Bharatiya Vidya Bhavan's
SARDAR PATEL INSTITUTE OF TECHNOLOGY
(Autonomous Institute Affiliated to University of Mumbai)
Munshi Nagar, Andheri (W), Mumbai – 400 058.
Department of Computer Science and Engineering

Experiment	6
Aim	Create an app for children where children can learn numbers and alphabets
Objective	● To Create App of children ● To use Upper Tabs in navigation bar ● To implement Splash Screen in Flutter
Name	Vedant Onkar
UCID	2024510036
Class	FYMCA
Batch	B
Date of Submission	23-04-2025

Technology used	**VS Code IDE, Flutter SDK**
Task	Children should be able to even recognize the numbers and alphabets by quiz form. Create 4 upper Tabs with Splash Screen at the start: 1) Practice/Lessons 2) Quiz 3) Leader Board 4) Profile

Code with proper label

```
// main.dart
import 'package:flutter/material.dart';
import 'package:shared_preferences/shared_preferences.dart';
import 'dart:math';

void main() {
  runApp(const KidsLearningApp());
}

class KidsLearningApp extends StatelessWidget {
  const KidsLearningApp({Key? key}) : super(key: key);

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Kids Learning App',
      theme: ThemeData(
        primarySwatch: Colors.blue,
        fontFamily: 'Comic Sans MS',
        colorScheme: ColorScheme.fromSwatch(
          primarySwatch: Colors.blue,
          accentColor: Colors.orange,
          backgroundColor: Colors.white,
          brightness: Brightness.light,
        ),
        textTheme: TextTheme(
          headlineLarge: TextStyle(fontSize: 32, fontWeight:
FontWeight.bold, color: Colors.indigo[900]),
          headlineMedium: TextStyle(fontSize: 24,
fontWeight: FontWeight.bold, color: Colors.indigo[800]),
          bodyLarge: TextStyle(fontSize: 18, color:
Colors.black87),
          bodyMedium: TextStyle(fontSize: 16, color:
Colors.black87),
        ),
        buttonTheme: ButtonThemeData(
          shape: RoundedRectangleBorder(borderRadius:
BorderRadius.circular(20)),
          buttonColor: Colors.orange,
        ),
      ),
      home: const SplashScreen(),
    );
  }
}

class SplashScreen extends StatefulWidget {
  const SplashScreen({Key? key}) : super(key: key);

  @override
  _SplashScreenState createState() => _SplashScreenState();
}

class _SplashScreenState extends State<SplashScreen> with
SingleTickerProviderStateMixin {
  late AnimationController _animationController;
  late Animation<double> _scaleAnimation;

  @override
  void initState() {
    super.initState();
    _animationController = AnimationController(
      vsync: this,
      duration: const Duration(seconds: 2),
    );

    _scaleAnimation = Tween<double>(begin: 0.0, end: 1.0)
        .animate(CurvedAnimation(parent:
_animationController, curve: Curves.elasticOut));
```

```
    _animationController.forward();

    Future.delayed(const Duration(seconds: 3), () {
      Navigator.pushReplacement(
        context,
        MaterialPageRoute(builder: (context) => const
MainScreen()),
      );
    });
  }

  @override
  void dispose() {
    _animationController.dispose();
    super.dispose();
  }

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      backgroundColor: Colors.lightBlue[50],
      body: Center(
        child: Column(
          mainAxisAlignment: MainAxisAlignment.center,
          children: [
            ScaleTransition(
              scale: _scaleAnimation,
              child: Column(
                children: [
                  Container(
                    width: 150,
                    height: 150,
                    decoration: BoxDecoration(
                      color: Colors.white,
                      borderRadius:
BorderRadius.circular(20),
                      boxShadow: [
                        BoxShadow(
                          color:
Colors.blue.withOpacity(0.3),
                          spreadRadius: 5,
                          blurRadius: 7,
                          offset: const Offset(0, 3),
                        ),
                      ],
                    ),
                    child: Center(
                      child: Row(
                        mainAxisAlignment:
MainAxisAlignment.center,
                        children: [
                          Text(
                            'A',
                            style: TextStyle(
                              fontSize: 48,
                              fontWeight: FontWeight.bold,
                              color: Colors.red,
                            ),
                          ),
                          Text(
                            '1',
                            style: TextStyle(
                              fontSize: 48,
                              fontWeight: FontWeight.bold,
                              color: Colors.blue,
                            ),
                          ),
                          Text(
                            'B',
                            style: TextStyle(
                              fontSize: 48,
                              fontWeight: FontWeight.bold,
                              color: Colors.green,
                            ),
                          ),
                          Text(
                            '2',
                            style: TextStyle(
                              fontSize: 48,
                              fontWeight: FontWeight.bold,
                              color: Colors.orange,
                            ),
                          ),
                        ],
                      ),
                    ),
                  ),
                  const SizedBox(height: 20),
                  Text(
                    'Learn & Play',
                    style: TextStyle(
                      fontSize: 32,
                      fontWeight: FontWeight.bold,
                      color: Colors.indigo[800],
                    ),
                  ),
                  const SizedBox(height: 10),
                  Text(
                    'Numbers & Alphabets',
                    style: TextStyle(
                      fontSize: 18,
                      fontWeight: FontWeight.bold,
                      color: Colors.indigo[600],
                    ),
                  ),
                ],
              ),
            ),
            const SizedBox(height: 40),
            CircularProgressIndicator(
              valueColor:
AlwaysStoppedAnimation<Color>(Colors.orange),
            ),
          ],
        ),
      ),
    );
  }
}

class MainScreen extends StatefulWidget {
  const MainScreen({Key? key}) : super(key: key);

  @override
  _MainScreenState createState() => _MainScreenState();
}

class _MainScreenState extends State<MainScreen> {
  int _currentIndex = 0;
  final List<Widget> _screens = [
    const PracticeLessonsScreen(),
    const QuizScreen(),
    const LeaderBoardScreen(),
    const ProfileScreen(),
  ];

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      body: _screens[_currentIndex],
      bottomNavigationBar: Container(
        decoration: BoxDecoration(
          borderRadius: BorderRadius.vertical(top:
Radius.circular(20)),
          boxShadow: [
            BoxShadow(
              color: Colors.grey.withOpacity(0.3),
              spreadRadius: 1,
              blurRadius: 10,
            ),
          ],
        ),
        child: ClipRRect(
          borderRadius: BorderRadius.vertical(top:
Radius.circular(20)),
          child: BottomNavigationBar(
            currentIndex: _currentIndex,
            onTap: (index) {
              setState(() {
                _currentIndex = index;
              });
            },
            type: BottomNavigationBarType.fixed,
            backgroundColor: Colors.white,
            selectedItemColor: Colors.orange,
            unselectedItemColor: Colors.grey,
            selectedLabelStyle: TextStyle(fontWeight:
FontWeight.bold),
            items: const [
              BottomNavigationBarItem(
                icon: Icon(Icons.book),
                label: 'Lessons',
              ),
              BottomNavigationBarItem(
                icon: Icon(Icons.quiz),
                label: 'Quiz',
              ),
              BottomNavigationBarItem(
                icon: Icon(Icons.leaderboard),
                label: 'Leaders',
              ),
              BottomNavigationBarItem(
                icon: Icon(Icons.person),
                label: 'Profile',
              ),
            ],
          ),
        ),
      ),
    );
  }
}

// Data Models
class User {
  String name;
  String avatar;
  int totalScore;
  int alphabetsLevel;
  int numbersLevel;
  List<QuizResult> quizHistory;

  User({
    required this.name,
    required this.avatar,
    this.totalScore = 0,
    this.alphabetsLevel = 1,
    this.numbersLevel = 1,
    required this.quizHistory,
  });

  Map<String, dynamic> toJson() {
    return {
      'name': name,
      'avatar': avatar,
      'totalScore': totalScore,
      'alphabetsLevel': alphabetsLevel,
      'numbersLevel': numbersLevel,
      'quizHistory': quizHistory.map((result) =>
result.toJson()).toList(),
    };
  }

  factory User.fromJson(Map<String, dynamic> json) {
    return User(
      name: json['name'],
      avatar: json['avatar'],
      totalScore: json['totalScore'],
      alphabetsLevel: json['alphabetsLevel'],
      numbersLevel: json['numbersLevel'],
      quizHistory: (json['quizHistory'] as List)
          .map((item) => QuizResult.fromJson(item))
          .toList(),
    );
  }
}

class QuizResult {
  final String type; // 'alphabets' or 'numbers'
  final int score;
  final DateTime dateTime;

  QuizResult({
    required this.type,
    required this.score,
    required this.dateTime,
  });

  Map<String, dynamic> toJson() {
    return {
      'type': type,
      'score': score,
      'dateTime': dateTime.toIso8601String(),
    };
  }

  factory QuizResult.fromJson(Map<String, dynamic> json) {
    return QuizResult(
      type: json['type'],
      score: json['score'],
      dateTime: DateTime.parse(json['dateTime']),
    );
  }
}

// Practice/Lessons Screen
class PracticeLessonsScreen extends StatefulWidget {
  const PracticeLessonsScreen({Key? key}) : super(key: key);

  @override
  _PracticeLessonsScreenState createState() =>
_PracticeLessonsScreenState();
}

class _PracticeLessonsScreenState extends
```

```
State<PracticeLessonsScreen> with
SingleTickerProviderStateMixin {
  late TabController _tabController;

  @override
  void initState() {
    super.initState();
    _tabController = TabController(length: 2, vsync: this);
  }

  @override
  void dispose() {
    _tabController.dispose();
    super.dispose();
  }

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: Text(
          'Learn & Practice',
          style: TextStyle(color: Colors.white, fontWeight:
FontWeight.bold),
        ),
        backgroundColor: Colors.blue,
        elevation: 0,
