AniListify

Group Leader:

Alexander Paul S. Lunas

Members:

Brix G. Mendoza

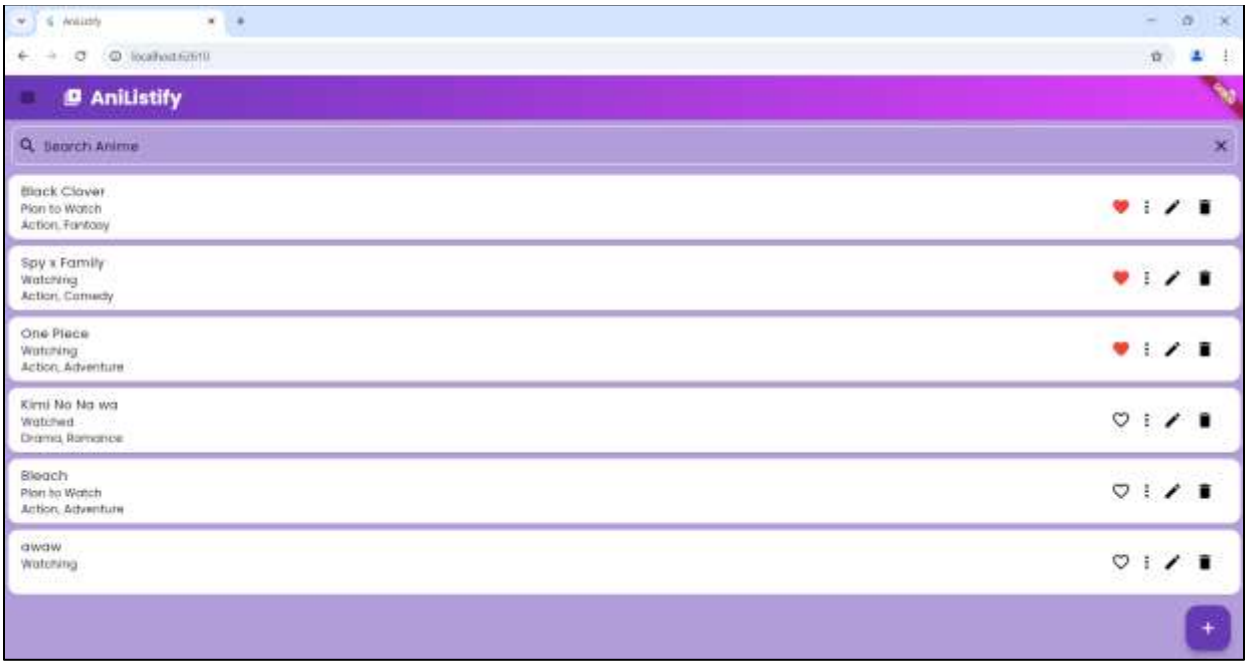
Jhonard Adam D. Mustacisa

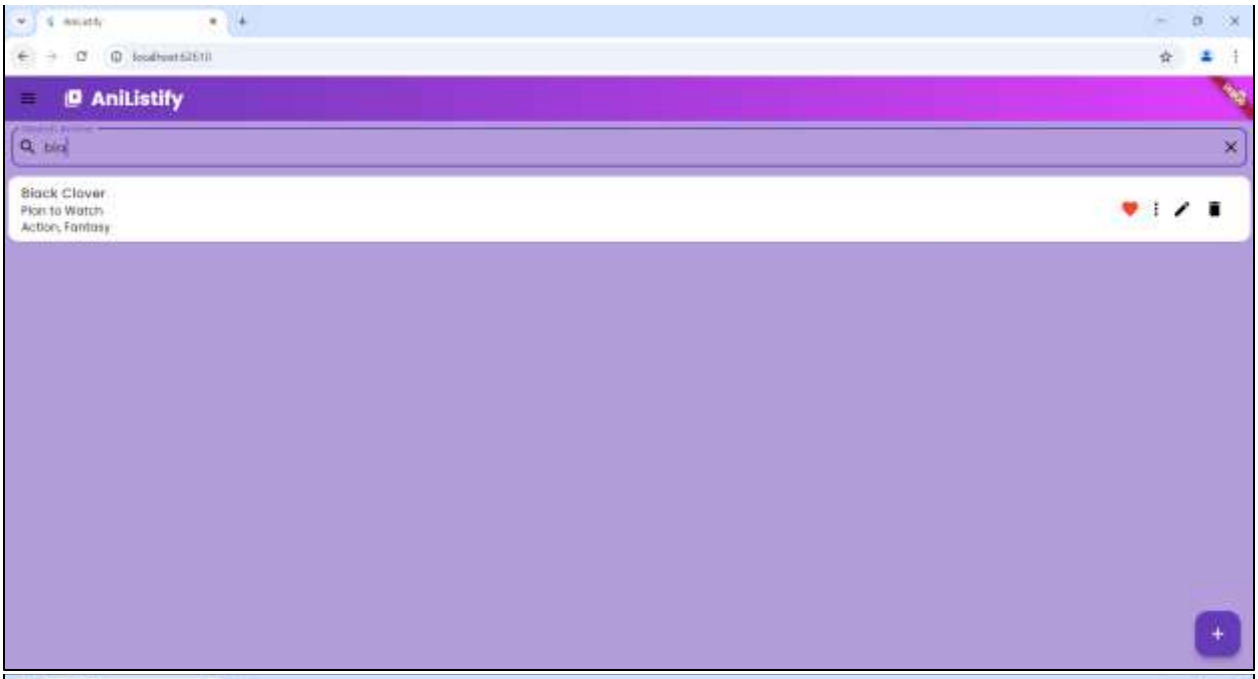
Jonh Cristhian G. Mariquina

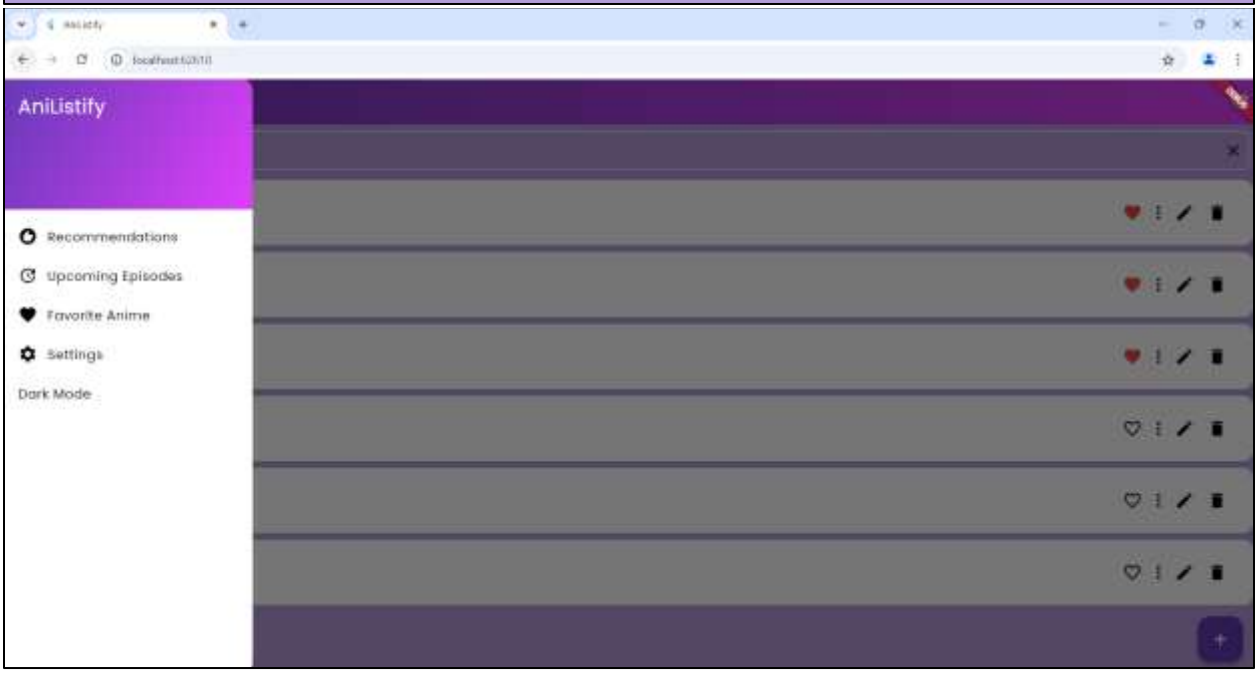
SUMMARY OF APPLICATION

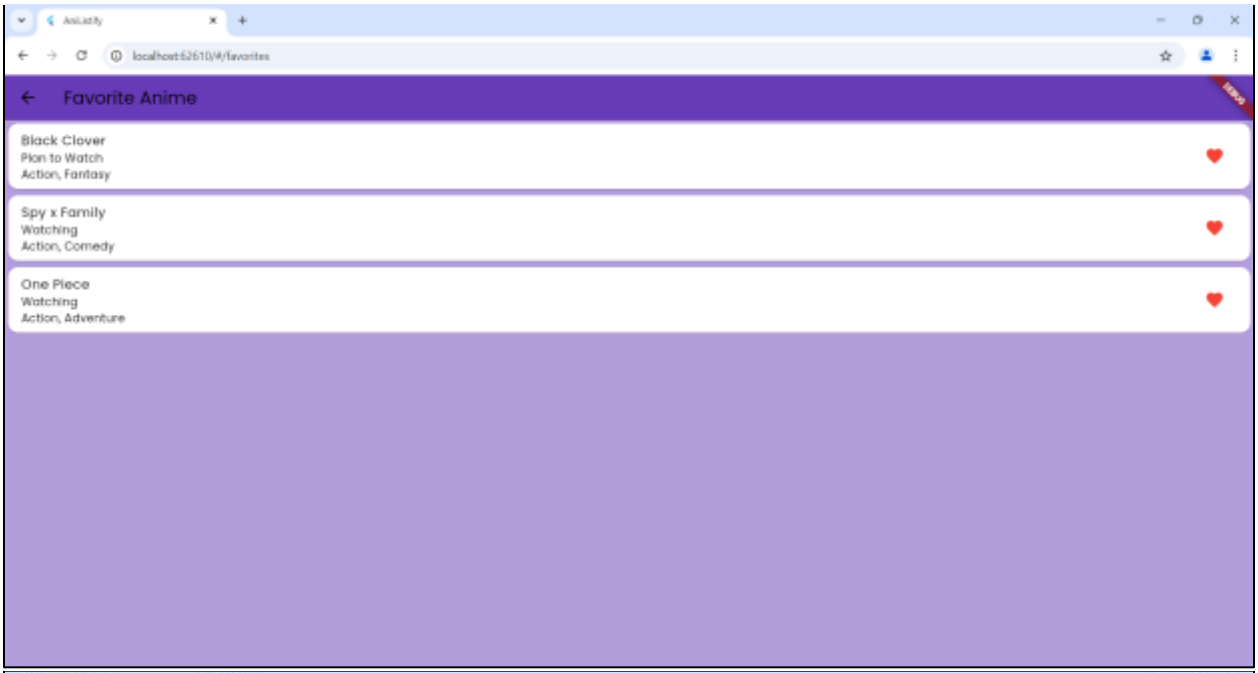
