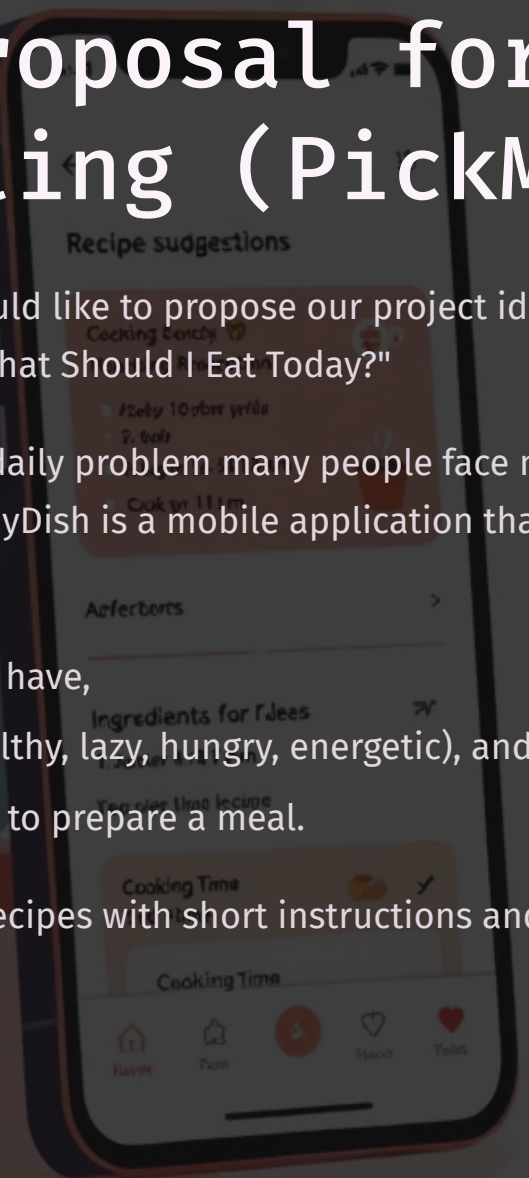
Project Proposal for Software Design and Modelling (PickMyDish)



We (I and my team member) would like to propose our project idea for the Software Design and Modelling course. The project is titled "PickMyDish – What Should I Eat Today?"

The idea came from a common daily problem many people face not knowing what to eat or cook, especially when time or ingredients are limited. PickMyDish is a mobile application that helps users decide what to eat based on three main factors:

- The ingredients they already have,
- Their current mood (e.g., healthy, lazy, hungry, energetic), and
- The time they have available to prepare a meal.

The app will suggest matching recipes with short instructions and images. Users will also be able to save their favorite meals for later.

Application Architecture Overview

The PickMyDish application is built using Flutter and follows a clean architecture pattern with clear separation of concerns. The main entry point is defined in Main.dart, which sets up the application with MultiProvider for state management.

Main Application Setup

The app initializes with two primary providers: UserProvider for managing user authentication and profile data, and RecipeProvider for handling recipe data and favorites. The application starts with a SplashScreen that transitions to the LoginScreen after 3 seconds.

State Management

Provider pattern is used throughout the application to manage state efficiently. UserProvider tracks the current authenticated user, their profile information, and settings. RecipeProvider manages the recipe catalog, favorites, and filtering logic.

Navigation Flow

Users begin at the SplashScreen, proceed to LoginScreen or RegisterScreen, and upon successful authentication, access the HomeScreen. From there, they can navigate to RecipesScreen, FavoritesScreen, ProfileScreen, or RecipeUploadScreen.

Core Constants and Styling

The application maintains consistent styling through a centralized constants file (Constants.dart) that defines text styles, colors, and UI dimensions used throughout the app.