        bottom: TabBar(
          controller: _tabController,
          labelColor: Colors.white,
          unselectedLabelColor: Colors.white60,
          indicatorColor: Colors.orange,
          indicatorWeight: 4,
          tabs: const [
            Tab(text: 'Alphabets'),
            Tab(text: 'Numbers'),
          ],
        ),
      ),
      body: TabBarView(
        controller: _tabController,
        children: [
          _buildAlphabetsLessons(),
          _buildNumbersLessons(),
        ],
      ),
    );
  }

  Widget _buildAlphabetsLessons() {
    return SingleChildScrollView(
      padding: const EdgeInsets.all(16),
      child: Column(
        crossAxisAlignment: CrossAxisAlignment.start,
        children: [
          _buildSectionTitle('Let\'s Learn Alphabets!'),
          const SizedBox(height: 16),
          GridView.builder(
            physics: NeverScrollableScrollPhysics(),
            shrinkWrap: true,
            gridDelegate:
SliverGridDelegateWithFixedCrossAxisCount(
              crossAxisCount: 3,
              childAspectRatio: 1,
              mainAxisSpacing: 16,
              crossAxisSpacing: 16,
            ),
            itemCount: 26,
            itemBuilder: (context, index) {
              String letter = String.fromCharCode(65 +
index);
              return _buildLetterCard(letter);
            },
          ),
        ],
      ),
    );
  }

  Widget _buildNumbersLessons() {
    return SingleChildScrollView(
      padding: const EdgeInsets.all(16),
      child: Column(
        crossAxisAlignment: CrossAxisAlignment.start,
        children: [
          _buildSectionTitle('Let\'s Learn Numbers!'),
          const SizedBox(height: 16),
          GridView.builder(
            physics: NeverScrollableScrollPhysics(),
            shrinkWrap: true,
            gridDelegate:
SliverGridDelegateWithFixedCrossAxisCount(
              crossAxisCount: 3,
              childAspectRatio: 1,
              mainAxisSpacing: 16,
              crossAxisSpacing: 16,
            ),
            itemCount: 20,
            itemBuilder: (context, index) {
              int number = index + 1;  // Start from 1
              return _buildNumberCard(number);
            },
          ),
        ],
      ),
    );
  }

  Widget _buildSectionTitle(String title) {
    return Text(
      title,
      style: TextStyle(
        fontSize: 24,
        fontWeight: FontWeight.bold,
        color: Colors.indigo[800],
      ),
    );
  }

  Widget _buildLetterCard(String letter) {
    List<Color> colors = [
      Colors.red[300]!,
      Colors.blue[300]!,
      Colors.green[300]!,
      Colors.orange[300]!,
      Colors.purple[300]!,
    ];
    final random = Random();
    final color = colors[random.nextInt(colors.length)];

    return InkWell(
      onTap: () {
        _showLetterDetailDialog(letter);
      },
      child: Card(
        elevation: 4,
        shape: RoundedRectangleBorder(borderRadius:
BorderRadius.circular(16)),
        color: Colors.white,
        child: Container(
          decoration: BoxDecoration(
            borderRadius: BorderRadius.circular(16),
            border: Border.all(color: color, width: 2),
          ),
          child: Center(
            child: Text(
              letter,
              style: TextStyle(
                fontSize: 48,
                fontWeight: FontWeight.bold,
                color: color,
              ),
            ),
          ),
        ),
      ),
    );
  }

  Widget _buildNumberCard(int number) {
    List<Color> colors = [
      Colors.red[300]!,
      Colors.blue[300]!,
      Colors.green[300]!,
      Colors.orange[300]!,
      Colors.purple[300]!,
    ];
    final random = Random();
    final color = colors[random.nextInt(colors.length)];

    return InkWell(
      onTap: () {
        _showNumberDetailDialog(number);
      },
      child: Card(
        elevation: 4,
        shape: RoundedRectangleBorder(borderRadius:
BorderRadius.circular(16)),
        color: Colors.white,
        child: Container(
          decoration: BoxDecoration(
            borderRadius: BorderRadius.circular(16),
            border: Border.all(color: color, width: 2),
          ),
          child: Center(
            child: Text(
              number.toString(),
              style: TextStyle(
                fontSize: 48,
                fontWeight: FontWeight.bold,
                color: color,
              ),
            ),
          ),
        ),
      ),
    );
  }

  void _showLetterDetailDialog(String letter) {
    final List<String> examples =
_getExamplesForLetter(letter);

    showDialog(
      context: context,
      builder: (context) => AlertDialog(
        shape: RoundedRectangleBorder(borderRadius:
BorderRadius.circular(20)),
        title: Center(
          child: Text(
            'Letter $letter',
            style: TextStyle(
              fontSize: 32,
              fontWeight: FontWeight.bold,
              color: Colors.indigo[800],
            ),
          ),
        ),
        content: Column(
          mainAxisSize: MainAxisSize.min,
          children: [
            Text(
              letter,
              style: TextStyle(
                fontSize: 72,
                fontWeight: FontWeight.bold,
                color: Colors.orange,
              ),
            ),
            const SizedBox(height: 20),
            Text(
              'Examples:',
              style: TextStyle(
                fontSize: 18,
                fontWeight: FontWeight.bold,
              ),
            ),
            const SizedBox(height: 10),
            Column(
              children: examples.map((example) => Padding(
                padding: const
EdgeInsets.symmetric(vertical: 4),
                child: Text(
                  example,
                  style: TextStyle(fontSize: 16),
                ),
              )).toList(),
            ),
          ],
        ),
        actions: [
          TextButton(
            onPressed: () => Navigator.pop(context),
            child: Text(
              'Close',
              style: TextStyle(color: Colors.blue),
            ),
          ),
        ],
      ),
    );
  }

  void _showNumberDetailDialog(int number) {
    showDialog(
      context: context,
      builder: (context) => AlertDialog(
        shape: RoundedRectangleBorder(borderRadius:
BorderRadius.circular(20)),
        title: Center(
          child: Text(
            'Number $number',
            style: TextStyle(
              fontSize: 32,
              fontWeight: FontWeight.bold,
              color: Colors.indigo[800],
            ),
          ),
        ),
        content: Column(
          mainAxisSize: MainAxisSize.min,
          children: [
            Text(
              number.toString(),
              style: TextStyle(
                fontSize: 72,
                fontWeight: FontWeight.bold,
                color: Colors.orange,
              ),
            ),
            const SizedBox(height: 20),
            Row(
              mainAxisAlignment: MainAxisAlignment.center,
              children: List.generate(
                number,
                (index) => Padding(
                  padding: const
EdgeInsets.symmetric(horizontal: 2),
                  child: Icon(
                    Icons.star,
                    color: Colors.amber,
                    size: 24,
                  ),
                ),
              ),
            ),
            const SizedBox(height: 20),
            Text(
              'Count with me: $number',
              style: TextStyle(fontSize: 16),
            ),
          ],
        ),
        actions: [
          TextButton(
            onPressed: () => Navigator.pop(context),
            child: Text(
              'Close',
              style: TextStyle(color: Colors.blue),
            ),
          ),
        ],
      ),
    );
  }

  List<String> _getExamplesForLetter(String letter) {
    final Map<String, List<String>> examples = {
      'A': ['Apple', 'Ant', 'Alligator'],
      'B': ['Ball', 'Banana', 'Butterfly'],
      'C': ['Cat', 'Cake', 'Carrot'],
      'D': ['Dog', 'Duck', 'Dolphin'],
      'E': ['Elephant', 'Egg', 'Eagle'],
      'F': ['Fish', 'Frog', 'Flower'],
      'G': ['Giraffe', 'Goat', 'Grapes'],
      'H': ['Horse', 'Hat', 'House'],
      'I': ['Ice cream', 'Igloo', 'Insect'],
      'J': ['Juice', 'Jellyfish', 'Jacket'],
      'K': ['Kite', 'Kangaroo', 'Key'],
      'L': ['Lion', 'Lemon', 'Leaf'],
      'M': ['Monkey', 'Moon', 'Milk'],
      'N': ['Nest', 'Nose', 'Nut'],
      'O': ['Orange', 'Owl', 'Octopus'],
      'P': ['Pig', 'Pencil', 'Parrot'],
      'Q': ['Queen', 'Quilt', 'Question mark'],
      'R': ['Rabbit', 'Rainbow', 'Robot'],
      'S': ['Sun', 'Snake', 'Star'],
      'T': ['Tiger', 'Tree', 'Train'],
      'U': ['Umbrella', 'Unicorn', 'Utensils'],
      'V': ['Violin', 'Vase', 'Vegetable'],
      'W': ['Watermelon', 'Whale', 'Watch'],
      'X': ['Xylophone', 'X-ray', 'Fox'],
      'Y': ['Yo-yo', 'Yarn', 'Yellow'],
      'Z': ['Zebra', 'Zoo', 'Zipper'],
    };

    return examples[letter] ?? [];
  }
}

// Quiz Screen
class QuizScreen extends StatefulWidget {
  const QuizScreen({Key? key}) : super(key: key);

  @override
  _QuizScreenState createState() => _QuizScreenState();
}
```

```
class _QuizScreenState extends State<QuizScreen> with SingleTickerProviderStateMixin {
  late TabController _tabController;

  @override
  void initState() {
    super.initState();
    _tabController = TabController(length: 2, vsync: this);
  }

  @override
  void dispose() {
    _tabController.dispose();
    super.dispose();
  }

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: Text(
          'Quiz Time!',
          style: TextStyle(color: Colors.white, fontWeight: FontWeight.bold),
        ),
        backgroundColor: Colors.purple,
        elevation: 0,
        bottom: TabBar(
          controller: _tabController,
          labelColor: Colors.white,
          unselectedLabelColor: Colors.white60,
          indicatorColor: Colors.orange,
          indicatorWeight: 4,
          tabs: const [
            Tab(text: 'Alphabets Quiz'),
            Tab(text: 'Numbers Quiz'),
          ],
        ),
      ),
      body: TabBarView(
        controller: _tabController,
        children: [
          AlphabetsQuizScreen(),
          NumbersQuizScreen(),
        ],
      ),
    );
  }
}

class AlphabetsQuizScreen extends StatefulWidget {
  const AlphabetsQuizScreen({Key? key}) : super(key: key);

  @override
  _AlphabetsQuizScreenState createState() =>
_AlphabetsQuizScreenState();
}

class _AlphabetsQuizScreenState extends
State<AlphabetsQuizScreen> {
  final Map<String, List<String>> _letterExamples = {
    'A': ['Apple', 'Ant', 'Alligator', 'Arrow'],
    'B': ['Ball', 'Banana', 'Butterfly', 'Book'],
    'C': ['Cat', 'Cake', 'Carrot', 'Crown'],
    'D': ['Dog', 'Duck', 'Dolphin', 'Donut'],