"**AniListify**" is a simple and user-friendly mobile app designed for anime enthusiasts to keep track of the anime series they have watched, are currently watching, or plan to watch. The app allows users to manage their anime list, receive personalized recommendations, and stay updated on upcoming episodes or new seasons.

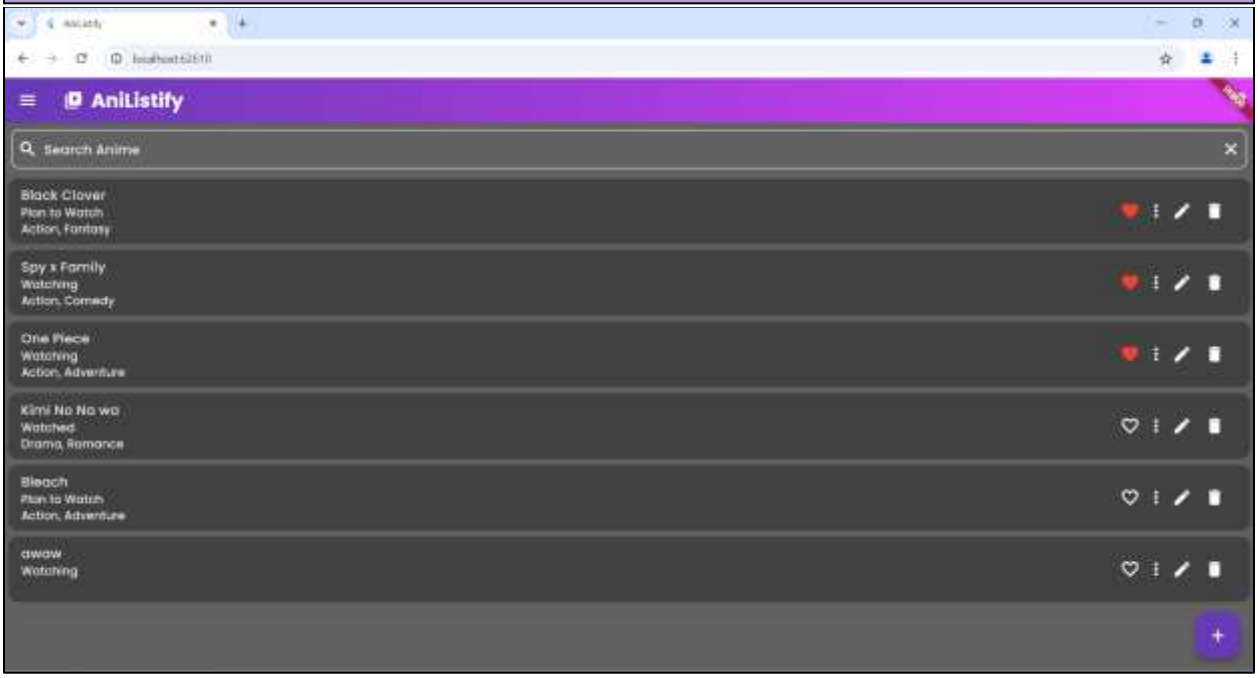
Screenshots of the application:

