TomeTrack Technical Specification

TomeTrack Documentation

Your Reading, Your Community

TomeTrack - Technical Specification

Project Overview

TomeTrack is a social reading platform that combines personal library tracking, an integrated ebook reader, and community features. Users upload their own book files (stored locally or in personal cloud storage), track reading progress, and connect with other readers through reviews, discussions, and groups.

Key differentiator: No file storage on platform servers. Users maintain responsibility for their own files.

Tech Stack Recommendation

Frontend (Mobile App)

Primary recommendation: React Native

- Cross-platform (iOS and Android from single codebase)
- Strong community and library ecosystem
- Good performance for this use case
- Easy integration with native file systems and cloud APIs

Alternative: Flutter

- Also cross-platform
- Slightly better performance
- Different language (Dart vs JavaScript/TypeScript)

For MVP: React Native

Backend

Primary recommendation: Node.js with Express

- JavaScript across full stack (easier to find developers)
- Fast development
- Good for real-time features (feeds, notifications)
- Excellent ecosystem for APIs

Alternative: Django with Django REST Framework

- Python backend

- Built-in admin panel
- Strong ORM
- Good for data-heavy applications

For MVP: Node.js with Express

Database

PostgreSQL

- Robust relational database
- Excellent for structured data (users, books, reviews)
- Good JSON support for flexible fields
- Reliable and well-documented

File Storage

None required for book files (user responsibility)

For app assets and user content:

- AWS S3 or Cloudflare R2 for:

- Book cover images
- User profile pictures
- Group icons

Authentication

JWT (JSON Web Tokens)

- Stateless authentication
- Works well with mobile apps
- Secure when implemented properly

Optional: OAuth integration

- Google Sign-In
- Apple Sign-In
- Can be added post-MVP

Reader Component

react-native-epub-reader or **rn-pdf-reader-js**

- Open-source libraries for EPUB and PDF rendering
- Support for highlights and bookmarks
- Customizable styling

Cloud Storage Integration

For allowing users to link files:

- Google Drive API
- Dropbox API
- iCloud (iOS native)
- OneDrive API

Database Schema

Users Table

```
CREATE TABLE users (
id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
username VARCHAR(50) UNIQUE NOT NULL,
email VARCHAR(255) UNIQUE NOT NULL,
password_hash VARCHAR(255) NOT NULL,
display_name VARCHAR(100),
bio TEXT,
profile_picture_url VARCHAR(500),
created_at TIMESTAMP DEFAULT NOW(),
updated_at TIMESTAMP DEFAULT NOW()
);
```

Books Table

```
CREATE TABLE books (
id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
title VARCHAR(500) NOT NULL,
author VARCHAR(255) NOT NULL,
isbn VARCHAR(20),
publisher VARCHAR(255),
publication_year INTEGER,
pages INTEGER,
synopsis TEXT,
cover_image_url VARCHAR(500),
created_at TIMESTAMP DEFAULT NOW(),
updated_at TIMESTAMP DEFAULT NOW()
);
```

Genres Table

```
CREATE TABLE genres (
id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
name VARCHAR(100) UNIQUE NOT NULL
);
```

Book_Genres (Junction Table)

```
CREATE TABLE book_genres (
book_id UUID REFERENCES books(id) ON DELETE CASCADE,
genre_id UUID REFERENCES genres(id) ON DELETE CASCADE,
PRIMARY KEY (book_id, genre_id)
);
```

User_Books (Personal Library)

```
CREATE TABLE user_books (
id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
user_id UUID REFERENCES users(id) ON DELETE CASCADE,
book_id UUID REFERENCES books(id) ON DELETE CASCADE,
status VARCHAR(20) NOT NULL, -- 'want_to_read', 'reading', 'finished'
file_path VARCHAR(500), -- Local device path or cloud URL
file_format VARCHAR(10), -- 'epub', 'pdf', 'mobi'
progress_percentage INTEGER DEFAULT 0,
progress_location TEXT, -- Last read position (chapter, page, etc)
started_at TIMESTAMP,
finished_at TIMESTAMP,
added_at TIMESTAMP DEFAULT NOW(),
updated_at TIMESTAMP DEFAULT NOW(),
UNIQUE(user_id, book_id)
);
```