    'E': ['Elephant', 'Egg', 'Eagle', 'Eye'],
    'F': ['Fish', 'Frog', 'Flower', 'Flag'],
    'G': ['Giraffe', 'Goat', 'Grapes', 'Gift'],
    'H': ['Horse', 'Hat', 'House', 'Heart'],
    'I': ['Ice cream', 'Igloo', 'Insect', 'Island'],
    'J': ['Juice', 'Jellyfish', 'Jacket', 'Jar'],
    'K': ['Kite', 'Kangaroo', 'Key', 'Kitchen'],
    'L': ['Lion', 'Lemon', 'Leaf', 'Lamp'],
    'M': ['Monkey', 'Moon', 'Milk', 'Mountain'],
    'N': ['Nest', 'Nose', 'Nut', 'Night'],
    'O': ['Orange', 'Owl', 'Octopus', 'Ocean'],
    'P': ['Pig', 'Pencil', 'Parrot', 'Pizza'],
    'Q': ['Queen', 'Quilt', 'Question mark', 'Quail'],
    'R': ['Rabbit', 'Rainbow', 'Robot', 'Rose'],
    'S': ['Sun', 'Snake', 'Star', 'Strawberry'],
    'T': ['Tiger', 'Tree', 'Train', 'Table'],
    'U': ['Umbrella', 'Unicorn', 'Utensils', 'UFO'],
    'V': ['Violin', 'Vase', 'Vegetable', 'Van'],
    'W': ['Watermelon', 'Whale', 'Watch', 'Window'],
    'X': ['Xylophone', 'X-ray', 'Fox', 'Box'],
    'Y': ['Yo-yo', 'Yarn', 'Yellow', 'Yogurt'],
    'Z': ['Zebra', 'Zoo', 'Zipper', 'Zero'],
  };

  bool _quizStarted = false;
  int _currentQuestionIndex = 0;
  int _score = 0;
  List<Map<String, dynamic>> _questions = [];
  bool _answered = false;
  int? _selectedAnswerIndex;

  void _startQuiz() {
    _questions = _generateQuestions();
    setState(() {
      _quizStarted = true;
      _currentQuestionIndex = 0;
      _score = 0;
      _answered = false;
      _selectedAnswerIndex = null;
    });
  }

  List<Map<String, dynamic>> _generateQuestions() {
    final List<String> alphabet =
'ABCDEFGHIJKLMNOPQRSTUVWXYZ'.split('');
    final random = Random();
    List<Map<String, dynamic>> questions = [];

    // Generate 10 questions
    for (int i = 0; i < 10; i++) {
      // Randomly select a letter
      final correctLetter =
alphabet[random.nextInt(alphabet.length)];

      // Get a random word starting with this letter
      final correctExample =
_letterExamples[correctLetter]![random.nextInt(_letterExampl
es[correctLetter]!.length)];

      // Generate 3 incorrect options (different letters)
      List<String> incorrectLetters = List.from(alphabet);
      incorrectLetters.remove(correctLetter);
      incorrectLetters.shuffle();
      incorrectLetters = incorrectLetters.take(3).toList();

      // Create options (1 correct + 3 incorrect) and
shuffle them
      List<String> options = [correctLetter,
...incorrectLetters];
      options.shuffle();

      questions.add({
        'question': 'What letter does "$correctExample"
start with?',
        'options': options,
        'correctAnswer': correctLetter,
      });
    }

    return questions;
  }

  void _checkAnswer(int selectedIndex) {
    if (_answered) return;

    final correctAnswer =
_questions[_currentQuestionIndex]['correctAnswer'];
    final selectedAnswer =
_questions[_currentQuestionIndex]['options'][selectedIndex];

    setState(() {
      _answered = true;
      _selectedAnswerIndex = selectedIndex;

      if (selectedAnswer == correctAnswer) {
        _score++;
      }
    });

    // Wait 2 seconds before moving to next question
    Future.delayed(Duration(seconds: 2), () {
      if (_currentQuestionIndex < _questions.length - 1) {
        setState(() {
          _currentQuestionIndex++;
          _answered = false;
          _selectedAnswerIndex = null;
        });
      } else {
        // Quiz completed, save result
        _saveQuizResult();
      }
    });
  }

  void _saveQuizResult() async {
    // In a real app, this would store the result in a
database or shared preferences
    final result = QuizResult(
      type: 'alphabets',
      score: _score,
      dateTime: DateTime.now(),
    );

    // For demonstration purposes, we'll show a result
dialog
    showDialog(
      context: context,
      barrierDismissible: false,
      builder: (context) => AlertDialog(
        shape: RoundedRectangleBorder(borderRadius:
BorderRadius.circular(20)),
        title: Center(
          child: Text(
            'Quiz Completed!',
            style: TextStyle(
              color: Colors.indigo[800],
              fontWeight: FontWeight.bold,
            ),
          ),
        ),
        content: Column(
          mainAxisSize: MainAxisSize.min,
          children: [
            Icon(
              _score >= 7 ? Icons.sentiment_very_satisfied :
              _score >= 5 ? Icons.sentiment_satisfied :
Icons.sentiment_dissatisfied,
              color: _score >= 7 ? Colors.green :
                     _score >= 5 ? Colors.orange :
Colors.red,
              size: 64,
            ),
            const SizedBox(height: 16),
            Text(
              'Your Score:',
              style: TextStyle(fontSize: 18),
            ),
            Text(
              '$_score / 10',
              style: TextStyle(
                fontSize: 32,
                fontWeight: FontWeight.bold,
                color: _score >= 7 ? Colors.green :
                       _score >= 5 ? Colors.orange :
Colors.red,
```

	```
	       ),
	      ),
	      const SizedBox(height: 16),
	      Text(
	       _score >= 7 ? 'Great job!' :
	       _score >= 5 ? 'Good effort!' : 'Keep
	practicing!',
	       style: TextStyle(fontSize: 18),
	      ),
	     ],
	    ),
	    actions: [
	     TextButton(
	      onPressed: () {
	       Navigator.pop(context);
	       setState(() {
	        _quizStarted = false;
	       });
	      },
	      child: Text('Try Again'),
	     ),
	     ElevatedButton(
	      onPressed: () {
	       Navigator.pop(context);
	       setState(() {
	        _quizStarted = false;
	       });
	       // Update leaderboard
	       // This would normally handle updating the
	user's profile and leaderboard
	      },
	      style: ElevatedButton.styleFrom(
	       backgroundColor: Colors.orange,
	       shape: RoundedRectangleBorder(
	        borderRadius: BorderRadius.circular(20),
	       ),
	      ),
	      child: Text('Done'),
	     ),
	    ],
	   ),
	  );
	 }

	 @override
	 Widget build(BuildContext context) {
	```

```
    if (!_quizStarted) {
      return _buildStartScreen();
    } else {
      return _buildQuizScreen();
    }
  }

  Widget _buildStartScreen() {
    return Center(
      child: Column(
        mainAxisAlignment: MainAxisAlignment.center,
        children: [
          Icon(
            Icons.quiz,
            size: 80,
            color: Colors.purple,
          ),
          const SizedBox(height: 20),
          Text(
            'Alphabets Quiz',
            style: TextStyle(
              fontSize: 28,
              fontWeight: FontWeight.bold,
              color: Colors.indigo[800],
            ),
          ),
          const SizedBox(height: 10),
          Padding(
padding: const EdgeInsets.symmetric(horizontal: 32),
            child: Text(
              'Test your knowledge of the alphabet with fun
questions! Identify which letters words begin with.',
              textAlign: TextAlign.center,
              style: TextStyle(fontSize: 16),
            ),
          ),
          const SizedBox(height: 30),
          ElevatedButton(
            onPressed: _startQuiz,
            style: ElevatedButton.styleFrom(
              backgroundColor: Colors.orange,
              padding: EdgeInsets.symmetric(horizontal: 40,
vertical: 15),
              shape: RoundedRectangleBorder(
                borderRadius: BorderRadius.circular(30),
              ),
            ),
            child: Text(
              'Start Quiz',
              style: TextStyle(
                fontSize: 18,
                fontWeight: FontWeight.bold,
                color: Colors.white,
              ),
            ),
          ),
        ],
      ),
    );
  }

  Widget _buildQuizScreen() {
    final currentQuestion =
_questions[_currentQuestionIndex];

    return Padding(
      padding: const EdgeInsets.all(16.0),
      child: Column(
        children: [
          // Progress bar
          LinearProgressIndicator(
            value: (_currentQuestionIndex + 1) /
_questions.length,
            backgroundColor: Colors.grey[300],
            valueColor:
AlwaysStoppedAnimation<Color>(Colors.purple),
            minHeight: 10,
            borderRadius: BorderRadius.circular(5),
          ),
          SizedBox(height: 8),
          // Question counter
          Row(
            mainAxisAlignment:
MainAxisAlignment.spaceBetween,
            children: [
              Text(
                'Question ${_currentQuestionIndex +
1}/${_questions.length}',
                style: TextStyle(
                  fontSize: 16,
                  fontWeight: FontWeight.bold,
                ),
       ),
       Text(
        'Score: $_score',
        style: TextStyle(
         fontSize: 16,
         fontWeight: FontWeight.bold,
         color: Colors.indigo,
        ),
       ),
      ],
     ),
     SizedBox(height: 30),
     // Question
     Container(
      padding: EdgeInsets.all(16),
      decoration: BoxDecoration(
       color: Colors.purple[50],
       borderRadius: BorderRadius.circular(20),
       border: Border.all(color: Colors.purple[200]!,
width: 2),
      ),
      child: Text(
       currentQuestion['question'],
       style: TextStyle(
        fontSize: 22,
        fontWeight: FontWeight.bold,
        color: Colors.indigo[800],
       ),
       textAlign: TextAlign.center,
      ),
     ),
     SizedBox(height: 30),
     // Options
     ...List.generate(
      currentQuestion['options'].length,
      (index) => Padding(
       padding: const EdgeInsets.only(bottom: 16.0),
       child: InkWell(
        onTap: () => _answered ? null :
_checkAnswer(index),
        child: Container(
         width: double.infinity,
         padding: EdgeInsets.symmetric(vertical:
16),
         decoration: BoxDecoration(
          color: _answered
```

```
                        ? _selectedAnswerIndex == index
                            ? 
currentQuestion['options'][index] == 
currentQuestion['correctAnswer']
                                ? Colors.green[100]
                                : Colors.red[100]
                            : 
currentQuestion['options'][index] == 
currentQuestion['correctAnswer']
                                ? Colors.green[100]
                                : Colors.white
                        : Colors.white,
                    borderRadius: BorderRadius.circular(15),
                    border: Border.all(
                      color: _answered
                          ? _selectedAnswerIndex == index
                              ? 
currentQuestion['options'][index] == 
currentQuestion['correctAnswer']
                                  ? Colors.green
                                  : Colors.red
                              : 
currentQuestion['options'][index] == 
currentQuestion['correctAnswer']
                                  ? Colors.green
                                  : Colors.grey[400]!
