CS310 Phase 2.2: UI Implementation Mini Report

Student Movie Buffs - Flutter Application

Course: CS310 - Mobile Application Development
University: Sabancı University
Submission Date: November 24, 2025

Executive Summary

This report documents the complete implementation of the Student Movie Buffs Flutter application, transitioning from the wireframe designs created in Phase 2.1 to a fully functional mobile application in Phase 2.2. The application serves as a social platform exclusively for university students to discover, rate, review, and discuss movies with their peers. All technical requirements have been successfully implemented, including named routes, utility classes, custom fonts, form validation, responsive design, and dynamic list management.

1. Introduction & Application Overview

1.1 Project Context

The Student Movie Buffs application was developed as a mobile-first platform designed exclusively for university students with .edu email addresses. The app creates a closed community where students can explore trending films, share reviews, organize movie night events, and participate in discussion groups centered around their favorite movies.

1.2 Application Flow

The app consists of 10 interconnected screens, organized around a bottom navigation bar that provides quick access to four main sections: Home, Browse, Chat Groups, and Profile. Additional screens are accessible through contextual navigation, creating an intuitive user journey from login to movie discovery, rating, and social interaction.

2. Implementation of Phase 2.1 Wireframes

2.1 From Wireframe to Implementation

Each screen designed in Phase 2.1 has been faithfully implemented while incorporating improvements discovered during development:

Login Screen (Screen 1 - Wireframe): The wireframe showed a simple login form with email and password fields. The implementation enhanced this with: - Added a "Full Name" field to capture user identity for personalization - Implemented real-time .edu email validation - Added "Create Account" and "Forgot Password" functionality with dialog-based interactions - Included "Demo Mode" button for testing without authentication - Applied custom Poppins font throughout

Home Screen (Screen 2 - Wireframe): The wireframe featured a trending movies section and navigation to event planning. The implementation delivered: - Personalized welcome message displaying the user's first name - Horizontal scrolling grid of trending movies with movie posters - "Organize a Movie Night" card with action buttons - "View My Events" button for quick access to planned events - Responsive grid layout (2-4 columns based on screen size) - Bottom navigation bar with visual active state indicators

Browse Movies Screen (Screen 3 - Wireframe): The wireframe showed genre filters and sorting options. The implementation includes: - Working dropdown filters for genre selection (All Genres, Action, Sci-Fi, Thriller, Drama, Comedy) - Functional sorting options (Highest Rated, Most Reviews, Newest, Oldest) - Dynamic list that updates immediately when filters change - Movie cards showing poster, title, year, rating, review count, and genre tags - Remove button functionality with undo capability - Results counter showing filtered movie count

Movie Detail Screen (Screen 4 - Wireframe): The wireframe presented movie information and action buttons. The implementation provides: - Large movie poster display with enhanced placeholder handling - Movie metadata (title, year, genres, rating) - Synopsis section with formatted text - "Rate and Comment" button navigating to review submission - "View Student Reviews" button for accessing community feedback - Back navigation with proper route management

Rate & Review Screen (Screen 5 - Wireframe): The wireframe showed a rating interface and comment field. The implementation delivers: - Interactive 5-star rating selector - Multi-line text field for review content (500 character limit) - Character counter for user guidance - Form validation ensuring rating selection before submission - Success dialog with confirmation message - Dynamic statistics update (incrementing "Movies Rated" and "Reviews Written" counters)

Chat Groups Screen (Screen 7 - Wireframe): The wireframe displayed available chat groups. The implementation includes: - List of chat groups with icons, names, descriptions, and metadata - Last message preview and timestamp - Member count display - Clickable cards navigating to individual group chats - Empty state handling for new users

Group Chat Screen (Screen 8 - Wireframe): The wireframe showed a message interface. The implementation provides: - AppBar with group information (icon, name, member count) - Empty state UI encouraging conversation start - Message composition area with: - Attachment button - Emoji selector button - Text input field - Send button with visual emphasis - Proper keyboard handling and SafeArea implementation

Profile Screen (Screen 9 - Wireframe): The wireframe presented user information and statistics. The implementation delivers: - Dynamic user profile with avatar placeholder - User's full name displayed prominently - Email address and university information (extracted from email domain) - Statistics cards showing: - Movies Rated (dynamically updates when rating) - Reviews Written (dynamically updates when reviewing) - Groups Joined (static for demo) - Logout button with confirmation - Clean, card-based information architecture