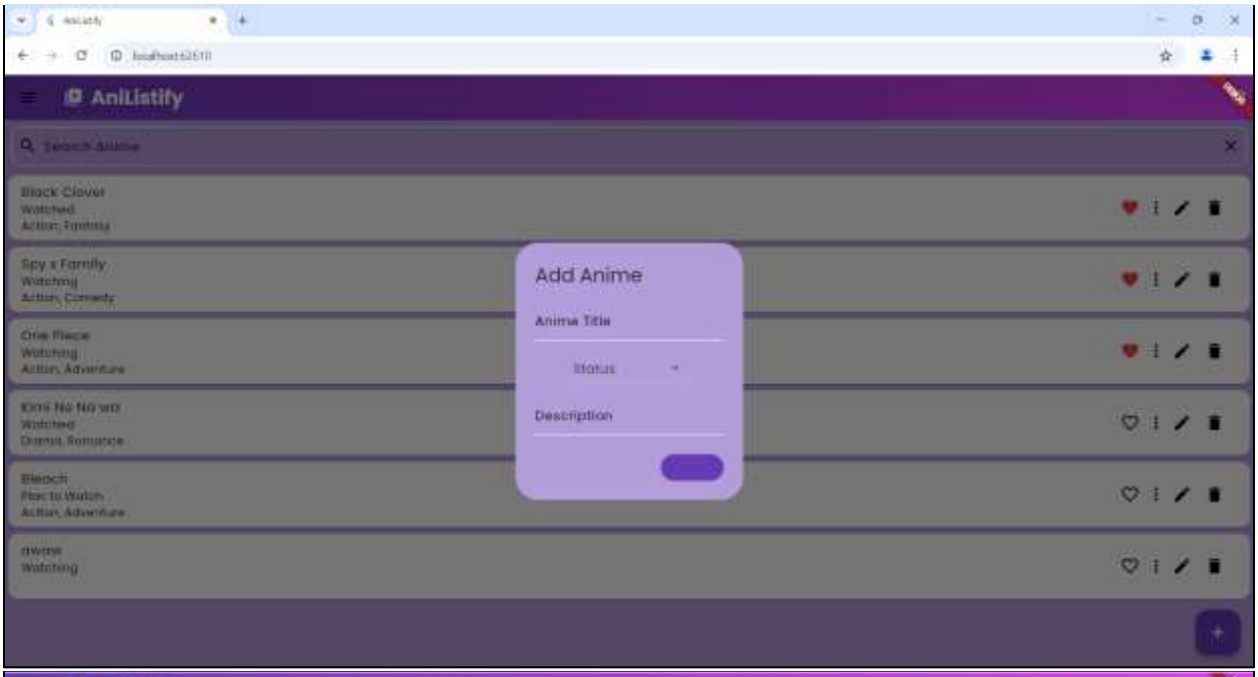


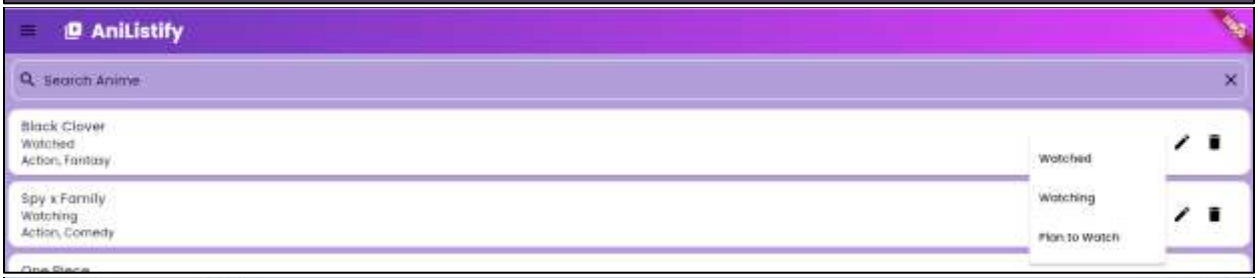

AniListify
blac
Black Clover
Plan to Watch
Action, Fantasy

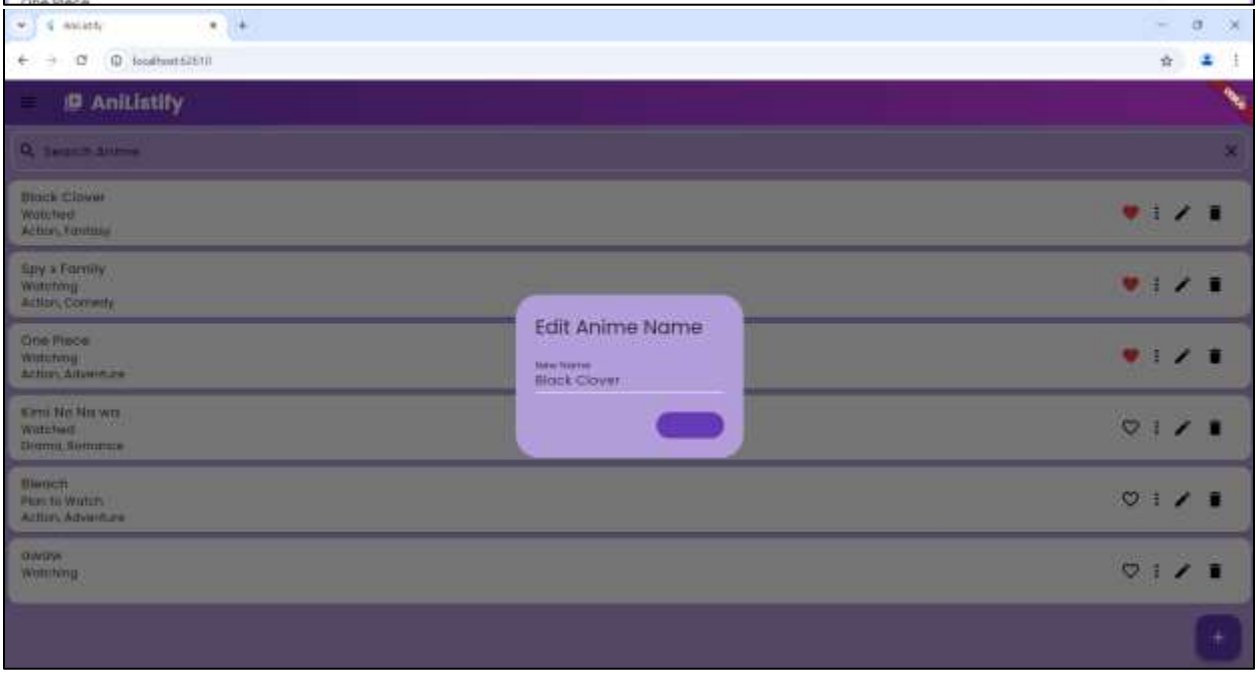

AniListify
Recommendations
Upcoming Episodes
Favorite Anime
Settings
Dark Mode

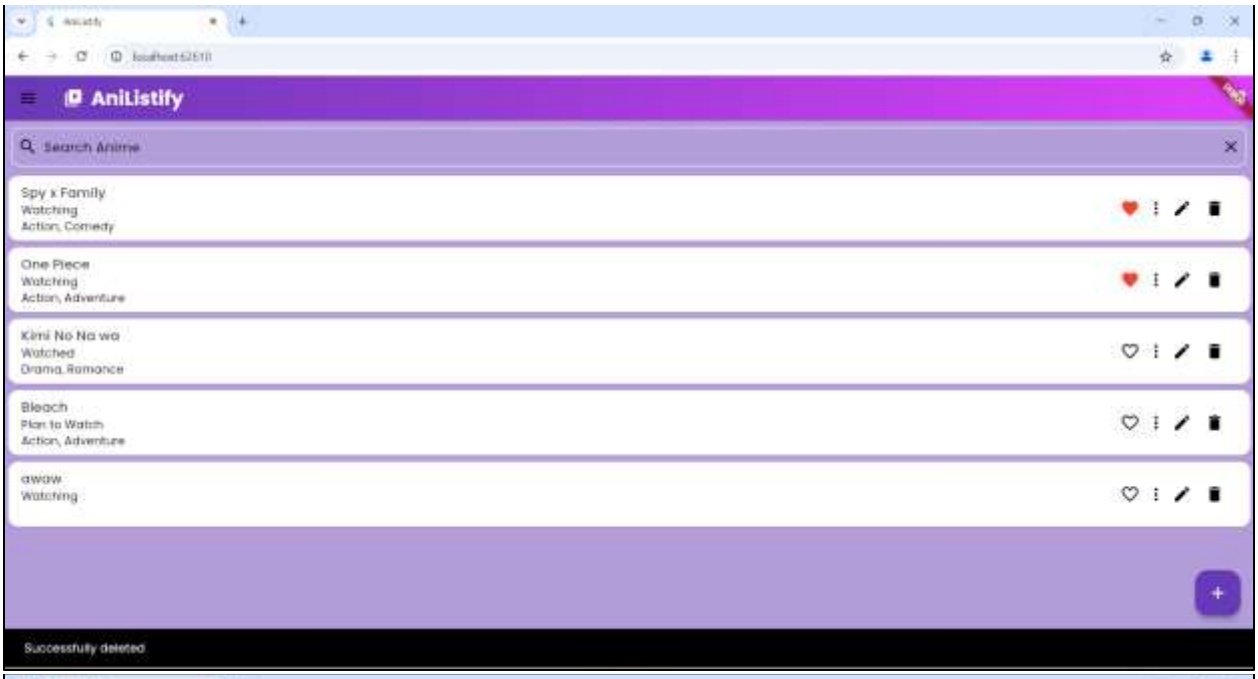

Favorite Anime
Black Clover
Plan to Watch
Action, Fantasy
Spy x Family
Watching
Action, Comedy
One Piece
Watching
Action, Adventure

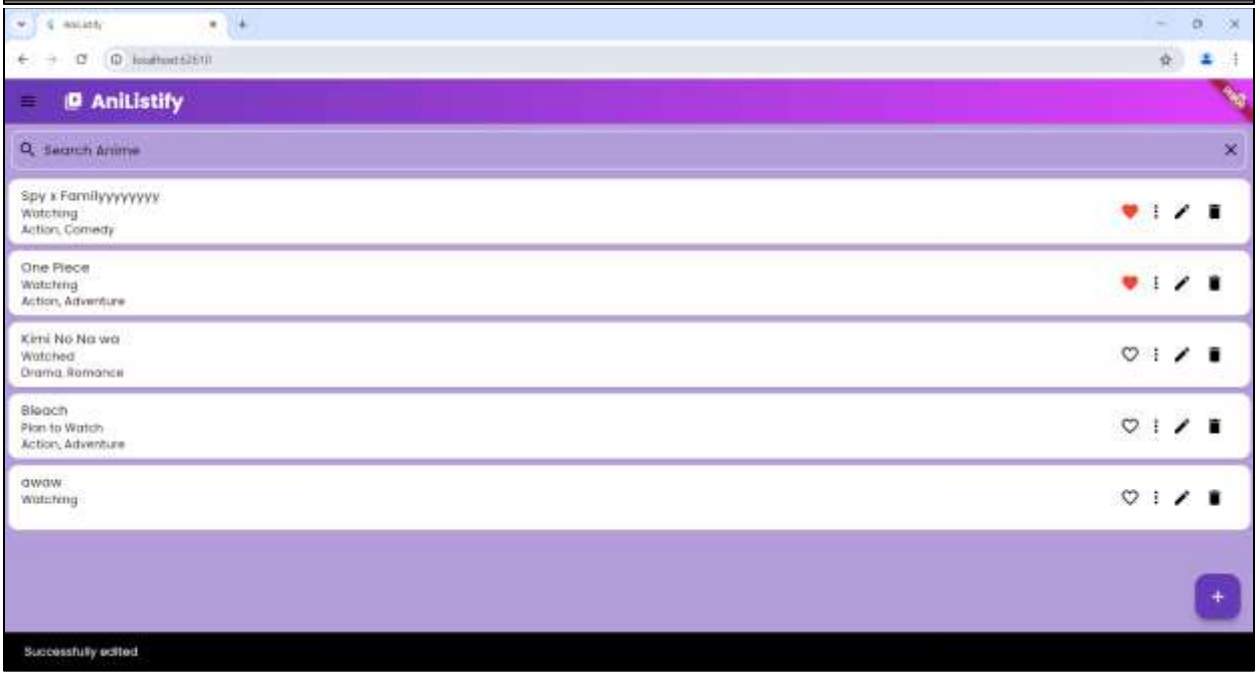

AniListify
Search Anime
Black Clover
Plan to Watch
Action, Fantasy
Spy x Family
Watching
Action, Comedy
One Piece
Watching
Action, Adventure
Kimi No Na wa
Watched
Drama, Romance
Bleach
Plan to Watch
Action, Adventure
awaw
Watching

