Notflix Architecture Overview

Developer Guide & Technical Documentation

Table of Contents

System Overview

Notflix is a cross-platform Flutter application for discovering movies and TV shows.

Core Functionality:

- Browse content by genre and category
- Search movies and TV shows
- View detailed information and trailers
- User authentication and watch lists
- Personalized recommendations

Platform Support:

- Android
- iOS
- Web

Technology Stack

Frontend:

- **Flutter** (Dart SDK 3.9.0+)
- **Material Design** with custom Netflix-inspired theme

Backend Services:

- **The Movie Database (TMDB) API** - Content data
- **PocketBase** - User authentication and watch lists

Key Dependencies:

- `http` - REST API calls
- `pocketbase` - Backend database
- `youtube_player_iframe` - Trailer playback

Architecture Patterns

Patterns Used:

1. **MVC (Model-View-Controller)**
   - Models: `Movie`, `TvShow`, `Episode`
   - Views: Screen widgets (`MovieList`, `Search`, `MovieDetail`)
   - Controllers: State management via `StatefulWidget`
2. **Repository Pattern**
   - `APIRunner` - API abstraction layer
   - `DbConnection` - Database abstraction layer
3. **Widget Composition**
   - Reusable components (`MovieCard`, `WatchListButtons`)

Project Structure

```
lib/
├── main.dart                # App entry point & theme
├── model/                   # Data models
│   ├── movie.dart
│   ├── tvShow.dart
│   └── episode.dart
├── util/                   # Utilities
│   ├── api.dart            # TMDB API client
│   └── db.dart             # PocketBase client
└── view/                   # UI screens
    ├── movie_list.dart     # Home screen
    ├── search.dart         # Search screen
    ├── movie_detail.dart # Details screen
    ├── signup_page.dart
    └── user_page/
        ├── log_in.dart
        └── profile.dart
```

Data Models

Movie Model:

```
class Movie {
  int id;
  String title;
  double voteAverage;
  String releaseDate;
  String overview;
  String posterPath;
  List genres;
}
```

TvShow Model:

- Similar structure to Movie
- Additional TV-specific fields

Episode Model:

API Integration - TMDB

APIRunner Class Responsibilities:

- Authentication with Bearer token
- Endpoint management
- JSON parsing to models
- Error handling

Key Methods:

- `getUpcoming()` - Popular content
- `getGenre()` - Genre-based content
- `searchMovie()` - Search functionality
- `getTrailerKey()` - Video trailers
- `getTvShowDetails()` - TV show metadata

API Endpoints Used

TMDB API Endpoints:

- `/discover/movie` - Discover movies
- `/discover/tv` - Discover TV shows
- `/search/movie` - Search movies
- `/movie/{id}/videos` - Movie trailers
- `/tv/{id}/videos` - TV show trailers
- `/tv/{id}` - TV show details
- `/tv/{id}/season/{season}` - Season episodes
- `/genre/movie/list` - Movie genres
- `/genre/tv/list` - TV genres

Base URL: `https://api.themoviedb.org/3`

Database Design - PocketBase

Collections:

1. **users**
    - `username` (String)
    - `email` (String, unique)
    - `password` (String, hashed)
2. **user_watch_lists**
    - `user` (Relation to users)
    - `recently_watched` (JSON object)
    - `watch_later` (JSON object)

Storage Format:

Database Operations

DbConnection Class Methods:

Authentication:

- `logInUser()` - User login
- `createUser()` - User registration
- `logoutUser()` - Clear session
- `checkUserLogStatus()` - Check auth state

Watch Lists:

- `addShowToWatchList()` - Add to list
- `removeShowFromWatchList()` - Remove from list
- `getWatchLaterShows()` - Retrieve watch later

Key Components - Home Screen

MovieList Widget:

- Main entry point
- Manages state for categories/genres
- Fetches and displays content
- Hero movie display
- Horizontal scrolling rows

Features:

- Category filtering (All/Movies/TV Shows)
- Genre filtering
- Dynamic content loading
- Loading states

Key Components - Search Screen

Search Widget:

- Real-time search functionality
- Grid layout for results
- Personalized recommendations (when logged in)
- "Because You Watched" section
- Upcoming movies section

Search Logic:

- Debounced input handling
- API call on text change
- Results displayed in grid
- Clear search functionality

Key Components - Details Screen

MovieDetail Widget:

- Displays movie/TV show information
- YouTube trailer integration
- Episode list for TV shows
- Season selector
- Watch list buttons

Features:

- Conditional rendering (Movie vs TV Show)
- Async data loading
- Trailer playback
- Episode browsing

Data Flow - Content Browsing

```
User Action
    ↓
MovieList Widget
    ↓
APIRunner.getUpcoming()
    ↓
TMDB API Request
    ↓
JSON Response
    ↓
Movie.fromJson() / TvShow.fromJson()
    ↓
Widget State Update
    ↓
UI Rendering
```