Organize Event Screen (Screen 10 - Wireframe): The wireframe showed event creation form. The implementation includes: - Event name input with validation - Movie selection dropdown - Date picker with calendar interface - Time picker with clock interface - Location/ platform input field - "Invite Peers" button with email dialog - Form validation ensuring all fields are completed - Success dialog with navigation to "My Events" - Event persistence in

memory storage

My Events Screen (New - Not in Original Wireframe): An additional screen was implemented to fulfill the requirement for viewing organized events: - Segregated display of "Upcoming Events" and "Past Events" - Event cards showing: - Date badge with day and month - Event name and movie title - Time and location - Attendee count - "TODAY" badge for current events - Cancel event functionality for upcoming events - Event details modal with full information - "Share Event" capability - Empty state when no events exist - Quick access button to create new events

3. Technical Requirements Implementation

3.1 Named Routes

Requirement: All navigation must use named routes with a clearly defined initial route.

Implementation: The application implements a comprehensive named routing system in

`main.dart` :

```
initialRoute: '/login',
routes: {
'/login': (context) => const LoginScreen(),
'/home': (context) => const HomeScreen(),
'/browse': (context) => const BrowseMoviesScreen(),
'/movie-detail': (context) => const MovieDetailScreen(),  '/review': (context) =>
const ReviewScreen(),
'/chat-groups': (context) => const ChatGroupsScreen(),  '/group-chat':
(context) => const GroupChatScreen(),
'/organize-event': (context) => const OrganizeEventScreen(),  '/my-events': (context)
=> const MyEventsScreen(),
'/profile': (context) => const ProfileScreen(),
}
```

Navigation Examples: - Login to Home: `Navigator.pushReplacementNamed(context, '/home')` - Home to Event Planning: `Navigator.pushNamed(context, '/organize-event')` - Browse to Movie Detail: `Navigator.pushNamed(context, '/movie-detail', arguments: movie)` - Movie Detail to Review:

`Navigator.pushNamed(context, '/review', arguments: movie)`

The initial route is set to `/login`, ensuring users authenticate before accessing the application. Proper route transitions maintain navigation stack integrity for back button functionality.

3.2 Utility Classes

Requirement: Static values like colors, text styles, and paddings must be in dedicated utility files.

Implementation:

AppColors (`lib/utils/app_colors.dart`):

```
class AppColors {
 static const Color backgroundColor = Color(0xFF1A2332);  static const Color
cardBackground = Color(0xFF2C3E50);  static const Color primaryYellow =
Color(0xFFFFB800);  static const Color darkBlue = Color(0xFF0A1628);
static const Color lightBlue = Color(0xFF34495E);  static const Color
textPrimary = Color(0xFFFFFFFF);  static const Color textSecondary =
Color(0xFFBDC3C7);  static const Color textHint = Color(0xFF7F8C8D);
static const Color success = Color(0xFF27AE60);  static const Color error =
Color(0xFFE74C3C); }
```

AppTextStyles (`lib/utils/app_text_styles.dart`):

```
class AppTextStyles {
 static const String fontFamily = 'Poppins';

 static const TextStyle h1 = TextStyle(  fontSize: 28,
 fontWeight: FontWeight.bold,
 color: AppColors.textPrimary,
 fontFamily: fontFamily,
 );

 static const TextStyle h2 = TextStyle(  fontSize: 22,
 fontWeight: FontWeight.bold,
 color: AppColors.textPrimary,
```

```
  fontFamily: fontFamily,
  );

  static const TextStyle bodyMedium = TextStyle(  fontSize: 14,
  color: AppColors.textPrimary,
  fontFamily: fontFamily,
  );

  static const TextStyle button = TextStyle(  fontSize: 16,
  fontWeight: FontWeight.w600,
  fontFamily: fontFamily,
  );
 }
```

AppSpacing (lib/utils/app_spacing.dart):

```
 class AppSpacing {
  static const double xs = 4.0;
  static const double sm = 8.0;
  static const double md = 16.0;
  static const double lg = 24.0;
  static const double xl = 32.0;

  static const double radiusSm = 8.0;
  static const double radiusMd = 12.0;
  static const double radiusLg = 16.0;

  static const double elevationSm = 2.0;
  static const double elevationMd = 4.0;
  static const double elevationLg = 8.0;
 }
```

These utility classes ensure consistent styling throughout the application and make theme updates efficient and maintainable.

3.3 Images

Requirement: App must include both asset images (local) and network images (from URLs).