Reviews Table

```
CREATE TABLE reviews (
id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
user_id UUID REFERENCES users(id) ON DELETE CASCADE,
book_id UUID REFERENCES books(id) ON DELETE CASCADE,
rating INTEGER CHECK (rating >= 1 AND rating <= 5),
review_text TEXT,
contains_spoilers BOOLEAN DEFAULT FALSE,
visibility VARCHAR(20) DEFAULT 'public', -- 'private', 'followers', 'public'
created_at TIMESTAMP DEFAULT NOW(),
updated_at TIMESTAMP DEFAULT NOW(),
UNIQUE(user_id, book_id)
);
```

Highlights Table

```
CREATE TABLE highlights (
id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
user_book_id UUID REFERENCES user_books(id) ON DELETE CASCADE,
highlighted_text TEXT NOT NULL,
location_start TEXT NOT NULL,
location_end TEXT NOT NULL,
color VARCHAR(20) DEFAULT 'yellow',
note TEXT,
created_at TIMESTAMP DEFAULT NOW()
);
```

Bookmarks Table

```
CREATE TABLE bookmarks (
id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
user_book_id UUID REFERENCES user_books(id) ON DELETE CASCADE,
location TEXT NOT NULL,
note TEXT,
created_at TIMESTAMP DEFAULT NOW()
);
```

Follows Table

```
CREATE TABLE follows (
follower_id UUID REFERENCES users(id) ON DELETE CASCADE,
following_id UUID REFERENCES users(id) ON DELETE CASCADE,
created_at TIMESTAMP DEFAULT NOW(),
PRIMARY KEY (follower_id, following_id),
CHECK (follower_id != following_id)
);
```

Groups Table

```
CREATE TABLE groups (
id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
name VARCHAR(200) NOT NULL,
description TEXT,
icon_url VARCHAR(500),
is_private BOOLEAN DEFAULT FALSE,
created_by UUID REFERENCES users(id),
created_at TIMESTAMP DEFAULT NOW(),
updated_at TIMESTAMP DEFAULT NOW()
);
```

Group_Members Table

```
CREATE TABLE group_members (
group_id UUID REFERENCES groups(id) ON DELETE CASCADE,
user_id UUID REFERENCES users(id) ON DELETE CASCADE,
role VARCHAR(20) DEFAULT 'member', -- 'admin', 'moderator', 'member'
joined_at TIMESTAMP DEFAULT NOW(),
PRIMARY KEY (group_id, user_id)
);
```

Discussions Table

```
CREATE TABLE discussions (
id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
group_id UUID REFERENCES groups(id) ON DELETE CASCADE,
user_id UUID REFERENCES users(id) ON DELETE CASCADE,
book_id UUID REFERENCES books(id),
title VARCHAR(300) NOT NULL,
content TEXT NOT NULL,
is_pinned BOOLEAN DEFAULT FALSE,
created_at TIMESTAMP DEFAULT NOW(),
updated_at TIMESTAMP DEFAULT NOW()
);
```

Discussion_Replies Table

```
CREATE TABLE discussion_replies (
id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
discussion_id UUID REFERENCES discussions(id) ON DELETE CASCADE,
user_id UUID REFERENCES users(id) ON DELETE CASCADE,
content TEXT NOT NULL,
created_at TIMESTAMP DEFAULT NOW(),
updated_at TIMESTAMP DEFAULT NOW()
);
```

Activity_Feed Table

```
CREATE TABLE activity_feed (
id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
user_id UUID REFERENCES users(id) ON DELETE CASCADE,
activity_type VARCHAR(50) NOT NULL, -- 'started_reading', 'finished_reading',
'posted_review', 'joined_group'
book_id UUID REFERENCES books(id),
review_id UUID REFERENCES reviews(id),
group_id UUID REFERENCES groups(id),
visibility VARCHAR(20) DEFAULT 'public', -- 'private', 'followers', 'public'
created_at TIMESTAMP DEFAULT NOW()
);
```

