# SpareTime - Calendar Availability App

## Project Requirements Document (PRD)

**Version:** 1.0
**Date:** October 24, 2025
**Domain:** sparetime.me

---

## Table of Contents

---

## 1. Executive Summary

SpareTime is a social availability layer application that transforms how people share and communicate their availability. Unlike traditional calendar sharing or scheduling tools, SpareTime allows users to create customized views of their availability for different audiences, protecting their boundaries while maintaining social connections.

The core innovation of SpareTime is the ability for users to:

- Import calendar data from existing sources
- Create customized "availability views" for different audiences

- Artificially mark time as busy to protect personal boundaries
- Allow others to see when you're "busy but flexible" for select people
- Enable time requests through an innovative "three nearest options" interface

This application solves the fundamental problem of time boundary management in both personal and professional contexts. It provides a middle ground between the binary options of either fully exposing your calendar or keeping it completely private, allowing for a more nuanced approach to sharing availability that respects both the user's need for privacy and the social norms of availability coordination.

---

# 2. Product Definition

## 2.1 Problem Statement

Current calendar and scheduling tools force users into a binary choice: either expose their entire calendar or engage in inefficient back-and-forth communication. This creates several problems:

1. **Privacy concerns:** Users don't want everyone to see everything on their calendar
2. **Boundary issues:** It's difficult to protect personal time without appearing unavailable
3. **Scheduling friction:** "When are you free?" conversations waste time and create social awkwardness
4. **Contextual needs:** Different relationships require different levels of calendar transparency

SpareTime solves these problems by providing a flexible layer between users' actual calendars and what others see, with fine-grained controls for different audiences.

## 2.2 Product Vision

**Vision:** A world where people have healthier relationships with their time and clearer boundaries in both professional and personal contexts.

**Mission:** To provide a social availability layer that helps users communicate when they're free while maintaining control over their personal time boundaries.

## 2.3 Core Value Proposition

1. **For individuals:** SpareTime lets you share your availability in a way that protects your privacy and personal boundaries while still appearing responsive to others.

2. **For teams:** SpareTime enables team members to coordinate efficiently without exposing private calendar details, providing just enough information to find mutual available times.

3. **For families:** SpareTime helps family members coordinate schedules while giving parents control over what information is shared with whom, protecting family time while allowing for coordination.

### 2.4 Key Differentiators

1. **Not a scheduling tool:** SpareTime focuses on communicating availability rather than direct booking, emphasizing the social aspect of time coordination.

2. **Boundary control:** Users can mark time as artificially busy, protecting personal time without revealing why.

3. **Audience-specific views:** Create different availability views for different groups of people.

4. **Negotiable time:** Mark time as "busy but flexible" for select viewers, allowing for social negotiation.

5. **Time request simplification:** The "three nearest options" interface reduces decision fatigue and screen real estate.

### 2.5 Target Users

- Professionals who manage multiple stakeholders but need to protect focus time
- Individuals juggling multiple social circles with different boundary needs
- Parents managing family schedules while protecting family time
- Teams needing to coordinate without overexposing individual schedules
- Anyone who values both personal boundaries and social responsiveness

---

# 3. Market Analysis

## 3.1 Competitive Landscape

### 3.1.1 Google Calendar

**Similarities:**

- Calendar integration
- Basic availability sharing
- Time zone support

**Differences:**

- Google Calendar exposes full calendar details when shared
- No concept of audience-specific views
- No artificial "busy" blocks for boundary protection
- No "busy but flexible" status

### 3.1.2 Calendly

**Similarities:**

- Shareable availability links
- Time slot selection
- Calendar integration

**Differences:**

- Transactional, booking-focused approach
- No audience-specific customization
- Binary busy/free status only
- Grid-based time slot selection using significant screen space
- No concept of "busy but flexible" time

### 3.1.3 When2Meet/Doodle

**Similarities:**

- Helps find mutual available times
- Simple interface

**Differences:**

- Event-specific rather than persistent
- No privacy controls or boundary protection
- No natural language time input
- No concept of negotiable time

### 3.1.4 Manual Coordination

**Similarities:**

- Personal, relationship-based
- Allows for social negotiation

**Differences:**

- Inefficient and repetitive
- No visual aids or availability context
- No persistent record of availability patterns

### 3.2 Market Opportunity

The calendar and scheduling tools market is large ($3B+ globally) but focused primarily on two types of solutions:

1. **Calendar apps:** Focused on creating and viewing events
2. **Scheduling tools:** Focused on booking specific time slots

SpareTime creates a new category: **Social Availability Layer** applications that focus on communicating time boundaries rather than event creation or direct booking. This addresses a gap in the market for solutions that respect both personal time boundaries and social coordination needs.

### 3.3 Business Potential

1. **Freemium model:**
    - Basic version: Single availability link, basic customization
    - Premium version ($5.99/month): Multiple audience links, advanced features
2. **Team/enterprise licenses:** For companies and organizations
3. **Target market size:**
    - 500M+ knowledge workers globally
    - 1B+ smartphone users who coordinate schedules
4. **Revenue projections:**
    - Year 1: 100,000 users, 5% conversion to premium = $359,400
    - Year 3: 1M users, 8% conversion to premium = $5.75M

---

# 4. User Personas

### 4.1 Maya - The Boundary-Setter

**Demographics:**

- 32-year-old UX designer
- Lives in a major city
- Active social life, side projects, and demanding job

**Goals:**

- Protect focused work time without appearing unavailable
- Manage different social groups without revealing full schedule
- Reduce time spent on "when are you free?" conversations

**Pain Points:**

- Feels guilty saying "no" to requests
- Calendar is too private to share but too full to coordinate manually
- Different social circles have different expectations about her time

**User Journey with SpareTime:**

1. Imports work and personal calendars
2. Creates different links for coworkers, close friends, and acquaintances
3. Blocks out focus time that appears as "busy" to everyone
4. Makes certain work commitments appear as "busy but flexible" to close friends

### 4.2 Alex - The Team Lead

**Demographics:**

- 41-year-old engineering manager
- Leads a team of 12 developers across 3 time zones
- Balances management duties with hands-on work

**Goals:**

- Protect focus time while being available to team
- Coordinate team meetings efficiently
- Signal when he's interruptible vs. deep in focused work

**Pain Points:**

- Calendar is filled with sensitive meetings he can't share broadly
- Team asks "do you have time?" throughout the day
- Struggles to find meeting times that work for distributed team

**User Journey with SpareTime:**

1. Imports work calendar

2. Creates team view showing general availability but hiding meeting details
3. Marks 1-on-1s as "busy but flexible" for direct reports
4. Sets up team availability view to find mutual free time

### 4.3 Sarah - The Family Coordinator

**Demographics:**

- 37-year-old parent with two children (8 and 11)
- Works part-time and manages family logistics
- Coordinates with extended family and children's activity leaders

**Goals:**

- Protect family dinner and weekend time
- Coordinate with spouse, children's schools, and activities
- Share appropriate availability with extended family

**Pain Points:**

- Family time constantly encroached upon by outside requests
- Manual coordination with multiple parties is exhausting
- Different levels of schedule detail needed for different people

**User Journey with SpareTime:**

1. Imports family calendar from Google
2. Creates protected time blocks for family dinners and weekends
3. Sets up different views for school contacts, activity leaders, and extended family
4. Uses team view to coordinate with immediate family members

---

# 5. Feature Requirements

## 5.1 Core Features (MVP)

### 5.1.1 User Account Management

1. **Registration & Authentication**
    - Email/password registration
    - Social login (Google, Apple)
    - Forgot password flow
    - Account settings

2. **Profile Management**

    - User profile (name, photo, time zone)
    - Username selection (for sparetime.me/username)
    - Default availability preferences

#### 5.1.2 Calendar Integration

1. **Calendar Connections**

    - Google Calendar OAuth integration
    - Apple Calendar integration (iOS only)
    - Manual calendar entry option
    - Calendar sync settings (frequency, scope)
2. **Calendar Management**

    - View/manage connected calendars
    - Set default visibility for each calendar
    - Color coding and categorization
    - Calendar refresh and sync status

#### 5.1.3 Availability Customization

1. **Protected Time Blocks**

    - Create artificial "busy" time blocks
    - Set recurring protected time
    - Exceptions to recurring patterns
    - Bulk protection (e.g., "make all evenings busy")
2. **Visibility Rules**

    - Default visibility settings
    - Calendar-specific visibility
    - Time-based visibility rules
    - Special date handling (holidays, etc.)

#### 5.1.4 Link Management

1. **Link Creation**

    - Generate unique availability links
    - Custom link naming
    - Link expiration settings
    - Link viewing and sharing options

2. **Link Customization**
    - Per-link visibility settings
    - Custom messages for link visitors
    - Time frame restrictions
    - Enable/disable time requests
3. **Link Analytics**
    - View count
    - Visitor statistics
    - Request patterns
    - Last accessed date

#### 5.1.5 Time Request System

1. **Request Interface**
    - Date selection
    - Natural language time input
    - Three nearest options display
    - Request purpose and details
2. **Request Management**
    - Incoming request notifications
    - Request approval/denial
    - Propose alternative times
    - Request history
3. **Time Selection Algorithm**
    - Nearest time slot calculation
    - Natural language time parsing
    - Availability calculation
    - Slot ranking and filtering

## 5.2 Premium Features

#### 5.2.1 Multiple Audience Views

1. **Audience Management**
    - Create audience categories
    - Assign links to audiences
    - Bulk update audience settings
    - Audience templates

2. **Audience-Specific Rules**
    - Per-audience visibility settings
    - Custom messages by audience
    - Time-block labeling by audience
    - Request permissions by audience

#### 5.2.2 Negotiable Time Slots

1. **Negotiable Status**
    - Mark time as "busy but movable"
    - Set movability by audience
    - Indicate priority levels
    - Conditional availability rules
2. **Negotiation Interface**
    - Visual indicators for negotiable time
    - Request flow for negotiable slots
    - Suggested alternatives
    - Priority-based access

#### 5.2.3 Rich Status Messages

1. **Status Creation**
    - Custom status messages
    - Timed status updates
    - Status by audience
    - Status templates
2. **Status Display**
    - Status visibility on links
    - Status integration with availability
    - Status history
    - Status notifications

#### 5.2.4 Advanced Time Protection

1. **Pattern Protection**
    - Smart pattern recognition
    - Protection suggestions
    - Automated protection rules
    - Protection exceptions

2. **Protection Management**

    - Protection calendar view
    - Protection statistics
    - Protection override
    - Protection templates

**5.2.5 Team/Group Coordination**

1. **Group Management**

    - Create and manage groups
    - Add/remove members
    - Set group permissions
    - Group availability view

2. **Mutual Availability**

    - Find mutual free times
    - Group request system
    - Majority-available times
    - Optimal meeting finder

---

# 6. UI/UX Specifications

## 6.1 Design Principles

1. **Calm & Clear**

    - Clean, minimal interface
    - Generous white space
    - Clear visual hierarchy
    - Subtle color coding for status

2. **Human & Personal**

    - Warm, conversational copy
    - Friendly illustrations
    - Accessible to diverse users
    - Emotionally intelligent microcopy

3. **Efficient & Focused**

    - Minimize clicks for common tasks
    - Progressive disclosure for complex features

    - Mobile-first design approach
    - Consistent navigation patterns

## 6.2 Color Scheme

1. **Primary Colors**
    - Primary Blue: #2E75B6 (buttons, links, key actions)
    - Background White: #FFFFFF (main background)
    - Text Black: #333333 (primary text)
2. **Status Colors**
    - Available: #4CAF50 (green)
    - Busy: #F44336 (red)
    - Negotiable: #FF9800 (orange)
    - Protected: #9C27B0 (purple)
3. **Neutral Colors**
    - Light Gray: #F5F5F5 (section backgrounds)
    - Medium Gray: #E0E0E0 (dividers, borders)
    - Dark Gray: #9E9E9E (secondary text)

## 6.3 Typography

1. **Font Families**
    - Primary: Inter (headers and body text)
    - Fallback: System font stack
2. **Type Scale**
    - H1: 32px, 700 weight
    - H2: 24px, 700 weight
    - H3: 20px, 600 weight
    - Body: 16px, 400 weight
    - Small: 14px, 400 weight
3. **Type Styles**
    - Headers: Left-aligned, Primary Blue
    - Body text: Left-aligned, Text Black
    - Links: Primary Blue, no underline
    - Buttons: 16px, 600 weight, uppercase

## 6.4 Screen Specifications

### 6.4.1 Onboarding Flow

**Screen 1: Welcome**

- **Elements:**
  - `welcomeTitle`: Main headline
  - `welcomeSubtitle`: Brief explanation
  - `signUpButton`: Primary CTA
  - `loginLink`: Secondary CTA
  - `featureCarousel`: Key feature showcase

**Screen 2: Sign Up**

- **Elements:**
  - `signUpForm`: Registration form
  - `nameField`: Full name input
  - `emailField`: Email input
  - `passwordField`: Password input with requirements
  - `createAccountButton`: Submit button
  - `socialLoginButtons`: Google, Apple buttons
  - `termsCheckbox`: Terms agreement

**Screen 3: Calendar Connection**

- **Elements:**
  - `connectionTitle`: Section header
  - `googleCalendarButton`: Connect Google Calendar
  - `appleCalendarButton`: Connect Apple Calendar (iOS only)
  - `skipStepLink`: Skip for now option
  - `calendarSelectionList`: If multiple calendars found
  - `syncOptionsForm`: Sync settings
  - `continueButton`: Proceed to next step

**Screen 4: Time Preferences**

- **Elements:**
  - `preferencesTitle`: Section header
  - `timezoneSelector`: User's time zone
  - `workHoursForm`: Default work hours
  - `protectedTimeForm`: Initial protected time
  - `visibilityDefaultSelector`: Default visibility
  - `finishButton`: Complete setup

**Screen 5: Welcome Tour**

- **Elements:**
    - `tourTitle`: Welcome message
    - `createLinkCTA`: Create first link
    - `tourSteps`: Tooltips for key features
    - `tourProgressIndicator`: Step indicators
    - `tourSkipButton`: Skip tour option

**Wireframe:**

```None
┌──────────────────────────────┐
│          Welcome to          │
│          SpareTime           │
│                              │
│   Your availability, your    │
│            way.              │
│                              │
│    [Feature Carousel]        │
│                              │
│     ┌──────────────────┐      │
│     │     Sign Up      │     │
│     └──────────────────┘      │
│                              │
│    Already have an account?  │
│          Log In              │
└──────────────────────────────┘
```