                          : Colors.grey[400]!,
                      width: 2,
                    ),
                    boxShadow: [
                      BoxShadow(
                        color: Colors.grey.withOpacity(0.2),
                        spreadRadius: 1,
                        blurRadius: 3,
                        offset: Offset(0, 2),
                      ),
                    ],
                  ),
                  child: Center(
                    child: Text(
                      currentQuestion['options'][index],
                      style: TextStyle(
                        fontSize: 24,
                        fontWeight: FontWeight.bold,
                        color: _answered
                            ? _selectedAnswerIndex == index
```

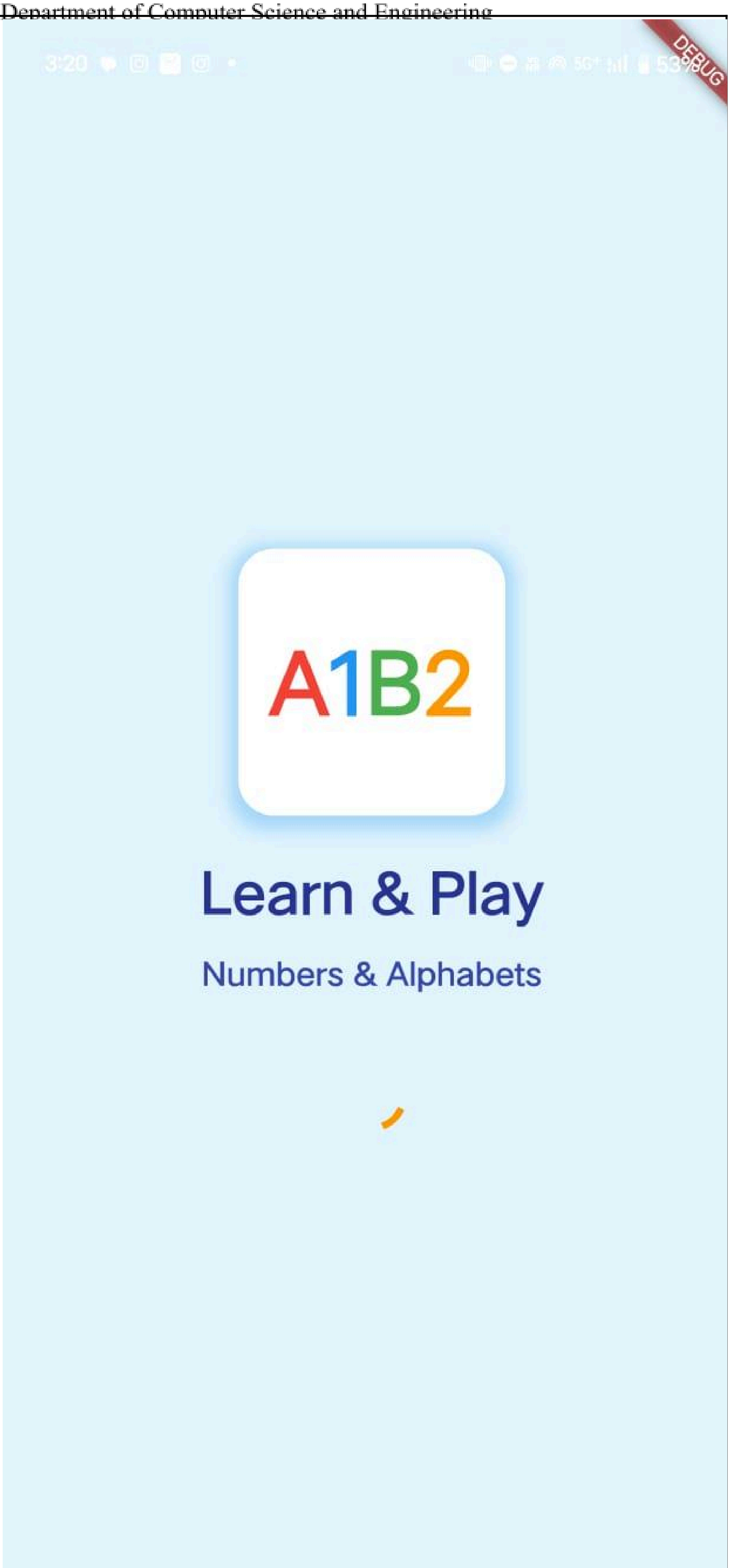
```
                                ?
currentQuestion['options'][index] ==
currentQuestion['correctAnswer']
                                    ? Colors.green[800]
                                    : Colors.red[800]
                                :
currentQuestion['options'][index] ==
currentQuestion['correctAnswer']
                                    ? Colors.green[800]
                                    : Colors.black87
                            : Colors.black87,
                      ),
                    ),
                  ),
                ),
              ),
            ),
          ),
        ],
      ),
    );
  }
}

class NumbersQuizScreen extends StatefulWidget {
  const NumbersQuizScreen({Key? key}) : super(key: key);

  @override
  _NumbersQuizScreenState createState() =>
_NumbersQuizScreenState();
}

class _NumbersQuizScreenState extends
State<NumbersQuizScreen> {
  bool _quizStarted = false;
  int _currentQuestionIndex = 0;
  int _score = 0;
  List<Map<String, dynamic>> _questions = [];
  bool _answered = false;
  int? _selectedAnswerIndex;

  void _startQuiz() {
    _questions = _generateQuestions();
    setState(() {
      _quizStarted = true;
      _currentQuestionIndex = 0;
      _score = 0;
      _answered = false;
      _selectedAnswerIndex = null;
    });
  }

  List<Map<String, dynamic>> _generateQuestions() {
    final random = Random();
    List<Map<String, dynamic>> questions = [];

    // Generate 10 questions
    for (int i = 0; i < 10; i++) {
      // Decide on question type
      final questionType = random.nextInt(3); // 0: count, 1: add, 2: subtract

      if (questionType == 0) {
        // Counting question
        final count = random.nextInt(10) + 1; // 1 to 10

        List<int> options = [count];
        while (options.length < 4) {
          final option = random.nextInt(10) + 1;
          if (!options.contains(option)) {
            options.add(option);
          }
        }
        options.shuffle();

        questions.add({
          'question': 'How many stars do you see?',
          'imageType': 'stars',
          'imageCount': count,
          'options': options,
          'correctAnswer': count,
        });
      } else if (questionType == 1) {
        // Addition question
        final a = random.nextInt(5) + 1; // 1 to 5
        final b = random.nextInt(5) + 1; // 1 to 5
        final sum = a + b;

        List<int> options = [sum];
        while (options.length < 4) {
          final option = random.nextInt(10) + 1;
          if (!options.contains(option)) {
```

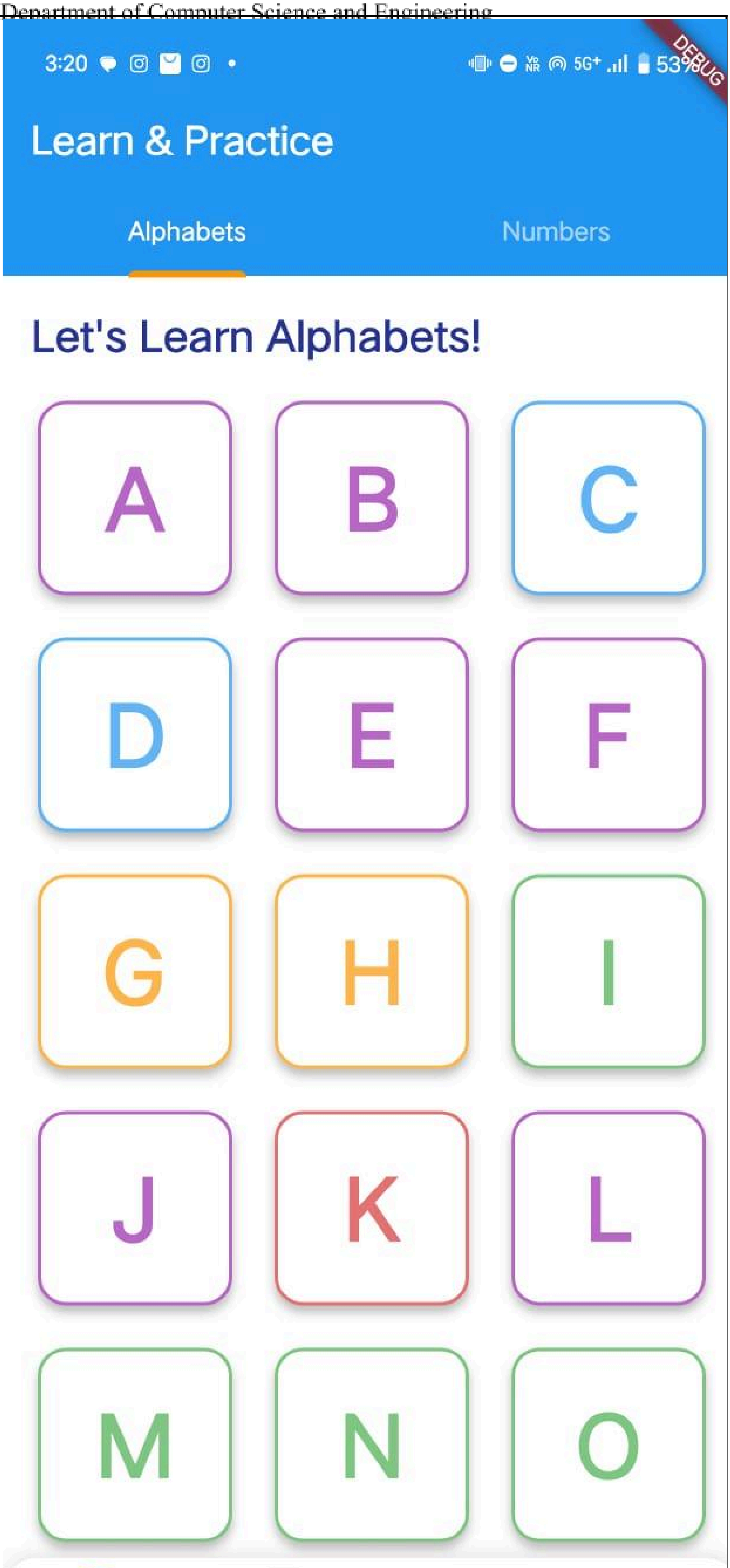
```
          options.add(option);
        }
      }
      options.shuffle();

      questions.add({
        'question': 'What is $a + $b?',
        'imageType': 'equation',
        'a': a,
        'b': b,
        'options': options,
        'correctAnswer': sum,
      });
    } else {
      // Subtraction question (ensure result is positive)
      final b = random.nextInt(5) + 1; // 1 to 5
      final result = random.nextInt(5) + 1; // 1 to 5
      final a = b + result; // ensures a > b

      List<int> options = [result];
      while (options.length < 4) {
        final option = random.nextInt(10) + 1;
        if (!options.contains(option)) {
          options.add(option);
        }
      }
      options.shuffle();

      questions.add({
        'question': 'What is $a - $b?',
        'imageType': 'equation',
        'a': a,
        'b': b,
        'operation': '-',
        'options': options,
        'correctAnswer': result,
      });
    }
  }

  return questions;
}

void _checkAnswer(int selectedIndex) {
  if (_answered) return;
```

```
    final correctAnswer =
_questions[_currentQuestionIndex]['correctAnswer'];
    final selectedAnswer =
_questions[_currentQuestionIndex]['options'][selectedIndex];

    setState(() {
      _answered = true;
      _selectedAnswerIndex = selectedIndex;

      if (selectedAnswer == correctAnswer) {
        _score++;
      }
    });

    // Wait 2 seconds before moving to next question
    Future.delayed(Duration(seconds: 2), () {
      if (_currentQuestionIndex < _questions.length - 1) {
        setState(() {
          _currentQuestionIndex++;
          _answered = false;
          _selectedAnswerIndex = null;
        });
      } else {
        // Quiz completed, save result
        _saveQuizResult();
      }
    });
  }

  void _saveQuizResult() async {
    // In a real app, this would store the result in a
database or shared preferences
    final result = QuizResult(
      type: 'numbers',
      score: _score,
      dateTime: DateTime.now(),
    );

    // For demonstration purposes, we'll show a result
dialog
    showDialog(
      context: context,
      barrierDismissible: false,
      builder: (context) => AlertDialog(
        shape: RoundedRectangleBorder(borderRadius:
BorderRadius.circular(20)),
        title: Center(
          child: Text(
            'Quiz Completed!',
            style: TextStyle(
              color: Colors.indigo[800],
              fontWeight: FontWeight.bold,
            ),
          ),
        ),
        content: Column(
          mainAxisSize: MainAxisSize.min,
          children: [
            Icon(
              _score >= 7 ? Icons.sentiment_very_satisfied :
              _score >= 5 ? Icons.sentiment_satisfied :
Icons.sentiment_dissatisfied,
              color: _score >= 7 ? Colors.green :
                    _score >= 5 ? Colors.orange :
Colors.red,
              size: 64,
            ),
            const SizedBox(height: 16),
            Text(
              'Your Score:',
              style: TextStyle(fontSize: 18),
            ),
            Text(
              '$_score / 10',
              style: TextStyle(
                fontSize: 32,
                fontWeight: FontWeight.bold,
                color: _score >= 7 ? Colors.green :
                      _score >= 5 ? Colors.orange :
Colors.red,
              ),
            ),
            const SizedBox(height: 16),
            Text(
              _score >= 7 ? 'Great job!' :
              _score >= 5 ? 'Good effort!' : 'Keep
practicing!',
              style: TextStyle(fontSize: 18),
            ),
          ],
        ),
        actions: [
          TextButton(
            onPressed: () {
              Navigator.pop(context);
              setState(() {
                _quizStarted = false;
              });
            },
            child: Text('Try Again'),
          ),
          ElevatedButton(
            onPressed: () {
              Navigator.pop(context);
              setState(() {
                _quizStarted = false;
              });
              // Update leaderboard
              // This would normally handle updating the
user's profile and leaderboard
            },
            style: ElevatedButton.styleFrom(
              backgroundColor: Colors.orange,
              shape: RoundedRectangleBorder(
                borderRadius: BorderRadius.circular(20),
              ),
            ),
            child: Text('Done'),
          ),
        ],
      ),
    );
  }

  @override
  Widget build(BuildContext context) {
    if (!_quizStarted) {
      return _buildStartScreen();
    } else {
      return _buildQuizScreen();
    }
  }

  Widget _buildStartScreen() {
    return Center(
      child: Column(
        mainAxisAlignment: MainAxisAlignment.center,
        children: [
          Icon(
            Icons.format_list_numbered,
            size: 80,
            color: Colors.blue,
          ),
          const SizedBox(height: 20),
          Text(
            'Numbers Quiz',
            style: TextStyle(
              fontSize: 28,
              fontWeight: FontWeight.bold,
              color: Colors.indigo[800],
            ),
          ),
          const SizedBox(height: 10),
          Padding(
            padding: const EdgeInsets.symmetric(horizontal: 32),
            child: Text(
              'Test your number skills with fun questions on counting, addition, and subtraction!',
              textAlign: TextAlign.center,
              style: TextStyle(fontSize: 16),
            ),
          ),
          const SizedBox(height: 30),
          ElevatedButton(
            onPressed: _startQuiz,
            style: ElevatedButton.styleFrom(
              backgroundColor: Colors.orange,
              padding: EdgeInsets.symmetric(horizontal: 40, vertical: 15),
              shape: RoundedRectangleBorder(
                borderRadius: BorderRadius.circular(30),
              ),
            ),
            child: Text(
              'Start Quiz',
              style: TextStyle(
                fontSize: 18,
                fontWeight: FontWeight.bold,