Reactions Table

```
CREATE TABLE reactions (
id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
user_id UUID REFERENCES users(id) ON DELETE CASCADE,
target_type VARCHAR(50) NOT NULL, -- 'review', 'discussion', 'reply'
target_id UUID NOT NULL,
reaction_type VARCHAR(20) NOT NULL, -- 'like', 'love', 'bookmark'
created_at TIMESTAMP DEFAULT NOW(),
UNIQUE(user_id, target_type, target_id, reaction_type)
);
```

Comments Table

```
CREATE TABLE comments (
id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
user_id UUID REFERENCES users(id) ON DELETE CASCADE,
target_type VARCHAR(50) NOT NULL, -- 'review', 'activity'
target_id UUID NOT NULL,
content TEXT NOT NULL,
created_at TIMESTAMP DEFAULT NOW(),
updated_at TIMESTAMP DEFAULT NOW()
);
```

API Endpoints

Authentication

- `POST /api/auth/register` - Create new user account
- `POST /api/auth/login` - Login and receive JWT
- `POST /api/auth/logout` - Invalidate token
- `POST /api/auth/refresh` - Refresh JWT
- `POST /api/auth/forgot-password` - Request password reset
- `POST /api/auth/reset-password` - Reset password with token

Users

- `GET /api/users/:id` - Get user profile
- `PUT /api/users/:id` - Update user profile (own only)
- `GET /api/users/:id/library` - Get user's library (respects privacy)
- `GET /api/users/:id/reviews` - Get user's reviews
- `GET /api/users/:id/activity` - Get user's activity feed
- `GET /api/users/:id/followers` - Get user's followers
- `GET /api/users/:id/following` - Get who user follows
- `POST /api/users/:id/follow` - Follow a user
- `DELETE /api/users/:id/follow` - Unfollow a user

Books

- `GET /api/books` - Search/browse books (with filters)
- `GET /api/books/:id` - Get book details

- `POST /api/books` - Create new book entry (if not exists)
- `PUT /api/books/:id` - Update book details (admin/moderator)
- `GET /api/books/:id/reviews` - Get book reviews
- `GET /api/books/:id/discussions` - Get book discussions
- `GET /api/books/trending` - Get trending books
- `GET /api/books/recommended` - Get personalized recommendations

User Library

- `GET /api/library` - Get current user's library
- `POST /api/library` - Add book to library
- `PUT /api/library/:id` - Update book status/progress
- `DELETE /api/library/:id` - Remove book from library
- `POST /api/library/:id/attach-file` - Record file location (local path or cloud URL)

Reading Progress

- `PUT /api/library/:id/progress` - Update reading progress
- `POST /api/library/:id/highlights` - Create highlight
- `GET /api/library/:id/highlights` - Get highlights for a book
- `DELETE /api/highlights/:id` - Delete highlight
- `POST /api/library/:id/bookmarks` - Create bookmark
- `GET /api/library/:id/bookmarks` - Get bookmarks for a book
- `DELETE /api/bookmarks/:id` - Delete bookmark

Reviews

- `POST /api/reviews` - Create review
- `PUT /api/reviews/:id` - Update review
- `DELETE /api/reviews/:id` - Delete review
- `GET /api/reviews/:id` - Get single review
- `POST /api/reviews/:id/react` - React to review
- `POST /api/reviews/:id/comment` - Comment on review
- `GET /api/reviews/:id/comments` - Get review comments

Groups

- `GET /api/groups` - Browse/search groups
- `GET /api/groups/:id` - Get group details
- `POST /api/groups` - Create new group
- `PUT /api/groups/:id` - Update group (admin only)
- `DELETE /api/groups/:id` - Delete group (admin only)
- `POST /api/groups/:id/join` - Join group

- `POST /api/groups/:id/leave` - Leave group
- `GET /api/groups/:id/members` - Get group members