**6.4.2 Dashboard (Main App)**

**Elements:**

- `appHeader`: App navigation and profile
    - `logoIcon`: App logo
    - `profileButton`: User profile access
    - `notificationsButton`: Notification bell
    - `settingsButton`: Settings access
- `tabBar`: Main navigation tabs

    - `linksTab`: Manage links
    - `calendarTab`: View calendar
    - `requestsTab`: Manage requests
    - `profileTab`: User profile
- `dashboardContent`: Main content area
    - `welcomeMessage`: Personalized greeting
    - `activeLinksSection`: Active links summary
    - `pendingRequestsSection`: Pending requests
    - `quickActionsRow`: Common actions
    - `createLinkButton`: Create new link
    - `protectedTimeSummary`: Protected time blocks

**Wireframe:**

```
None
┌─────────────────────────────┐
│ SpareTime     👤 🔔 ⚙️       │
├─────────────────────────────┤
│                             │
│ Welcome, [Name]             │
│                             │
│ Active Links (3)            │
│ ┌─────────────────────────┐ │
│ │ Friends Link            │ │
│ │ 12 views · 2 requests   │ │
│ └─────────────────────────┘ │
│ ┌─────────────────────────┐ │
│ │ Work Link               │ │
│ │ 5 views · 1 request     │ │
│ └─────────────────────────┘ │
│                             │
│ Pending Requests (2)        │
│ ┌─────────────────────────┐ │
│ │ Coffee with Sam         │ │
│ │ Tomorrow at 3:30pm      │ │
│ └─────────────────────────┘ │
│                             │
│ ┌─────────────────────────┐ │
│ │      Create New Link     │ │
│ └─────────────────────────┘ │
│                             │
├─────────────────────────────┤
│ 🔗 Links   📅 Calendar       │
│ 🔔 Requests  👤 Profile      │
└─────────────────────────────┘
```

### 6.4.3 Link Creation/Edit Screen

**Elements:**

- `linkHeader`: Section header
    - `backButton`: Return to dashboard
    - `titleText`: "Create Link" or "Edit Link"
    - `saveButton`: Save changes
- `linkDetailsForm`: Link configuration form
    - `linkNameField`: Custom name for link
    - `linkRecipientField`: Optional recipient/group
    - `linkExpirationPicker`: Set optional expiration
    - `linkURLPreview`: Generated URL display
    - `copyLinkButton`: Copy to clipboard
- `visibilitySection`: Visibility settings
    - `showLabelsToggle`: Show event labels
    - `showNegotiableToggle`: Show negotiable status
    - `detailLevelSelector`: Amount of detail to show
- `calendarVisibilitySection`: Per-calendar settings
    - `calendarList`: List of connected calendars
    - `visibilitySelector`: Per-calendar visibility
- `customMessageField`: Optional message for viewers
- `requestSettingsSection`: Time request settings
    - `allowRequestsToggle`: Allow/disallow requests
    - `defaultDurationPicker`: Default duration
    - `timeConstraintsForm`: Earliest/latest times
- `previewButton`: Preview link view
- `shareButton`: Share link

**Wireframe:**

```
None

┌─────────────────────────────┐
│ ← Create Link      Save     │
├─────────────────────────────┤
│                             │
│ Link Details                │
│ ┌──────────────────────────┐ │
│ │ Link Name                │ │
│ └──────────────────────────┘ │
│                             │
│ ┌──────────────────────────┐ │
│ │ For (optional)           │ │
│ └──────────────────────────┘ │
│                             │
│ Expires: Never ▼            │
│                             │
│ sparetime.me/you/link-id    │
│ [Copy]                      │
│                             │
│ Visibility                  │
│ Show event labels      ⚪ ⚫    │
│ Show negotiable        ⚪ ⚫    │
│                             │
│ Detail level: Minimal   ▼   │
│                             │
│ Calendars                   │
│ Work         Busy      ▼    │
│ Personal     Hidden    ▼    │
│                             │
│ Custom Message              │
│ ┌──────────────────────────┐ │
│ │ Hi! Here's my...         │ │
│ └──────────────────────────┘ │
│                             │
│ ┌──────────────────────────┐ │
│ │          Preview         │ │
│ └──────────────────────────┘ │
│                              │
│ ┌──────────────────────────┐ │
│ │          Share           │ │
│ └──────────────────────────┘ │
└──────────────────────────────┘
```

#### 6.4.4 Calendar View

**Elements:**

- `calendarHeader`: Section header
  - `backButton`: Return to dashboard
  - `titleText`: "My Calendar"
  - `todayButton`: Jump to today
  - `viewSelector`: Day/Week/Month toggle
- `dateNavigator`: Date navigation controls
  - `prevButton`: Previous period
  - `dateDisplay`: Current date/range
  - `nextButton`: Next period
- `calendarGrid`: Calendar display
  - `timeLabels`: Hour markers
  - `eventBlocks`: Calendar events
  - `protectedBlocks`: Protected time blocks
  - `artificialBusyBlocks`: Artificial busy time
- `addButtonsRow`: Action buttons
  - `addEventButton`: Add calendar event
  - `addProtectedTimeButton`: Add protected time
  - `addArtificialBusyButton`: Add artificial busy
- `calendarSourceList`: Calendar filter list
  - `sourceItem`: Individual calendar source
  - `sourceToggle`: Show/hide toggle
  - `sourceColorIndicator`: Calendar color
- `dayDetailSection`: Selected day detail (mobile)
  - `dayEventsTimeline`: Events for selected day
  - `dayProtectedTimeBlocks`: Protected time

    ◦ `dayArtificialBusyBlocks`: Artificial busy

**Wireframe:**

```None
┌──────────────────────────────┐
| ← My Calendar    Today       |
├──────────────────────────────┤
| Day | Week | Month           |
├──────────────────────────────┤
| < October 2025 >             |
├──────────────────────────────┤
|    |                         |
| 8  |  ┌───────────────┐      |
|    |  | Team Meeting  |      |
| 9  |  └───────────────┘      |
|    |                         |
|10  |                         |
|    |                         |
|11  |  ┌───────────────────┐  |
|    |  | Protected Time    |  |
|12  |  |                   |  |
|    |  |                   |  |
| 1  |  └───────────────────┘  |
|    |                         |
| 2  |  ┌───────────────┐      |
|    |  | Client Call   |      |
| 3  |  └───────────────┘      |
|    |                         |
| 4  |  ┌───────────────────┐  |
|    |  | Artificial Busy   |  |
| 5  |  └───────────────────┘  |
|    |                         |
├──────────────────────────────┤
| + Event  + Protected  + Busy|
├──────────────────────────────┤
| Calendars:                   |
| ● Work   ● Personal          |
│ ● Protected  ● Artificial  │
└────────────────────────────┘
```

**6.4.5 Availability View (Link Recipient)**

**Elements:**

- `viewHeader`: Page header
  - `ownerNameDisplay`: Calendar owner's name
  - `statusMessageDisplay`: Custom status message
- `dateSelector`: Date selection controls
  - `miniCalendarButton`: Open mini calendar
  - `prevDayButton`: Previous day
  - `dateDisplay`: Current date
  - `nextDayButton`: Next day
- `availabilityTimeline`: Time availability display
  - `timeLabels`: Hour markers
  - `busyBlocks`: Busy time periods
  - `negotiableBlocks`: "Busy but flexible" periods
  - `freeBlocks`: Available time periods
- `timeRequestSection`: Request time controls
  - `datePillSelector`: Date selection pill
  - `timeInputField`: Preferred time input
  - `findTimesButton`: Search for available times
- `timeOptionsContainer`: Available time options
  - `timeOptionChip`: Individual time option
  - `timeLabel`: Time display
  - `durationLabel`: Duration indicator
  - `statusBadge`: Special status (if any)
  - `moreTimesLink`: Show additional options
- `requestFormSection`: Time request form
  - `nameField`: Requester name
  - `emailField`: Requester email
  - `purposeField`: Purpose of meeting
  - `messageField`: Additional message
  - `submitButton`: Submit request

**Wireframe:**

```None
┌──────────────────────────────┐
│ Maya's Availability          │
│                              │
│ October is mostly packed!    │
├──────────────────────────────┤
│ < Oct 24, 2025 >             │
├──────────────────────────────┤
│ Morning                      │
│  ┌──────────────────────────┐ │
│  │ 9:00 - 10:30  Busy       │ │
│  └──────────────────────────┘ │
│                              │
│  ┌──────────────────────────┐ │
│  │ 11:00 - 12:00 Negotiable│ │
│  └──────────────────────────┘ │
│                              │
│ Afternoon                    │
│                              │
│  ┌──────────────────────────┐ │
│  │ 2:00 - 3:30  Busy        │ │
│  └──────────────────────────┘ │
│                              │
│ Evening                      │
│                              │
│ Request a time:              │
│  ┌─────────┐ ┌──────────────┐ │
│  │ Oct 24  ││e.g., 4pm     │ │
│  └─────────┘ └──────────────┘ │
│  ┌──────────────────────────┐ │
│  │        Find Times        │ │
│  └──────────────────────────┘ │
│                              │
│ Available times:             │
│  ┌──────────────────────────┐ │
│ │ 4:00 PM • 30m          │ │
│ └────────────────────────┘ │
│ ┌────────────────────────┐ │
│ │ 5:15 PM • 30m          │ │
│ └────────────────────────┘ │
│ ┌────────────────────────┐ │
│ │ 6:30 PM • 30m  Movable │ │
│ └────────────────────────┘ │
│                            │
│ Show more times            │
└────────────────────────────┘
```

**6.4.6 Time Request Interface (Detail)**

**Elements:**

- `requestHeader`: Section header
    - `backButton`: Return to availability view
    - `titleText`: "Request Time"
- `selectedTimeDisplay`: Chosen time details
    - `dateLabel`: Selected date
    - `timeLabel`: Selected time
    - `durationLabel`: Meeting duration
- `requestForm`: Request details form
    - `nameField`: Requester name
    - `emailField`: Requester email
    - `purposeSelector`: Meeting purpose
    - `purposeOtherField`: Custom purpose (if "Other")
    - `messageField`: Additional notes
    - `urgencyIndicator`: Optional importance
- `submitRequestButton`: Submit the request
- `cancelButton`: Cancel and return

**Wireframe:**

None

```
┌──────────────────────────────┐
│ ← Request Time               │
├──────────────────────────────┤
│                              │
│ Friday, Oct 24, 2025         │
│ 4:00 PM for 30 minutes       │
│                              │
│ Your Details                 │
│ ┌──────────────────────────┐ │
│ │ Your Name                │ │
│ └──────────────────────────┘ │
│                              │
│ ┌──────────────────────────┐ │
│ │ Your Email               │ │
│ └──────────────────────────┘ │
│                              │
│ Purpose: Coffee Chat      ▼  │
│                              │
│ Message (optional)           │
│ ┌──────────────────────────┐ │
│ │                          │ │
│ │                          │ │
│ └──────────────────────────┘ │
│                              │
│ Urgency: Regular         ▼   │
│                              │
│ ┌──────────────────────────┐ │
│ │      Submit Request      │ │
│ └──────────────────────────┘ │
│                              │
│ Cancel                       │
└──────────────────────────────┘
```

### 6.4.7 Request Management Screen

**Elements:**

- `requestsHeader`: Section header
  - `backButton`: Return to dashboard
  - `titleText`: "Time Requests"
  - `filterButton`: Filter options
- `requestsTabBar`: Request type tabs
  - `incomingTab`: Requests from others
  - `outgoingTab`: Requests you've sent
  - `completedTab`: Past requests
- `requestsList`: List of requests
  - `requestCard`: Individual request
    - `requesterInfo`: Name and contact
    - `timeInfo`: Requested date and time
    - `purposeLabel`: Stated purpose
    - `messagePreview`: Request message
    - `urgencyIndicator`: Request urgency
    - `actionButtonsRow`: Response buttons
    - `acceptButton`: Accept request
    - `proposeButton`: Propose alternative
    - `declineButton`: Decline request
  - `emptyStateMessage`: When no requests
  - `loadMoreButton`: Load older requests
- `bulkActionBar`: Actions for multiple selections
  - `selectAllCheckbox`: Select all visible
  - `bulkAcceptButton`: Accept selected
  - `bulkDeclineButton`: Decline selected
  - `selectCountLabel`: Selection counter

**Wireframe:**

```
None
┌─────────────────────────────┐
│ ← Time Requests    Filter   │
├─────────────────────────────┤
│ Incoming | Outgoing | Past  │
├─────────────────────────────┤
│                             │
│  ┌─────────────────────────┐ │
│ │ Coffee with Alex        │ │
│ │ Friday at 4:00 PM       │ │
│ │ "Would love to catch up"│ │
│ │                         │ │
│ │ Accept  Propose  Decline│ │
│ └─────────────────────────┘ │
│                             │
│ ┌─────────────────────────┐ │
│ │ Project Discussion      │ │
│ │ Monday at 11:00 AM      │ │
│ │ "Need to discuss the..."│ │
│ │ ⚠️ Urgent               │ │
│ │ Accept  Propose  Decline│ │
│ └─────────────────────────┘ │
│                             │
│ ┌─────────────────────────┐ │
│ │ Team Check-in           │ │
│ │ Next Thursday at 2:00 PM│ │
│ │ "Weekly sync"           │ │
│ │                         │ │
│ │ Accept  Propose  Decline│ │
│ └─────────────────────────┘ │
│                             │
│ Load more...                │
└─────────────────────────────┘
```

**6.4.8 Protected Time Creation Screen**

**Elements:**

- `protectedHeader`: Section header
  - `backButton`: Return to calendar view
  - `titleText`: "Add Protected Time"
  - `saveButton`: Save protected time
- `timeDetailsSection`: Time configuration
  - `titleField`: Name for protected block
  - `dateRangePicker`: Date selection
  - `startTimePicker`: Start time

    - `endTimePicker`: End time
    - `recurrenceToggle`: Make recurring
- `recurrenceSection`: Recurring options
    - `recurrencePatternSelector`: Frequency
    - `recurrenceDaysSelector`: Days of week
    - `recurrenceEndSelector`: End condition
- `visibilitySection`: Visibility settings
    - `defaultVisibilitySelector`: Default view
    - `audienceOverridesList`: Per-audience settings
    - `addAudienceButton`: Add audience override
- `prioritySection`: Priority settings
    - `prioritySelector`: Importance level
    - `movabilityToggle`: Allow moving
    - `exceptionRulesEditor`: Create exceptions