                color: Colors.white,
              ),
            ),
          ),
        ],
      ),
    );
  }

  Widget _buildQuizScreen() {
    final currentQuestion =
_questions[_currentQuestionIndex];

    return Padding(
      padding: const EdgeInsets.all(16.0),
      child: Column(
        children: [
          // Progress bar
          LinearProgressIndicator(
            value: (_currentQuestionIndex + 1) /
_questions.length,
            backgroundColor: Colors.grey[300],
            valueColor:
AlwaysStoppedAnimation<Color>(Colors.blue),
            minHeight: 10,
            borderRadius: BorderRadius.circular(5),
          ),
          SizedBox(height: 8),
          // Question counter
          Row(
            mainAxisAlignment:
MainAxisAlignment.spaceBetween,
            children: [
              Text(
                'Question ${_currentQuestionIndex +
1}/${_questions.length}',
                style: TextStyle(
                  fontSize: 16,
                  fontWeight: FontWeight.bold,
                ),
              ),
              Text(
                'Score: $_score',
                style: TextStyle(
                  fontSize: 16,
                  fontWeight: FontWeight.bold,
                  color: Colors.indigo,
                ),
              ),
            ],
          ),
          SizedBox(height: 30),
          // Question
          Container(
            padding: EdgeInsets.all(16),
            decoration: BoxDecoration(
              color: Colors.blue[50],
              borderRadius: BorderRadius.circular(20),
              border: Border.all(color: Colors.blue[200]!,
width: 2),
            ),
            child: Text(
              currentQuestion['question'],
              style: TextStyle(
                fontSize: 22,
                fontWeight: FontWeight.bold,
                color: Colors.indigo[800],
              ),
              textAlign: TextAlign.center,
            ),
          ),
          SizedBox(height: 20),
          // Image or Visual representation
          _buildQuestionVisual(currentQuestion),
          SizedBox(height: 30),
          // Options
          ...List.generate(
            currentQuestion['options'].length,
            (index) => Padding(
              padding: const EdgeInsets.only(bottom: 16.0),
              child: InkWell(
                onTap: () => _answered ? null :
_checkAnswer(index),
                child: Container(
                  width: double.infinity,
                  padding: EdgeInsets.symmetric(vertical:
16),
                  decoration: BoxDecoration(
                    color: _answered
                        ? _selectedAnswerIndex == index
                            ?
currentQuestion['options'][index] ==
currentQuestion['correctAnswer']
                                ? Colors.green[100]
                                : Colors.red[100]
                            :
currentQuestion['options'][index] ==
currentQuestion['correctAnswer']
                                ? Colors.green[100]
                                : Colors.white
                        : Colors.white,
                    borderRadius: BorderRadius.circular(15),
                    border: Border.all(
                      color: _answered
                          ? _selectedAnswerIndex == index
                              ?
currentQuestion['options'][index] ==
currentQuestion['correctAnswer']
                                  ? Colors.green
                                  : Colors.red
                              :
currentQuestion['options'][index] ==
currentQuestion['correctAnswer']
                                  ? Colors.green
                                  : Colors.grey[400]!
                          : Colors.grey[400]!,
                      width: 2,
                    ),
                    boxShadow: [
                      BoxShadow(
                        color: Colors.grey.withOpacity(0.2),
                        spreadRadius: 1,
                        blurRadius: 3,
                        offset: Offset(0, 2),
                      ),
                    ],
                  ),
                  child: Center(
                    child: Text(

currentQuestion['options'][index].toString(),
                      style: TextStyle(
                        fontSize: 24,
                        fontWeight: FontWeight.bold,
                        color: _answered
                            ? _selectedAnswerIndex == index
                                ?
currentQuestion['options'][index] ==
currentQuestion['correctAnswer']
                                    ? Colors.green[800]
                                    : Colors.red[800]
                                :
currentQuestion['options'][index] ==
currentQuestion['correctAnswer']
                  ? Colors.green[800]
                  : Colors.black87
              : Colors.black87,
           ),
          ),
         ),
        ),
       ),
      ),
     ),
    ],
   ),
  );
 }

 Widget _buildQuestionVisual(Map<String, dynamic> question)
{
  if (question['imageType'] == 'stars') {
   return Container(
    height: 120,
    alignment: Alignment.center,
    child: Wrap(
     spacing: 8,
     runSpacing: 8,
     alignment: WrapAlignment.center,
     children: List.generate(
      question['imageCount'],
      (index) => Icon(
       Icons.star,
       color: Colors.amber,
       size: 32,
      ),
     ),
    ),
   );
  } else if (question['imageType'] == 'equation') {
   final String operation =
question.containsKey('operation') ? question['operation'] :
'+';
   return Container(
    height: 120,
    alignment: Alignment.center,
    child: Row(
     mainAxisAlignment: MainAxisAlignment.center,
     children: [
          _buildNumberWithDots(question['a']),
          SizedBox(width: 16),
          Text(
            operation,
            style: TextStyle(
              fontSize: 36,
              fontWeight: FontWeight.bold,
            ),
          ),
          SizedBox(width: 16),
          _buildNumberWithDots(question['b']),
          SizedBox(width: 16),
          Text(
            '=',
            style: TextStyle(
              fontSize: 36,
              fontWeight: FontWeight.bold,
            ),
          ),
          SizedBox(width: 16),
          Text(
            '?',
            style: TextStyle(
              fontSize: 36,
              fontWeight: FontWeight.bold,
              color: Colors.orange,
            ),
          ),
        ],
      ),
    );
  }

  return SizedBox.shrink();
}

Widget _buildNumberWithDots(int number) {
  return Column(
    mainAxisSize: MainAxisSize.min,
    children: [
      Text(
        number.toString(),
        style: TextStyle(
          fontSize: 32,
          fontWeight: FontWeight.bold,
        ),
        ),
        SizedBox(height: 8),
        Wrap(
          spacing: 4,
          runSpacing: 4,
          children: List.generate(
            number,
            (index) => Container(
              width: 12,
              height: 12,
              decoration: BoxDecoration(
                color: Colors.blue,
                shape: BoxShape.circle,
              ),
            ),
          ),
        ),
      ],
    );
  }
}

// LeaderBoard Screen
class LeaderBoardScreen extends StatefulWidget {
  const LeaderBoardScreen({Key? key}) : super(key: key);

  @override
  _LeaderBoardScreenState createState() =>
_LeaderBoardScreenState();
}

class _LeaderBoardScreenState extends
State<LeaderBoardScreen> with SingleTickerProviderStateMixin
{
  late TabController _tabController;

  // Sample leaderboard data
  final List<Map<String, dynamic>> _alphabetsLeaders = [
    {'name': 'Emma', 'avatar': 'assets/avatar1.png',
'score': 95},
    {'name': 'Noah', 'avatar': 'assets/avatar2.png',
'score': 92},
    {'name': 'Olivia', 'avatar': 'assets/avatar3.png',
'score': 88},
    {'name': 'Liam', 'avatar': 'assets/avatar4.png',
'score': 85},
    {'name': 'Ava', 'avatar': 'assets/avatar5.png', 'score': 82},
    {'name': 'William', 'avatar': 'assets/avatar6.png', 'score': 80},
    {'name': 'Sophia', 'avatar': 'assets/avatar7.png', 'score': 78},
    {'name': 'James', 'avatar': 'assets/avatar8.png', 'score': 75},
    {'name': 'Isabella', 'avatar': 'assets/avatar9.png', 'score': 72},
    {'name': 'Benjamin', 'avatar': 'assets/avatar10.png', 'score': 70},
  ];

  final List<Map<String, dynamic>> _numbersLeaders = [
    {'name': 'Sophia', 'avatar': 'assets/avatar7.png', 'score': 98},
    {'name': 'Liam', 'avatar': 'assets/avatar4.png', 'score': 94},
    {'name': 'Emma', 'avatar': 'assets/avatar1.png', 'score': 90},
    {'name': 'Noah', 'avatar': 'assets/avatar2.png', 'score': 86},
    {'name': 'Isabella', 'avatar': 'assets/avatar9.png', 'score': 84},
    {'name': 'William', 'avatar': 'assets/avatar6.png', 'score': 81},
    {'name': 'Olivia', 'avatar': 'assets/avatar3.png', 'score': 79},
    {'name': 'Benjamin', 'avatar': 'assets/avatar10.png', 'score': 77},
    {'name': 'Ava', 'avatar': 'assets/avatar5.png', 'score': 74},
    {'name': 'James', 'avatar': 'assets/avatar8.png', 'score': 71},
  ];

  @override
  void initState() {
    super.initState();
    _tabController = TabController(length: 2, vsync: this);
  }

  @override
  void dispose() {
    _tabController.dispose();
    super.dispose();
  }

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: Text(
          'Leaderboard',
          style: TextStyle(color: Colors.white, fontWeight:
FontWeight.bold),
        ),
