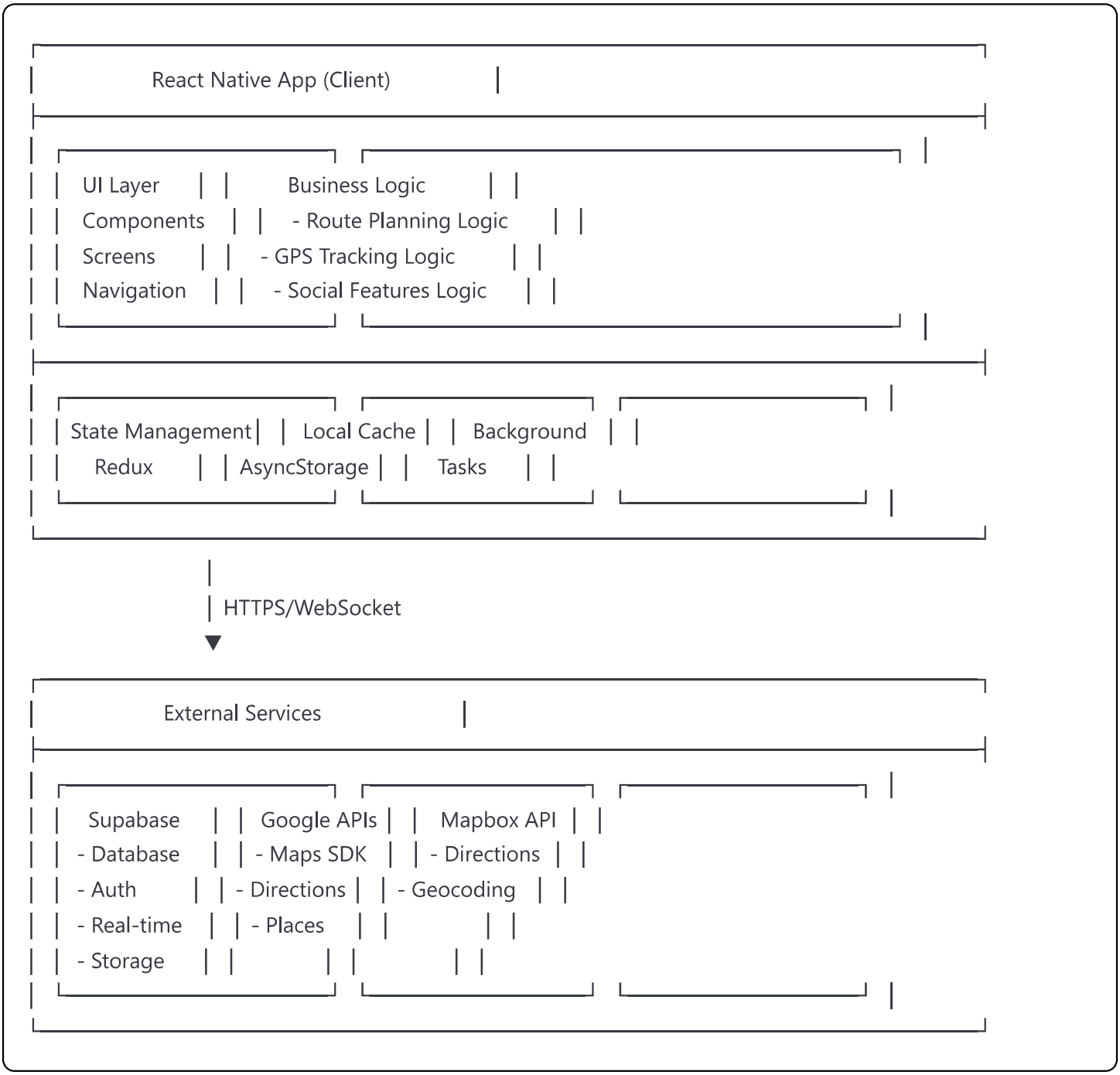
RunRoute - Technical Architecture Document

1. System Overview

RunRoute is a cross-platform mobile application built with React Native that enables runners to plan custom routes, track runs with GPS, and share their activities socially. The app follows a client-server architecture with cloud-based data storage and real-time synchronization.