**Wireframe:**

None

```
┌───────────────────────────────┐
│ ← Add Protected Time    Save │
├───────────────────────────────┤
│                              │
│ ┌──────────────────────────┐ │
│ │ Title (e.g., Focus Time)│ │
│ └──────────────────────────┘ │
│                              │
│ Date                         │
│ ┌──────────────────────────┐ │
│ │ Oct 24, 2025            │ │
│ └──────────────────────────┘ │
│                              │
│ Time                         │
│ ┌──────────┐ ┌──────────┐    │
│ │ 1:00 PM │ │ 3:00 PM │    │
│ └──────────┘ └──────────┘    │
│                              │
│ Repeats  ⚪ ⚫                │
│                              │
```

```
| Repeat frequency: Weekly ▼  |
|                             |
| On these days:              |
| M  T  W  Th  F  Sa  Su      |
| ⚪ ⚫ ⚪ ⚪ ⚫ ⚪ ⚪          |
|                             |
| Ends: Never              ▼  |
|                             |
| Visibility                  |
| Default: Busy            ▼  |
|                             |
| Friends: Negotiable      ▼  |
| Family: Free             ▼  |
| + Add audience override     |
|                             |
| Priority: High           ▼  |
| Allow moving  ⚪ ⚫          |
└─────────────────────────────┘
```

**6.4.9 Settings Screen**

**Elements:**

- `settingsHeader`: Section header
  - `backButton`: Return to dashboard
  - `titleText`: "Settings"
- `accountSection`: Account settings
  - `profileSettings`: User profile options
  - `notificationSettings`: Notification preferences
  - `privacySettings`: Privacy controls
  - `subscriptionSettings`: Plan and billing
- `calendarSection`: Calendar settings
  - `calendarConnections`: Manage connections
  - `syncSettings`: Sync preferences
  - `defaultVisibility`: Default settings
- `appSection`: Application settings
  - `timeZoneSettings`: Time zone preferences

    - `themeSelector`: App theme
    - `languageSelector`: App language
    - `startDaySelector`: Week start day
- `advancedSection`: Advanced options
    - `dataExport`: Export user data
    - `deleteAccount`: Account deletion
    - `debugTools`: Troubleshooting options
- `helpSection`: Support resources
    - `helpCenter`: Help documentation
    - `contactSupport`: Support contact
    - `feedbackForm`: Submit feedback
    - `aboutApp`: App version and info

**Wireframe:**

None

```
┌─────────────────────────────┐
│ ← Settings                  │
├─────────────────────────────┤
│                             │
│ Account                     │
│ ┌─────────────────────────┐ │
│ │ Profile                >│ │
│ └─────────────────────────┘ │
│ ┌─────────────────────────┐ │
│ │ Notifications          >│ │
│ └─────────────────────────┘ │
│ ┌─────────────────────────┐ │
│ │ Privacy                >│ │
│ └─────────────────────────┘ │
│ ┌─────────────────────────┐ │
│ │ Premium Plan           >│ │
│ └─────────────────────────┘ │
│                             │
│ Calendars                   │
│ ┌─────────────────────────┐ │
│ │ Manage Calendars       >│ │
│ └─────────────────────────┘ │
│ ┌─────────────────────────┐ │
│ │ Sync Settings          >│ │
│ └─────────────────────────┘ │
│                             │
│ App Settings                │
│ ┌─────────────────────────┐ │
│ │ Time Zone: US Eastern  >│ │
│ └─────────────────────────┘ │
│ ┌─────────────────────────┐ │
│ │ Theme: Light           >│ │
│ └─────────────────────────┘ │
│ ┌─────────────────────────┐ │
│ │ Language: English      >│ │
│ └─────────────────────────┘ │
│                             │
│ Help & Support              │
│ ┌─────────────────────────┐ │
│ │ Help Center            >│ │
│ └─────────────────────────┘ │
│ ┌─────────────────────────┐ │
│ │ Contact Support        >│ │
│ └─────────────────────────┘ │
│                             │
│ Log Out                     │
└─────────────────────────────┘
```

## 6.5 User Flows

### 6.5.1 New User Registration Flow

1. User downloads app or visits website
2. User clicks "Sign Up" on welcome screen
3. User enters email, password, and name
4. User verifies email (link sent to inbox)
5. User connects Google Calendar or Apple Calendar
    - If skipped, shown empty state with connect prompt
6. User selects which calendars to import
7. User sets up basic preferences (time zone, work hours)

8. User is presented with dashboard and guided tour

### 6.5.2 Link Creation Flow

1. User clicks "Create Link" button on dashboard
2. User enters link name (e.g., "Friends" or "Work Team")
3. User configures visibility settings
   - Select which calendars to include
   - Set detail level (none, minimal, full)
   - Toggle event labels on/off
   - Toggle negotiable status visibility
4. User adds optional custom message
5. User configures request settings (if allowing requests)
6. User previews how link will appear to recipients
7. User saves link and receives shareable URL
8. User shares link via preferred channel

### 6.5.3 Time Request Flow (Requester)

1. Requester receives and opens availability link
2. Requester browses recipient's availability
3. Requester selects date from mini-calendar
4. Requester enters preferred time in text field (e.g., "around 3pm")
5. System shows three nearest available options
6. Requester selects preferred option
7. Requester fills out request form
   - Name
   - Email
   - Purpose
   - Optional message
   - Optional urgency
8. Requester submits request
9. Requester sees confirmation screen with next steps

### 6.5.4 Time Request Flow (Recipient)

1. Recipient receives notification of new time request
2. Recipient views request details in Requests tab
3. Recipient has three options:
   - Accept (automatically adds to calendar if integrated)
   - Propose alternative (opens time selection)
   - Decline (with optional reason)
4. If accepting, recipient can add note and specify location
5. If proposing alternative, recipient selects new options
6. If declining, recipient can provide reason

7. Requester is notified of recipient's response
8. Request is moved to "Completed" tab with outcome

#### 6.5.5 Protected Time Creation Flow

1. User navigates to Calendar tab
2. User clicks "Add Protected Time" button
3. User enters details for protected time block
    - Title
    - Date or date range
    - Start and end times
    - Recurrence pattern (if recurring)
4. User configures visibility settings
    - Default visibility (busy, free, negotiable)
    - Per-audience overrides
5. User sets priority and movability
6. User saves protected time block
7. Protected time appears on calendar view
8. Protected time is reflected in availability links according to settings

---

# 7. Technical Architecture

## 7.1 Architecture Overview

SpareTime will follow a modern client-server architecture with native mobile clients and a cloud-based backend:

#### 7.1.1 High-Level Components

1. **Mobile Applications**

    - Native iOS application (Swift/SwiftUI)
    - Native Android application (Kotlin/Jetpack Compose)
2. **Backend Services**

    - RESTful API server (Node.js/Express)
    - Authentication service
    - Calendar integration service
    - Availability calculation engine
    - Link management service
    - Request processing service

3. **Database**
    - Primary database (MongoDB)
    - Caching layer (Redis)
    - File storage (AWS S3)
4. **External Integrations**
    - Google Calendar API
    - Apple Calendar API (via iOS)
    - Push notification services
    - Analytics service

### 7.1.2 System Architecture Diagram

None

```
┌─────────────────┐     ┌─────────────────┐
│  iOS App        │     │  Android App    │
│  (Swift/SwiftUI)│     │  (Kotlin/Compose)
└─────────┬───────┘     └─────────┬───────┘
          │                      │
          │                      │
          │     ┌────────────────┐
          │     │                │
          ├─────┤  API Gateway   │
          │     │                │
          │     └────────┬───────┘
          │              │
┌─────────┴──────────────┴────────┐
│                                 │
│      Backend Services           │
│                                 │
│  ┌───────┐  ┌─────────────┐      │
│  │Auth   │  │Calendar     │     │
│  │Service│  │Integration  │     │
│  └───────┘  └─────────────┘      │
│                                 │
│  ┌───────┐  ┌─────────────┐      │
│  │Link   │  │Availability │     │
│  │Service│  │Engine       │     │
│  └───────┘  └─────────────┘      │
│                                    │
│   ┌─────────┐   ┌──────────────┐    │
│   │Request  │   │Notification  │    │
│   │Service  │   │Service       │    │
│   └─────────┘   └──────────────┘    │
│                                    │
└────────────┬───────────────┬────────┘
             │               │
┌────────────┴──────────┐ ┌──┴────────────────┐
│                       │ │                   │
│   MongoDB             │ │   Redis Cache     │
│   Database            │ │                   │
│                       │ │                   │
└────────────┬──────────┘ └──┬────────────────┘
             │               │
             │        ┌──────┴────────────┐
             │        │                   │
             └────────┤  AWS S3 Storage   │
                      │                   │
                      └───────────────────┘
```

## 7.2 Backend Architecture

### 7.2.1 API Server

The backend will be built using Node.js with Express framework, providing a RESTful API for the mobile clients:

**Technology Stack:**

- Node.js (v18+)
- Express.js
- TypeScript for type safety
- JWT for authentication
- Mongoose for MongoDB ODM

**Key Services:**

1. **Authentication Service**
    - User registration and login
    - JWT token generation and validation
    - Social login integration
    - Password reset functionality
2. **Calendar Integration Service**
    - Google Calendar OAuth flow
    - Calendar data synchronization
    - Event processing and normalization
    - Sync state management
3. **Availability Engine**
    - Calendar data aggregation
    - Busy/free time calculation
    - Protected time application
    - Time slot recommendation algorithm
    - Natural language time parsing
4. **Link Management Service**
    - Link generation and validation
    - Audience-specific visibility rules
    - Link analytics tracking
    - Expiration handling
5. **Request Processing Service**
    - Time request creation and validation
    - Request notification handling
    - Request state management
    - Calendar event creation (if accepted)

### 7.2.2 Database Design

The primary database will be MongoDB, with collections structured to support the application's data model:

**Key Collections:**

1. **Users**
    - User account information
    - Authentication details
    - Profile data
    - Preferences

2. **Calendars**
    - Connected calendar sources
    - Sync metadata
    - Visibility settings
    - Color coding
3. **Events**
    - Calendar events from all sources
    - Recurring event patterns
    - Event metadata
    - Visibility status
4. **ProtectedTimes**
    - User-defined protected time blocks
    - Recurrence patterns
    - Visibility settings
    - Priority information
5. **Links**
    - Generated availability links
    - Configuration settings
    - Access statistics
    - Expiration data
6. **Requests**
    - Time requests from link viewers
    - Request status and history
    - Requester information
    - Response details
7. **Audiences**
    - Named groups of link viewers
    - Audience-specific settings
    - Member management
8. **Notifications**
    - User notifications
    - Delivery status
    - Read status
    - Action references

#### 7.2.3 Caching Strategy

Redis will be used as a caching layer to improve performance:

**Cache Types:**

1. **API Response Cache**
    - Frequently accessed endpoints
    - Short TTL (5-15 minutes)
    - Cache invalidation on data updates
2. **Availability Cache**
    - Pre-calculated availability for active links
    - Invalidated on calendar sync or settings change
    - Daily refresh for active links
3. **User Session Cache**
    - Active user sessions
    - Authentication state
    - Recently accessed links
4. **Rate Limiting**
    - API request limiting
    - Abuse prevention
    - Throttling for heavy operations

## 7.3 Mobile Architecture

### 7.3.1 iOS Application

**Technology Stack:**

- Swift 5.9+
- SwiftUI for UI components
- Combine for reactive programming
- Core Data for local storage
- URLSession for networking
- EventKit for local calendar access

**Architecture Pattern:**

- MVVM (Model-View-ViewModel)
- Repository pattern for data access
- Service-oriented approach for business logic

**Key Components:**

1. **UI Layer**
   - SwiftUI views
   - View models
   - UI state management
   - Navigation coordinator
2. **Domain Layer**
   - Business logic
   - Use cases
   - Domain models
   - Validators
3. **Data Layer**
   - API clients
   - Repository implementations
   - Local storage (Core Data)
   - Calendar integration
4. **Core Layer**
   - Common utilities
   - Extensions
   - Constants
   - Analytics

**7.3.2 Android Application**

**Technology Stack:**

- Kotlin 1.8+
- Jetpack Compose for UI
- Kotlin Coroutines for async operations
- Flow for reactive streams
- Room for local storage
- Retrofit for networking
- Calendar Provider for calendar access

**Architecture Pattern:**

- MVVM with Clean Architecture
- Repository pattern for data access
- Use case pattern for business logic

**Key Components:**

1. **UI Layer**
    - Compose UI components
    - ViewModels
    - UI state management
    - Navigation graph
2. **Domain Layer**
    - Use cases
    - Domain models
    - Business rules
    - Interface definitions
3. **Data Layer**
    - Repository implementations
    - Remote data sources (API)
    - Local data sources (Room)
    - Data mappers
4. **Core Layer**
    - Common utilities
    - Extensions
    - Constants
    - DI configuration

## 7.4 External Integrations

### 7.4.1 Google Calendar API

The application will integrate with Google Calendar using their official API:

**Integration Flow:**

1. User authenticates with Google OAuth
2. Application receives OAuth tokens
3. Tokens are stored securely
4. Application requests calendar list
5. User selects calendars to sync
6. Application fetches events with incremental sync
7. Events are processed and stored locally
8. Periodic sync maintains data freshness

**API Endpoints Used:**

- `calendar.list` - Get list of user's calendars

- `events.list` - Get events from a calendar
- `events.watch` - Set up push notifications for changes
- `freebusy.query` - Check availability efficiently

#### 7.4.2 Push Notification Services

The application will use platform-specific push notification services:

**iOS (APNs):**

1. Register device with APNs
2. Get device token
3. Send token to backend
4. Backend sends push via APNs
5. iOS app processes incoming notifications

**Android (FCM):**

1. Register device with FCM
2. Get FCM token
3. Send token to backend
4. Backend sends push via FCM
5. Android app processes incoming notifications

**Notification Types:**

- New time requests
- Request responses
- Link viewed notifications
- Calendar sync completed
- System announcements

