Australia Trip Scheduler - Technical Documentation v3.0

Last Updated: December 12, 2025
Version: 3.0 - Complete Cloud Database Integration
Repository: https://github.com/petehep/interactive-Aus-map
Live URL: https://petehep.github.io/interactive-Aus-map/

Table of Contents

Project Overview

Purpose: Interactive trip planning application for Australia with real-time cloud synchronization across devices.

Key Features: - Interactive map with 100,000+ Australian locations - Route planning with real-time driving distances - Cloud-synced favorites across all devices - Persistent visited places tracking - Campsite discovery (free & paid) - Essential services (fuel, dumps, water) - Multi-user authentication - Real-time database synchronization - Fully responsive design

---

What's New in v3.0

Cloud Database Integration

Previous: Data stored only in browser localStorage (single device)
Now: All data synced to Firebase Firestore (cloud-based, multi-device)

Key Improvements

1. **Real-time Synchronization**

   - Changes sync instantly across all devices
   - No manual refresh needed
   - Built-in conflict resolution
2. **Multi-Device Support**
   - Log in from phone, tablet, computer
   - Same account = same data everywhere
   - Automatic background syncing
3. **Data Persistence**
   - Backed up in the cloud
   - Never lose trip data
   - 99.99% uptime SLA
4. **Automatic Migration**
   - Existing localStorage data automatically moves to cloud on first login
   - Zero data loss
   - One-time process per user
5. **Enhanced Security**
   - Firestore security rules enforce user isolation
   - Users can only access their own data
   - Encrypted in transit and at rest

Technology Stack

Frontend

- **React 18.3.1** - UI framework with hooks
- **TypeScript 5.6.3** - Type-safe development
- **Vite 5.4.8** - Fast build tooling

Mapping & Geospatial

- **Leaflet 1.9.4** - Interactive maps
- **React-Leaflet 4.2.1** - React wrapper
- **React-Leaflet-Cluster 2.1.0** - Marker clustering
- **Overpass API** - OSM data queries
- **OSRM** - Open routing machine
- **Nominatim** - Geocoding service

Backend & Database

- **Firebase 10.14.0**
  - Authentication (Email/Password)
  - Firestore (Cloud database)
  - Real-time listeners
  - Security rules

External APIs

- **MeteoBlue** - Weather forecasts
- **Google Maps** - Location viewing
- **OpenStreetMap** - Map tiles and POI data

Hosting

- **GitHub Pages** - Static hosting
- **GitHub Actions** - CI/CD pipeline

Architecture

System Diagram

```
                React Application

            UI Layer (Components)
- MapView | Favorites | Itinerary | Login | etc

                      ↑↓

         State Management (React Hooks)
- useState, useEffect, useCallback, useReducer

                      ↑↓

       Service Layer (firestoreService.ts)
- saveFavorite() | saveVisitedPlace() | etc
- Real-time subscriptions (onSnapshot)
- Automatic ID sanitization



               ↓              ↓              ↓

         Firebase       Overpass        OSRM
          (Auth +         API          (Routing)
         Firestore)     (Places)
```

Data Flow

1. **User Interaction** → Component updates state

2. **Service Layer Call** → firestoreService function invoked
3. **Firestore Operation** → Write/read to cloud database
4. **Real-time Listener** → Subscription updates state
5. **React Re-render** → UI reflects latest data

Cloud Database (Firestore)

Database Structure

```
Firestore Root
  users/ (Collection)
      {userId}/ (Document)
          favorites/ (Subcollection)
             {sanitizedPlaceId}/ (Document)
                 id: "node/8443821294"
                 name: "Sydney"
                 type: "city"
                 lat: -33.8688
                 lon: 151.2093
                 visited: true
                 visitedAt: 1702400000000
                 updatedAt: 1702400000000

          visited/ (Subcollection)
             {sanitizedPlaceId}/ (Document)
                 id: "node/8443821294"
                 name: "Sydney"
                 visitedAt: 1702300000000
                 updatedAt: 1702400000000

          itineraries/ (Subcollection)
              current/ (Document)
                  items: [
                   {
                     id: "node/123456",
                     name: "Sydney",
                     lat: -33.8688,
                     lon: 151.2093,
                     addedAt: 1702200000000
                   }
                 ]
                  updatedAt: 1702400000000
```

ID Sanitization

Problem: Firestore document IDs cannot contain slashes (/), but OSM IDs do (e.g., node/8443821294)

Solution: Replace slashes with underscores in document IDs

```
function sanitizeId(id: string): string {
  return id.replace(/\//g, '_')
}
// "node/8443821294" → "node_8443821294"
```