2. High-Level Architecture



3. Technology Stack

Frontend (React Native)

- **Framework**: React Native with Expo managed workflow
- **Navigation**: React Navigation v6 (Stack + Tab navigators)
- **State Management**: Redux Toolkit with RTK Query
- **Maps**: react-native-maps with Google Maps SDK
- **Location Services**: expo-location for GPS tracking
- **Local Storage**: AsyncStorage for caching
- **Background Tasks**: expo-background-fetch and expo-task-manager
- **UI Components**: React Native Elements or NativeBase
- **Charts/Analytics**: Victory Native for run statistics

Backend Services

- **Database & Auth**: Supabase (PostgreSQL + Auth)
- **Real-time Updates**: Supabase Realtime
- **File Storage**: Supabase Storage for profile pictures
- **Push Notifications**: Expo Push Notifications

External APIs

- **Mapping**: Google Maps SDK (iOS/Android)
- **Routing**: Google Directions API + Mapbox Directions API (fallback)
- **Places Search**: Google Places API
- **Social Sharing**: Platform-specific sharing APIs

4. Data Architecture

Database Schema (Supabase/PostgreSQL)

Users Table

```sql
```

```
users (
  id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
  email VARCHAR UNIQUE NOT NULL,
  username VARCHAR UNIQUE,
  full_name VARCHAR,
  avatar_url VARCHAR,
  created_at TIMESTAMP DEFAULT NOW(),
  updated_at TIMESTAMP DEFAULT NOW()
)
```

Runs Table

sql

```
runs (
  id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
  user_id UUID REFERENCES users(id),
  title VARCHAR,
  distance_meters FLOAT NOT NULL,
  duration_seconds INT NOT NULL,
  average_pace FLOAT,
  route_polyline TEXT, -- Encoded polyline
  start_latitude FLOAT,
  start_longitude FLOAT,
  end_latitude FLOAT,
  end_longitude FLOAT,
  started_at TIMESTAMP NOT NULL,
  completed_at TIMESTAMP NOT NULL,
  created_at TIMESTAMP DEFAULT NOW()
)
```

Planned Routes Table

sql

```
planned_routes (
  id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
  user_id UUID REFERENCES users(id),
  name VARCHAR,
  target_distance_meters FLOAT NOT NULL,
  is_loop BOOLEAN DEFAULT false,
  start_latitude FLOAT NOT NULL,
  start_longitude FLOAT NOT NULL,
  end_latitude FLOAT,
  end_longitude FLOAT,
  waypoints JSONB, -- Array of lat/lng waypoints
  route_polyline TEXT,
  created_at TIMESTAMP DEFAULT NOW()
)
```

Social Feed Table

sql

```
run_posts (
  id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
  run_id UUID REFERENCES runs(id),
  user_id UUID REFERENCES users(id),
  caption TEXT,
  is_public BOOLEAN DEFAULT true,
  likes_count INT DEFAULT 0,
  comments_count INT DEFAULT 0,
  created_at TIMESTAMP DEFAULT NOW()
)

post_likes (
  user_id UUID REFERENCES users(id),
  post_id UUID REFERENCES run_posts(id),
  created_at TIMESTAMP DEFAULT NOW(),
  PRIMARY KEY (user_id, post_id)
)

post_comments (
  id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
  post_id UUID REFERENCES run_posts(id),
  user_id UUID REFERENCES users(id),
  comment_text TEXT NOT NULL,
  created_at TIMESTAMP DEFAULT NOW()
)
```

User Relationships Table

```sql
user_follows (
  follower_id UUID REFERENCES users(id),
  following_id UUID REFERENCES users(id),
  created_at TIMESTAMP DEFAULT NOW(),
  PRIMARY KEY (follower_id, following_id)
)
```