Discussions

- `GET /api/groups/:id/discussions` - Get group discussions
- `POST /api/groups/:id/discussions` - Create discussion
- `GET /api/discussions/:id` - Get discussion details
- `PUT /api/discussions/:id` - Update discussion
- `DELETE /api/discussions/:id` - Delete discussion
- `POST /api/discussions/:id/reply` - Reply to discussion
- `GET /api/discussions/:id/replies` - Get discussion replies

Feed

- `GET /api/feed` - Get personalized activity feed
- `GET /api/feed/global` - Get global activity (trending)

Search

- `GET /api/search` - Universal search (books, users, groups)

- Query params: `q` (query), `type` (books/users/groups), `page`, `limit`

File Handling Strategy

User Flow for File Upload

1. **User adds book to library**

- App prompts: "Do you want to attach your copy?"

- User can skip or proceed

2. **If user chooses to attach file:**

- **Option A: From Device**

- App opens native file picker

- User selects file (EPUB, PDF, MOBI)

- App reads file metadata

- App stores local file path in database

- **Option B: From Cloud Storage**

- User authenticates with cloud provider
- User navigates to file location
- App receives access token and file URL
- App stores cloud URL in database

3. **Reading the file:**
- When user opens book, app:
- Checks if file still exists at stored path
- If local: reads directly
- If cloud: downloads temporarily or streams
- Opens in integrated reader

4. **Sync considerations:**
- Progress, highlights, bookmarks sync to server
- Actual file never touches server
- If user switches devices, they need to attach file again on new device

File Access Permissions

iOS:

- Request file system access permission
- Use Document Picker for file selection
- Store reference to user-selected file

Android:

- Request storage permission
- Use Storage Access Framework
- Store content URI

Cloud Integration

Google Drive:

- Use Google Drive API
- OAuth 2.0 authentication
- Store file ID and access token (encrypted)

Dropbox:

- Use Dropbox API
- OAuth 2.0 authentication

- Store file path and access token (encrypted)

iCloud:

- iOS native integration
- Use CloudKit framework

Security Considerations

Authentication

- Password hashing: bcrypt with salt rounds >= 10
- JWT expiration: 7 days (refresh token), 1 hour (access token)
- HTTPS only for all API calls
- Rate limiting on auth endpoints

Data Privacy

- User email addresses never exposed in API responses
- Reading history private by default
- User can control visibility of library and activity
- File paths/URLs never exposed to other users

File Security

- Never store or transmit book files through server
- Cloud access tokens encrypted in database
- Regular token refresh and validation
- Clear user responsibility in Terms of Service

Input Validation

- All user inputs sanitized
- SQL injection prevention (parameterized queries)
- XSS prevention (escape output)
- File type validation (only allowed formats)
- Max file size checks

Moderation

- User reporting system
- Admin moderation panel

- Automated flagging for suspicious content
- DMCA compliance tools

Performance Optimization

Database

- Index on frequently queried fields:

- `users.username`
- `books.title`, `books.author`
- `user_books.user_id`, `user_books.book_id`
- `follows.follower_id`, `follows.following_id`
- `activity_feed.user_id`, `activity_feed.created_at`

- Query optimization:

- Use joins efficiently
- Limit result sets (pagination)
- Cache frequently accessed data (Redis)

API

- Response compression (gzip)
- Pagination on all list endpoints (default 20 items)
- Rate limiting to prevent abuse
- CDN for static assets (images)

Mobile App

- Image caching (React Native Fast Image)
- Lazy loading for lists
- Offline support for reader
- Background sync for progress updates

MVP Feature Priority

Must Have (Phase 1)

1. User authentication (register, login, logout)
2. User profiles (basic)
3. Book database and search
4. Add books to library
5. Library tracking (want to read, reading, finished)
6. Integrated reader (EPUB and PDF)
7. Reading progress tracking
8. Basic highlights and bookmarks
9. Reviews (create, read)
10. Simple activity feed (following-based)
11. Follow/unfollow users