---

# 8. Data Models

## 8.1 Core Data Entities

### 8.1.1 User

```json
{
  "id": "string",
  "email": "string",
  "username": "string",
  "passwordHash": "string",
  "displayName": "string",
  "avatar": "string (URL)",
  "timezone": "string",
  "preferences": {
    "defaultProtectedTimes": [
      {
        "dayOfWeek": "number (0-6)",
        "startTime": "string (HH:MM)",
        "endTime": "string (HH:MM)"
      }
    ],
    "defaultLinkSettings": {
      "showLabels": "boolean",
      "showNegotiable": "boolean",
      "detailLevel": "enum (none, minimal, full)"
    },
    "notifications": {
      "email": "boolean",
      "push": "boolean",
      "requestTypes": ["array of enum (all, incoming, outgoing,
responses)"]
    },
    "workHours": {
      "start": "string (HH:MM)",
      "end": "string (HH:MM)",
      "workDays": ["array of numbers (0-6)"]
    }
  },
  "subscription": {
    "level": "enum (free, premium)",
    "expiresAt": "date",
    "features": ["array of strings"]
  },
  "createdAt": "date",
  "updatedAt": "date"
}
```

### 8.1.2 Calendar

JSON
```json
{
  "id": "string",
  "userId": "string",
  "source": "enum (google, apple, manual)",
  "sourceId": "string",
  "name": "string",
  "color": "string (hex)",
  "isActive": "boolean",
  "lastSynced": "date",
  "syncToken": "string",
  "visibility": {
    "default": "enum (busy, free, hidden)",
    "overrides": [
      {
        "linkId": "string",
        "visibility": "enum (busy, free, hidden)"
      }
    ]
  },
  "authData": {
    "accessToken": "string (encrypted)",
    "refreshToken": "string (encrypted)",
    "expiresAt": "date"
  },
  "createdAt": "date",
  "updatedAt": "date"
}
```

### 8.1.3 Event

```json
{
  "id": "string",
  "calendarId": "string",
  "sourceId": "string",
  "title": "string",
  "description": "string",
  "location": "string",
  "startTime": "date",
  "endTime": "date",
  "isAllDay": "boolean",
  "recurrence": {
    "frequency": "enum (daily, weekly, monthly, yearly, custom)",
    "interval": "number",
    "byDay": ["array of strings (MO, TU, WE, TH, FR, SA, SU)"],
    "byMonthDay": ["array of numbers"],
    "endDate": "date",
    "count": "number",
    "exceptionDates": ["array of dates"]
  },
  "visibility": "enum (public, private)",
  "status": "enum (confirmed, tentative, cancelled)",
  "lastUpdated": "date"
}
```

### 8.1.4 ProtectedTime

```json
{
  "id": "string",
  "userId": "string",
  "title": "string",
  "startTime": "date",
  "endTime": "date",
  "recurrence": {
    "frequency": "enum (daily, weekly, monthly, yearly, custom)",
    "interval": "number",
    "byDay": ["array of strings (MO, TU, WE, TH, FR, SA, SU)"],
    "byMonthDay": ["array of numbers"],
    "endDate": "date",
    "count": "number",
    "exceptionDates": ["array of dates"]
  },
  "visibility": {
    "default": "enum (busy, free, negotiable)",
    "overrides": [
      {
        "audienceId": "string",
        "visibility": "enum (busy, free, negotiable)"
      }
    ]
  },
  "priority": "enum (low, medium, high)",
  "isMovable": "boolean",
  "createdAt": "date",
  "updatedAt": "date"
}
```

**8.1.5 Link**

```json
{
  "id": "string",
  "userId": "string",
  "name": "string",
  "slug": "string",
  "fullUrl": "string",
  "expiresAt": "date",
  "settings": {
    "showLabels": "boolean",
    "showNegotiable": "boolean",
    "detailLevel": "enum (none, minimal, full)",
    "calendarVisibility": {
      "calendarId": "enum (busy, free, hidden)"
    },
    "customMessage": "string",
    "allowRequests": "boolean",
    "defaultDuration": "number (minutes)",
    "timeConstraints": {
      "earliestTime": "string (HH:MM)",
      "latestTime": "string (HH:MM)",
      "daysInAdvance": "number"
    }
  },
  "audience": {
    "id": "string",
    "name": "string"
  },
  "stats": {
    "views": "number",
    "requests": "number",
    "lastViewed": "date"
  },
  "createdAt": "date",
  "updatedAt": "date"
}
```

**8.1.6 Request**

```json
{
  "id": "string",
  "linkId": "string",
  "fromUser": {
    "name": "string",
    "email": "string",
    "message": "string"
  },
  "toUserId": "string",
  "startTime": "date",
  "endTime": "date",
  "purpose": "string",
  "urgency": "enum (low, normal, high)",
  "status": "enum (pending, approved, rejected, alternative)",
  "alternativeTime": {
    "startTime": "date",
    "endTime": "date"
  },
  "responseMessage": "string",
  "createdAt": "date",
  "updatedAt": "date"
}
```

#### 8.1.7 Audience

```json
{
  "id": "string",
  "userId": "string",
  "name": "string",
  "description": "string",
  "links": ["array of link ids"],
  "settings": {
    "defaultVisibility": "enum (busy, free, negotiable)",
    "calendarVisibility": {
      "calendarId": "enum (busy, free, hidden)"
    },
    "requestPermissions": {
      "canRequest": "boolean",
      "defaultDuration": "number (minutes)"
    }
  },
  "createdAt": "date",
  "updatedAt": "date"
}
```

#### 8.1.8 Notification

```json
{
  "id": "string",
  "userId": "string",
  "type": "enum (request, response, view, system)",
  "title": "string",
  "message": "string",
  "relatedId": "string",
  "relatedType": "enum (request, link)",
  "read": "boolean",
  "delivered": "boolean",
  "action": {
    "type": "enum (view, respond, settings)",
    "data": "object"
  },
  "createdAt": "date"
}
```

## 8.2 Database Indexes

To ensure optimal performance, the following database indexes should be created:

#### 8.2.1 User Collection

- `email`: Unique index
- `username`: Unique index

#### 8.2.2 Calendar Collection

- `userId`: Index
- `userId_sourceId`: Compound unique index

#### 8.2.3 Event Collection

- `calendarId`: Index
- `calendarId_sourceId`: Compound unique index
- `startTime_endTime`: Compound index for date range queries

#### 8.2.4 ProtectedTime Collection

- `userId`: Index
- `userId_startTime_endTime`: Compound index for availability queries

#### 8.2.5 Link Collection

- `userId`: Index
- `slug`: Unique index
- `expiresAt`: Index for expiration cleanup

#### 8.2.6 Request Collection

- `linkId`: Index
- `toUserId`: Index
- `status`: Index
- `startTime`: Index for date-based filtering

#### 8.2.7 Audience Collection

- `userId`: Index

#### 8.2.8 Notification Collection

- `userId`: Index
- `userId_read`: Compound index for unread queries

### 8.3 Database Relationships

The application will use MongoDB, which doesn't enforce relationships like a relational database. However, the following conceptual relationships will be maintained:

1. **User to Calendars**: One-to-many
    - User has multiple calendars
    - Calendars reference user by userId
2. **Calendar to Events**: One-to-many
    - Calendar has multiple events
    - Events reference calendar by calendarId
3. **User to ProtectedTimes**: One-to-many
    - User has multiple protected time blocks
    - Protected times reference user by userId
4. **User to Links**: One-to-many

    - User has multiple availability links
    - Links reference user by userId
5. **Link to Requests**: One-to-many
    - Link has multiple time requests
    - Requests reference link by linkId
6. **User to Audiences**: One-to-many
    - User has multiple audiences
    - Audiences reference user by userId
7. **User to Notifications**: One-to-many
    - User has multiple notifications
    - Notifications reference user by userId

---

# 9. API Specifications

## 9.1 API Overview

The SpareTime API will follow RESTful principles with the following characteristics:

- Base URL: `https://api.sparetime.me/v1`
- Authentication: JWT tokens in Authorization header
- Response format: JSON
- Error handling: Consistent error objects with status codes
- Rate limiting: Implemented for all endpoints
- Pagination: Used for list endpoints

## 9.2 Authentication Endpoints

### 9.2.1 Register New User

**Endpoint:** `POST /auth/register`

**Request:**

```json
{
  "email": "user@example.com",
  "password": "securePassword123",
  "username": "username",
  "displayName": "User's Name",
  "timezone": "America/New_York"
}
```

**Response:** `201 Created`

```json
{
  "token": "jwt_token_here",
  "user": {
    "id": "user_id",
    "email": "user@example.com",
    "username": "username",
    "displayName": "User's Name"
  }
}
```

#### 9.2.2 Login

**Endpoint:** `POST /auth/login`

**Request:**

```json
{
  "email": "user@example.com",
  "password": "securePassword123"
}
```

**Response:** `200 OK`

```json
{
  "token": "jwt_token_here",
  "user": {
    "id": "user_id",
    "email": "user@example.com",
    "username": "username",
    "displayName": "User's Name"
  }
}
```

#### 9.2.3 Refresh Token

**Endpoint:** `POST /auth/refresh`

**Request:**

```json
{
  "refreshToken": "refresh_token_here"
}
```

**Response:** `200 OK`

```json
{
  "token": "new_jwt_token",
  "refreshToken": "new_refresh_token"
}
```

#### 9.2.4 Google OAuth

**Endpoint:** `POST /auth/google`

**Request:**

```JSON
{
  "code": "google_auth_code",
  "redirectUri": "app_redirect_uri"
}
```

**Response:** `200 OK`

```JSON
{
  "token": "jwt_token_here",
  "user": {
    "id": "user_id",
    "email": "user@example.com",
    "username": "username",
    "displayName": "User's Name",
    "isNew": false
  }
}
```

### 9.3 Calendar Endpoints

#### 9.3.1 List Calendars

**Endpoint:** `GET /calendars`

**Headers:** `Authorization: Bearer {token}`

**Response:** `200 OK`

```JSON
{
  "data": [
    {
      "id": "calendar_id_1",
      "source": "google",
      "name": "Work Calendar",
      "color": "#4285F4",
      "isActive": true,
      "lastSynced": "2025-10-23T14:30:00Z",
      "visibility": {
        "default": "busy"
      }
    },
    {
      "id": "calendar_id_2",
      "source": "google",
      "name": "Personal Calendar",
      "color": "#34A853",
      "isActive": true,
      "lastSynced": "2025-10-23T14:30:00Z",
      "visibility": {
        "default": "hidden"
      }
    }
  ]
}
```

#### 9.3.2 Connect Google Calendar

**Endpoint:** `POST /calendars/connect/google`

**Headers:** `Authorization: Bearer {token}`

**Request:**

```json
{
  "code": "google_auth_code",
  "redirectUri": "app_redirect_uri"
}
```

**Response:** `201 Created`

```json
{
  "data": [
    {
      "id": "calendar_id_1",
      "source": "google",
      "name": "Work Calendar",
      "color": "#4285F4",
      "isActive": true,
      "lastSynced": "2025-10-23T14:30:00Z"
    },
    {
      "id": "calendar_id_2",
      "source": "google",
      "name": "Personal Calendar",
      "color": "#34A853",
      "isActive": true,
      "lastSynced": "2025-10-23T14:30:00Z"
    }
  ]
}
```

#### 9.3.3 Update Calendar Settings

**Endpoint:** `PUT /calendars/{id}`

**Headers:** `Authorization: Bearer {token}`

**Request:**

```json
{
  "name": "Updated Name",
  "color": "#EA4335",
  "isActive": true,
  "visibility": {
    "default": "busy"
  }
}
```

**Response:** `200 OK`

```JSON
{
  "data": {
    "id": "calendar_id_1",
    "source": "google",
    "name": "Updated Name",
    "color": "#EA4335",
    "isActive": true,
    "lastSynced": "2025-10-23T14:30:00Z",
    "visibility": {
      "default": "busy"
    }
  }
}
```

#### 9.3.4 Sync Calendar

**Endpoint:** `POST /calendars/{id}/sync`

**Headers:** `Authorization: Bearer {token}`

**Response:** `200 OK`

```JSON
{
  "success": true,
  "lastSynced": "2025-10-24T10:15:30Z",
  "eventCount": 42
}
```

### 9.4 Protected Time Endpoints

#### 9.4.1 Create Protected Time

**Endpoint:** `POST /protected-times`

**Headers:** `Authorization: Bearer {token}`

**Request:**

```JSON
{
  "title": "Focus Time",
  "startTime": "2025-10-25T13:00:00Z",
  "endTime": "2025-10-25T15:00:00Z",
  "recurrence": {
    "frequency": "weekly",
    "byDay": ["MO", "WE", "FR"],
    "endDate": "2025-12-31T23:59:59Z"
  },
  "visibility": {
    "default": "busy",
    "overrides": [
      {
        "audienceId": "audience_id_1",
        "visibility": "negotiable"
      }
    ]
  },
  "priority": "high",
  "isMovable": false
}
```

**Response:** `201 Created`

```JSON
{
  "data": {
    "id": "protected_time_id",
    "userId": "user_id",
    "title": "Focus Time",
    "startTime": "2025-10-25T13:00:00Z",
    "endTime": "2025-10-25T15:00:00Z",
    "recurrence": {
      "frequency": "weekly",
      "byDay": ["MO", "WE", "FR"],
      "endDate": "2025-12-31T23:59:59Z"
    },
    "visibility": {
      "default": "busy",
      "overrides": [
        {
          "audienceId": "audience_id_1",
          "visibility": "negotiable"
        }
      ]
    },
    "priority": "high",
    "isMovable": false,
    "createdAt": "2025-10-24T10:30:00Z",
    "updatedAt": "2025-10-24T10:30:00Z"
  }
}
```

#### 9.4.2 List Protected Times

**Endpoint:** `GET /protected-times?startDate=2025-10-01&endDate=2025-10-31`

**Headers:** `Authorization: Bearer {token}`

**Response:** `200 OK`

```json
{
  "data": [
    {
      "id": "protected_time_id_1",
      "title": "Focus Time",
      "startTime": "2025-10-25T13:00:00Z",
      "endTime": "2025-10-25T15:00:00Z",
      "recurrence": {
        "frequency": "weekly",
        "byDay": ["MO", "WE", "FR"]
      },
      "visibility": {
        "default": "busy"
      },
      "priority": "high"
    },
    {
      "id": "protected_time_id_2",
      "title": "Family Time",
      "startTime": "2025-10-26T18:00:00Z",
      "endTime": "2025-10-26T21:00:00Z",
      "recurrence": {
        "frequency": "weekly",
        "byDay": ["SU"]
      },
      "visibility": {
        "default": "busy"
      },
      "priority": "high"
    }
  ]
}
```