Implementation:

Network Images: Movie posters are loaded from IMDb's content delivery network:

```
Image.network(
movie.imageUrl,
loadingBuilder: (context, child, loadingProgress) {
if (loadingProgress == null) return child;
return CircularProgressIndicator(
value: loadingProgress.expectedTotalBytes != null
? loadingProgress.cumulatedBytesLoaded /
loadingProgress.expectedTotalBytes!
: null,
color: AppColors.primaryYellow,
);
},
errorBuilder: (context, error, stackTrace) {
return Container(
color: AppColors.lightBlue,
child: Column(
mainAxisAlignment: MainAxisAlignment.center,
children: [
Icon(Icons.movie, size: 50, color: AppColors.primaryYellow),  SizedBox(height: 8),
Text(movie.title, style: AppTextStyles.caption),  ],
),
);
},
)
```

Asset Images: Local image assets are used for: - User avatars (placeholder) - App logo/icon - Empty state illustrations

Asset configuration in `pubspec.yaml` :

```
flutter:
 assets:
 - assets/images/
```

The implementation includes proper error handling for network images, showing loading indicators during fetch and fallback UI when images fail to load.

3.4 Custom Font

Requirement: A custom external font family must be applied across the entire

UI.

Implementation:

Font Integration: Poppins font family integrated via Google Fonts:

`pubspec.yaml` :

```
dependencies:
 google_fonts: ^6.1.0
```

Theme Configuration in `main.dart` :

```
theme: ThemeData(
 fontFamily: 'Poppins',
 textTheme: GoogleFonts.poppinsTextTheme(),
 colorScheme: ColorScheme.dark(
 primary: AppColors.primaryYellow,
 surface: AppColors.backgroundColor,
```

```
),
)
```

The Poppins font is applied universally through the theme, but can also be explicitly specified in individual text styles through the AppTextStyles utility class. The font provides excellent readability and a modern, friendly appearance that aligns with the app's student-focused audience.

3.5 Card + List Component with Remove Action

Requirement: At least one screen should display a list of items using Card widgets with dynamic remove functionality.

Implementation:

Browse Movies Screen: This requirement is implemented in `browse_movies_screen.dart` :

```
ListView.builder(
shrinkWrap: true,
physics: const NeverScrollableScrollPhysics(),
itemCount: filteredMovies.length,
itemBuilder: (context, index) {
return Padding(
padding: const EdgeInsets.only(bottom: AppSpacing.md),  child: Card(
color: AppColors.cardBackground,
elevation: AppSpacing.elevationMd,
child: ListTile(
leading: Image.network(movie.imageUrl),
title: Text(movie.title),
subtitle: Column(
crossAxisAlignment: CrossAxisAlignment.start,  children: [
Text(movie.year.toString()),
Row(
children: [
Icon(Icons.star, color: AppColors.primaryYellow),  Text('${movie.rating}'),
Text('${movie.reviewCount} reviews'),  ],
),
```

```
],
),
trailing: IconButton(
icon: Icon(Icons.remove_circle, color: AppColors.error),  onPressed: () {
setState(() {
Movie removedMovie = filteredMovies[index];
removedMovies.add(removedMovie);
filteredMovies.removeAt(index);
movies.remove(removedMovie);
});
ScaffoldMessenger.of(context).showSnackBar(  SnackBar(
content: Text('${movie.title} removed'),  action: SnackBarAction(
label: 'Undo',
onPressed: () {
setState(() {
movies.add(removedMovie);
_applyFilters();
});
},
),
),
);
},
),
),
),
);
},
)
```

Key Features: - Cards display comprehensive movie information - Remove button triggers immediate visual feedback - Removed items are tracked for undo functionality - SnackBar provides undo action for accidental removals - Filtering logic updates automatically after removal - State management ensures list consistency

Model Class (lib/models/movie.dart):

```
class Movie {
 final String title;
 final int year;
 final List<String> genres;
 final double rating;
 final int reviewCount;
 final String synopsis;
 final String imageUrl;

 Movie({
 required this.title,
 required this.year,
 required this.genres,
 required this.rating,
 required this.reviewCount,
 required this.synopsis,
 required this.imageUrl,
 });
}
```

3.6 Form & Validation

Requirement: Application must contain at least one form screen with validation and AlertDialog on successful submission.

