Recipe Slot App - Flutter Mobile Application

Project Overview

This is a complete Flutter mobile application that replicates the Recipe Slot App design and functionality. The app features a slot machine-style recipe discovery experience with Tinder-like swipe gestures, comprehensive recipe management, and a beautiful dark theme UI.

Features Implemented

Spin Mode

- **Interactive Slot Machine**: Custom slot machine widget with cuisine, meal type, and cooking time reels
- **Smooth Animations**: Using `slot_machine_roller` package for engaging visual feedback
- **Random Recipe Generation**: Fetches random recipes from Spoonacular API based on slot results
- **Swipeable Recipe Cards**: Tinder-like interface for recipe interaction

Ingredient Mode

- **Smart Search**: Search recipes based on available ingredients
- **Dynamic Ingredient Management**: Add/remove ingredients with intuitive chip interface
- **Real-time Results**: Live recipe matching using Spoonacular API
- **Filter Integration**: Respects user dietary preferences and restrictions

Recipe Management

- **Swipe Gestures**:
- Swipe Right: Save to favorites
- Swipe Left: Dismiss recipe
- Swipe Up: Mark as tried
- **Local Storage**: Uses Hive for offline access to saved recipes
- **Recipe Collections**: Separate tabs for saved and tried recipes
- **Detailed Recipe View**: Full recipe information with ingredients and instructions

⚙ Settings & Preferences

- **API Configuration**: Spoonacular API key management
- **Dietary Restrictions**: Support for allergies, diets, and cuisine preferences
- **Cooking Preferences**: Maximum cooking time settings
- **Persistent Storage**: All settings saved locally using Hive

UI/UX Design

- **Dark Theme**: Modern dark UI matching contemporary design trends
- **Material Design 3**: Latest Material Design components and patterns
- **Responsive Layout**: Optimized for various screen sizes
- **Smooth Animations**: Engaging transitions and micro-interactions
- **Bottom Navigation**: Easy access to all app features

Technical Architecture

State Management

- **Riverpod**: Modern, compile-safe state management
- **Provider Pattern**: Clean separation of business logic and UI
- **Reactive Updates**: Automatic UI updates when data changes

Data Layer

- **Hive Database**: Fast, lightweight local storage
- **Type Adapters**: Custom serialization for complex objects
- **API Service**: Clean abstraction for Spoonacular API calls
- **Dio HTTP Client**: Robust networking with error handling

Project Structure

```
lib/
├── main.dart                 # App entry point with Hive initialization
├── models/                   # Data models with Hive adapters
│   ├── recipe.dart          # Recipe model with JSON serialization
│   ├── settings.dart        # User settings model
│   └── *.g.dart            # Generated code files
├── services/                 # Business logic layer
│   └── api_service.dart     # Spoonacular API integration
├── providers/               # State management
│   └── recipe_provider.dart # Riverpod providers for app state
├── screens/                 # App screens
│   ├── main_screen.dart     # Bottom navigation container
│   ├── spin_mode_screen.dart # Slot machine interface
│   ├── ingredient_mode_screen.dart # Ingredient search
│   ├── saved_recipes_screen.dart # Recipe collections
│   ├── settings_screen.dart # App configuration
│   └── recipe_detail_screen.dart # Detailed recipe view
├── widgets/                 # Reusable UI components
│   ├── slot_machine_widget.dart # Custom slot machine
│   ├── recipe_card_stack.dart # Swipeable card stack
│   ├── recipe_card.dart     # Individual recipe card
│   └── recipe_list_item.dart # List view recipe item
└── theme/                   # App theming
    └── app_theme.dart       # Dark theme configuration
```

Platform Support

Android

- **Minimum SDK**: API 21 (Android 5.0)
- **Target SDK**: Latest stable
- **Permissions**: Internet access for API calls
- **Build Configuration**: Gradle-based build system

iOS

- **Minimum Version**: iOS 11.0
- **Architecture**: Universal (ARM64 + x86_64)

- **Capabilities**: Network access
- **Build Configuration**: Xcode project with Swift

Dependencies

Core Dependencies

- `flutter_riverpod: ^2.4.9` - State management
- `hive: ^2.2.3` - Local database
- `hive_flutter: ^1.1.0` - Flutter integration for Hive
- `dio: ^5.4.0` - HTTP client for API calls