Data Flow - User Actions

```
User Login
    ↓
DbConnection.logInUser()
    ↓
PocketBase Authentication
    ↓
Auth Token Stored
    ↓
Session Management
    ↓
Personalized Features Enabled
```

State Management

Current Approach:

- `StatefulWidget` with `setState()`
- Local state management
- No external state management library

State Variables:

- Loading states (`isLoading`)
- Content lists (`movies` , `moviesTvShows`)
- Filter selections (`_typeValue` , `_genreValue`)
- User authentication state

Theme & Styling

Custom Theme (Netflix-inspired):

- Dark background (`#141414`)
- Netflix red primary (`#E50914`)
- Dark surface (`#181818`)
- White text with opacity variations

Defined in: `lib/main.dart`

Components:

- AppBar theme
- Text theme
- Button theme
- Card theme

Error Handling

API Errors:

- HTTP status code checking
- Try-catch blocks
- Null safety handling
- Default values for missing data

Database Errors:

- Connection error handling
- Authentication error messages
- User-friendly error display

UI Error States:

Development Setup

Prerequisites:

- Flutter SDK 3.9.0+
- Dart SDK
- PocketBase server (local)

Steps:

1. Clone repository
2. Run `flutter pub get`
3. Start PocketBase: `cd server/pocketbase && .\start-pocketbase.ps1`
4. Configure API key in `lib/util/api.dart`
5. Run `flutter run`

PocketBase Setup

Local Development:

- PocketBase binary in `server/pocketbase/`
- Start script: `start-pocketbase.ps1`
- Default URL: `http://localhost:8090`
- Android emulator: `http://10.0.2.2:8090`

Collections Setup:

- Create `users` collection
- Create `user_watch_lists` collection
- Configure field types and relations

Testing Strategy

Test Structure:

- Unit tests: `test/` directory
- Widget tests: `test/widget/`
- Integration tests: `integration_test/`

Test Coverage:

- Model parsing (`movie_test.dart`, `tv_show_test.dart`)
- Database operations (`db_test.dart`)
- Widget rendering (`movie_card_test.dart`)
- API integration (mocked)

API Key Management

Current Implementation:

- API key stored in `lib/util/api.dart`
- **Security Note:** Should be moved to environment variables
- Consider using `flutter_dotenv` package

Best Practices:

- Never commit API keys to version control
- Use environment-specific keys
- Rotate keys regularly

Image Loading

Image Sources:

- TMDB CDN: `https://image.tmdb.org/t/p/w500/`
- Poster paths from API
- Default fallback image
- Network image widgets with error handling

Sizes Used:

- `w500` - Standard posters
- `w780` - Hero images

Performance Considerations

Optimizations:

- Lazy loading of content rows
- Image caching (Flutter default)
- Pagination for large lists
- Debounced search input

Areas for Improvement:

- Implement proper pagination
- Add image caching strategy
- Optimize list rendering
- Consider state management library

Future Enhancements

Potential Improvements:

1. State management (Provider/Riverpod/Bloc)
2. Offline caching
3. Advanced filtering
4. User ratings and reviews
5. Social features
6. Push notifications
7. Better error handling
8. Analytics integration

Code Organization Best Practices

Current Structure:

- Separation of concerns (models, views, utils)
- Reusable widgets
- Clear naming conventions

Recommendations:

- Consider feature-based structure
- Extract constants to separate files
- Create service layer for business logic
- Implement dependency injection

Deployment Considerations

Build Targets:

- Android: APK/AAB
- iOS: IPA
- Web: Static hosting
- Desktop: Platform-specific executables

Environment Configuration:

- Production API keys
- Production PocketBase instance
- Error tracking (Sentry, Firebase Crashlytics)
- Analytics integration

Security Considerations

Current Security:

- Password hashing (PocketBase)
- HTTPS for API calls
- Bearer token authentication

Recommendations:

- Implement API rate limiting
- Add input validation
- Secure storage for sensitive data
- Regular security audits

Documentation Standards

Code Documentation:

- Inline comments for complex logic
- Class and method documentation
- README files for setup

Architecture Documentation:

- This presentation
- User manual
- API documentation
- Database schema

Contributing Guidelines

Development Workflow:

1. Create feature branch
2. Write tests
3. Implement feature
4. Submit pull request
5. Code review required

Code Style:

- Follow Dart style guide
- Use `flutter analyze`
- Run tests before committing

Resources & References

Documentation:

- Flutter: https://flutter.dev/docs
- TMDB API: https://developers.themoviedb.org
- PocketBase: https://pocketbase.io/docs

Project Files:

- `README.md` - Project overview
- `TEAM_RULES.md` - Team guidelines
- `pubspec.yaml` - Dependencies

Questions & Support

For Developers:

- Review code comments
- Check test files for examples
- Consult team documentation
- Reach out to team members

Thank you for contributing to Notflix! 🚀