Should Have (Phase 2)

1. Groups (create, join, discussions)
2. Advanced search and filters
3. Trending books section
4. Notifications
5. Comments on reviews
6. Reactions (likes) on content
7. Cloud storage integration (Google Drive, Dropbox)
8. Profile customization
9. Privacy controls

Could Have (Phase 3)

1. Reading challenges
2. Book recommendations algorithm
3. Reading statistics and insights
4. Curated lists
5. Import from Goodreads
6. Dark mode refinements
7. Accessibility features
8. Multiple theme options in reader
9. Export reading data

Development Timeline Estimate

Phase 1: MVP (12-16 weeks)

- Week 1-2: Project setup, database design, API foundation
- Week 3-4: Authentication system
- Week 5-7: Book database and library features
- Week 8-10: Reader integration
- Week 11-12: Basic social features (feed, follow, reviews)
- Week 13-14: Testing and bug fixes
- Week 15-16: Polish and deployment preparation

Phase 2: Enhanced Features (8-10 weeks)

- Weeks 1-3: Groups and discussions
- Weeks 4-5: Advanced search and recommendations
- Weeks 6-7: Notifications and engagement features
- Weeks 8-10: Testing and refinement

Phase 3: Advanced Features (6-8 weeks)

- As needed based on user feedback

Testing Strategy

Unit Tests

- All API endpoints
- Database queries
- Authentication logic
- Validation functions

Integration Tests

- User flows (signup to first book read)
- API + database interactions
- File handling

User Acceptance Testing

- Beta testing with 50-100 users
- Focus on reader usability
- Social feature engagement
- Performance on various devices

Performance Testing

- Load testing (simulated concurrent users)
- Reader performance with large files
- API response times
- Database query optimization

Deployment Strategy

Backend

Recommended: Railway or Render

- Easy deployment
- Automatic scaling
- Reasonable pricing for MVP
- PostgreSQL included

Alternative: AWS (more complex but scalable)

- EC2 for app server
- RDS for PostgreSQL
- S3 for static assets
- CloudFront for CDN

Mobile App

iOS:

- Apple Developer account required ($99/year)
- Submit to App Store
- Review process ~2-7 days

Android:

- Google Play Developer account ($25 one-time)
- Submit to Play Store
- Review process ~1-3 days

Monitoring

- Error tracking: Sentry
- Analytics: Mixpanel or Amplitude
- Server monitoring: Datadog or New Relic
- User feedback: In-app feedback form

Legal Requirements

Terms of Service

Must include:

- User responsibility for file legality
- No file sharing between users
- Platform is not liable for copyrighted material
- Prohibited content and behavior
- Account termination conditions

Privacy Policy

Must include:

- What data is collected
- How data is used
- Third-party services (cloud storage)
- User rights (data access, deletion)
- Cookie usage
- GDPR compliance (if serving EU users)

Community Guidelines

Must include:

- Respectful behavior expectations
- Prohibited content (harassment, spam, piracy links)
- Reporting process
- Moderation actions

DMCA Compliance

- Designated agent for copyright notices
- Takedown process for infringing content

- Counter-notification process
- Repeat infringer policy

Cost Estimates (Monthly, Post-MVP)

Infrastructure

- Server hosting: $25-50 (Railway/Render)
- Database: Included in hosting
- Storage (S3 for images): $5-20
- CDN: $0-10
- Total: ~$30-80/month for first 1000 users

Services

- Apple Developer: $8.25/month ($99/year)
- Google Play Developer: $2/month (amortized)
- Domain name: $1-2/month
- Email service: $0-5/month
- Total: ~$11-17/month

Scaling Costs

- At 10k users: ~$200-400/month
- At 100k users: ~$1000-2000/month
- Primarily server and database costs

This technical specification provides the complete foundation for developing TomeTrack. Next steps would be:

1. Set up development environment
2. Initialize Git repository
3. Create database
4. Build API scaffolding
5. Develop React Native app structure
6. Iterate on features according to MVP priority