UI Components

- `flutter_card_swiper: ^7.0.2` - Tinder-like swipe cards
- `slot_machine_roller: ^1.0.0` - Slot machine animations
- `cached_network_image: ^3.3.1` - Image caching and loading

Utilities

- `shared_preferences: ^2.2.2` - Simple key-value storage
- `path_provider: ^2.1.2` - File system paths
- `logger: ^2.0.2+1` - Logging utility

Development

- `build_runner: ^2.4.7` - Code generation
- `json_serializable: ^6.7.1` - JSON serialization
- `hive_generator: ^2.0.1` - Hive adapter generation
- `flutter_lints: ^3.0.0` - Linting rules

Getting Started

Prerequisites

1. **Flutter SDK** (3.0.0 or higher)
2. **Android Studio** or **Xcode** for mobile development
3. **Spoonacular API Key** (free at spoonacular.com (https://spoonacular.com/food-api))

Installation Steps

1. **Clone and Setup**:
   ```
   bash
      cd recipe_slot_app
      chmod +x setup.sh
      ./setup.sh
   ```

2. **Manual Setup** (if script fails):
   ```
   bash
      flutter pub get
      flutter packages pub run build_runner build
      flutter analyze
      flutter test
   ```

3. **Run the App**:
   `bash`
   `flutter run`

Configuration

1. **API Key Setup**:
   - Open the app
   - Navigate to Settings tab
   - Enter your Spoonacular API key
   - Configure dietary preferences

2. **Development Setup**:
   - For hot reload: `flutter run`
   - For debugging: Use your IDE's debug configuration
   - For profiling: `flutter run --profile`

API Integration

Spoonacular API Features Used

- **Random Recipes**: `/recipes/random` endpoint
- **Ingredient Search**: `/recipes/findByIngredients` endpoint
- **Recipe Details**: `/recipes/{id}/information` endpoint
- **Dietary Filters**: Support for diets, allergies, and cuisine preferences

API Service Architecture

- **Error Handling**: Comprehensive error catching and user feedback
- **Rate Limiting**: Respectful API usage patterns
- **Caching**: Local storage of fetched recipes
- **Offline Support**: Graceful degradation when offline

Testing

Test Coverage

- **Widget Tests**: UI component testing
- **Unit Tests**: Business logic validation
- **Integration Tests**: End-to-end user flows

Running Tests

```
flutter test                       # Run all tests
flutter test test/widget_test.dart  # Run specific test file
flutter test --coverage           # Generate coverage report
```

Building for Production

Android Release

```
flutter build apk --release          # APK for distribution
flutter build appbundle --release    # AAB for Play Store
```

iOS Release

```
flutter build ios --release          # iOS build
# Then use Xcode to archive and distribute
```

Security Considerations

API Key Management

- API keys stored securely in local storage
- No hardcoded secrets in source code
- User-provided API key configuration

Data Privacy

- All user data stored locally
- No personal information sent to external services
- Recipe preferences kept private

Troubleshooting

Common Issues

1. **Flutter Not Found**:
   `bash`
   ```
   export PATH="$PATH:[PATH_TO_FLUTTER_GIT_DIRECTORY]/flutter/bin"
   ```

2. **Build Errors**:
   `bash`
   ```
   flutter clean
   flutter pub get
   flutter packages pub run build_runner build --delete-conflicting-outputs
   ```

3. **API Issues**:
   - Verify API key is correct
   - Check internet connection
   - Ensure API quota not exceeded

Performance Optimization

- **Image Caching**: Automatic with `cached_network_image`
- **State Management**: Efficient with Riverpod
- **Database**: Fast local storage with Hive
- **Memory Management**: Proper widget disposal

Contributing

Development Workflow

1. Fork the repository
2. Create a feature branch
3. Make changes with tests
4. Run `flutter analyze` and `flutter test`
5. Submit a pull request

Code Style

- Follow Dart/Flutter conventions
- Use `flutter format` for consistent formatting
- Add documentation for public APIs
- Write tests for new features

License

Acknowledgments

- **Spoonacular API** for comprehensive recipe data
- **Flutter Team** for the amazing framework
- **Open Source Community** for excellent packages:
- Riverpod for state management
- Hive for local storage
- Card Swiper for gesture interactions
- Slot Machine Roller for animations

Support

For issues and questions:
1. Check the troubleshooting section
2. Review Flutter documentation
3. Check Spoonacular API documentation
4. Create an issue in the repository

Happy Cooking!