Technical Documentation

 # Technical Documentation for Movie Management Rails Application

This is a Ruby on Rails 8.1.1 web application designed for movie enthusiasts to manage their movie-watching experiences. The application integrates with The Movie Database (TMDB) API to fetch movie data and provides features for user registration, movie discovery, watchlists, ratings, reviews, favorites, and diary entries.

Technology Stack

- **Framework**: Ruby on Rails 8.1.1

- **Database**: SQLite3 (development/test), configurable for production

- **Frontend**: HTML, CSS (Tailwind CSS), JavaScript (ES6 modules via importmap-rails)

- **Authentication**: Custom authentication using bcrypt for password hashing

- **External API**: TMDB API for movie data

- **Testing**: RSpec, Cucumber, Capybara

- **Deployment**: Docker, Kamal for containerized deployment

- **Background Jobs**: Solid Queue

- **Caching**: Solid Cache

- **Web Server**: Puma

- **Asset Pipeline**: Propshaft

Prerequisites

Before setting up the application, ensure you have the following installed on your system:

- Ruby 3.2+ (check `.ruby-version` for exact version)

- Node.js and npm (for Tailwind CSS)

- SQLite3 (or PostgreSQL/MySQL for production)

Step 1: Clone the Repository

bash

git clone <repository-url>

cd CSCE-606-700-Group-9-Project-3

Step 2: Install Dependencies

The project includes a setup script that handles most of the initial configuration:

bash

bin/setup

Step 3: Configure TMDB API

The application requires a TMDB API key for movie search functionality. Configure it in one of two ways:

Option A: Environment Variable

```bash
export TMDB_API_KEY=your_tmdb_api_key_here
```

Option B: Rails Credentials

```bash
bin/rails credentials:edit
```

Add the following to the credentials file:

```yaml
tmdb:
  api_key: your_tmdb_api_key_here
```

Step 4: Database Setup

```bash
```

bin/rails db:setup
bin/rails db:migrate
bin/rails db:seed # loads sample data
```
### Step 5: Start Development Server
```bash
bin/dev
```
This uses Foreman to run both the Rails server and Tailwind CSS watcher.
Step 6: Access the Application
Open your browser and navigate to `http://localhost:3000`

System Architecture

The application follows a typical Rails MVC (Model-View-Controller) architecture with the following components:

- **Web Layer**: Puma web server handling HTTP requests

- **Application Layer**: Rails controllers processing business logic

- **Data Layer**: ActiveRecord models interfacing with SQLite database

- **External Services**: TMDB API for movie data retrieval

- **Background Processing**: Solid Queue for asynchronous jobs

- **Caching**: Solid Cache for performance optimization

- **Asset Pipeline**: Propshaft for static asset management
```

Models

User Model

- Inherits from ApplicationRecord

- Uses `has_secure_password` for authentication

- Associations: has_many diary_entries, favorites, ratings, watchlists, review_reactions

- Through associations: movies (via watchlists), favorite_movies (via favorites)

- Validations: email format and uniqueness, username length and uniqueness, password strength

Services

- **TMDB Service**: Handles API integration for fetching movie data

Key Features

1. **User Management**: Registration, login, profile editing

2. **Movie Discovery**: Search and browse movies via TMDB API

3. **Watchlists**: Add movies to watch later

4. **Ratings & Reviews**: Rate movies and write reviews

5. **Favorites**: Mark movies as favorites

6. **Diary Entries**: Log movie watching experiences

7. **Tags**: Categorize movies with custom tags

8. **Review Reactions**: React to other users' reviews with emojis

9. **Dashboard**: Personalized user dashboard

Security Considerations

- Password hashing using bcrypt
- CSRF protection via Rails
- Content Security Policy configured
- Parameter filtering for sensitive data
- Brakeman for security vulnerability scanning

Testing Strategy

- **Unit Tests**: RSpec for models and services
- **Integration Tests**: Cucumber for feature testing
- **System Tests**: Capybara for end-to-end testing
- **Code Quality**: RuboCop for style enforcement
- **Coverage**: SimpleCov for test coverage reporting

- Tailwind CSS for optimized styling
- Import maps for JavaScript module management