Important: The original ID is still stored in the id field for reference

Security Rules

Location: firestore.rules

```
rules_version = '2';
service cloud.firestore {
  match /databases/{database}/documents {
    // Users can only access their own data
    match /users/{userId} {
      allow read, write: if request.auth != null && request.auth.uid == userId;

      match /favorites/{favoriteId} {
        allow read, write: if request.auth != null && request.auth.uid == userId;
      }

      match /visited/{visitedId} {
        allow read, write: if request.auth != null && request.auth.uid == userId;
      }

      match /itineraries/{itineraryId} {
        allow read, write: if request.auth != null && request.auth.uid == userId;
      }
    }
  }
}
```

How It Works: - Only authenticated users can read/write - Users can only access documents under their own {userId} - Prevents unauthorized access and data tampering - Database is secure by default (production mode)

Firestore Service Layer

Location: src/services/firestoreService.ts

Key Functions **Favorites:**

```
saveFavorite(userId: string, place: Place)
  // Add/update favorite in cloud

deleteFavorite(userId: string, placeId: string)
  // Remove favorite from cloud

getFavorites(userId: string): Promise<Place[]>
  // One-time fetch of all favorites

subscribeFavorites(userId: string, callback): Unsubscribe
  // Real-time listener - callback fires on any change
```

Visited Places:

```
saveVisitedPlace(userId: string, place: Place)
deleteVisitedPlace(userId: string, placeId: string)
getVisitedPlaces(userId: string): Promise<Place[]>
subscribeVisitedPlaces(userId: string, callback): Unsubscribe
```

Itineraries:

```
saveItinerary(userId: string, itinerary: ItineraryItem[])
getItinerary(userId: string): Promise<ItineraryItem[]>
subscribeItinerary(userId: string, callback): Unsubscribe
```

Migration:

```
migrateLocalStorageToFirestore(userId: string): Promise<boolean>
  // Automatically moves localStorage data to Firestore on first login

isMigrationComplete(): boolean
  // Check if user has already migrated (localStorage flag)
```

Authentication System

Firebase Auth Flow

1. **User Opens App**

   ```
   App → onAuthStateChanged(auth)
   ↓
   No User → Show Login Screen
   ↓
   User Enters Credentials → createUserWithEmailAndPassword or signInWithEmailAndPassword
   ↓
   Firebase Validates → Generates Auth Token
   ↓
   Auth State Updates → setUser(currentUser)
   ```

```
↓
Firestore Subscriptions Start → Load cloud data
```

2. **Sign Up Process**

```
async function handleSignUp(email: string, password: string) {
  const { user } = await createUserWithEmailAndPassword(auth, email, password)
  // User automatically logged in
  // Migration checks and loads Firestore data
}
```

3. **Sign In Process**

```
async function handleSignIn(email: string, password: string) {
  const { user } = await signInWithEmailAndPassword(auth, email, password)
  // User logged in
  // Firestore subscriptions load their data
}
```

4. **Sign Out**

```
async function handleLogout() {
  await signOut(auth)
  // User state cleared
  // Firestore unsubscribers called
  // App returns to Login screen
}
```

Session Persistence

- Firebase automatically persists auth state in localStorage
- Page refresh → auth state restored automatically
- Users stay logged in across browser sessions
- Credentials never stored in app code

Data Synchronization

Real-time Updates

How it works:

```
useEffect(() => {
  if (!user) return

  // Subscribe to favorites
  const unsubFavorites = subscribeFavorites(user.uid, (favorites) => {
    setFavorites(favorites)  // Triggers re-render
  })
  // Subscribe to visited places
  const unsubVisited = subscribeVisitedPlaces(user.uid, (visited) => {
    setVisitedPlaces(visited)
  })

  // Subscribe to itinerary
  const unsubItinerary = subscribeItinerary(user.uid, (items) => {
    setItinerary(items)
  })

  // Clean up on unmount
  return () => {
    unsubFavorites()
    unsubVisited()
    unsubItinerary()
  }
}, [user])
```

Behavior: - Multiple browser tabs/windows on same device = instant sync - Different devices (phone, tablet, computer) = sync within seconds - Changes appear without refresh - Optimistic UI updates for instant feedback

Migration Process

First Login:

```
User Logs In
↓
Check: localStorage has 'firestore-migrated' flag?
↓
NO → Migration Needed
  ↓
  Read all localStorage data
  ↓
  Save each item to Firestore with user ID
  ↓
  Set 'firestore-migrated' flag
  ↓
  Done!
↓
YES → Already Migrated
  ↓
  Load from Firestore directly
```

One-time per user - Never runs again for that account

Component Structure

App.tsx (Main Component)