#### 9.4.3 Update Protected Time

**Endpoint:** `PUT /protected-times/{id}`

**Headers:** `Authorization: Bearer {token}`

**Request:**

```json
{
  "title": "Deep Work",
  "startTime": "2025-10-25T14:00:00Z",
  "endTime": "2025-10-25T17:00:00Z",
  "priority": "medium"
}
```

**Response:** `200 OK`

```json
{
  "data": {
    "id": "protected_time_id",
    "title": "Deep Work",
    "startTime": "2025-10-25T14:00:00Z",
    "endTime": "2025-10-25T17:00:00Z",
    "recurrence": {
      "frequency": "weekly",
      "byDay": ["MO", "WE", "FR"]
    },
    "visibility": {
      "default": "busy"
    },
    "priority": "medium",
    "updatedAt": "2025-10-24T11:15:00Z"
  }
}
```

#### 9.4.4 Delete Protected Time

**Endpoint:** `DELETE /protected-times/{id}`

**Headers:** `Authorization: Bearer {token}`

**Response:** `204 No Content`

## 9.5 Link Endpoints

### 9.5.1 Create Link

**Endpoint:** `POST /links`

**Headers:** `Authorization: Bearer {token}`

**Request:**

```json
{
  "name": "Work Team",
  "settings": {
    "showLabels": false,
    "showNegotiable": true,
    "detailLevel": "minimal",
    "calendarVisibility": {
      "calendar_id_1": "busy",
      "calendar_id_2": "hidden"
    },
    "customMessage": "My availability for work meetings",
    "allowRequests": true,
    "defaultDuration": 30,
    "timeConstraints": {
      "earliestTime": "09:00",
      "latestTime": "17:00",
      "daysInAdvance": 14
    }
  },
  "audienceId": "audience_id_1",
  "expiresAt": "2026-01-01T00:00:00Z"
}
```

**Response:** `201 Created`

JSON

```json
{
  "data": {
    "id": "link_id",
    "userId": "user_id",
    "name": "Work Team",
    "slug": "work-team-xyz123",
    "fullUrl": "https://sparetime.me/username/work-team-xyz123",
    "expiresAt": "2026-01-01T00:00:00Z",
    "settings": {
      "showLabels": false,
      "showNegotiable": true,
      "detailLevel": "minimal",
      "calendarVisibility": {
        "calendar_id_1": "busy",
        "calendar_id_2": "hidden"
      },
      "customMessage": "My availability for work meetings",
      "allowRequests": true,
      "defaultDuration": 30,
      "timeConstraints": {
        "earliestTime": "09:00",
        "latestTime": "17:00",
        "daysInAdvance": 14
      }
    },
    "audience": {
      "id": "audience_id_1",
      "name": "Work Team"
    },
    "stats": {
      "views": 0,
      "requests": 0
    },
    "createdAt": "2025-10-24T12:00:00Z",
    "updatedAt": "2025-10-24T12:00:00Z"
  }
}
```

#### 9.5.2 List Links

**Endpoint:** `GET /links`

**Headers:** `Authorization: Bearer {token}`

**Response:** `200 OK`

```json
{
  "data": [
    {
      "id": "link_id_1",
      "name": "Work Team",
      "slug": "work-team-xyz123",
      "fullUrl":
"https://sparetime.me/username/work-team-xyz123",
      "expiresAt": "2026-01-01T00:00:00Z",
      "settings": {
        "showLabels": false,
        "detailLevel": "minimal",
        "allowRequests": true
      },
      "stats": {
        "views": 12,
        "requests": 3,
        "lastViewed": "2025-10-23T16:45:00Z"
      }
    },
    {
      "id": "link_id_2",
      "name": "Friends",
      "slug": "friends-abc456",
      "fullUrl": "https://sparetime.me/username/friends-abc456",
      "settings": {
        "showLabels": true,
        "detailLevel": "full",
        "allowRequests": true
      },
      "stats": {
        "views": 8,
        "requests": 2,
        "lastViewed": "2025-10-22T20:15:00Z"
      }
    }
  ]
}
```

**9.5.3 Get Link Details**

**Endpoint:** `GET /links/{id}`

**Headers:** `Authorization: Bearer {token}`

**Response:** `200 OK`

```json
{
  "data": {
    "id": "link_id",
    "userId": "user_id",
    "name": "Work Team",
    "slug": "work-team-xyz123",
    "fullUrl": "https://sparetime.me/username/work-team-xyz123",
    "expiresAt": "2026-01-01T00:00:00Z",
    "settings": {
      "showLabels": false,
      "showNegotiable": true,
      "detailLevel": "minimal",
      "calendarVisibility": {
        "calendar_id_1": "busy",
        "calendar_id_2": "hidden"
      },
      "customMessage": "My availability for work meetings",
      "allowRequests": true,
      "defaultDuration": 30,
      "timeConstraints": {
        "earliestTime": "09:00",
        "latestTime": "17:00",
        "daysInAdvance": 14
      }
    },
    "audience": {
      "id": "audience_id_1",
      "name": "Work Team"
    },
    "stats": {
      "views": 12,
      "requests": 3,
      "lastViewed": "2025-10-23T16:45:00Z"
    },
    "createdAt": "2025-10-24T12:00:00Z",
    "updatedAt": "2025-10-24T12:00:00Z"
  }
}
```

#### 9.5.4 Update Link

**Endpoint:** `PUT /links/{id}`

**Headers:** `Authorization: Bearer {token}`

**Request:**

```JSON
{
  "name": "Updated Name",
  "settings": {
    "showLabels": true,
    "customMessage": "Updated message"
  },
  "expiresAt": null
}
```

**Response:** `200 OK`

```json
{
  "data": {
    "id": "link_id",
    "name": "Updated Name",
    "settings": {
      "showLabels": true,
      "showNegotiable": true,
      "detailLevel": "minimal",
      "calendarVisibility": {
        "calendar_id_1": "busy",
        "calendar_id_2": "hidden"
      },
      "customMessage": "Updated message",
      "allowRequests": true,
      "defaultDuration": 30,
      "timeConstraints": {
        "earliestTime": "09:00",
        "latestTime": "17:00",
        "daysInAdvance": 14
      }
    },
    "expiresAt": null,
    "updatedAt": "2025-10-24T14:30:00Z"
  }
}
```

#### 9.5.5 Delete Link

**Endpoint:** `DELETE /links/{id}`

**Headers:** `Authorization: Bearer {token}`

**Response:** `204 No Content`

### 9.6 Availability Endpoints

#### 9.6.1 Get Public Availability

**Endpoint:** `GET /public/{username}/{slug}/availability?date=2025-10-25`

**Response:** `200 OK`

```json
{
  "data": {
    "date": "2025-10-25",
    "timezone": "America/New_York",
    "owner": {
      "displayName": "User's Name",
      "message": "My availability for work meetings"
    },
    "busyBlocks": [
      {
        "start": "2025-10-25T09:00:00-04:00",
        "end": "2025-10-25T10:30:00-04:00",
        "isNegotiable": false
      },
      {
        "start": "2025-10-25T12:00:00-04:00",
        "end": "2025-10-25T13:00:00-04:00",
        "isNegotiable": false
      },
      {
        "start": "2025-10-25T14:00:00-04:00",
        "end": "2025-10-25T17:00:00-04:00",
        "isNegotiable": true
      }
    ]
  }
}
```

#### 9.6.2 Find Nearest Time Slots

**Endpoint:** `GET /public/{username}/{slug}/nearest-times?date=2025-10-25&preferredTime=15:00&duration=30`

**Response:** `200 OK`

```json
{
  "data": {
    "date": "2025-10-25",
    "timezone": "America/New_York",
    "preferredTime": "15:00",
    "duration": 30,
    "options": [
      {
        "start": "2025-10-25T13:15:00-04:00",
        "end": "2025-10-25T13:45:00-04:00",
        "isNegotiable": false
      },
      {
        "start": "2025-10-25T17:15:00-04:00",
        "end": "2025-10-25T17:45:00-04:00",
        "isNegotiable": false
      },
      {
        "start": "2025-10-25T14:30:00-04:00",
        "end": "2025-10-25T15:00:00-04:00",
        "isNegotiable": true
      }
    ]
  }
}
```

## 9.7 Request Endpoints

### 9.7.1 Create Time Request

**Endpoint:** `POST /public/{username}/{slug}/request`

**Request:**

```JSON
{
  "startTime": "2025-10-25T17:15:00-04:00",
  "endTime": "2025-10-25T17:45:00-04:00",
  "name": "Requester Name",
  "email": "requester@example.com",
  "purpose": "Quick catch-up",
  "message": "Would love to discuss the new project",
  "urgency": "normal"
}
```

**Response:** `201 Created`

```JSON
{
  "data": {
    "id": "request_id",
    "status": "pending",
    "message": "Your request has been sent. You'll be notified
when User's Name responds."
  }
}
```

**9.7.2 List Requests**

**Endpoint:** `GET /requests?status=pending`

**Headers:** `Authorization: Bearer {token}`

**Response:** `200 OK`

```JSON
{
  "data": [
    {
      "id": "request_id_1",
      "linkId": "link_id_1",
      "linkName": "Work Team",
      "fromUser": {
        "name": "Requester 1",
        "email": "requester1@example.com"
      },
      "startTime": "2025-10-25T17:15:00-04:00",
      "endTime": "2025-10-25T17:45:00-04:00",
      "purpose": "Quick catch-up",
      "message": "Would love to discuss the new project",
      "urgency": "normal",
      "status": "pending",
      "createdAt": "2025-10-23T14:30:00Z"
    },
    {
      "id": "request_id_2",
      "linkId": "link_id_2",
      "linkName": "Friends",
      "fromUser": {
        "name": "Requester 2",
        "email": "requester2@example.com"
      },
      "startTime": "2025-10-26T12:00:00-04:00",
      "endTime": "2025-10-26T13:00:00-04:00",
      "purpose": "Coffee",
      "urgency": "low",
      "status": "pending",
      "createdAt": "2025-10-24T09:15:00Z"
    }
  ]
}
```

#### 9.7.3 Update Request Status

**Endpoint:** `PUT /requests/{id}/status`

**Headers:** `Authorization: Bearer {token}`

**Request:**

JSON
```
{
  "status": "approved",
  "message": "Looking forward to our meeting!"
}
```

**Response:** `200 OK`

JSON
```
{
  "data": {
    "id": "request_id",
    "status": "approved",
    "responseMessage": "Looking forward to our meeting!",
    "updatedAt": "2025-10-24T15:30:00Z"
  }
}
```

**9.7.4 Propose Alternative Time**

**Endpoint:** `PUT /requests/{id}/propose`

**Headers:** `Authorization: Bearer {token}`

**Request:**

```JSON
{
  "startTime": "2025-10-26T15:00:00-04:00",
  "endTime": "2025-10-26T15:30:00-04:00",
  "message": "I can't make your requested time. How about this
alternative?"
}
```

**Response:** `200 OK`

```JSON
{
  "data": {
    "id": "request_id",
    "status": "alternative",
    "alternativeTime": {
      "startTime": "2025-10-26T15:00:00-04:00",
      "endTime": "2025-10-26T15:30:00-04:00"
    },
    "responseMessage": "I can't make your requested time. How
about this alternative?",
    "updatedAt": "2025-10-24T15:45:00Z"
    }
}
```

---

# 10. Mobile Development Guidelines

## 10.1 Common Development Approach

### 10.1.1 Development Principles

1. **Offline-First Approach**
    - Cache data for offline use
    - Sync when connection is available
    - Handle offline state gracefully
2. **Responsive Design**
    - Support multiple screen sizes
    - Adapt to orientation changes
    - Consider split-screen (iPad/tablet)
3. **Performance Optimization**
    - Minimize network requests
    - Efficiently use system resources
    - Optimize for battery life
4. **Security Best Practices**
    - Secure storage of sensitive data
    - Proper authentication flow
    - Input validation and sanitization
5. **Accessibility**
    - Support screen readers
    - Provide adequate contrast
    - Ensure keyboard/alternative navigation

### 10.1.2 Code Organization

1. **Feature-Based Structure**
    - Organize code by feature/module

    - Maintain clear boundaries
    - Avoid cross-module dependencies
2. **Clean Architecture**
    - Separation of concerns
    - Dependency inversion
    - Testable components
3. **Shared Utilities**
    - Common helpers
    - Reusable components
    - Platform-specific abstractions
4. **Documentation**
    - Code comments for complex logic
    - README files for modules
    - Architecture diagrams

## 10.2 iOS Development

### 10.2.1 Project Setup

1. **Xcode Configuration**
    - iOS Deployment Target: iOS 15.0+
    - Swift version: 5.9+
    - Build configuration: Debug and Release
    - Code signing setup
2. **Dependencies Management**
    - Swift Package Manager for dependencies
    - Minimize third-party libraries
    - Document all dependencies
3. **Project Structure**

```
None
SpareTime/
├── App/
│   ├── AppDelegate.swift
│   ├── SceneDelegate.swift
│   └── SpareTimeApp.swift
```

```
├── Features/
│   ├── Auth/
│   ├── Calendar/
│   ├── Links/
│   ├── Requests/
│   ├── Settings/
│   └── ProtectedTime/
├── Core/
│   ├── Models/
│   ├── Services/
│   ├── Repositories/
│   └── Utils/
├── UI/
│   ├── Components/
│   ├── Styles/
│   └── Extensions/
└── Resources/
    ├── Assets.xcassets/
    ├── Localizable.strings
    └── Info.plist
```