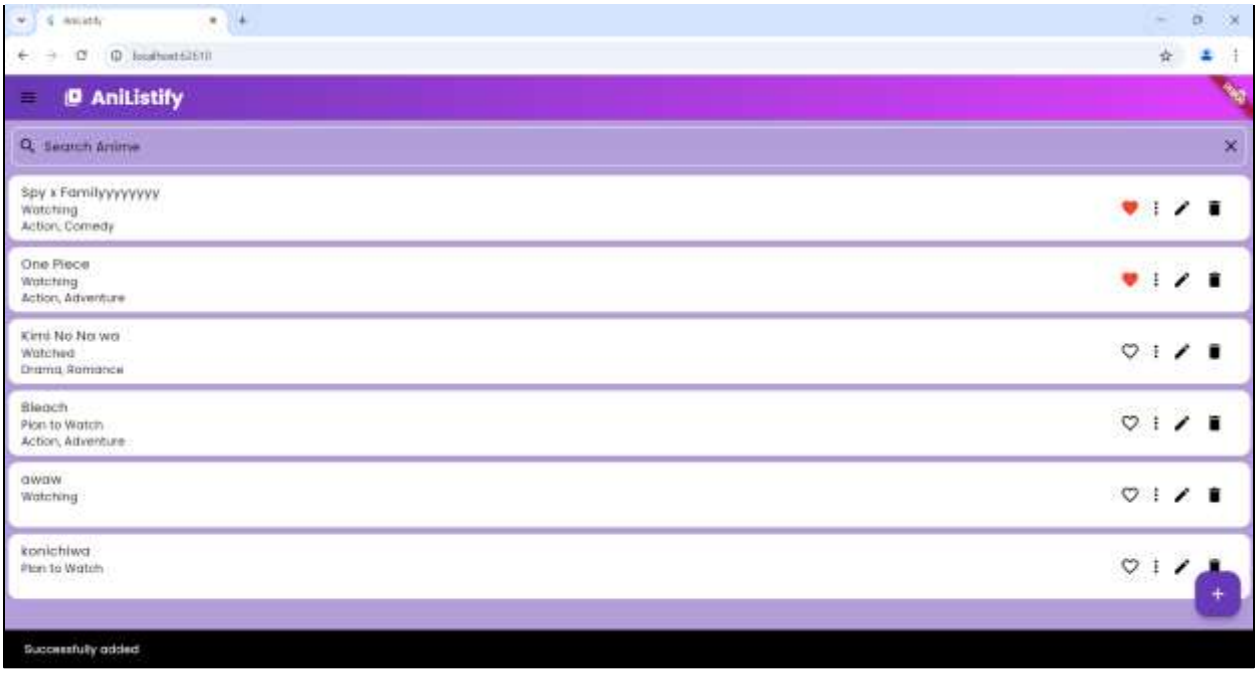

AniListify
Search Anime
Black Clover
Watched
Action, Fantasy
Spy x Family
Watching
Action, Comedy
One Piece
Watching
Action, Adventure
Kimi No Na wa
Watched
Drama, Romance
Bleach
Plan to Watch
Action, Adventure
awaw
Watching
Add Anime
Anime Title
Status
Description


AniListify
Search Anime
Black Clover
Watched
Action, Fantasy
Spy x Family
Watching
Action, Comedy
Watched
Watching
Plan to Watch


AniListify
Black Clover
Watched
Action, Fantasy
Spy x Family
Watching
Action, Comedy
One Piece
Watching
Action, Adventure
Kimi No Na wa
Watched
Drama, Romance
Bleach
Plan to Watch
Action, Adventure
awaw
Watching
Edit Anime Name
Black Clover