        backgroundColor: Colors.green,
        elevation: 0,
        bottom: TabBar(
          controller: _tabController,
          labelColor: Colors.white,
          unselectedLabelColor: Colors.white60,
          indicatorColor: Colors.orange,
          indicatorWeight: 4,
          tabs: const [
            Tab(text: 'Alphabets'),
            Tab(text: 'Numbers'),
          ],
        ),
      ),
      body: TabBarView(
        controller: _tabController,
        children: [
          _buildLeaderboardTab(_alphabetsLeaders),
          _buildLeaderboardTab(_numbersLeaders),
        ],
      ),
    );
  }

  Widget _buildLeaderboardTab(List<Map<String, dynamic>>
leaders) {
    return Column(
      children: [
        const SizedBox(height: 16),
        // Top 3 card
        Container(
          margin: EdgeInsets.symmetric(horizontal: 16),
          padding: EdgeInsets.all(16),
          decoration: BoxDecoration(
            gradient: LinearGradient(
              colors: [Colors.green[300]!,
Colors.green[100]!],
              begin: Alignment.topLeft,
              end: Alignment.bottomRight,
            ),
            borderRadius: BorderRadius.circular(20),
            boxShadow: [
              BoxShadow(
                color: Colors.green.withOpacity(0.3),
                spreadRadius: 1,
                blurRadius: 10,
                offset: Offset(0, 4),
              ),
            ],
          ),
          child: Row(
            mainAxisAlignment:
MainAxisAlignment.spaceEvenly,
            children: [
              _buildTopPlayer(leaders[1], 2,
Colors.grey[400]!),
              _buildTopPlayer(leaders[0], 1, Colors.amber),
              _buildTopPlayer(leaders[2], 3,
Colors.brown[300]!),
            ],
          ),
        ),
        const SizedBox(height: 16),
        // Rest of the leaderboard
        Expanded(
          child: ListView.builder(
            padding: EdgeInsets.symmetric(horizontal: 16),
            itemCount: leaders.length - 3,
            itemBuilder: (context, index) {
              final playerIndex = index + 3;
              final player = leaders[playerIndex];

              return Container(
                margin: EdgeInsets.only(bottom: 8),
                decoration: BoxDecoration(
                  color: Colors.white,
                  borderRadius: BorderRadius.circular(12),
                  boxShadow: [
                    BoxShadow(
                      color: Colors.grey.withOpacity(0.2),
                      spreadRadius: 1,
                      blurRadius: 3,
                      offset: Offset(0, 2),
                    ),
                  ],
                ),
                child: ListTile(
                  leading: Container(
                    width: 36,
                    height: 36,
                    alignment: Alignment.center,
                    decoration: BoxDecoration(
                      color: Colors.green[100],
                      shape: BoxShape.circle,
                    ),
                    child: Text(
                      '${playerIndex + 1}',
                      style: TextStyle(
                        fontWeight: FontWeight.bold,
                        color: Colors.green[800],
                      ),
                    ),
                  ),
                  title: Text(
                    player['name'],
                    style: TextStyle(fontWeight:
FontWeight.bold),
                  ),
                  trailing: Container(
                    padding:
EdgeInsets.symmetric(horizontal: 12, vertical: 6),
                    decoration: BoxDecoration(
                      color: Colors.green[50],
                      borderRadius:
BorderRadius.circular(20),
                      border: Border.all(color:
Colors.green[300]!),
                    ),
                    child: Text(
                      '${player['score']}',
                      style: TextStyle(
                        fontWeight: FontWeight.bold,
                        color: Colors.green[800],
                      ),
                    ),
                  ),
                ),
              );
            },
          ),
        ),
      ],
    );
  }

  Widget _buildTopPlayer(Map<String, dynamic> player, int 
position, Color medalColor) {
    return Column(
      mainAxisSize: MainAxisSize.min,
      children: [
        // Medal
        position == 1
            ? Stack(
                alignment: Alignment.center,
                children: [
                  Icon(
                    Icons.star,
                    size: 50,
                    color: medalColor,
                  ),
                  Text(
                    '$position',
                    style: TextStyle(
                      fontWeight: FontWeight.bold,
                      color: Colors.white,
                      fontSize: 16,
                    ),
                  ),
                ],
              )
            : Container(
                width: 34,
                height: 34,
                alignment: Alignment.center,
                decoration: BoxDecoration(
                  color: medalColor,
                  shape: BoxShape.circle,
                ),
                child: Text(
                  '$position',
                  style: TextStyle(
                    fontWeight: FontWeight.bold,
                    color: Colors.white,
                  ),
                ),
              ),
        const SizedBox(height: 8),
        // Avatar placeholder
        Container(
          width: position == 1 ? 60 : 45,
          height: position == 1 ? 60 : 45,
          decoration: BoxDecoration(
            shape: BoxShape.circle,
            color: Colors.white,
            border: Border.all(
              color: position == 1 ? Colors.amber : 
Colors.grey[300]!,
              width: 2,
            ),
          ),
          child: Icon(
            Icons.person,
            size: position == 1 ? 40 : 30,
            color: Colors.green[300],
          ),
        ),
        const SizedBox(height: 8),
        // Name
        Text(
          player['name'],
          style: TextStyle(
            fontWeight: FontWeight.bold,
            fontSize: position == 1 ? 16 : 14,
          ),
        ),
        // Score
        Text(
          '${player['score']}',
          style: TextStyle(
            fontWeight: FontWeight.bold,
            color: Colors.green[800],
            fontSize: position == 1 ? 18 : 16,
          ),
        ),
      ],
    );
  }
}
```

```
// Profile Screen
class ProfileScreen extends StatefulWidget {
  const ProfileScreen({Key? key}) : super(key: key);

  @override
  _ProfileScreenState createState() =>
_ProfileScreenState();
}

class _ProfileScreenState extends State<ProfileScreen> {
  // Sample user profile data
  final User _user = User(
    name: 'Alex',
    avatar: 'assets/user_avatar.png',
    totalScore: 487,
    alphabetsLevel: 4,
    numbersLevel: 5,
    quizHistory: [
      QuizResult(type: 'alphabets', score: 8, dateTime:
DateTime.now().subtract(Duration(days: 1))),
      QuizResult(type: 'numbers', score: 9, dateTime:
DateTime.now().subtract(Duration(days: 2))),
      QuizResult(type: 'alphabets', score: 7, dateTime:
DateTime.now().subtract(Duration(days: 3))),
      QuizResult(type: 'numbers', score: 8, dateTime:
DateTime.now().subtract(Duration(days: 4))),
      QuizResult(type: 'alphabets', score: 6, dateTime:
DateTime.now().subtract(Duration(days: 5))),
    ],
  );

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: Text(
          'My Profile',
          style: TextStyle(color: Colors.white, fontWeight:
FontWeight.bold),
        ),
        backgroundColor: Colors.orange,
        elevation: 0,
      ),
      body: SingleChildScrollView(
        child: Column(
          children: [
            // Profile header
            Container(
              padding: EdgeInsets.all(16),
              decoration: BoxDecoration(
                color: Colors.orange,
                borderRadius: BorderRadius.only(
                  bottomLeft: Radius.circular(30),
                  bottomRight: Radius.circular(30),
                ),
              ),
              child: Column(
                children: [
                  // Avatar
                  Container(
                    width: 100,
                    height: 100,
                    decoration: BoxDecoration(
                      shape: BoxShape.circle,
                      color: Colors.white,
                      border: Border.all(color:
Colors.white, width: 3),
                    ),
                    child: Icon(
                      Icons.person,
                      size: 70,
                      color: Colors.orange[300],
                    ),
                  ),
                  const SizedBox(height: 16),
                  // Name
                  Text(
                    _user.name,
                    style: TextStyle(
                      fontSize: 24,
                      fontWeight: FontWeight.bold,
                      color: Colors.white,
                    ),
                  ),
                  const SizedBox(height: 8),
                  // Total score
                  Container(
                    padding:
EdgeInsets.symmetric(horizontal: 16, vertical: 6),
                    decoration: BoxDecoration(
                      color: Colors.white,
                      borderRadius: 
BorderRadius.circular(20),
                    ),
                    child: Text(