#### 10.2.2 UI Implementation

1. **SwiftUI Components**
    - Create reusable view components
    - Implement design system components
    - Use ViewModifiers for consistent styling
2. **SwiftUI Views Implementation**

```
None
struct TimeSelectionView: View {
    @ObservedObject var viewModel: TimeSelectionViewModel
    @State private var preferredTime: String = ""
    var body: some View {
        VStack(spacing: 16) {
            // Date pill and time input
            HStack {
                DatePillView(
                    date: $viewModel.selectedDate,
                    onDateSelected: viewModel.onDateSelected
                )

                TextField("e.g., 7pm or 'after 3'", text:
$preferredTime)

.textFieldStyle(RoundedBorderTextFieldStyle())
                    .onSubmit {
                        viewModel.findNearestTimes(preferredTime:
preferredTime)
                    }
            }

            // Results
            if viewModel.isLoading {
                ProgressView()
            } else if viewModel.errorMessage != nil {
                ErrorView(message: viewModel.errorMessage ?? "")
            } else if !viewModel.timeOptions.isEmpty {
                TimeOptionsView(
                    options: viewModel.timeOptions,
                    selectedOption: $viewModel.selectedOption
                )
            }

            // Request button
            if viewModel.selectedOption != nil {
                Button("Request This Time") {
                    viewModel.showRequestForm = true
                }
                .buttonStyle(PrimaryButtonStyle())
            }
        }
        .padding()
        .sheet(isPresented: $viewModel.showRequestForm) {
            RequestFormView(
                viewModel: RequestFormViewModel(
                    startTime: viewModel.selectedOption?.start ??
Date(),
                    endTime: viewModel.selectedOption?.end ??
Date(),
                    onSubmit: viewModel.submitRequest
                )
            )
        }
    }
}
```

3. **MVVM Pattern**

```
None
class TimeSelectionViewModel: ObservableObject {
    @Published var selectedDate: Date = Date()
    @Published var timeOptions: [TimeOption] = []
    @Published var selectedOption: TimeOption?
    @Published var isLoading = false
    @Published var errorMessage: String?
    @Published var showRequestForm = false

    private let apiService: APIService
    private let linkId: String

    init(apiService: APIService, linkId: String) {
        self.apiService = apiService
        self.linkId = linkId
    }

    func onDateSelected(date: Date) {
        selectedDate = date
        // Reset other state
    }

    func findNearestTimes(preferredTime: String) {
        guard !preferredTime.isEmpty else { return }

        isLoading = true
        errorMessage = nil

        let dateFormatter = DateFormatter()
        dateFormatter.dateFormat = "yyyy-MM-dd"
        let dateString = dateFormatter.string(from: selectedDate)

        apiService.findNearestTimes(
            linkId: linkId,
            date: dateString,
            preferredTime: preferredTime
        ) { [weak self] result in
            DispatchQueue.main.async {
                self?.isLoading = false

                switch result {
                case .success(let options):
                    self?.timeOptions = options
                case .failure(let error):
                    self?.errorMessage =
error.localizedDescription
                }
            }
        }
    }

    func submitRequest(name: String, email: String, purpose:
String, message: String?) {
        guard let option = selectedOption else { return }

        // Implementation details
    }
}
```

### 10.2.3 Data and Network Layer

1. **API Service**

```
None
protocol APIServiceProtocol {
    func findNearestTimes(linkId: String, date: String,
preferredTime: String, completion: @escaping
(Result<[TimeOption], Error>) -> Void)
    func submitRequest(request: RequestModel, completion:
@escaping (Result<RequestResponse, Error>) -> Void)
}

class APIService: APIServiceProtocol {
    private let baseURL = URL(string:
"https://api.sparetime.me/v1")!
    private let session: URLSession

    init(session: URLSession = .shared) {
        self.session = session
    }

    func findNearestTimes(linkId: String, date: String,
preferredTime: String, completion: @escaping
(Result<[TimeOption], Error>) -> Void) {
        var components = URLComponents(url:
baseURL.appendingPathComponent("public/links/\(linkId)/nearest-ti
mes"), resolvingAgainstBaseURL: true)!
        components.queryItems = [
            URLQueryItem(name: "date", value: date),
            URLQueryItem(name: "preferredTime", value:
preferredTime)
        ]

        let request = URLRequest(url: components.url!)

        let task = session.dataTask(with: request) { data,
response, error in
            if let error = error {
                completion(.failure(error))
                return
            }

            guard let data = data else {
                completion(.failure(NSError(domain: "APIError",
code: 0, userInfo: [NSLocalizedDescriptionKey: "No data
received"])))
                return
            }

            do {
                let decoder = JSONDecoder()
                decoder.dateDecodingStrategy = .iso8601

                let response = try
decoder.decode(TimeOptionsResponse.self, from: data)
                completion(.success(response.data.options))
            } catch {
                completion(.failure(error))
            }
        }
        task.resume()
    }

    func submitRequest(request: RequestModel, completion:
@escaping (Result<RequestResponse, Error>) -> Void) {
        // Implementation details
    }
}
```

2. **Local Storage**

```
None
// Core Data model for Link
@objc(CDLink)
public class CDLink: NSManagedObject {
    @NSManaged public var id: String
    @NSManaged public var name: String
    @NSManaged public var slug: String
    @NSManaged public var fullUrl: String
    @NSManaged public var expiresAt: Date?
    @NSManaged public var createdAt: Date
    @NSManaged public var updatedAt: Date
    @NSManaged public var settings: NSData
}

// Repository pattern
protocol LinkRepository {
    func getLinks() -> AnyPublisher<[Link], Error>
    func getLink(id: String) -> AnyPublisher<Link?, Error>
    func saveLink(_ link: Link) -> AnyPublisher<Link, Error>
    func deleteLink(id: String) -> AnyPublisher<Void, Error>
}
```

```
class CoreDataLinkRepository: LinkRepository {
    private let coreDataStack: CoreDataStack

    init(coreDataStack: CoreDataStack) {
        self.coreDataStack = coreDataStack
    }

    func getLinks() -> AnyPublisher<[Link], Error> {
        Future<[Link], Error> { promise in
            self.coreDataStack.performBackgroundTask { context in
                let fetchRequest: NSFetchRequest<CDLink> =
CDLink.fetchRequest()
                fetchRequest.sortDescriptors =
[NSSortDescriptor(key: "updatedAt", ascending: false)]

                do {
                    let cdLinks = try context.fetch(fetchRequest)
                    let links = cdLinks.map {
self.mapToDomain($0) }
                    promise(.success(links))
                } catch {
                    promise(.failure(error))
                }
            }
        }
        .eraseToAnyPublisher()
    }

    // Other repository methods
}
```

## 10.3 Android Development

### 10.3.1 Project Setup

1. **Android Studio Configuration**
    - Minimum SDK: API 26 (Android 8.0)
    - Target SDK: API 33 (Android 13)
    - Kotlin version: 1.8+
    - Compose compiler version: 1.4+
2. **Gradle Configuration**

Kotlin
```kotlin
plugins {
    id("com.android.application")
    id("kotlin-android")
    id("kotlin-kapt")
    id("dagger.hilt.android.plugin")
}

android {
    compileSdk = 33

    defaultConfig {
        applicationId = "me.sparetime.app"
        minSdk = 26
        targetSdk = 33
        versionCode = 1
        versionName = "1.0.0"

        testInstrumentationRunner =
"androidx.test.runner.AndroidJUnitRunner"
    }

    buildFeatures {
        compose = true
    }

    composeOptions {
        kotlinCompilerExtensionVersion = "1.4.7"
    }
    buildTypes {
        getByName("release") {
            isMinifyEnabled = true

proguardFiles(getDefaultProguardFile("proguard-android-optimize.t
xt"), "proguard-rules.pro")
        }
    }

    compileOptions {
        sourceCompatibility = JavaVersion.VERSION_1_8
        targetCompatibility = JavaVersion.VERSION_1_8
    }

    kotlinOptions {
        jvmTarget = "1.8"
    }
}

dependencies {
    // Core libraries
    implementation("androidx.core:core-ktx:1.10.1")
    implementation("androidx.appcompat:appcompat:1.6.1")

implementation("androidx.lifecycle:lifecycle-runtime-ktx:2.6.1")

    // Compose
    implementation("androidx.compose.ui:ui:1.4.3")
    implementation("androidx.compose.material3:material3:1.1.1")

implementation("androidx.compose.ui:ui-tooling-preview:1.4.3")
    implementation("androidx.activity:activity-compose:1.7.2")

implementation("androidx.navigation:navigation-compose:2.6.0")
```

3.

```
    // Architecture Components

implementation("androidx.lifecycle:lifecycle-viewmodel-compose:2.
6.1")
    implementation("androidx.hilt:hilt-navigation-compose:1.0.0")

    // Networking
    implementation("com.squareup.retrofit2:retrofit:2.9.0")

implementation("com.squareup.retrofit2:converter-moshi:2.9.0")

implementation("com.squareup.okhttp3:logging-interceptor:4.11.0")

    // Local Storage
    implementation("androidx.room:room-runtime:2.5.2")
    implementation("androidx.room:room-ktx:2.5.2")
    kapt("androidx.room:room-compiler:2.5.2")

    // DI
    implementation("com.google.dagger:hilt-android:2.44.2")
    kapt("com.google.dagger:hilt-android-compiler:2.44.2")

    // Testing
    testImplementation("junit:junit:4.13.2")
    androidTestImplementation("androidx.test.ext:junit:1.1.5")

androidTestImplementation("androidx.test.espresso:espresso-core:3
.5.1")

androidTestImplementation("androidx.compose.ui:ui-test-junit4:1.4
.3")
    debugImplementation("androidx.compose.ui:ui-tooling:1.4.3")
}
```

3.

**Project Structure**

None

```
me.sparetime.app/
├── data/
│   ├── api/
│   ├── db/
│   ├── models/
│   └── repositories/
├── di/
│   └── modules/
├── domain/
│   ├── models/
│   ├── repositories/
│   └── usecases/
├── presentation/
│   ├── auth/
│   ├── calendar/
│   ├── links/
│   ├── requests/
│   ├── settings/
│   ├── components/
│   ├── theme/
│   └── utils/
└── SpareTimeApplication.kt
```

### 10.3.2 UI Implementation

1. **Composable Components**

Kotlin

```kotlin
@Composable
fun TimeSelectionView(
    viewModel: TimeSelectionViewModel = hiltViewModel(),
    onRequestSubmitted: () -> Unit
) {
    val uiState by viewModel.uiState.collectAsState()
    var preferredTime by remember { mutableStateOf("") }
    Column(
        modifier = Modifier
            .fillMaxWidth()
            .padding(16.dp),
        verticalArrangement = Arrangement.spacedBy(16.dp)
    ) {
        // Date and time input
        Row(
            modifier = Modifier.fillMaxWidth(),
            horizontalArrangement = Arrangement.spacedBy(8.dp)
        ) {
            DatePill(
                selectedDate = uiState.selectedDate,
                onDateSelected = viewModel::onDateSelected,
                modifier = Modifier.weight(0.4f)
            )

            OutlinedTextField(
                value = preferredTime,
                onValueChange = { preferredTime = it },
                label = { Text("e.g., 7pm or 'after 3'") },
                modifier = Modifier.weight(0.6f),
                singleLine = true,
                keyboardActions = KeyboardActions(
                    onDone = {
                        viewModel.findNearestTimes(preferredTime)
                    }
                )
            )
        }

        Button(
            onClick = { viewModel.findNearestTimes(preferredTime)
},
            modifier = Modifier.align(Alignment.End)
```

```kotlin
        ) {
            Text("Find Times")
        }

        // Results
        when {
            uiState.isLoading -> {
                CircularProgressIndicator(
                    modifier =
Modifier.align(Alignment.CenterHorizontally)
                )
            }
            uiState.errorMessage != null -> {
                Text(
                    text = uiState.errorMessage ?: "An error
occurred",
                    color = MaterialTheme.colorScheme.error
                )
            }
            uiState.timeOptions.isNotEmpty() -> {
                TimeOptionsSection(
                    options = uiState.timeOptions,
                    selectedOption = uiState.selectedOption,
                    onOptionSelected =
viewModel::selectTimeOption
                )
            }
        }

        // Request button
        if (uiState.selectedOption != null) {
            Button(
                onClick = { viewModel.showRequestForm() },
                modifier = Modifier.fillMaxWidth()
            ) {
                Text("Request This Time")
            }
        }
    }

    // Request form dialog
    if (uiState.showingRequestForm) {
        RequestFormDialog(
            timeOption = uiState.selectedOption!!,
            onDismiss = viewModel::hideRequestForm,
            onSubmit = { name, email, purpose, message ->
                viewModel.submitRequest(name, email, purpose,
message)
                onRequestSubmitted()
            }
        )
    }
}

@Composable
fun DatePill(
    selectedDate: LocalDate,
    onDateSelected: (LocalDate) -> Unit,
    modifier: Modifier = Modifier
) {
    val dateFormatter = DateTimeFormatter.ofPattern("MMM d")

    Surface(
        shape = RoundedCornerShape(24.dp),
        color =
MaterialTheme.colorScheme.surfaceVariant.copy(alpha = 0.3f),
        modifier = modifier.clickable {
            // Show date picker in real implementation
        }
    ) {
        Row(
            modifier = Modifier.padding(horizontal = 12.dp,
vertical = 8.dp),
            verticalAlignment = Alignment.CenterVertically,
            horizontalArrangement = Arrangement.Center
        ) {
            Text(
                text = selectedDate.format(dateFormatter),
                style = MaterialTheme.typography.bodyMedium,
                modifier = Modifier.padding(end = 4.dp)
            )
            Icon(
                imageVector = Icons.Default.ArrowDropDown,
                contentDescription = "Select Date"
            )
        }
    }
}

@Composable
fun TimeOptionsSection(
    options: List<TimeOption>,
    selectedOption: TimeOption?,
    onOptionSelected: (TimeOption) -> Unit
) {
    Column(
        verticalArrangement = Arrangement.spacedBy(8.dp)
    ) {
        Text(
            text = "Available Times",
            style = MaterialTheme.typography.titleMedium,
            fontWeight = FontWeight.Bold
        )

        options.forEach { option ->
            TimeOptionChip(
                option = option,
                isSelected = selectedOption?.id == option.id,
                onClick = { onOptionSelected(option) }
            )
        }
    }
}
```