Implementation:

Primary Form: Login Screen

The login form in `login_screen.dart` implements comprehensive validation:

```
Form(
 key: _formKey,
 child: Column(
 children: [
 // Name Field
```

```
TextFormField(
controller: _nameController,
decoration: InputDecoration(
labelText: 'Full Name (Optional)',  hintText: 'Enter your
name',
),
),

// Email Field with Validation
TextFormField(
controller: _emailController,
keyboardType: TextInputType.emailAddress,  validator: (value) {
if (value == null || value.isEmpty) {  return 'Email is required';
}
if (!value.endsWith('.edu')) {
return 'Must be a valid .edu email address';  }
if (!value.contains('@')) {
return 'Invalid email format';  }
return null;
},
decoration: InputDecoration(
labelText: 'Student Email',
hintText: 'you@university.edu',
errorStyle: TextStyle(color: AppColors.error),  ),
),
// Password Field with Validation
TextFormField(
controller: _passwordController,
obscureText: _obscurePassword,
validator: (value) {
if (value == null || value.isEmpty) {
return 'Password is required';
}
if (value.length < 6) {
return 'Password must be at least 6 characters';
```

```
}
return null;
},
decoration: InputDecoration(
labelText: 'Password',
suffixIcon: IconButton(
icon: Icon(_obscurePassword ? Icons.visibility : Icons.visibility_off),  onPressed: () {
setState(() {
_obscurePassword = !_obscurePassword;
});
},
),
),
),

// Submit Button
ElevatedButton(
onPressed: () {
if (_formKey.currentState!.validate()) {
// Save user info
UserService.login(
name: _nameController.text.isEmpty
? 'Student Name'
: _nameController.text,
email: _emailController.text,
);

// Show success dialog
showDialog(
context: context,
builder: (context) => AlertDialog(
backgroundColor: AppColors.cardBackground,  title: Text('Login Successful', style:
AppTextStyles.h3),  content: Text(
'Welcome back, ${_emailController.text}',  style: AppTextStyles.bodyMedium,
),
```

```
actions: [
TextButton(
onPressed: () {
Navigator.of(context).pop();
Navigator.pushReplacementNamed(context, '/home');  },
child: Text(
'Continue',
style: AppTextStyles.button.copyWith(  color: AppColors.primaryYellow,  ),
),
),
],
),
);
}
},
child: Text('Login'),
),
],
),
)
```

Secondary Form: Create Account Dialog

The Create Account feature includes a nested form with three validated fields: - Full Name (required, non-empty) - Student Email (required, must end with .edu) - Password (required, minimum 6 characters)

Tertiary Form: Organize Event Screen

Event creation form includes: - Event Name (required text field) - Movie Selection (required dropdown) - Date (required date picker) - Time (required time picker) - Location/Platform (required text field)

All forms display inline error messages in red text below invalid fields and only allow submission when all validations pass.

3.7 Responsiveness

Requirement: UI layout should adapt smoothly to various device sizes and orientations.

Implementation:

Responsive Grid Layout (Home Screen):

```
final size = MediaQuery.of(context).size;
final crossAxisCount = size.width < 600 ? 2 :
 size.width < 900 ? 3 : 4;

GridView.builder(
 gridDelegate: SliverGridDelegateWithFixedCrossAxisCount(  crossAxisCount:
crossAxisCount,
 childAspectRatio: 0.7,
 crossAxisSpacing: AppSpacing.md,
 mainAxisSpacing: AppSpacing.md,
 ),
 itemBuilder: (context, index) => MovieCard(movie: movies[index]), )
```

Responsive Padding:

```
final isSmallScreen = size.width < 600;
padding: EdgeInsets.all(
 isSmallScreen ? AppSpacing.md : AppSpacing.xl
 )
```

Constrained Content Width:

```
ConstrainedBox(
 constraints: BoxConstraints(maxWidth: 600),
 child: loginForm,
 )
```

Flexible Layouts:

```
Row(
 children: [
 Expanded(child: movieTitle), // Takes available space  Icon(Icons.star), // Fixed
size
 ],
 )
```

Text Overflow Handling:

```
Text(
 movie.title,
 maxLines: 2,
 overflow: TextOverflow.ellipsis,
 )
```

Responsive Testing: The application has been tested on: - Small phones (iPhone SE - 375x667) - Standard phones (iPhone 15 - 393x852) - Large phones (iPhone 15 Pro Max - 430x932) - Tablets (iPad - 768x1024) - Desktop (macOS - 1920x1080)
All screens maintain proper layout and functionality across these dimensions, with appropriate grid columns, spacing, and text sizing.

4. Additional Features & Enhancements

4.1 User Management System

UserService (`lib/services/user_service.dart`): A centralized service manages user authentication and profile data: - Login functionality with name and email storage - Automatic university name extraction from email domain - Session persistence during app runtime -

Logout capability

User Model (`lib/models/user.dart`): Structured data model containing: - Full name - Email address - University affiliation - Bio/description - Helper methods (getFirstName(), getInitials())

This enables personalized experiences throughout the app, such as greeting users by name on the Home screen and displaying complete profile information.