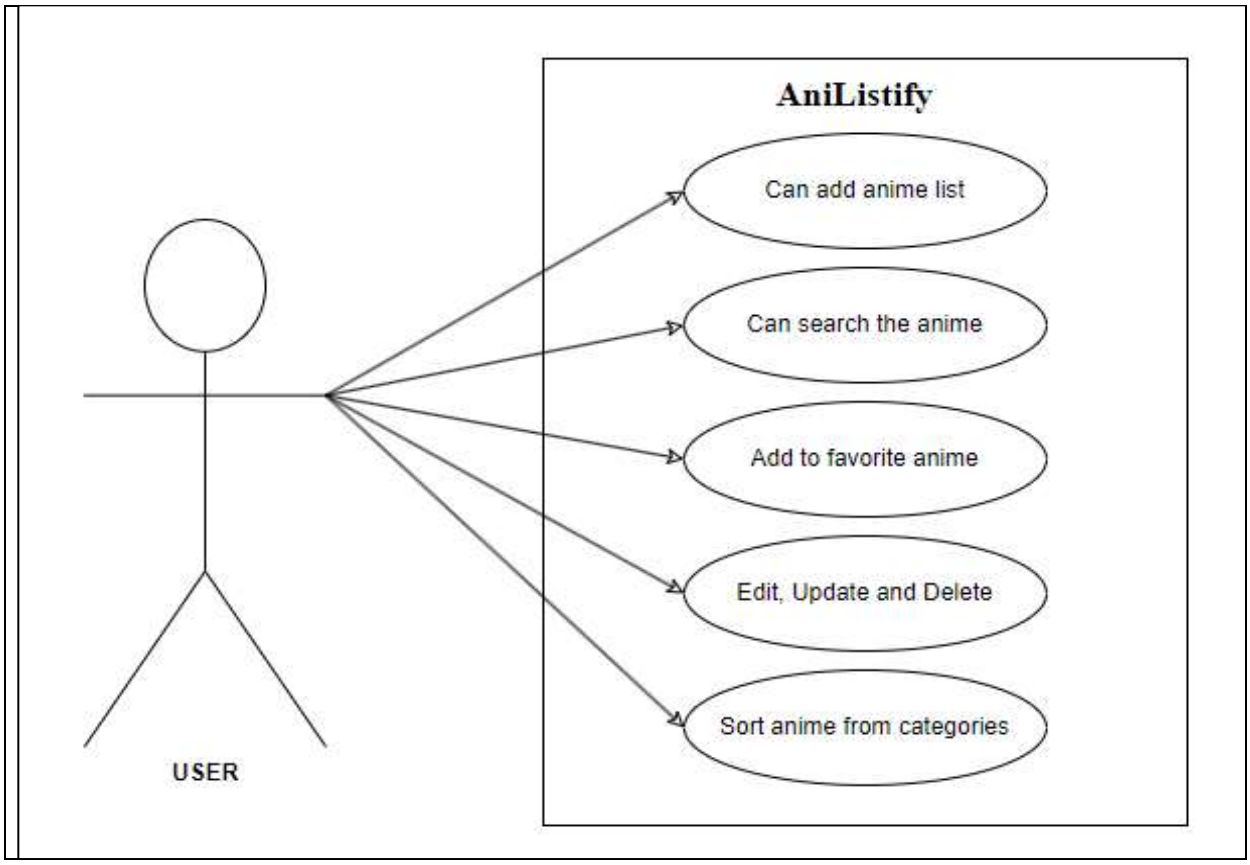

AniListify
Search Anime
Spy x Family
Watching
Action, Comedy
One Piece
Watching
Action, Adventure
Kimi No Na wa
Watched
Drama, Romance
Bleach
Plan to Watch
Action, Adventure
awaw
Watching
Successfully deleted


AniListify
Search Anime
Spy x Familyyyyyyyy
Watching
Action, Comedy
One Piece
Watching
Action, Adventure
Kimi No Na wa
Watched
Drama, Romance
Bleach
Plan to Watch
Action, Adventure
awaw
Watching
Successfully edited