                      'Total Score: ${_user.totalScore}',
                      style: TextStyle(
                        fontWeight: FontWeight.bold,
                        color: Colors.orange,
                      ),
                    ),
                  ),
                ],
              ),
            ),

            const SizedBox(height: 24),

            // Learning progress
            Padding(
              padding: const 
EdgeInsets.symmetric(horizontal: 16),
              child: Column(
                crossAxisAlignment: 
CrossAxisAlignment.start,
                children: [
                  Text(
                    'Learning Progress',
                    style: TextStyle(
                      fontSize: 20,
                      fontWeight: FontWeight.bold,
                      color: Colors.indigo[800],
                    ),
                  ),
                  const SizedBox(height: 16),
                  // Alphabets progress
                  _buildProgressCard(
                    title: 'Alphabets',
                    level: _user.alphabetsLevel,
                    progress: 0.8,
                    color: Colors.purple,
                    icon: Icons.text_fields,
                  ),
                  const SizedBox(height: 16),
                  // Numbers progress
                  _buildProgressCard(
                    title: 'Numbers',
                    level: _user.numbersLevel,
                    progress: 0.65,
                    color: Colors.blue,
                    icon: Icons.format_list_numbered,
                  ),
                ],
              ),
            ),

            const SizedBox(height: 24),

            // Recent activities
            Padding(
              padding: const
EdgeInsets.symmetric(horizontal: 16),
              child: Column(
                crossAxisAlignment:
CrossAxisAlignment.start,
                children: [
                  Text(
                    'Recent Activities',
                    style: TextStyle(
                      fontSize: 20,
                      fontWeight: FontWeight.bold,
                      color: Colors.indigo[800],
                    ),
                  ),
                  const SizedBox(height: 16),
                  // Activity list
                  ..._buildRecentActivities(),
                ],
              ),
            ),

            const SizedBox(height: 24),

            // Edit profile button
            Padding(
              padding: const
EdgeInsets.symmetric(horizontal: 16),
              child: ElevatedButton(
                onPressed: () {
                  _showEditProfileDialog();
                },
                style: ElevatedButton.styleFrom(
                  backgroundColor: Colors.orange,
                  padding: EdgeInsets.symmetric(vertical:
16),
                  shape: RoundedRectangleBorder(
                    borderRadius: BorderRadius.circular(15),
                  ),
                  minimumSize: Size(double.infinity, 50),
                ),
                child: Text(
                  'Edit Profile',
                  style: TextStyle(
                    fontSize: 16,
                    fontWeight: FontWeight.bold,
                    color: Colors.white,
                  ),
                ),
              ),
            ),

            const SizedBox(height: 24),

            // Achievements section
            Padding(
              padding: const
EdgeInsets.symmetric(horizontal: 16),
              child: Column(
                crossAxisAlignment:
CrossAxisAlignment.start,
                children: [
                  Text(
                    'Achievements',
                    style: TextStyle(
                      fontSize: 20,
                      fontWeight: FontWeight.bold,
                      color: Colors.indigo[800],
                    ),
                  ),
                  const SizedBox(height: 16),
                  _buildAchievementsGrid(),
                ],
              ),
            ),

            const SizedBox(height: 32),
          ],
        ),
      ),
    );
  }

  Widget _buildProgressCard({
    required String title,
    required int level,
    required double progress,
    required Color color,
    required IconData icon,
  }) {
    return Container(
      padding: EdgeInsets.all(16),
      decoration: BoxDecoration(
        color: Colors.white,
        borderRadius: BorderRadius.circular(15),
        boxShadow: [
          BoxShadow(
            color: Colors.grey.withOpacity(0.2),
            spreadRadius: 1,
            blurRadius: 5,
            offset: Offset(0, 3),
          ),
        ],
      ),
      child: Column(
        crossAxisAlignment: CrossAxisAlignment.start,
        children: [
          Row(
            children: [
              Container(
                padding: EdgeInsets.all(8),
                decoration: BoxDecoration(
                  color: color.withOpacity(0.2),
                  borderRadius: BorderRadius.circular(10),
                ),
                child: Icon(
                  icon,
                  color: color,
                  size: 24,
                ),
              ),
              const SizedBox(width: 12),
              Text(
                title,
                style: TextStyle(
                  fontSize: 18,
                  fontWeight: FontWeight.bold,
                ),
              ),
              Spacer(),
              Container(
                padding: EdgeInsets.symmetric(horizontal:
12, vertical: 6),
                decoration: BoxDecoration(
                  color: color.withOpacity(0.2),
                  borderRadius: BorderRadius.circular(20),
                ),
                child: Text(
                  'Level $level',
                  style: TextStyle(
                    fontWeight: FontWeight.bold,
                    color: color,
                  ),
                ),
              ),
            ],
          ),
          const SizedBox(height: 16),
          LinearProgressIndicator(
            value: progress,
            backgroundColor: Colors.grey[200],
            valueColor:
AlwaysStoppedAnimation<Color>(color),
            minHeight: 10,
            borderRadius: BorderRadius.circular(5),
          ),
          const SizedBox(height: 8),
          Text(
            '${(progress * 100).toInt()}% to next level',
            style: TextStyle(
              color: Colors.grey[600],
              fontSize: 12,
            ),
          ),
        ],
      ),
    );
  }

  List<Widget> _buildRecentActivities() {
    return _user.quizHistory.map((result) {
      final isAlphabets = result.type == 'alphabets';
      final date = _formatDate(result.dateTime);

      return Container(
        margin: EdgeInsets.only(bottom: 10),
        padding: EdgeInsets.all(12),
        decoration: BoxDecoration(
          color: Colors.white,
          borderRadius: BorderRadius.circular(12),
          boxShadow: [
            BoxShadow(
              color: Colors.grey.withOpacity(0.1),
              spreadRadius: 1,
              blurRadius: 3,
              offset: Offset(0, 2),
            ),
          ],
        ),
        child: Row(
          children: [
            Container(
              padding: EdgeInsets.all(8),
              decoration: BoxDecoration(
                color: isAlphabets ? 
Colors.purple.withOpacity(0.2) : 
Colors.blue.withOpacity(0.2),
                borderRadius: BorderRadius.circular(10),
              ),
              child: Icon(
                isAlphabets ? Icons.text_fields : 
Icons.format_list_numbered,
                color: isAlphabets ? Colors.purple : 
Colors.blue,
                size: 20,
              ),
            ),
            const SizedBox(width: 12),
            Expanded(
              child: Column(
                crossAxisAlignment: 
CrossAxisAlignment.start,
                children: [
                  Text(
                    'Completed ${isAlphabets ? 'Alphabets' : 
'Numbers'} Quiz',
                    style: TextStyle(
                      fontWeight: FontWeight.bold,
                    ),
                  ),
                  const SizedBox(height: 2),
                  Text(
                    date,
                    style: TextStyle(
                      color: Colors.grey[600],
                      fontSize: 12,
                    ),
                  ),
                ],
              ),
            ),
            Container(
              padding: EdgeInsets.symmetric(horizontal: 12,
vertical: 6),
              decoration: BoxDecoration(
                color: result.score >= 7 ? Colors.green[100]
:
                      result.score >= 5 ? Colors.orange[100]
: Colors.red[100],
                borderRadius: BorderRadius.circular(20),
              ),
              child: Text(
                '${result.score}/10',
                style: TextStyle(
                  fontWeight: FontWeight.bold,
                  color: result.score >= 7 ?
Colors.green[800] :
                        result.score >= 5 ?
Colors.orange[800] : Colors.red[800],
                ),
              ),
            ),
          ],
        ),
      );
    }).toList();
  }

  String _formatDate(DateTime dateTime) {
    final now = DateTime.now();
    final difference = now.difference(dateTime);

    if (difference.inDays == 0) {
      return 'Today';
    } else if (difference.inDays == 1) {
      return 'Yesterday';
    } else {