5. Component Architecture

App Structure

```
src/
```

```
├── components/          # Reusable UI components
│   ├── common/          # Button, Input, Loading, etc.
│   ├── map/             # MapView, RoutePolyline, Markers
│   └── social/          # RunCard, CommentList, etc.
├── screens/             # Screen components
│   ├── auth/            # Login, Register
│   ├── planning/        # RoutePlanning, RoutePreview
│   ├── tracking/        # LiveTracking, RunSummary
│   ├── history/         # RunHistory, RunDetails
│   └── social/          # Feed, Profile, Friends
├── services/            # API calls and external services
│   ├── supabase.js      # Supabase client configuration
│   ├── google-maps.js   # Google Maps API calls
│   ├── location.js      # GPS tracking utilities
│   └── notifications.js # Push notification handling
├── store/               # Redux store configuration
│   ├── slices/          # Redux slices (auth, runs, routes)
│   └── api/             # RTK Query API endpoints
├── utils/               # Utility functions
│   ├── distance.js      # Distance calculations
│   ├── polyline.js      # Polyline encoding/decoding
│   └── formatting.js    # Data formatting utilities
└── navigation/          # Navigation configuration
    ├── AppNavigator.js  # Main navigation setup
    └── TabNavigator.js  # Bottom tab navigation
```

6. Key Workflows

Route Planning Flow

1. User selects start location on map
2. User sets target distance and loop preference
3. App calls Google Directions API for initial route
4. Route distance is calculated and adjusted if needed
5. User can drag waypoints to customize route
6. Modified route triggers new Directions API call
7. Route is saved to planned_routes table

Live Run Tracking Flow

1. User starts tracking from planned or ad-hoc route

2. expo-location begins GPS tracking with high accuracy
3. Background task continues tracking when app is minimized
4. Real-time polyline is drawn on map
5. Distance and pace are calculated continuously
6. User can pause/resume tracking
7. Completed run is saved to runs table
8. Optional social post is created

Social Feed Flow

1. User creates run post after completing run
2. Post is saved to run_posts table
3. Supabase Realtime broadcasts new post to followers
4. Feed updates in real-time for connected users
5. Users can like/comment with optimistic updates

7. Security Considerations

Authentication & Authorization

- Supabase Auth handles user authentication with JWT tokens
- Row Level Security (RLS) policies protect user data
- API keys are stored in environment variables
- Sensitive operations require re-authentication

Data Privacy

- Location data is encrypted at rest
- Users control run visibility (public/private)
- Option to hide exact start/end locations near home
- GDPR-compliant data export/deletion

API Security

- Google Maps API keys restricted by bundle ID/package name
- Rate limiting implemented for route generation
- Input validation on all user-provided data

8. Performance Optimization

Client-Side

- Route polylines cached in AsyncStorage
- Image lazy loading for social feed
- Debounced search for places API
- Optimized map rendering with clustering

Server-Side

- Database indexes on frequently queried fields
- Supabase connection pooling
- Polyline compression for storage efficiency
- CDN for static assets

Background Tasks

- Efficient GPS tracking with configurable intervals
- Smart battery optimization
- Graceful handling of location permission changes

9. Monitoring & Analytics

Performance Monitoring

- Expo Application Services for crash reporting
- Custom analytics for route planning success rates
- GPS accuracy monitoring
- API response time tracking

User Analytics

- Feature usage tracking
- Route completion rates
- Social engagement metrics
- Retention and churn analysis

10. Development & Deployment

Development Environment

- Expo CLI for development and testing
- Flipper for debugging (React Native)
- Supabase local development setup
- Environment-specific configuration

CI/CD Pipeline

- GitHub Actions for automated testing
- EAS Build for app compilation
- EAS Submit for store deployment
- Automated database migrations

App Store Deployment

- iOS: App Store Connect with TestFlight beta testing
- Android: Google Play Console with internal testing
- Over-the-air updates via Expo Updates

11. Scalability Considerations

Database Scaling

- Read replicas for analytics queries
- Partitioning for large runs table
- Archiving old data beyond retention period

API Rate Limiting

- Intelligent caching of route requests
- Fallback between Google and Mapbox APIs
- User-based rate limiting for expensive operations

Real-time Performance

- Supabase Realtime connection management
- WebSocket connection pooling

- Selective subscriptions based on user activity

This architecture provides a solid foundation for building RunRoute while maintaining flexibility for future enhancements and scaling needs.