2.
   **ViewModel**

Kotlin
```kotlin
@HiltViewModel
class TimeSelectionViewModel @Inject constructor(
    private val linkRepository: LinkRepository,
    private val timeRepository: TimeRepository,
    private val requestRepository: RequestRepository
) : ViewModel() {

    private val _uiState =
MutableStateFlow(TimeSelectionUiState())
    val uiState: StateFlow<TimeSelectionUiState> =
_uiState.asStateFlow()

    private val linkId: String = /* get from saved state handle
*/

    fun onDateSelected(date: LocalDate) {
        _uiState.update { it.copy(selectedDate = date,
timeOptions = emptyList(), selectedOption = null) }
    }

    fun findNearestTimes(preferredTime: String) {
        if (preferredTime.isBlank()) return
        _uiState.update { it.copy(isLoading = true, errorMessage
= null) }

        viewModelScope.launch {
            try {
                val dateStr =
uiState.value.selectedDate.format(DateTimeFormatter.ISO_DATE)
                val options =
timeRepository.findNearestTimes(linkId, dateStr, preferredTime)

                _uiState.update {
                    it.copy(
                        isLoading = false,
                        timeOptions = options,
                        errorMessage = if (options.isEmpty()) "No
available times found" else null
                    )
                }
            } catch (e: Exception) {
                _uiState.update { it.copy(isLoading = false,
errorMessage = e.message) }
            }
        }
    }

    fun selectTimeOption(option: TimeOption) {
        _uiState.update { it.copy(selectedOption = option) }
    }

    fun showRequestForm() {
        _uiState.update { it.copy(showingRequestForm = true) }
    }

    fun hideRequestForm() {
        _uiState.update { it.copy(showingRequestForm = false) }
    }
    fun submitRequest(name: String, email: String, purpose:
String, message: String?) {
        val option = uiState.value.selectedOption ?: return

        viewModelScope.launch {
            try {
                val request = RequestModel(
                    linkId = linkId,
                    startTime = option.start,
                    endTime = option.end,
                    name = name,
                    email = email,
                    purpose = purpose,
                    message = message
                )

                val result =
requestRepository.submitRequest(request)
                _uiState.update {
                    it.copy(
                        requestSubmitted = true,
                        showingRequestForm = false
                    )
                }
            } catch (e: Exception) {
                _uiState.update { it.copy(errorMessage =
e.message) }
            }
        }
    }
}

data class TimeSelectionUiState(
    val selectedDate: LocalDate = LocalDate.now(),
    val timeOptions: List<TimeOption> = emptyList(),
    val selectedOption: TimeOption? = null,
    val isLoading: Boolean = false,
    val errorMessage: String? = null,
    val showingRequestForm: Boolean = false,
    val requestSubmitted: Boolean = false
)
```

### 10.3.3 Data and Network Layer

1. **API Service**

```kotlin
interface ApiService {
    @GET("public/{username}/{slug}/nearest-times")
    suspend fun findNearestTimes(
        @Path("username") username: String,
        @Path("slug") slug: String,
        @Query("date") date: String,
        @Query("preferredTime") preferredTime: String,
        @Query("duration") duration: Int = 30
    ): TimeOptionsResponse

    @POST("public/{username}/{slug}/request")
    suspend fun submitRequest(
        @Path("username") username: String,
        @Path("slug") slug: String,
        @Body request: RequestApiModel
    ): RequestResponse
}

@Singleton
class ApiClient @Inject constructor(
    okHttpClient: OkHttpClient,
    moshiConverterFactory: MoshiConverterFactory
) {
    val retrofit: Retrofit = Retrofit.Builder()
        .baseUrl("https://api.sparetime.me/v1/")
        .client(okHttpClient)
        .addConverterFactory(moshiConverterFactory)
        .build()

    val apiService: ApiService =
retrofit.create(ApiService::class.java)
}
```

2. **Repository Implementation**

```kotlin
interface TimeRepository {
    suspend fun findNearestTimes(linkId: String, date: String,
preferredTime: String): List<TimeOption>
}

class TimeRepositoryImpl @Inject constructor(
    private val apiService: ApiService,
    private val linkRepository: LinkRepository
) : TimeRepository {

    override suspend fun findNearestTimes(linkId: String, date:
String, preferredTime: String): List<TimeOption> {
        // Get link details to extract username and slug
        val link = linkRepository.getLink(linkId) ?: throw
IllegalArgumentException("Link not found")

        // Extract username and slug from fullUrl
        // Format: https://sparetime.me/{username}/{slug}
        val urlParts =
link.fullUrl.removePrefix("https://sparetime.me/").split("/")
```

```
        if (urlParts.size < 2) throw
IllegalArgumentException("Invalid link URL format")

        val username = urlParts[0]
        val slug = urlParts[1]

        return try {
            val response = apiService.findNearestTimes(
                username = username,
                slug = slug,
                date = date,
                preferredTime = preferredTime
            )

            response.data.options.map { it.toDomain() }
        } catch (e: Exception) {
            throw e
        }
    }
}
```

3. **Local Storage**

Kotlin

```kotlin
@Entity(tableName = "links")
data class LinkEntity(
    @PrimaryKey val id: String,
    val name: String,
    val slug: String,
    val fullUrl: String,
    val expiresAt: Long?,
    val settingsJson: String,
    val createdAt: Long,
    val updatedAt: Long
)

@Dao
interface LinkDao {
    @Query("SELECT * FROM links ORDER BY updatedAt DESC")
    fun getAll(): Flow<List<LinkEntity>>

    @Query("SELECT * FROM links WHERE id = :id")
    suspend fun getById(id: String): LinkEntity?

    @Insert(onConflict = OnConflictStrategy.REPLACE)
    suspend fun insert(link: LinkEntity)

    @Delete
    suspend fun delete(link: LinkEntity)
}

class LinkRepositoryImpl @Inject constructor(
    private val linkDao: LinkDao,
    private val apiService: ApiService,
    private val authManager: AuthManager
) : LinkRepository {

    override fun getLinks(): Flow<List<Link>> {
        return linkDao.getAll().map { entities ->
            entities.map { it.toDomain() }
        }
    }

    override suspend fun getLink(id: String): Link? {
        return linkDao.getById(id)?.toDomain()
    }

    override suspend fun refreshLinks() {
        try {
            val token = authManager.getAuthToken() ?: throw
IllegalStateException("Not authenticated")
            val response = apiService.getLinks("Bearer $token")

            response.data.forEach { apiLink ->
                linkDao.insert(apiLink.toEntity())
            }
        } catch (e: Exception) {
            Log.e("LinkRepository", "Failed to refresh links", e)
            throw e
        }
    }

    private fun LinkEntity.toDomain(): Link {
        val moshi = Moshi.Builder().build()
        val settingsAdapter =
moshi.adapter(LinkSettings::class.java)

        return Link(
            id = id,
            name = name,
            slug = slug,
            fullUrl = fullUrl,
            expiresAt = expiresAt?.let { Date(it) },
            settings = settingsAdapter.fromJson(settingsJson) ?:
LinkSettings(),
            createdAt = Date(createdAt),
            updatedAt = Date(updatedAt)
        )
    }

    private fun ApiLink.toEntity(): LinkEntity {
        val moshi = Moshi.Builder().build()
        val settingsAdapter =
moshi.adapter(LinkSettings::class.java)
```

```
        return LinkEntity(
            id = id,
            name = name,
            slug = slug,
            fullUrl = fullUrl,
            expiresAt = expiresAt?.time,
            settingsJson = settingsAdapter.toJson(settings),
            createdAt = createdAt.time,
            updatedAt = updatedAt.time
        )
    }
}
```

---

# 11. Testing Requirements

## 11.1 Testing Strategy

A comprehensive testing strategy will be implemented to ensure the quality and reliability of the SpareTime application:

### 11.1.1 Unit Testing

**Coverage Target:** 80% code coverage for business logic

**Key Areas to Test:**

- Availability calculation algorithm
- Time parsing and formatting
- Link visibility rules
- Protected time application
- Time slot recommendation logic

**Testing Frameworks:**

- iOS: XCTest
- Android: JUnit, Mockito, Turbine
- Backend: Jest, Supertest

**Example Unit Test (iOS):**

None

```
func testFindNearestTimeSlots() {
    // Given
    let startOfDay = Calendar.current.startOfDay(for: Date())
    let busyBlocks = [
        BusyBlock(
            start: Calendar.current.date(byAdding: .hour, value:
9, to: startOfDay)!,
            end: Calendar.current.date(byAdding: .hour, value:
11, to: startOfDay)!,
            isNegotiable: false
        ),
        BusyBlock(
            start: Calendar.current.date(byAdding: .hour, value:
13, to: startOfDay)!,
            end: Calendar.current.date(byAdding: .hour, value:
14, to: startOfDay)!,
            isNegotiable: false
        ),
        BusyBlock(
            start: Calendar.current.date(byAdding: .hour, value:
16, to: startOfDay)!,
            end: Calendar.current.date(byAdding: .hour, value:
18, to: startOfDay)!,
            isNegotiable: true
        )
    ]

    let preferredTime = Calendar.current.date(byAdding: .hour,
value: 15, to: startOfDay)!
    let duration = 30 // minutes

    let availabilityService = AvailabilityService()

    // When
    let slots = availabilityService.findNearestTimeSlots(
        busyBlocks: busyBlocks,
        preferredTime: preferredTime,
        duration: duration,
        maxResults: 3
    )

    // Then
    XCTAssertEqual(slots.count, 3)

    // Verify slots are ordered by proximity to preferred time
    XCTAssertLessThanOrEqual(
        abs(slots[0].start.timeIntervalSince(preferredTime)),
        abs(slots[1].start.timeIntervalSince(preferredTime))
    )

    // Verify slots don't overlap with busy blocks
    for slot in slots {
        for block in busyBlocks {
            if !block.isNegotiable {
                XCTAssertFalse(
                    (slot.start < block.end) && (slot.end >
block.start),
                    "Slot should not overlap with non-negotiable
busy block"
                )
            }
        }
    }
}
```

**Example Unit Test (Android):**

```kotlin
@Test
fun `parse natural language time input correctly`() {
    // Given
```

```kotlin
val parser = TimeInputParser()
val referenceDate = LocalDate.of(2025, 10, 24)

// When/Then - Test absolute times
assertEquals(
    LocalTime.of(14, 0),
    parser.parseTimeInput("2pm", referenceDate)
)

assertEquals(
    LocalTime.of(19, 30),
    parser.parseTimeInput("7:30pm", referenceDate)
)

assertEquals(
    LocalTime.of(15, 0),
    parser.parseTimeInput("15:00", referenceDate)
)

// When/Then - Test relative times
assertEquals(
    LocalTime.of(15, 0),
    parser.parseTimeInput("after 3", referenceDate)
)

assertEquals(
    LocalTime.of(11, 0),
    parser.parseTimeInput("before noon", referenceDate)
)

// When/Then - Test fuzzy times
assertEquals(
    LocalTime.of(9, 0),
    parser.parseTimeInput("morning", referenceDate)
)
```

```
    assertEquals(
        LocalTime.of(12, 0),
        parser.parseTimeInput("lunch", referenceDate)
    )

    assertEquals(
        LocalTime.of(18, 0),
        parser.parseTimeInput("evening", referenceDate)
    )

    // When/Then - Test invalid input
    assertNull(parser.parseTimeInput("not a time",
referenceDate))
}
```

#### 11.1.2 Integration Testing

**Coverage Target:** Key API endpoints and data flows

**Key Areas to Test:**

- API request/response cycle
- Database operations
- Calendar sync process
- Authentication flow
- Push notification delivery

**Testing Frameworks:**

- Backend: Jest, Supertest
- iOS: XCTest with integration testing target
- Android: Espresso with Hilt testing

**Example Integration Test (Backend):**

```javascript
describe('Availability API', () => {
  beforeEach(async () => {
    // Set up test database with fixture data
    await setupTestDatabase();
  });

  afterEach(async () => {
    // Clean up test database
    await cleanupTestDatabase();
  });

  it('should find nearest time options correctly', async () => {
    // Given
    const username = 'testuser';
    const linkSlug = 'test-link';
    const date = '2025-10-24';
    const preferredTime = '15:00';

    // When
    const response = await request(app)
      .get(`/v1/public/${username}/${linkSlug}/nearest-times`)
      .query({
        date,
        preferredTime,
        duration: 30
      });

    // Then
    expect(response.status).toBe(200);
    expect(response.body.data).toBeDefined();
    expect(response.body.data.options).toBeInstanceOf(Array);
    expect(response.body.data.options.length).toBeGreaterThan(0);

    // Verify options are close to preferred time
    const preferredTimeObj = new
Date(`${date}T${preferredTime}:00`);
    const firstOption = new
Date(response.body.data.options[0].start);
```

```
    const timeDiff = Math.abs(firstOption - preferredTimeObj);

    // Should be within 3 hours of preferred time
    expect(timeDiff).toBeLessThanOrEqual(3 * 60 * 60 * 1000);
  });
});
```

**11.1.3 UI Testing**

**Coverage Target:** Critical user flows

**Key Areas to Test:**

- Onboarding flow
- Link creation process
- Time request process
- Calendar integration
- Settings changes

**Testing Frameworks:**

- iOS: XCUITest
- Android: Espresso, Compose UI Testing

**Example UI Test (iOS):**

None
```
func testTimeRequestFlow() {
    // Given
    let app = XCUIApplication()
    app.launch()

    // When

    // Navigate to link view
    app.buttons["viewLinkButton"].tap()

    // Select date
    app.buttons["datePillButton"].tap()
    app.buttons["nextDayButton"].tap()
    app.buttons["doneButton"].tap()

    // Enter preferred time
    let timeField = app.textFields["preferredTimeField"]
    timeField.tap()
    timeField.typeText("3pm")
    app.buttons["findTimesButton"].tap()

    // Wait for results and select first option
    let timeOptions = app.buttons.matching(identifier:
"timeOptionChip")
    XCTAssert(timeOptions.element.waitForExistence(timeout: 5))
    timeOptions.element(boundBy: 0).tap()

    // Fill request form
    app.buttons["requestTimeButton"].tap()

    app.textFields["nameField"].tap()
    app.textFields["nameField"].typeText("Test User")

    app.textFields["emailField"].tap()
    app.textFields["emailField"].typeText("test@example.com")

    app.textFields["purposeField"].tap()
    app.textFields["purposeField"].typeText("Test Meeting")

    app.buttons["submitRequestButton"].tap()

    // Then

XCTAssert(app.staticTexts["requestConfirmationText"].waitForExist
ence(timeout: 5))
```

```
XCTAssertTrue(app.staticTexts["requestConfirmationText"].label.co
ntains("Your request has been sent"))
}
```

