styled toast using toast and toastText styles in HomeScreenStyles.js Toast auto-hides after 2 seconds with setTimeout
4/12/25 11:00 AM	0.25	Adjust the position of Fab button upon opening keyboard modal on the edit Profile Screen.

4/12/25 11:15 AM	0.25	Added onWeatherNudge prop to trigger weather widget animation from checklist Created nudge animation using withTiming and useSharedValue Wrapped weather widget in Animated.View with nudge style Passed onWeatherNudge from HomeScreen to OnboardingChecklist Triggered onWeatherNudge from "Check Weather Widget" checklist item Increased top spacing before first checklist item in onboarding checklist
4/13/25 1:00 AM	. 5	Updated user ID retrieval from AsyncStorage to match login storage Corrected owner key from userId to owner in Firebase structure Verified collaborator inclusion using object key check Filtered saved places by both owner and collaborators Fixed marker color to green when place is already saved
4/13/25 12:30 AM	0.25	What we did in ItineraryFormScreen: Added Firebase write logic to save new itinerary under /live_itineraries Stored owner field in Firebase instead of collaborators Removed creator from being saved as a collaborator What we did in ItineraryDetailScreen: Added Firebase delete logic when itinerary is deleted Ensured removal of /live_itineraries/{itineraryId} on deletion Kept UI and database states in sync after delete action
4/12/25 1:00 PM	2	Create presentation slides. Brainstorm presentation flow.
4/12/25 9:30 PM	3	Create presentation flow and scripts. Test reading to match the maximum duration allowed in the presentation.

Concluding Remarks

In conclusion, the development journey for Waypoint proved to be both challenging and immensely rewarding. Overcoming technical hurdles—such as integrating multiple third-party APIs, addressing data synchronization issues between PostgreSQL and Firebase, and refining

caching strategies—pushed us to continuously innovate and adapt our processes. Balancing these complexities with the need for a seamless, visually engaging user interface underscored the importance of iterative testing and agile project management, which ultimately led to meaningful enhancements in both functionality and user experience. Our commitment to a feature-first MVP approach, coupled with the invaluable insights garnered from real user feedback, has culminated in a travel planning solution that not only streamlines itinerary creation but also delivers personalized, AI-driven recommendations. This project stands as a testament to our team's resilience and collaborative spirit, setting a strong foundation for future developments that will continue to elevate the travel experience for visitors to British Columbia.

References:

- Google Maps SDK for iOS: https://developers.google.com/maps/documentation/ios-sdk/overview
- Google Places API: https://developers.google.com/maps/documentation/places/web-service/overview
- Firebase Realtime Database: https://firebase.google.com/docs/database
- Firebase Installation: https://firebase.google.com/docs/ios/setup
- React Native Maps for iOS: https://github.com/react-native-maps/react-native-maps
- React Native CLI: https://reactnative.dev/docs/environment-setup
- FastAPI: https://fastapi.tiangolo.com/
- Xcode: https://developer.apple.com/xcode/
- Apple Developer Program: https://developer.apple.com/programs/
- SQLAlchemy: https://docs.sqlalchemy.org/

- Pydantic: https://docs.pydantic.dev/latest/
- email-validator: https://pypi.org/project/email-validator/
- Openai syntax usage: https://pypi.org/project/openai/
- Openai documentation: https://platform.openai.com/docs/guides/text-generation
- iOS Permissions guide: https://rossbulat.medium.com/react-native-managing-app-permissions-for-ios-4204e2286598
- iOS Permissions guide: https://www.npmjs.com/package/react-native-permissions#ios
- Geolocation permissions: https://blog.logrocket.com/react-native-geolocation-complete-tutorial/

Tools Used

- ChatGPT: https://chatgpt.com/
    - Data analysis – Analysis of user testing
    - Debugging – Coding solutions, alternative fixes
    - Coding Logic – Introduce ways to approach a problem
    - Reports – Summarization for report writing
- Canva: https://www.canva.com/ : Image background removal
- Gemini: https://gemini.google.com/app : Badges generation
- Google Slides

Appendix A: Installation Guide

Part 1: Software Needed

Install the following:

1. VSCode https://code.visualstudio.com/

2. Node.js and npm https://nodejs.org/

3. React Native CLI https://reactnative.dev/docs/environment-setup

4. Xcode https://apps.apple.com/ca/app/xcode/id497799835

5. CocoaPods https://guides.cocoapods.org/using/getting-started.html

6. PostgreSQL (Postgres.app or Homebrew) https://postgresapp.com/ or

https://formulae.brew.sh/formula/postgresql

7. pgAdmin (GUI for PostgreSQL) – https://www.pgadmin.org/download/

8. Ruby https://www.ruby-lang.org/en/downloads/

Part 2: Backend Setup

Step 1: Clone the GitHub repository and open the backend folder.

Repository URL:

https://github.com/russellhanj/W25_4495_S1_SimoneL

Step 2: Open the `Implementation/backend` folder in VSCode.

Step 3: Create a Python virtual environment.

In your terminal (inside the backend folder), run the following:

- For macOS:

```
python3 -m venv venv
```

Then activate it using:

```
source venv/bin/activate
```

Step 4: Install backend dependencies.

With the virtual environment activated, run:

```
pip install -r requirements.txt
```

Step 5: Create a new PostgreSQL database using pgAdmin.