Responsibilities: - Authentication state management - Firestore subscription setup - Route/itinerary calculations - User event handling - UI layout and sidebar

Key State:

```
const [user, setUser] = useState<any>(null)           // Current auth user
const [favorites, setFavorites] = useState<Place[]>([])
const [visitedPlaces, setVisitedPlaces] = useState<Place[]>([])
const [itinerary, setItinerary] = useState<ItineraryItem[]>([])
const [route, setRoute] = useState<RouteResult>(null)
const [startLocation, setStartLocation] = useState<GeoLocation>()
// ... more UI state
```

Key Functions:

```
toggleFavorite(place: Place)      // async - add/remove from cloud
toggleVisited(id: string)         // async - mark visited in cloud
unvisitPlace(id: string)          // async - unmark visited
onAddPlace(place: Place)          // Add to itinerary
onRemove(id: string)              // Remove from itinerary
updateRoute()                     // Calculate driving route
```

MapView.tsx (Map Component)

Responsibilities: - Render Leaflet map with markers - Handle map interactions - Display popups with place details - Filter markers based on zoom level - Show route and service markers

Features:

```
// Zoom-based place loading
zoom < 4      → No places
zoom < 6      → Major cities only (1M+ population)
zoom < 8      → All cities and towns
zoom 8+       → All places, villages, hamlets
zoom 12+      → Attraction points
```

Marker Styling: - **Red/Pink** - Standard places - **Purple** - Free campsites - **Green** - Paid campsites - **Orange** - Fuel stations - **Blue** - Dump points - **Cyan** - Water points

Visited Indicators: - Checkmark prefix - Strikethrough text - Reduced opacity (0.6)

Login.tsx (Authentication Component)

Features: - Email/password form - Toggle between sign-in and sign-up - Form validation (6+ char password) - Error messages - Loading states

Favorites.tsx (Favorites List)

Features: - State-organized display (NSW, VIC, etc.) - Alphabetical sorting - Click to center map - Weather links - Google Maps links - Add to itinerary - Visited toggle

VisitedPlaces.tsx (Visited Modal)

Features: - Modal overlay - State organization - Visit timestamps - Unvisit functionality - Click to center map

Itinerary.tsx (Trip List)

Features: - Display all stops - Distances between stops - Total drive time - Remove stop button

---

API Integrations

Overpass API (OpenStreetMap)

Purpose: Query geospatial data (cities, towns, campsites, etc.)

Endpoints (with fallback): - `https://overpass-api.de/api/interpreter` (primary) - `https://lz4.overpass-api.de/api/interpreter` (fallback) - `https://overpass.kumi.systems/api/interpreter` (fallback)

Example Query:

```
[out:json][timeout:25];
(
  node["place"="city"]["population"~"[0-9]{6,}"](south,west,north,east);
  node["place"="town"](south,west,north,east);
);
out body;
```

Rate Limiting: - 1000ms debounce on map movement - Multiple endpoints for redundancy - Graceful error handling

OSRM (Open Source Routing Machine)

Purpose: Calculate driving routes and distances

Endpoint: `https://router.project-osrm.org/trip/v1/driving`

Request Format:

```
/trip/v1/driving/lon1,lat1;lon2,lat2;lon3,lat3
?source=first&roundtrip=false&overview=full
```

Response:

```
{
  "trips": [{
    "geometry": { "coordinates": [[lon,lat], ...] },
    "legs": [
      { "distance": 123456, "duration": 7890 }
    ],
    "distance": 246912,
    "duration": 15780
  }]
}
```

Nominatim (Geocoding)

Purpose: Convert place names to coordinates

Endpoint: `https://nominatim.openstreetmap.org/search`

Parameters: - `q` - Search query - `countrycodes=au` - Australia only - `format=json` - JSON format - `limit=5` - Max 5 results

Rate Limited - Subject to usage policies

MeteoBlue (Weather)

Integration: External link (no API calls)

URL Format:

```
https://www.meteoblue.com/en/weather/forecast/week/{lat}N{lon}E
```

Google Maps

Integration: External link (no API calls)

URL Format:

```
https://www.google.com/maps/search/?api=1&query={lat},{lon}
```

Deployment

GitHub Pages

Configuration:

```
// vite.config.ts
export default defineConfig({
  base: '/interactive-Aus-map/',
  plugins: [react()],
})
```

Build Process:

```
npm run build  # Compiles React + TypeScript → dist/
```

Deployment: - GitHub Actions auto-triggered on `main` push - Workflow: `.github/workflows/deploy.yml` - Target: `gh-pages` branch - Live in 2-3 minutes

Firebase Configuration

Authorized Domains: - `localhost` (development) - `petehep.github.io` (production)