4.2 Dynamic Statistics System

UserStats (`lib/utils/user_stats.dart`): In-memory statistics tracking: - Movies Rated counter - Reviews Written counter - Groups Joined counter - Increment methods called when users perform actions - Reset functionality for testing

Integration Points: - Review submission increments both Movies Rated and Reviews Written - Profile screen displays real-time statistics - Statistics update immediately upon action completion

4.3 Event Management System

Event Model (`lib/models/event.dart`): Complete event data structure: - Unique ID generation - Event name and associated movie - Date and time - Location/platform information - Organizer name - Attendee count - Helper methods for date formatting and status checking
My Events Screen: - Segregates upcoming and past events - Cancel functionality for upcoming events - Event detail modal with comprehensive information - Share event capability - Empty state handling - Quick navigation to event creation

4.4 Enhanced Navigation Flow

Bottom Navigation Bar: Persistent navigation across four main sections: - Home (yellow icon when active) - Browse (gray icon when inactive) - Chat Groups - Profile

Visual feedback through icon color changes and active state indicators.

Contextual Navigation: - Movie cards throughout app link to Movie Detail - Movie Detail provides paths to Review and Student Reviews - Home screen offers direct access to Event

Organization - Event success flows to My Events viewing

4.5 Improved Error Handling

Network Image Loading: - Loading indicator with progress tracking - Graceful fallback UI showing movie title and icon - No broken image placeholders

Form Submission: - Inline validation messages - Prevention of incomplete submissions - Clear error communication

Empty States: - Chat screen: "Start the conversation" message - My Events: "No Events Yet" with creation prompt - Reviews page: "Be the first to review" button

5. Code Organization & Architecture

5.1 Project Structure

```
lib/
├── main.dart # App entry point, theme, routes ├── models/ # Data models
│ ├── movie.dart # Movie data structure │ ├── chat_group.dart # Chat
group model
│ ├── event.dart # Event model
│ └── user.dart # User profile model ├── screens/ # UI screens (10 total)
│ ├── login_screen.dart
│ ├── home_screen.dart
│ ├── browse_movies_screen.dart
│ ├── movie_detail_screen.dart
│ ├── review_screen.dart
│ ├── chat_groups_screen.dart
│ ├── group_chat_screen.dart
│ ├── organize_event_screen.dart
│ ├── my_events_screen.dart
│ └── profile_screen.dart
├── widgets/ # Reusable components │ ├── movie_card.dart
│ └── movie_list_card.dart
├── services/ # Business logic
```

```
│ └── user_service.dart
└── utils/ # Constants & helpers ├── app_colors.dart
├── app_text_styles.dart
├── app_spacing.dart
└── user_stats.dart
```

5.2 State Management

The application uses **StatefulWidget** for state management: - Each screen manages its own local state - setState() triggers UI updates - UserService and UserStats provide shared state - Model classes encapsulate data logic

6. Conclusion

The Student Movie Buffs application successfully transforms the Phase 2.1 wireframes into a fully functional Flutter application that meets and exceeds all Phase 2.2 requirements. The implementation demonstrates:

Technical Competency: - Proper use of named routes for navigation management - Clean code organization with utility classes - Integration of both asset and network images with error handling - Custom font application via theme - Dynamic list with Card widgets and remove functionality - Comprehensive form validation with AlertDialog feedback - Responsive design adapting to multiple screen sizes

Enhanced Features: - User authentication and profile management - Dynamic statistics tracking - Event organization and viewing system - Personalized user experience - Professional UI with Material Design 3 - Robust error handling and empty states

Best Practices: - Separation of concerns (models, screens, widgets, services, utilities) - Consistent code style and naming conventions - Proper state management - Null safety throughout - Comprehensive documentation

The application provides a solid foundation for future enhancements and demonstrates the successful translation of design concepts into a polished, functional mobile application. All screens are fully implemented, all features work as intended, and the code runs without errors on multiple platforms including macOS desktop, Chrome web, and would run on iOS/ Android devices with proper SDK configuration.

The project showcases not only technical implementation skills but also attention to user experience, code quality, and professional development practices. It stands as a complete, production-ready prototype that effectively serves its target audience of university students seeking to discover and discuss movies with their peers.

End of Report

Total Screens Implemented: 10
Total Lines of Code: ~4,500+
GitHub Repository: Movie-Application-Project-Repository/Phase 2.2
Submission Date: November 24, 2025

Screenshots

Screenshots were explained and provided in 2.1 Project Step in detail.

Course: CS310 - Mobile Application Development
Instructor: Saima Gül
Sabancı University
Fall 2025