Use Case Diagram

1. Use case of the user.



Source code of application

```
name: anilistify
description: "A new Flutter project."
publish_to: 'none'  # Prevents accidental
publication to pub.dev

version: 1.0.0+1

environment:
  sdk: '>=3.4.4 <4.0.0'

dependencies:
  flutter:
    sdk: flutter
  google_fonts: ^6.0.0

  cupertino_icons: ^1.0.6
  provider: ^6.0.4  # State management
package

dev_dependencies:
  flutter_test:
    sdk: flutter

  flutter_lints: ^3.0.0

flutter:
  uses-material-design: true

  # Uncomment and list the assets below
if you have assets to include:
  # assets:
  #   - images/anilistify_logo.png

  # Uncomment and list the fonts below if
you have custom fonts:
  # fonts:
  #   - family: CustomFont
  #     fonts:
  #       - asset: fonts/CustomFont-
Regular.ttf
  #       - asset: fonts/CustomFont-
Bold.ttf
  #         weight: 700
```


```
    import
'package:flutter/material.dart';
    import
'package:google_fonts/google_fonts.dart';
    import
'package:provider/provider.dart';
    import 'package:http/http.dart' as
http;
    import 'dart:convert';

    void main() {
      runApp(AniListifyApp());
    }

    class Anime {
      final String title;
      final String status;
      final List<String> genres;
      final double rating;
      final String description; // Added
description field
      bool isFavorite; // Added
isFavorite field

      Anime(this.title, this.status,
          {this.genres = const [],
this.rating = 0.0, this.description = '',
this.isFavorite = false});

      Anime copyWith({
        String? title,
        String? status,
        List<String>? genres,
        double? rating,
        String? description,
        bool? isFavorite,
      }) {
        return Anime(
          title ?? this.title,
          status ?? this.status,
          genres: genres ?? this.genres,
```

```
12:32
        return Anime(
          title ?? this.title,
          status ?? this.status,
          genres: genres ?? this.genres,
          rating: rating ?? this.rating,
          description: description ??
this.description,
          isFavorite: isFavorite ??
this.isFavorite,
        );
      }
    }

    class AnimeListModel extends
ChangeNotifier {
      final List<Anime> _animeList = [
        Anime('Naruto', 'Watching',
genres: ['Action', 'Adventure'], rating:
8.5, description: 'A story about a
ninja.'),
        Anime('Dragon Ball', 'Watching',
genres: ['Action', 'Adventure'], rating:
9.0, description: 'The adventures of
Goku.'),
        Anime('Black Clover', 'Plan to
Watch', genres: ['Action', 'Fantasy'],
rating: 7.5, description: 'A boy aiming
to be the wizard king.'),
        Anime('Spy x Family', 'Watching',
genres: ['Action', 'Comedy'], rating:
8.0, description: 'A spy with a fake
family.'),
        Anime('One Piece', 'Watching',
genres: ['Action', 'Adventure'], rating:
9.5, description: 'Pirates searching for
the ultimate treasure.'),
        Anime('Kimi No Na wa', 'Watched',
genres: ['Drama', 'Romance'], rating:
9.2, description: 'Two teenagers swap
bodies.'),
        Anime('Bleach', 'Plan to Watch',
genres: ['Action', 'Adventure'], rating:
8.0, description: 'A soul reaper fighting
hollows.'),
      ];
12:32
      String _searchQuery = '';

      List<Anime> get animeList =>
_searchQuery.isEmpty
          ? _animeList
          : _animeList.where((anime) =>
anime.title.toLowerCase().contains(_searc
hQuery.toLowerCase())).toList();
      List<Anime> get favoriteAnime =>
_animeList.where((anime) =>
anime.isFavorite).toList();

      void addAnime(Anime anime) {
        _animeList.add(anime);
        notifyListeners();
      }

      void removeAnime(Anime anime) {
        _animeList.remove(anime);
        notifyListeners();
      }

      void updateAnime(Anime anime,
String newStatus) {
        final updatedAnime =
anime.copyWith(status: newStatus);
        final index =
_animeList.indexOf(anime);
        if (index != -1) {
          _animeList[index] =
updatedAnime;
          notifyListeners();
        }
      }

      void editAnimeName(Anime anime,
String newName) {
        final updatedAnime =
anime.copyWith(title: newName);
        final index =
_animeList.indexOf(anime);
        if (index != -1) {
          _animeList[index] =
updatedAnime;
```

```
updatedAnime;
          notifyListeners();
        }
      }

      void setSearchQuery(String query) {
        _searchQuery = query;
        notifyListeners();
      }

      void toggleFavorite(Anime anime) {
        final index =
_animeList.indexOf(anime);
        if (index != -1) {
          _animeList[index] =
anime.copyWith(isFavorite:
!anime.isFavorite);
          notifyListeners();
        }
      }
    }

    class ThemeNotifier extends
ChangeNotifier {
      bool _isDarkMode = false;

      bool get isDarkMode => _isDarkMode;

      void toggleTheme() {
        _isDarkMode = !_isDarkMode;
        notifyListeners();
      }
    }

    class AniListifyApp extends
StatelessWidget {
      @override
      Widget build(BuildContext context)
{
        return MultiProvider(
          providers: [

ChangeNotifierProvider(create: (_) =>
ThemeNotifier()),
ChangeNotifierProvider(create: (_) =>
AnimeListModel()),
          ],
          child: Consumer<ThemeNotifier>(
            builder: (context,
themeNotifier, child) {
              return MaterialApp(
                title: 'AniListify',
                theme: ThemeData(
                  colorScheme:
ColorScheme.fromSwatch(
                    primarySwatch:
Colors.deepPurple,
                    brightness:
themeNotifier.isDarkMode ?
Brightness.dark : Brightness.light, //
Set the brightness here
                  ).copyWith(secondary:
Colors.amber),
                  textTheme:
GoogleFonts.poppinsTextTheme(

Theme.of(context).textTheme,
                  ),
                  brightness:
themeNotifier.isDarkMode ?
Brightness.dark : Brightness.light, //
Ensure brightness matches
                ),
                home: AnimeListScreen(),
                routes: {
                  '/recommendations':
(context) => RecommendationsScreen(),
                  '/upcoming': (context)
=> UpcomingEpisodesScreen(),
                  '/favorites': (context)
=> FavoriteAnimeScreen(),
                  '/settings': (context)
=> SettingsScreen(),
                },
              );
            },
          ),
        );
```

```
12:32
    }
  }

  class AnimeListScreen extends
StatefulWidget {
    @override
    _AnimeListScreenState createState()
=> _AnimeListScreenState();
  }

  class _AnimeListScreenState extends
State<AnimeListScreen> {
    final TextEditingController
_titleController =
TextEditingController();
    final TextEditingController
_descriptionController =
TextEditingController();
    final TextEditingController
_searchController =
TextEditingController();
    String? _selectedStatus;

    void _showEditDialog(Anime anime) {
      final TextEditingController
_editController =
TextEditingController(text: anime.title);

      showDialog(
        context: context,
        builder: (context) {
          return AlertDialog(
            title: Text('Edit Anime
Name'),
            content: TextField(
              controller:
_editController,
              decoration:
InputDecoration(labelText: 'New Name'),
            ),
            actions: [
              ElevatedButton(
                onPressed: () {
                  if