Environment Variables:

```
VITE_FIREBASE_API_KEY
VITE_FIREBASE_AUTH_DOMAIN
VITE_FIREBASE_PROJECT_ID
VITE_FIREBASE_STORAGE_BUCKET
VITE_FIREBASE_MESSAGING_SENDER_ID
VITE_FIREBASE_APP_ID
VITE_FIREBASE_MEASUREMENT_ID
```

Stored in: - `.env` file (local development) - GitHub Secrets (production via Actions)

Development Guide

Setup

```
# Clone repository
git clone https://github.com/petehep/interactive-Aus-map.git
cd interactive-Aus-map

# Install dependencies
npm install

# Set up environment
cp .env.example .env
# Edit .env with your Firebase credentials
# Start dev server
npm run dev
```

Development Server

Command: `npm run dev`
URL: `http://localhost:5173/interactive-Aus-map/`
HMR: Hot module replacement enabled

Building for Production

```
npm run build    # Create dist/ folder
npm run preview  # Test production build locally
```

File Structure

```
src/
  App.tsx                    # Main component
  firebase.ts                # Firebase configuration
  index.css                  # Global styles
  main.tsx                   # React entry point
  vite-env.d.ts             # Type definitions

  components/
     Favorites.tsx          # Favorites list
     Itinerary.tsx          # Trip itinerary
     Login.tsx              # Auth UI
     MapView.tsx            # Leaflet map
     VisitedPlaces.tsx      # Visited modal

  services/
     firestoreService.ts    # Cloud database operations

  types/
      react-leaflet-cluster.d.ts  # Type definitions
```

Code Standards

- **Language:** TypeScript strict mode
- **Styling:** Inline styles + CSS
- **Components:** Functional with hooks
- **State:** useState, useEffect, useCallback, useMemo
- **Async:** Async/await with error handling

Common Tasks

Add a new favorite:

```
const handleAddFavorite = async (place: Place) => {
  try {
    await saveFavorite(user.uid, place)
    // Real-time listener automatically updates state
  } catch (error) {
    console.error('Failed to save favorite:', error)
  }
}
```

Subscribe to data changes:

```
useEffect(() => {
  const unsubscribe = subscribeFavorites(user.uid, (newFavorites) => {
    setFavorites(newFavorites)
  })
  return () => unsubscribe()
}, [user.uid])
```

Query the map:

```
const fetchPlaces = async (bbox: [number, number, number, number]) => {
  const query = `[out:json];
    (node["place"="city"](${bbox[1]},${bbox[0]},${bbox[3]},${bbox[2]}); );
    out body;`
  const response = await fetch('https://overpass-api.de/api/interpreter', {
    method: 'POST',
    body: query
  })
  return response.json()
}
```

Performance Optimizations

Frontend

- **Marker Clustering** - Efficient large datasets
- **Zoom-based Loading** - Progressive data fetching
- **Debouncing** - 1000ms for map interactions
- **Memoization** - useCallback, useMemo

Database

- **Real-time Subscriptions** - Only listen when needed
- **Unsubscribe on Unmount** - Prevent memory leaks
- **Pagination** - Future: limit initial data fetch

Network

- **Multiple API Endpoints** - Redundancy
- **Error Recovery** - Graceful degradation
- **Caching** - Browser cache for static assets

Troubleshooting

Firestore Issues

Q: Favorites not saving - Check: User is logged in - Check: Browser console for errors - Check: Firestore rules published - Check: Firestore database created in production mode

Q: Data not syncing between devices - Check: Using same account on both devices - Check: Both devices have internet - Check: Firestore security rules allow access

Q: Migration failed - Check: localStorage data exists - Check: User authenticated - Check: Firestore quota not exceeded

Map Issues

Q: Markers not appearing - Zoom in closer (place loading is zoom-dependent) - Check filters/toggles are enabled - Refresh page

Q: Route not calculating - Need at least 2 stops in itinerary - Wait 5-10 seconds - Check internet connection

Authentication Issues

Q: Can't sign in - Verify email exists and password correct - Check Email/Password auth enabled in Firebase - Check domain authorized in Firebase

---

Future Roadmap

- ☐ Offline map caching
- ☐ Drag-to-reorder itinerary
- ☐ Distance/fuel calculations
- ☐ Photo uploads
- ☐ Trip sharing with other users
- ☐ Advanced filtering
- ☐ Expense tracking
- ☐ Mobile native app

Support & Contributing

Repository: https://github.com/petehep/interactive-Aus-map
Issues: GitHub Issues
Documentation: See included manuals

Australia Trip Scheduler - Making journey planning easy.