Text Styles

- **title**: White, TimesNewRoman, 37px, weight 600 - used for main headings
- **mediumtitle**: White, TimesNewRoman, 23px, weight 600 - used for section headings
- **footer**: White, TimesNewRoman, 22px, weight 600 - used for footer text
- **footerClickable**: Orange, TimesNewRoman, 22px, weight 600 - used for clickable footer links
- **text**: White, TimesNewRoman, 18px, weight 600 - used for body text
- **caloriesText**: White, TimesNewRoman, 14px - used for calorie information
- **categoryText**: Orange, TimesNewRoman, 16px - used for recipe categories
- **placeHolder**: White (193 opacity), TimesNewRoman, 20px - used for input placeholders

Global Variables

isPasswordVisible: Boolean flag controlling password visibility in input fields

iconSize: Constant value of 40 for consistent icon sizing across the application

Design Philosophy

The styling emphasizes a dark theme with orange accents, creating a modern and appetizing visual experience. The TimesNewRoman font family provides a classic, readable appearance that works well for recipe content.

Image Handling and Caching

The application implements sophisticated image handling through the CachedProfileImage widget (Cached_image.dart), which efficiently manages both local assets and remote server images with caching capabilities.

01

Image Source Detection

The widget automatically detects whether an image path points to a local asset (starting with 'assets/') or a remote server image. For server images, it constructs the full URL using the base path 'http://38.242.246.126:3000/'.

02

Caching Strategy

Remote images are cached using the CachedNetworkImage package, which stores downloaded images locally to reduce network requests and improve loading times. This is particularly important for recipe images that users view repeatedly.

03

Flexible Display Options

The widget supports both profile pictures (circular avatars) and regular images (rectangular with rounded corners). Parameters include radius, width, height, fit, and shape customization.

04

Loading and Error States

While images load, a placeholder with a CircularProgressIndicator is displayed. If loading fails, an error widget with a broken_image icon appears, ensuring the UI remains functional even when images are unavailable.

Backend API Service Integration

The ApiService class (Api_service.dart) manages all communication with the backend server, handling authentication, user profile updates, and recipe operations. The backend server base URL is configured as "http://38.242.246.126:3000".

Authentication Endpoints

testConnection(): Verifies backend connectivity and database connection status

login(email, password): Authenticates users and returns user data including userId, username, email, and profile image path. Returns error messages for failed attempts.

register(userName, email, password): Creates new user accounts with validation. Returns boolean success status.

testAuth(): Development method for testing authentication flow

User Profile Management

updateUsername(newUsername, userId): Updates the username for a specific user. Sends PUT request with JSON body containing username and userId.

uploadProfilePicture(imageFile, userId): Uploads profile pictures using multipart form data. Accepts File object and userId, returns boolean success status.

getProfilePicture(userId): Retrieves the profile picture path from the database for a specific user. Returns the imagePath string or null if not found.

Recipe Operations

getRecipes(): Fetches all recipes from the server. Returns a List of Map objects containing recipe data including id, name, category, time, calories, image_path, ingredients, instructions, mood, and userId.

uploadRecipe(recipeData, imageFile): Uploads new recipes with optional images. Uses multipart request to send recipe details (name, category, time, calories, ingredients, instructions, userId) along with an image file. Returns boolean success status.

Local Database Management

The DatabaseService class (Database_service.dart) implements local data persistence using SQLite through the sqflite package. This provides offline access to recipes and manages the local recipe database.

Database Initialization

The database is created at 'recipes.db' with a single table structure:

- id (INTEGER PRIMARY KEY)
- name (TEXT)
- category (TEXT)
- time (TEXT)
- calories (TEXT)
- image (TEXT)
- ingredients (TEXT - JSON encoded)
- mood (TEXT - JSON encoded)
- difficulty (TEXT)
- steps (TEXT - JSON encoded)
- isFavorite (INTEGER - 0 or 1)

Initial data is loaded from 'data/recipes.json' during database creation, populating the database with default recipes.

Core Database Operations

getRecipes(): Returns all recipes as List<Recipe> objects, converting database maps to Recipe model instances.

getFilteredRecipes(ingredients, mood, time): Filters recipes based on user preferences. Supports filtering by available ingredients, current mood, and cooking time constraints.

getFavoriteRecipes(): Returns only recipes marked as favorites (isFavorite = 1).

toggleFavorite(recipeId, isFavorite): Updates the favorite status of a specific recipe in the database.