(_editController.text.isNotEmpty) {

context.read<AnimeListModel>
().editAnimeName(anime,
_editController.text);

Navigator.of(context).pop();

ScaffoldMessenger.of(context).showSnackBa
r(
                      SnackBar(
                        content:
Text('Successfully edited'),
                      ),
                    );
                  }
                },
                child: Text('Save'),
                style:
ElevatedButton.styleFrom(
                  backgroundColor:
Colors.deepPurple,
                ),
              ),
            ],
          );
        },
      );
    }

    void _showAddAnimeDialog() {
      showDialog(
        context: context,
        builder: (context) {
          return AlertDialog(
            title: Text('Add Anime'),
            content: Column(
              mainAxisSize:
MainAxisSize.min,
              children: [
                TextField(
                  controller:
_titleController,
                  decoration:
```

```
InputDecoration(labelText: 'Anime
Title'),
                ),
                SizedBox(height: 10),
                DropdownButton<String>(
                  value:
_selectedStatus,
                  items: <String>
['Watched', 'Watching', 'Plan to Watch']
                      .map((String
value) {
                    return
DropdownMenuItem<String>(
                      value: value,
                      child:
Text(value),
                    );
                  }).toList(),
                  onChanged: (String?
newValue) {
                    setState(() {
                      _selectedStatus =
newValue;
                    });
                  },
                  hint: Text('Status'),
                ),
                SizedBox(height: 10),
                TextField(
                  controller:
_descriptionController,
                  decoration:
InputDecoration(labelText:
'Description'),
                ),
              ],
            ),
            actions: [
              ElevatedButton(
                onPressed: () {
                  if (_selectedStatus
!= null &&
_titleController.text.isNotEmpty) {
```

```
context.read<AnimeListModel>().addAnime(

Anime(_titleController.text,
_selectedStatus!,

description:
_descriptionController.text));

_titleController.clear();

_descriptionController.clear();
                      setState(() {
                        _selectedStatus =
null;
                      });

Navigator.of(context).pop();

ScaffoldMessenger.of(context).showSnackBa
r(
                        SnackBar(
                          content:
Text('Successfully added'),
                        ),
                      );
                    }
                  },
                  child: Text('Add'),
                  style:
ElevatedButton.styleFrom(
                    backgroundColor:
Colors.deepPurple,
                  ),
                ),
              ],
            );
          },
        );
      }

      void _toggleFavorite(Anime anime) {
        context.read<AnimeListModel>
().toggleFavorite(anime);
      }
```

```
12:35
      void _toggleFavorite(Anime anime) {
        context.read<AnimeListModel>
().toggleFavorite(anime);
      }

      @override
      Widget build(BuildContext context) {
        return Scaffold(
          appBar: AppBar(
            flexibleSpace: Container(
              decoration: BoxDecoration(
                gradient: LinearGradient(
                  colors:
[Colors.deepPurple, Colors.purpleAccent],
                  begin:
Alignment.topLeft,
                  end:
Alignment.bottomRight,
                ),
              ),
            ),
            title: Row(
              children: [
                Icon(Icons.video_library,
color: Colors.white),
                SizedBox(width: 10),
                Text(
                  'AniListify',
                  style:
GoogleFonts.poppins(
                    fontSize: 24,
                    fontWeight:
FontWeight.bold,
                    color: Colors.white,
                  ),
                ),
              ],
            ),
            actions: [
              IconButton(
                icon: Icon(Icons.favorite,
color: Colors.white),
                onPressed: () {
                  Navigator.push(
```

```
12:35
                  Navigator.push(
                    context,

MaterialPageRoute(builder: (context) =>
FavoriteAnimeScreen()),
                  );
                },
              ),
            ],
          ),
          body: Consumer<AnimeListModel>(
            builder: (context,
animeListModel, child) {
              final animeList =
animeListModel.animeList;

              return Column(
                children: [
                  Padding(
                    padding: const
EdgeInsets.all(8.0),
                    child: TextField(
                      controller:
_searchController,
                      decoration:
InputDecoration(
                        labelText: 'Search
Anime',
                        prefixIcon:
Icon(Icons.search),
                        suffixIcon:
IconButton(
                          icon:
Icon(Icons.clear),
                          onPressed: () {

_searchController.clear();

context.read<AnimeListModel>
().setSearchQuery('');
                          },
                        ),
                        border:
OutlineInputBorder(
```

```
                        borderRadius:
BorderRadius.circular(10),
                      ),
                    ),
                    onChanged: (query) {

context.read<AnimeListModel>
().setSearchQuery(query);
                    },
                  ),
                ),
                Expanded(
                  child:
ListView.builder(
                    itemCount:
animeList.length,
                    itemBuilder:
(context, index) {
                      final anime =
animeList[index];
                      return Card(
                        child: ListTile(
                          title:
Text(anime.title),
                          subtitle:
Text('${anime.status}\n${anime.genres.join
(', ')}'),
                          trailing: Row(

mainAxisSize: MainAxisSize.min,
                            children: [

IconButton(
                                icon:
Icon(

anime.isFavorite ? Icons.favorite :
Icons.favorite_border,
                                  color:
anime.isFavorite ? Colors.red : null,
                                ),

onPressed: () => _toggleFavorite(anime),
                              ),
```

```
PopupMenuButton<String>(

onSelected: (String newValue) {

context.read<AnimeListModel>
().updateAnime(anime, newValue);

ScaffoldMessenger.of(context).showSnackBar
(

SnackBar(

content: Text('Status updated'),
                                    ),
                                  );
                                },

itemBuilder: (BuildContext context) {
                                  return
<String>['Watched', 'Watching', 'Plan to
Watch']

.map((String value) {

return PopupMenuItem<String>(

value: value,

child: Text(value),
                                    );

}).toList();
                                },
                                child:
Icon(Icons.more_vert),
                              ),

IconButton(
                                icon:
Icon(Icons.edit),

onPressed: () => _showEditDialog(anime),
                              ),
```