**Example UI Test (Android):**

```kotlin
@HiltAndroidTest
@RunWith(AndroidJUnit4::class)
class TimeRequestFlowTest {

    @get:Rule
    val composeTestRule =
createAndroidComposeRule<MainActivity>()

    @Inject
    lateinit var fakeApiService: FakeApiService

    @Before
    fun setup() {
        hiltRule.inject()
        // Setup fake data
        fakeApiService.setupFakeTimeOptions()
    }

    @Test
    fun timeRequestFlow_completesSuccessfully() {
        // Navigate to link view
        composeTestRule.onNodeWithText("View
Link").performClick()

        // Select date
        composeTestRule.onNodeWithTag("datePill").performClick()
        composeTestRule.onNodeWithText("Next Day").performClick()
        composeTestRule.onNodeWithText("Done").performClick()

        // Enter preferred time

composeTestRule.onNodeWithTag("preferredTimeInput").performTextIn
put("3pm")
        composeTestRule.onNodeWithText("Find
Times").performClick()

        // Wait for results
        composeTestRule.waitUntil(5000) {

composeTestRule.onAllNodesWithTag("timeOptionChip").fetchSemantic
sNodes().size > 0
        }

        // Select first option

composeTestRule.onAllNodesWithTag("timeOptionChip")[0].performCli
ck()

        // Request time
        composeTestRule.onNodeWithText("Request This
Time").performClick()

        // Fill form

composeTestRule.onNodeWithTag("nameInput").performTextInput("Test
User")

composeTestRule.onNodeWithTag("emailInput").performTextInput("tes
t@example.com")

composeTestRule.onNodeWithTag("purposeInput").performTextInput("T
est Meeting")
```

```
        // Submit request
        composeTestRule.onNodeWithText("Submit
Request").performClick()

        // Verify confirmation
        composeTestRule.onNodeWithText("Your request has been
sent").assertIsDisplayed()
    }
}
```

### 11.2 Performance Testing

#### 11.2.1 API Response Time

- Benchmark target: < 200ms for most API endpoints
- Load testing with simulated users (10, 100, 1000 concurrent users)
- Monitoring response time under load
- Database query performance optimization

#### 11.2.2 Mobile App Performance

- App startup time: < 2 seconds on mid-range devices
- Screen transition time: < 300ms
- Memory usage: < 150MB
- Battery usage monitoring
- Frame rate monitoring for animations (target: consistent 60fps)

### 11.3 Security Testing

- Authentication flow penetration testing
- API endpoint security testing
- Data encryption verification
- Session management testing
- Input validation and sanitization testing
- OWASP Top 10 vulnerability assessment

---

## 12. Deployment Plan

## 12.1 Backend Deployment

### 12.1.1 Infrastructure Setup (AWS)

1. **API Server**
   - AWS Elastic Beanstalk for Node.js deployment
   - Auto-scaling configuration
   - Load balancer setup
   - Health check endpoints
2. **Database**
   - MongoDB Atlas for primary database
   - Backup and restore procedures
   - Replication setup for redundancy
3. **Caching Layer**
   - Amazon ElastiCache (Redis)
   - Cache invalidation strategy
   - Connection pooling configuration
4. **Storage**
   - Amazon S3 for file storage
   - CDN configuration for static assets
   - Lifecycle policies for cost optimization

### 12.1.2 CI/CD Pipeline

1. **Continuous Integration**
   - GitHub Actions for automated testing
   - Code quality checks (ESLint, Prettier)
   - Test coverage reports
   - Dependency security scanning
2. **Continuous Deployment**
   - Automated staging environment deployment
   - Production deployment approval process
   - Rollback procedures
   - Version tagging

### 12.1.3 Monitoring and Logging

1. **Application Monitoring**

    - AWS CloudWatch metrics
    - Custom metrics for business logic
    - Alerting thresholds configuration
2. **Logging System**

    - Centralized logging with ELK stack
    - Log retention policies
    - Error aggregation and analysis
3. **Performance Monitoring**

    - Application Performance Monitoring (APM)
    - Database query monitoring
    - API response time tracking

## 12.2 Mobile App Deployment

### 12.2.1 iOS Deployment

1. **App Store Submission**

    - App Store Connect setup
    - Metadata preparation
    - Screenshots and promotional materials
    - App review guidelines compliance
2. **Beta Testing**

    - TestFlight distribution
    - Internal testing group setup
    - External testing group management
    - Feedback collection process
3. **Release Management**

    - Phased rollout strategy
    - Version update planning
    - In-app update prompting

### 12.2.2 Android Deployment

1. **Google Play Store Submission**

    - Google Play Console setup
    - Store listing preparation
    - Screenshot and graphic assets
    - Content rating questionnaire

2. **Beta Testing**
    - Google Play internal testing track
    - Open and closed testing tracks
    - Alpha/beta tester management
    - Crash reporting integration
3. **Release Management**
    - Staged rollout percentage configuration
    - Release notes management
    - Android App Bundle optimization
    - Play Store review process planning

### 12.3 Post-Launch Operations

1. **User Support**
    - Help documentation
    - In-app support system
    - Email support workflow
    - Bug reporting system
2. **Analytics**
    - User acquisition metrics
    - Engagement tracking
    - Feature usage analytics
    - Conversion rate monitoring
3. **Update Cadence**
    - Bi-weekly bug fix releases
    - Monthly feature updates
    - Quarterly major updates
    - Hotfix protocol for critical issues

---

# 13. Advanced Use Cases

### 13.1 Team Availability Coordination

#### 13.1.1 Use Case Overview

**Scenario:** A product development team of 8 people needs to coordinate meetings while respecting individual focus time and personal boundaries.

**Challenges:**

- Team members are in different time zones
- Some team members need large blocks of uninterrupted focus time
- Certain meetings are higher priority than others
- Some team members have flexible personal commitments

### 13.1.2 Implementation with SpareTime

1. **Team Setup Process**
    - Team lead creates a Premium account
    - Creates a team in SpareTime and invites members
    - Sets team defaults for visibility and request permissions
    - Creates different audience views (All Team, Direct Reports, Leadership)
2. **Individual Member Configuration**
    - Each member connects their work calendar
    - Sets protected focus time that appears as "busy"
    - Marks certain commitments as "busy but negotiable for team"
    - Configures personal boundary preferences
3. **Team Availability Features**
    - Combined team view showing when all members are available
    - Filters for "must-attend" vs. "optional" participants
    - Color-coding for fully free vs. negotiable time
    - Time zone overlays for distributed teams
4. **Meeting Coordination Workflow**
    - Team lead views combined availability
    - Selects optimal meeting time where most/all required members are free
    - System shows which members have negotiable conflicts
    - Meeting request sent with context about why this time was chosen
    - Members with negotiable conflicts can easily accept or propose alternatives
5. **Advanced Team Analytics**
    - Meeting load distribution across team
    - Focus time protection effectiveness
    - Meeting request patterns
    - Schedule density visualization

### 13.1.3 Specific Team Features

1. **Smart Meeting Finder**

    - Algorithm to find optimal meeting times based on:
        - Required vs. optional attendees
        - Protected focus time preservation
        - Fairness across time zones
        - Meeting type priority
2. **Meeting Budget System**
    - Teams can set "meeting budgets" (maximum hours per week)
    - Visual indicators when approaching budget limit
    - Different budgets for different meeting types
3. **Team Calendar Annotations**
    - Shared team markers and milestones
    - Sprint boundaries
    - Deadline indicators
    - Company holidays

## 13.2 Family Coordination Hub

### 13.2.1 Use Case Overview

**Scenario:** A family with two working parents and three children needs to coordinate schedules while maintaining boundaries with extended family, school activities, and work.

**Challenges:**

- Complex, overlapping schedules
- Different visibility needs for different relationships
- Need to protect family time while enabling coordination
- Various external parties requesting time (schools, activities, family)

### 13.2.2 Implementation with SpareTime

1. **Family Setup Process**
    - Parent creates Premium account
    - Sets up family group with appropriate permissions
    - Connects family shared calendar and individual calendars
    - Creates audience tiers (Immediate Family, Extended Family, School, Activities)
2. **Protected Family Time**
    - Designate dinner time and weekend blocks as protected
    - Mark family vacations and special events
    - Set recurring protected time for family activities
    - Create buffer times around important family events

3. **Audience-Specific Views**
    - Immediate family sees complete schedule with details
    - Extended family sees general availability plus family events
    - School contacts see school-related availability
    - Activity coordinators see activity-specific availability
4. **Request Management for Families**
    - School can request time for parent-teacher meetings
    - Coaches can request practice schedule changes
    - Extended family can see good times for visits
    - Children's friends' parents can request playdate times
5. **Family-Specific Features**
    - Child-friendly calendar views for older children
    - Visual indicators for pick-up/drop-off responsibilities
    - Meal planning integration
    - Special event countdown indicators

### 13.2.3 Family Coordination Example

**Weekend Planning Scenario:**

1. Parents block Sunday dinner as protected family time (visible to all)
2. Extended family can see that Sunday evening is blocked
3. Grandparents looking to visit use SpareTime to see that Saturday afternoon is free
4. Grandparents send time request for Saturday visit
5. Parents receive request and approve
6. Children's extracurricular activities are visible to appropriate audience
7. Parents can see complete weekend schedule with all commitments

## 13.3 Professional Services Time Management

### 13.3.1 Use Case Overview

**Scenario:** A consultant or service provider needs to manage client appointments while protecting work-life balance and administrative time.

**Challenges:**

- Need to appear available to clients while protecting personal time
- Different types of clients require different levels of access
- Some time blocks should be available only for high-priority clients
- Need buffer time between client appointments

**13.3.2 Implementation with SpareTime**

1. **Initial Setup**
    - Professional creates Premium account
    - Connects work calendar
    - Sets up protected time for administrative work
    - Creates audience tiers (VIP Clients, Regular Clients, Prospects)
2. **Client-Specific Availability**
    - VIP clients see more available slots including priority times
    - Regular clients see standard availability
    - Prospects see limited availability
    - All views respect core protected time
3. **Appointment Request Workflow**
    - Client uses personalized link to view availability
    - Selects preferred time from available options
    - Professional receives request with client context
    - Can approve, deny, or suggest alternative times
    - Automatic buffer time insertion between appointments
4. **Professional Service Features**
    - Automatic travel time calculation between appointments
    - Location-based availability
    - Service type duration templates
    - Preparation time protection

**13.3.3 Advanced Professional Features**

1. **Dynamic Availability Rules**
    - Automatically protect recovery time after intense meetings
    - Limit number of client meetings per day
    - Ensure minimum focus time blocks are preserved
    - Seasonal availability adjustments
2. **Client Relationship Management**
    - Track request patterns by client
    - Note preferred meeting times
    - Record cancellation history
    - Customize availability by client relationship stage

# 14. Implementation Roadmap

## 14.1 Phase 1: Foundation (Weeks 1-4)

### 14.1.1 Backend Core

- Set up development environment
- Implement user authentication system
- Create basic API structure
- Set up database and schema
- Implement basic calendar integration

### 14.1.2 iOS MVP Skeleton

- Create project structure
- Implement authentication screens
- Build basic UI components
- Create navigation flow

### 14.1.3 Android MVP Skeleton

- Create project structure
- Implement authentication screens
- Build basic UI components
- Create navigation flow

### 14.1.4 Deliverables

- Working authentication system
- API documentation
- Basic UI navigation
- Development environment setup

## 14.2 Phase 2: Core Functionality (Weeks 5-8)

### 14.2.1 Backend Features

- Complete calendar integration
- Implement availability calculation engine
- Create protected time functionality
- Develop link generation system
- Implement basic analytics

### 14.2.2 iOS Development

- Calendar integration
- Availability view
- Protected time management
- Link creation and management
- Settings screens

#### 14.2.3 Android Development

- Calendar integration
- Availability view
- Protected time management
- Link creation and management
- Settings screens

#### 14.2.4 Deliverables

- Functional calendar sync
- Working availability displays
- Link creation and sharing
- Protected time management

### 14.3 Phase 3: Time Request System (Weeks 9-12)

#### 14.3.1 Backend Features

- Implement time request API
- Develop natural language time parsing
- Create time slot recommendation algorithm
- Build notification system
- Add request management functionality

#### 14.3.2 iOS Development

- Implement time request interface
- Build natural language time input
- Create time options display
- Develop request management screens
- Add notification handling

#### 14.3.3 Android Development

- Implement time request interface
- Build natural language time input
- Create time options display
- Develop request management screens
- Add notification handling

#### 14.3.4 Deliverables

- Complete time request system
- Natural language time parsing
- Three nearest options display
- Request management interface
- Notification system

## 14.4 Phase 4: Refinement and Testing (Weeks 13-16)

#### 14.4.1 Quality Assurance

- Comprehensive testing plan
- Unit test implementation
- Integration test suite
- UI test automation
- Performance optimization

#### 14.4.2 User Experience Improvements

- UI polish and refinement
- Animation and transitions
- Error handling improvements
- Edge case management
- Accessibility compliance

#### 14.4.3 Backend Optimization

- API performance tuning
- Database query optimization
- Caching implementation
- Security hardening
- Scaling preparation

#### 14.4.4 Deliverables

- Test coverage report
- Performance benchmark results
- UI/UX refinement documentation
- Security assessment report

## 14.5 Phase 5: Premium Features (Weeks 17-20)

#### 14.5.1 Backend Premium Features

- Multiple audience views

- Team/group functionality
- Advanced time protection
- Rich status messages
- Negotiable time slots

#### 14.5.2 iOS Premium Implementation

- Audience management UI
- Team views and coordination
- Advanced protection settings
- Status message management
- Premium subscription handling

#### 14.5.3 Android Premium Implementation

- Audience management UI
- Team views and coordination
- Advanced protection settings
- Status message management
- Premium subscription handling

#### 14.5.4 Deliverables

- Complete premium feature set
- Subscription management system
- Team coordination functionality
- Advanced protection capabilities

### 14.6 Phase 6: Launch Preparation (Weeks 21-24)

#### 14.6.1 Final Testing

- End-to-end testing
- Beta testing program
- Bug fixing sprint
- Final performance optimization
- Security penetration testing

#### 14.6.2 Deployment Preparation

- Production environment setup
- CI/CD pipeline finalization
- Monitoring and alerting setup
- Backup and disaster recovery testing
- Launch checklist verification

### 14.6.3 App Store Preparation

- App store listings creation
- Screenshot and marketing materials
- App review guidelines compliance
- Rating and content classification
- In-app purchase configuration

### 14.6.4 Deliverables

- Production-ready applications
- Complete deployment documentation
- App store submission packages
- Launch marketing materials
- Support documentation and FAQs

---

This comprehensive PRD provides a detailed blueprint for the SpareTime application, covering everything from the core concept and market differentiation to the technical architecture and implementation roadmap. The document serves as a guide for developers, investors, and stakeholders to understand the application's purpose, functionality, and development plan.