_mapToRecipe(map): Helper method that converts database map entries to Recipe objects, handling JSON decoding for complex fields like ingredients, steps, and moods.

_convertTimeToMinutes(time): Utility method converting time strings ('15 mins', '30 mins', '1 hour', '2+ hours') to numeric minute values for comparison.

Data Models: Recipe and User

The application uses two primary data models to structure information: Recipe (Recipe_model.dart) and User (User_model.dart). These models provide type safety and consistent data handling throughout the app.

Recipe Model Structure

The Recipe class contains the following properties:

- **id** (int): Unique identifier for the recipe
- **name** (String): Recipe name/title
- **category** (String): Recipe category (e.g., 'Main Course', 'Breakfast')
- **cookingTime** (String): Time required to prepare (e.g., '30 mins')
- **calories** (String): Caloric content
- **imagePath** (String): Path to recipe image (local or remote)
- **ingredients** (List<String>): List of required ingredients
- **steps** (List<String>): Cooking instructions as ordered steps
- **moods** (List<String>): Associated moods/emotions (e.g., 'Comfort', 'Healthy')
- **userId** (int): ID of user who created the recipe
- **isFavorite** (bool): Favorite status flag

The Recipe model includes factory constructor **fromJson()** with robust parsing that handles both List and JSON string formats for complex fields. The **copyWith()** method enables immutable updates, and **toJson()** serializes the object for API transmission.

User Model Structure

The User class contains the following properties:

- **id** (String): Unique user identifier
- **username** (String): Display name
- **email** (String): User email address
- **profileImage** (String?): Optional profile picture path
- **joinedDate** (DateTime): Account creation timestamp

The User model provides **fromJson()** factory constructor that parses API responses, handling both 'profile_image_path' and 'profileImage' keys for backward compatibility. The 'created_at' timestamp is parsed into a DateTime object. The **copyWith()** method includes ALL fields for complete immutability. The model overrides **==** operator and **hashCode** for proper equality comparison, and includes **toString()** for debugging purposes.

State Management with Providers

The application uses the Provider pattern for state management, implementing two main providers: RecipeProvider and UserProvider. These providers extend ChangeNotifier to notify listeners of state changes.

RecipeProvider (Recipe_provider.dart)

State Variables:

- _recipes: List<Recipe> - Complete recipe catalog
- _favoriteIds: List<int> - IDs of favorited recipes
- _isLoading: bool - Loading state indicator
- _error: String? - Error message storage

Key Methods:

loadRecipes(): Fetches recipes from API via ApiService.getRecipes(), converts Map objects to Recipe instances, and updates state.

toggleFavorite(recipeId): Adds/removes recipe from favorites list, updates the recipe's isFavorite status using copyWith(), and notifies listeners.

filterRecipes(query): Searches recipes by name, category, or mood tags. Returns filtered list without modifying state.

personalizeRecipes(ingredients, mood, time): Advanced filtering that matches recipes based on available ingredients, user mood, and cooking time constraints.

getRecipeById(id): Retrieves specific recipe by ID, returns null if not found.

clear(): Resets all state (used during logout).

UserProvider (User_provider.dart)

State Variables:

- _user: User? - Current authenticated user
- _userId: int - User ID for API calls
- _profilePicture: String - Profile image path
- _joined: DateTime - Join date
- _userRecipes: List - User's created recipes
- _userFavorites: List<int> - User's favorites
- _userSettings: Map - User preferences

Key Methods:

setUser(user): Sets current user and notifies listeners.

setUserFromJson(userData): Creates User from JSON and sets as current user.

updateUsername(newUsername, userId): Updates username using copyWith() to preserve other fields.

updateProfilePicture(imagePath): Updates profile picture path and notifies listeners.

setUserId(userId): Sets user ID for API operations.

clearAllUserData(): Comprehensive cleanup that resets all user data, clears image cache using DefaultCacheManager, and optionally clears local storage.

logout(): Calls clearAllUserData() and logs the action.

printUserState(): Debug utility for logging current user state.