- Open pgAdmin
- Connect to your PostgreSQL server
- Right-click on "Databases" → Select Create → Database
- Name your database (e.g., `waypoint_db`)
- Keep the default owner as your local username (e.g., `russellhanjosef`)
- Click Save

Step 6: Create a `.env` file in the root of the backend folder.

Include the following variables (replace values with your own):

```
# PostgreSQL Database
DATABASE_URL=postgresql://<your_db_username>@localhost:5432/<your_db_name>

# API Keys
GOOGLE_PLACES_API_KEY=<your_google_places_api_key>
GOOGLE_MAPS_API_KEY=<your_google_maps_api_key>
EVENTBRITE_API_KEY=<your_eventbrite_api_key>
OPENWEATHERMAP_API_KEY=<your_openweathermap_api_key>
```

```
OPENAI_API_KEY=<your_openai_api_key>


# Environment Settings
DEBUG=True  # Change to False in production


# AWS S3 (Image Upload)
AWS_ACCESS_KEY_ID=<your_aws_access_key>
AWS_SECRET_ACCESS_KEY=<your_aws_secret_key>
AWS_BUCKET_NAME=<your_s3_bucket_name>
AWS_REGION=<your_s3_region>
```

Step 7: Start the PostgreSQL server on your local machine.

- If you installed Postgres via Postgres.app, open the app and make sure the server is running.
- If you used Homebrew, run:

  `brew services start postgresql`
- You can confirm it's running by connecting through a tool like `psql` or your DB client.

Step 8: Once PostgreSQL is running, start the FastAPI backend.

Use the command:

```
uvicorn main:app --reload
```

Backend GitHub Reference:

https://github.com/russellhanj/W25_4495_S1_SimoneL/tree/main/Implementation/backend

Part 3: Frontend Setup

Step 1: Open the `Implementation/frontend` folder using VSCode.

Step 2**:** Install the project's JavaScript dependencies.

In the terminal, type `npm install` and press Enter.

Step 3: Install Bundler if it's not already installed.

In the terminal, type `gem install bundler` and press Enter.

Step 4: Inside the `frontend` folder, install the Ruby dependencies using Bundler by typing `bundle install`.

Step 5: Navigate to the `ios` folder and install iOS-specific dependencies using CocoaPods.

Type `cd ios`, then type `bundle exec pod install`, and finally return to the root folder by typing `cd ..`

Step 6: Create a `.env` file inside the root of the `frontend` folder.

Paste the template below and replace the values with your actual API keys and Firebase configuration:

```
GOOGLE_PLACES_API_KEY=<your_google_places_api_key>
GOOGLE_MAPS_API_KEY=<your_google_maps_api_key>

OPENAI_API_KEY=<your_openai_api_key>

FIREBASE_API_KEY=<your_firebase_api_key>
FIREBASE_PROJECT_ID=<your_firebase_project_id>
FIREBASE_DATABASE_URL=<your_firebase_database_url>
FIREBASE_STORAGE_BUCKET=<your_firebase_storage_bucket>
```

```
FIREBASE_MESSAGING_SENDER_ID=<your_firebase_messaging_sender_id>

FIREBASE_APP_ID=<your_firebase_app_id>

FIREBASE_IOS_BUNDLE_ID=<your_ios_bundle_id>
```

Step 7: Run the app on the iOS simulator by typing `npx react-native run-ios`.

Make sure the backend and PostgreSQL server are already running before launching the app.

Frontend GitHub Reference:

https://github.com/russellhanj/W25_4495_S1_SimoneL/tree/main/Implementation/frontend

Appendix B: User Guide

WayPoint User Guide

Step 1: Launch the App

Once both the backend and frontend are running, the app will open to the Login screen.

Login Screen

- Login using your email and password.
- If you don't have an account yet, tap "Don't have an account? Sign Up".

Step 2: Create an Account (if new)

Signup Screen

- Fill in your name, email, and password (minimum 6 characters).
- The app will check if the email already exists.
- A default travel style will be assigned.
- Tap Sign Up to register.
- Once successful, you'll be redirected back to the Login screen to sign in.

Step 3: Home Screen Overview

After logging in, you'll land on the Home screen — the central hub of the app.

Key Features:

- Weather Widget

  Shows the current weather for your location.

  Tap the suitcase icon to get AI-powered packing suggestions.
- Start Your Journey (Quiz Prompt)

  If you haven’t taken the Travel Style Quiz, you’ll see a banner.

  Tap it to discover your travel preferences.
- Onboarding Checklist

  Helps guide new users through setting up and exploring key features.
- Feature Highlights Carousel

  Swipable section introducing helpful app features.
- Action Tiles

  Quick access to:

  - Trip Planning
  - Check-in (Log visit)
  - Friends
- Trip Plans Section

  Displays your recent personal and shared itineraries.

  Tap a trip to open the full itinerary.

Step 4: Bottom Navigation Tabs

- Home – Main dashboard with weather, checklist, trip plans, etc.
- Map – Discover personalized recommendations nearby (Interactive Map).
- Itinerary – Manage your trip plans and schedules.
- More – Tap to open additional features.

Step 5: More Menu Options

Tap the More icon in the bottom navigation to access:

- Profile – View your info and profile image.
- Settings – App configuration (future updates).
- Chatbot – Ask for trip suggestions, FAQs, or packing help.
- Log Visit – Check-in at a location (for gamification).
- Achievements – View badges and milestones earned.
- Friends – Add or manage travel friends.
- Log Out – Securely sign out of the app.