1:01
Anilistify_main_...
content://media/extern
onPressed: () => _showEditDialog(anime),
),
IconButton(
icon:
Icon(Icons.delete),
onPressed: () {
context.read<AnimeListModel>
().removeAnime(anime);
ScaffoldMessenger.of(context).showSnackBar
(
SnackBar(
content: Text('Successfully deleted'),
),
);
},
),
],
),
),
);
},
),
),
],
);
},
),
floatingActionButton:
FloatingActionButton(
onPressed:
_showAddAnimeDialog,
child: Icon(Icons.add),
backgroundColor:
Colors.deepPurple,
1:02
Colors.deepPurple,
),
drawer: Drawer(
child: ListView(
padding: EdgeInsets.zero,
children: <Widget>[
DrawerHeader(
child: Text(
'AniListify',
style: TextStyle(
color: Colors.white,
fontSize: 24,
),
),
decoration:
BoxDecoration(
gradient:
LinearGradient(
colors:
[Colors.deepPurple, Colors.purpleAccent],
begin:
Alignment.topLeft,
end:
Alignment.bottomRight,
),
),
),
ListTile(
leading:
Icon(Icons.recommend),
title:
Text('Recommendations'),
onTap: () {
Navigator.pushNamed(context,
'/recommendations');
},
),
ListTile(
leading:
Icon(Icons.update),
title: Text('Upcoming
Episodes'),
onTap: () {

1:02

```
Navigator.pushNamed(context, '/upcoming');
                  },
                ),
                ListTile(
                  leading:
Icon(Icons.favorite),
                  title: Text('Favorite
Anime'),
                  onTap: () {

Navigator.pushNamed(context,
'/favorites');
                  },
                ),
                ListTile(
                  leading:
Icon(Icons.settings),
                  title: Text('Settings'),
                  onTap: () {

Navigator.pushNamed(context, '/settings');
                  },
                ),
                Consumer<ThemeNotifier>(
                  builder: (context,
themeNotifier, child) {
                    return SwitchListTile(
                      title: Text('Dark
Mode'),
                      value:
themeNotifier.isDarkMode,
                      onChanged: (value) {

themeNotifier.toggleTheme();
                      },
                    );
                  },
                ),
              ],
            ),
          ),
        );
      }
    }
```

1:03

```
    class RecommendationsScreen extends
StatelessWidget {
      Future<List<Anime>>
fetchRecommendations() async {
        final response = await
http.get(Uri.parse('https://api.example.co
m/recommendations'));

        if (response.statusCode == 200) {
          final data =
json.decode(response.body);
          return (data['recommendations']
as List)
              .map((item) => Anime(
            item['title'],
            item['status'],
            genres:
List<String>.from(item['genres']),
            rating:
item['rating'].toDouble(),
          ))
              .toList();
        } else {
          throw Exception('Failed to load
recommendations');
        }
      }

      @override
      Widget build(BuildContext context) {
        return Scaffold(
          appBar: AppBar(
            title:
Text('Recommendations'),
            backgroundColor:
Colors.deepPurple,
          ),
          body: FutureBuilder<List<Anime>>
(
            future:
fetchRecommendations(),
            builder: (context, snapshot) {
              if (snapshot.connectionState
== ConnectionState.waiting) {
```

```
                return Center(child:
CircularProgressIndicator());
              } else if
(snapshot.hasError) {
                return Center(child:
Text('Error: ${snapshot.error}'));
              } else if (!snapshot.hasData
|| snapshot.data!.isEmpty) {
                return Center(child:
Text('No recommendations found.'));
              } else {
                final recommendations =
snapshot.data!;
                return ListView.builder(
                  itemCount:
recommendations.length,
                  itemBuilder: (context,
index) {
                    final anime =
recommendations[index];
                    return Card(
                      child: ListTile(
                        title:
Text(anime.title),
                        subtitle:
Text('${anime.status}\n${anime.genres.join
(', ')}'),
                      ),
                    );
                  },
                );
              }
            },
          ),
        );
      }
    }

    class UpcomingEpisodesScreen extends
StatelessWidget {
      @override
      Widget build(BuildContext context) {
        return Scaffold(
          appBar: AppBar(
```

```
    class UpcomingEpisodesScreen extends
StatelessWidget {
      @override
      Widget build(BuildContext context) {
        return Scaffold(
          appBar: AppBar(
            title: Text('Upcoming
Episodes'),
            backgroundColor:
Colors.deepPurple,
          ),
          body: Center(
            child: Text(
              'Upcoming Episodes Feature
Coming Soon!',
              style: TextStyle(fontSize:
20),
            ),
          ),
        );
      }
    }

    class SettingsScreen extends
StatelessWidget {
      @override
      Widget build(BuildContext context) {
        return Scaffold(
          appBar: AppBar(
            title: Text('Settings'),
            backgroundColor:
Colors.deepPurple,
          ),
          body: Center(
            child: Text(
              'Settings Feature Coming
Soon!',
              style: TextStyle(fontSize:
20),
            ),
          ),
        );
      }
    }
```