User Interface Screens

The application consists of eight main screens that provide the complete user experience: SplashScreen, LoginScreen, RegisterScreen, HomeScreen, RecipesScreen, RecipeDetailScreen, FavoritesScreen, ProfileScreen, and RecipeUploadScreen.

SplashScreen

Initial loading screen displaying the app logo and tagline "What should I eat today?". Uses a 3-second Timer to automatically navigate to LoginScreen. Implements proper cleanup by canceling the timer in dispose().

LoginScreen & RegisterScreen

LoginScreen provides email/password authentication with password visibility toggle, Google sign-in option, and "Login As Guest" functionality. RegisterScreen includes username, email, password, and confirm password fields with real-time password strength indicator (weak/medium/strong) and email validation. Both screens feature gradient backgrounds with food imagery.

HomeScreen

Main dashboard with personalized recipe generation based on ingredients, mood, and time. Features a side menu drawer with profile picture, navigation options, and logout. Displays "Today's Fresh Recipe" section and personalized results. Includes search functionality and quick access to favorites and recipe upload.

RecipesScreen

Displays all recipes in a 2-column grid layout. Each card shows recipe image, name, category, cooking time, and favorite icon. Includes search bar for filtering by name or category. Features pull-to-refresh functionality and loading/error states.

RecipeDetailScreen

Full recipe view with expandable image header, recipe name, category, cooking time badge, and calorie information. Displays complete ingredients list with bullet points and numbered cooking steps. Shows mood tags and includes "Start Cooking" button.

ProfileScreen & FavoritesScreen

ProfileScreen allows users to edit username, upload profile pictures via image picker, view account information (email, join date, favorite count), and logout. FavoritesScreen displays user's saved recipes in a scrollable list with quick access to recipe details.

Testing Infrastructure and Project Configuration

The project includes comprehensive testing setup and configuration files to ensure code quality and proper dependency management.

Widget Testing (Widget_test.dart)

Comprehensive test suite covering:

- **Screen Rendering Tests**: Verifies all screens (Splash, Home, Login, Register, Recipe, Favorite, Profile) render without crashing
- **Key UI Elements Tests**: Validates presence of welcome text, titles, and section headers
- **Form Elements Tests**: Checks email fields, password fields, and username editing functionality
- **Button Tests**: Ensures login, register, and save buttons are present
- **Icon Tests**: Verifies favorite icons and back buttons exist
- **Layout Tests**: Confirms personalization sections and search fields are displayed
- **Input Tests**: Tests text entry in login and search fields
- **Constants Test**: Validates text styles and global variables

Uses test helper functions like wrapWithProviders() to properly initialize widgets with required providers.

Test Helpers (Test_helper.dart)

Provides utility functions and mock objects:

- **FakeFavoriteProvider**: Mock provider for testing favorite functionality
- **createTestRecipes(count)**: Generates test recipe data
- **createTestableWidget(child)**: Wraps widgets in MaterialApp for testing
- **wrapWithProviders(child)**: Wraps widgets with MultiProvider including UserProvider

Project Configuration (Pubspec.yaml)

Key Dependencies:

- sqflite ^2.4.2 - Local database
- provider ^6.0.5 - State management
- http ^1.1.0 - API communication
- image_picker ^1.0.4 - Image selection
- cached_network_image ^3.3.0 - Image caching
- dropdown_search ^5.0.2 - Enhanced dropdowns
- intl ^0.18.1 - Internationalization
- flutter_cache_manager ^3.3.0 - Cache management

Dev Dependencies: flutter_test, integration_test, mockito ^5.4.0, build_runner ^2.4.0

Assets: Organized in folders for logo, login, sideMenu, and recipes images

Fonts: TimesNewRoman (times.ttf) for consistent typography

The project is configured for SDK version ^3.9.2 and includes comprehensive testing infrastructure with both unit and integration test capabilities. The pubspec.yaml file is well-organized with clear separation of dependencies, dev dependencies, assets, and fonts.