      return '${difference.inDays} days ago';
    }
  }

  void _showEditProfileDialog() {
    final nameController = TextEditingController(text:
_user.name);

    showDialog(
      context: context,
      builder: (context) => AlertDialog(
        shape: RoundedRectangleBorder(borderRadius:
BorderRadius.circular(20)),
        title: Text(
          'Edit Profile',
          textAlign: TextAlign.center,
          style: TextStyle(
            fontWeight: FontWeight.bold,
            color: Colors.indigo[800],
          ),
        ),
        content: Column(
          mainAxisSize: MainAxisSize.min,
          children: [
            // Avatar edit
            Stack(
              alignment: Alignment.bottomRight,
              children: [
                Container(
                  width: 100,
                  height: 100,
                  decoration: BoxDecoration(
                    shape: BoxShape.circle,
                    color: Colors.grey[200],
                    border: Border.all(color: Colors.orange,
width: 3),
                  ),
                  child: Icon(
                    Icons.person,
                    size: 70,
                    color: Colors.orange[300],
                  ),
                ),
                Container(
                  decoration: BoxDecoration(
                    color: Colors.orange,
                    shape: BoxShape.circle,
                  ),
                  child: IconButton(
                    icon: Icon(Icons.camera_alt, color:
Colors.white, size: 20),
                    onPressed: () {
                      // Image picker functionality would go
here
                    },
                  ),
                ),
              ],
            ),
            const SizedBox(height: 16),
            // Name field
            TextField(
              controller: nameController,
              decoration: InputDecoration(
                labelText: 'Name',
                border: OutlineInputBorder(
                  borderRadius: BorderRadius.circular(15),
                ),
                focusedBorder: OutlineInputBorder(
                  borderRadius: BorderRadius.circular(15),
                  borderSide: BorderSide(color:
Colors.orange, width: 2),
                ),
              ),
            ),
          ],
        ),
        actions: [
          TextButton(
            onPressed: () => Navigator.pop(context),
            child: Text(
              'Cancel',
              style: TextStyle(color: Colors.grey[600]),
            ),
          ),
          ElevatedButton(
            onPressed: () {
              final newName = nameController.text.trim();
              if (newName.isNotEmpty) {
                setState(() {
                  // This is where we would update the
user's name in a real app
                  // _user.name = newName;
                });
              }
              Navigator.pop(context);
            },
            style: ElevatedButton.styleFrom(
              backgroundColor: Colors.orange,
              shape: RoundedRectangleBorder(
                borderRadius: BorderRadius.circular(15),
              ),
            ),
            child: Text('Save'),
          ),
        ],
      ),
    );
  }

  Widget _buildAchievementsGrid() {
    final achievements = [
      {
        'title': 'First Steps',
        'description': 'Complete your first quiz',
        'icon': Icons.emoji_events,
        'unlocked': true,
      },
      {
        'title': 'Perfect Score',
        'description': 'Get 10/10 in any quiz',
        'icon': Icons.star,
        'unlocked': true,
      },
      {
        'title': 'Fast Learner',
        'description': 'Complete 5 quizzes in one day',
        'icon': Icons.speed,
        'unlocked': false,
      },
      {
        'title': 'Alphabet Master',
        'description': 'Reach level 10 in Alphabets',
        'icon': Icons.text_fields,
        'unlocked': false,
      },
      {
        'title': 'Number Wizard',
        'description': 'Reach level 10 in Numbers',
        'icon': Icons.format_list_numbered,
        'unlocked': false,
      },
      {
        'title': 'Consistent Learner',
        'description': 'Practice for 7 days in a row',
        'icon': Icons.calendar_today,
        'unlocked': false,
      },
    ];

    return GridView.builder(
      physics: NeverScrollableScrollPhysics(),
      shrinkWrap: true,
      gridDelegate:
SliverGridDelegateWithFixedCrossAxisCount(
        crossAxisCount: 2,
        crossAxisSpacing: 10,
        mainAxisSpacing: 10,
        childAspectRatio: 1.2,
      ),
      itemCount: achievements.length,
      itemBuilder: (context, index) {
        final achievement = achievements[index];
        final bool unlocked = achievement['unlocked'] as
bool;

        return Container(
          padding: EdgeInsets.all(10),
          decoration: BoxDecoration(
            color: Colors.white,
            borderRadius: BorderRadius.circular(15),
            border: Border.all(
              color: unlocked ? Colors.orange :
Colors.grey[300]!,
              width: 2,
            ),
            boxShadow: [
              BoxShadow(
                color: Colors.grey.withOpacity(0.1),
                spreadRadius: 1,
                blurRadius: 3,
                offset: Offset(0, 2),
              ),
            ],
          ),
          child: Column(
            mainAxisAlignment: MainAxisAlignment.center,
            children: [
              Icon(
                achievement['icon'] as IconData,
                color: unlocked ? Colors.orange :
Colors.grey[400],
                size: 32,
              ),
              const SizedBox(height: 8),
              Text(
                achievement['title'] as String,
                textAlign: TextAlign.center,
                style: TextStyle(
                  fontWeight: FontWeight.bold,
                  color: unlocked ? Colors.black87 :
Colors.grey[600],
                ),
              ),
              const SizedBox(height: 4),
              Text(
                achievement['description'] as String,
                textAlign: TextAlign.center,
                style: TextStyle(
                  fontSize: 10,
                  color: Colors.grey[600],
                ),
              ),
            ],
          ),
        );
      },
    );
  }
}

// Data Service (to manage data persistence)
class DataService {
  static Future<void> saveUser(User user) async {
    final prefs = await SharedPreferences.getInstance();
    final userJson = user.toJson();
    await prefs.setString('user', userJson.toString());
  }

  static Future<User?> getUser() async {
    final prefs = await SharedPreferences.getInstance();
    final userJson = prefs.getString('user');
    if (userJson != null) {
      return User.fromJson(Map<String,
dynamic>.from(userJson as Map));
    }
    return null;
  }

  static Future<void> saveQuizResult(QuizResult result)
async {
    final user = await getUser();
    if (user != null) {
      user.quizHistory.add(result);
      if (result.type == 'alphabets') {
        user.alphabetsLevel =
_calculateNewLevel(user.alphabetsLevel, result.score);
      } else {
        user.numbersLevel =
_calculateNewLevel(user.numbersLevel, result.score);
      }
      user.totalScore += result.score;
      await saveUser(user);
    }
  }

  static int _calculateNewLevel(int currentLevel, int score)
{
    // Simple level calculation logic
    if (score >= 9) {
      return currentLevel + 1;
    } else if (score >= 7) {
      return currentLevel + (currentLevel % 2 == 0 ? 1 : 0);
    }
    return currentLevel;
  }

  static Future<List<Map<String, dynamic>>>
getLeaderboard(String type) async {
    // In a real app, this would fetch from a database
    // Here we return mock data
    if (type == 'alphabets') {
      return [
        {'name': 'Emma', 'avatar': 'assets/avatar1.png',
'score': 95},
        {'name': 'Noah', 'avatar': 'assets/avatar2.png',
'score': 92},
        {'name': 'Olivia', 'avatar': 'assets/avatar3.png',
'score': 88},
        {'name': 'Liam', 'avatar': 'assets/avatar4.png',
'score': 85},
        {'name': 'Ava', 'avatar': 'assets/avatar5.png',
'score': 82},
        {'name': 'William', 'avatar': 'assets/avatar6.png',
'score': 80},
        {'name': 'Sophia', 'avatar': 'assets/avatar7.png',
'score': 78},
        {'name': 'James', 'avatar': 'assets/avatar8.png',
'score': 75},
        {'name': 'Isabella', 'avatar': 'assets/avatar9.png',
'score': 72},
        {'name': 'Benjamin', 'avatar':
'assets/avatar10.png', 'score': 70},
      ];
    } else {
      return [
        {'name': 'Sophia', 'avatar': 'assets/avatar7.png',
'score': 98},
        {'name': 'Liam', 'avatar': 'assets/avatar4.png',
'score': 94},
        {'name': 'Emma', 'avatar': 'assets/avatar1.png',
'score': 90},
        {'name': 'Noah', 'avatar': 'assets/avatar2.png',
'score': 86},
        {'name': 'Isabella', 'avatar': 'assets/avatar9.png',
'score': 84},
        {'name': 'William', 'avatar': 'assets/avatar6.png',
'score': 81},
        {'name': 'Olivia', 'avatar': 'assets/avatar3.png',
'score': 79},
        {'name': 'Benjamin', 'avatar':
'assets/avatar10.png', 'score': 77},
        {'name': 'Ava', 'avatar': 'assets/avatar5.png',
'score': 74},
        {'name': 'James', 'avatar': 'assets/avatar8.png',
'score': 71},
      ];
    }
  }
```

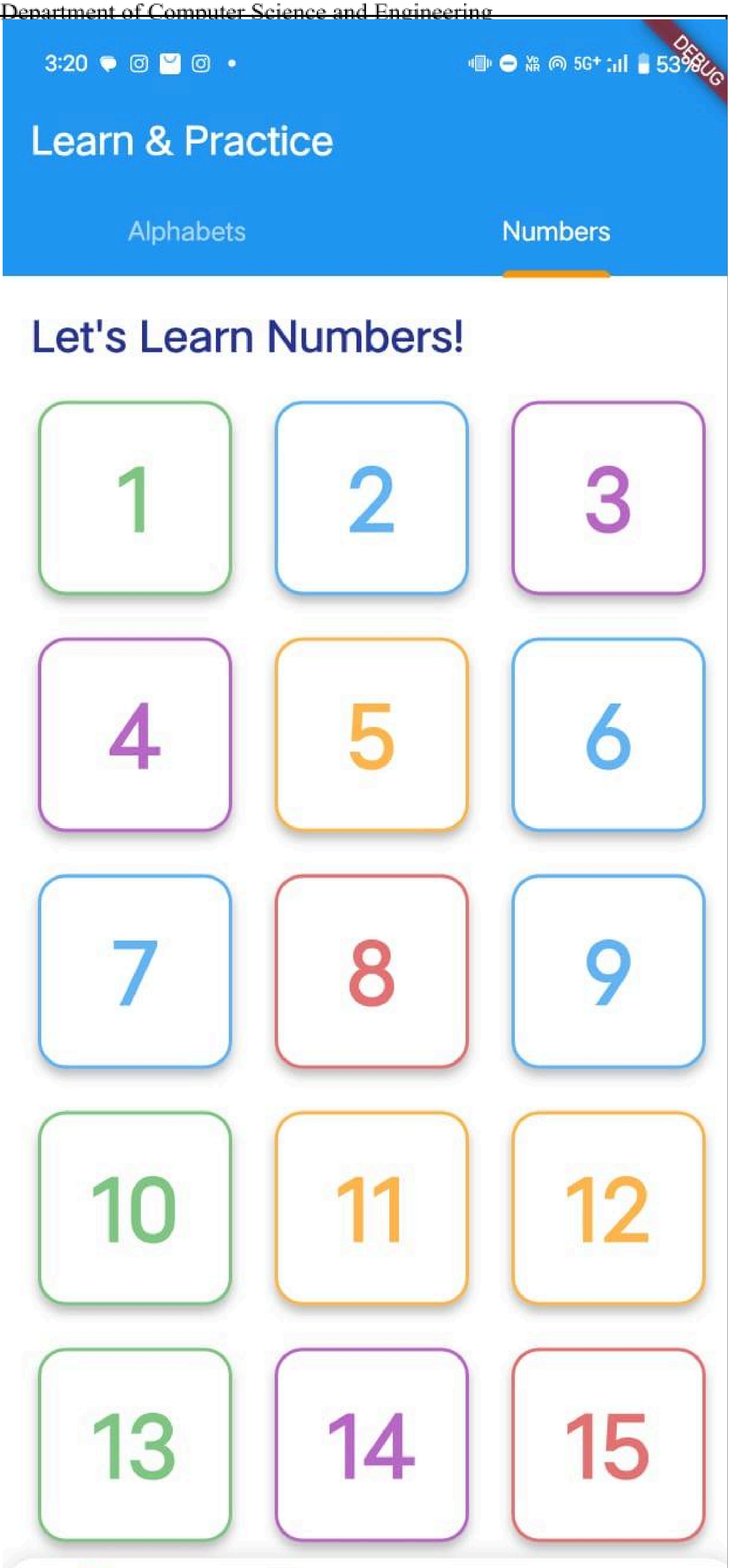
	`}`

Screenshots	



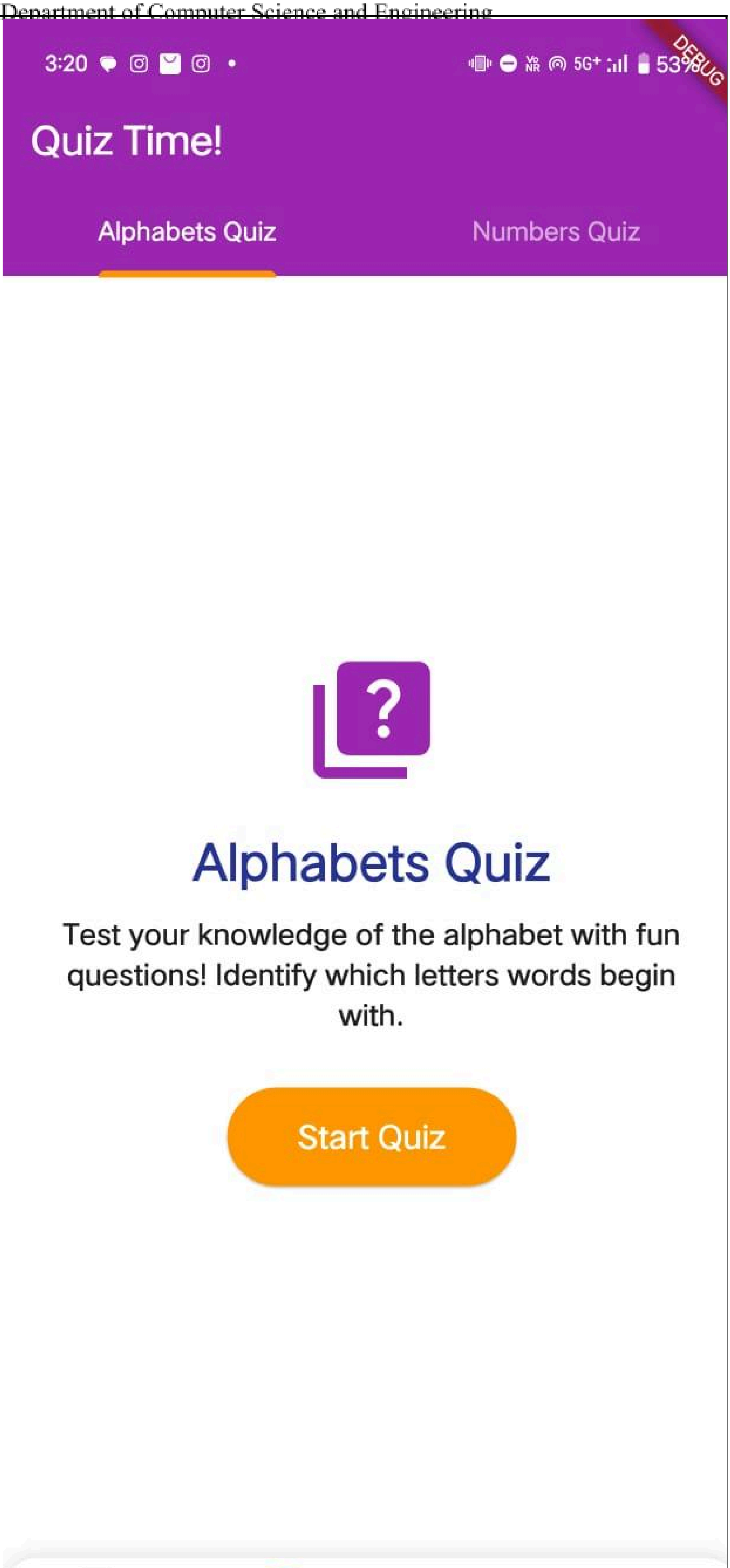
Learn & Practice

Let's Learn Alphabets!



Learn & Practice

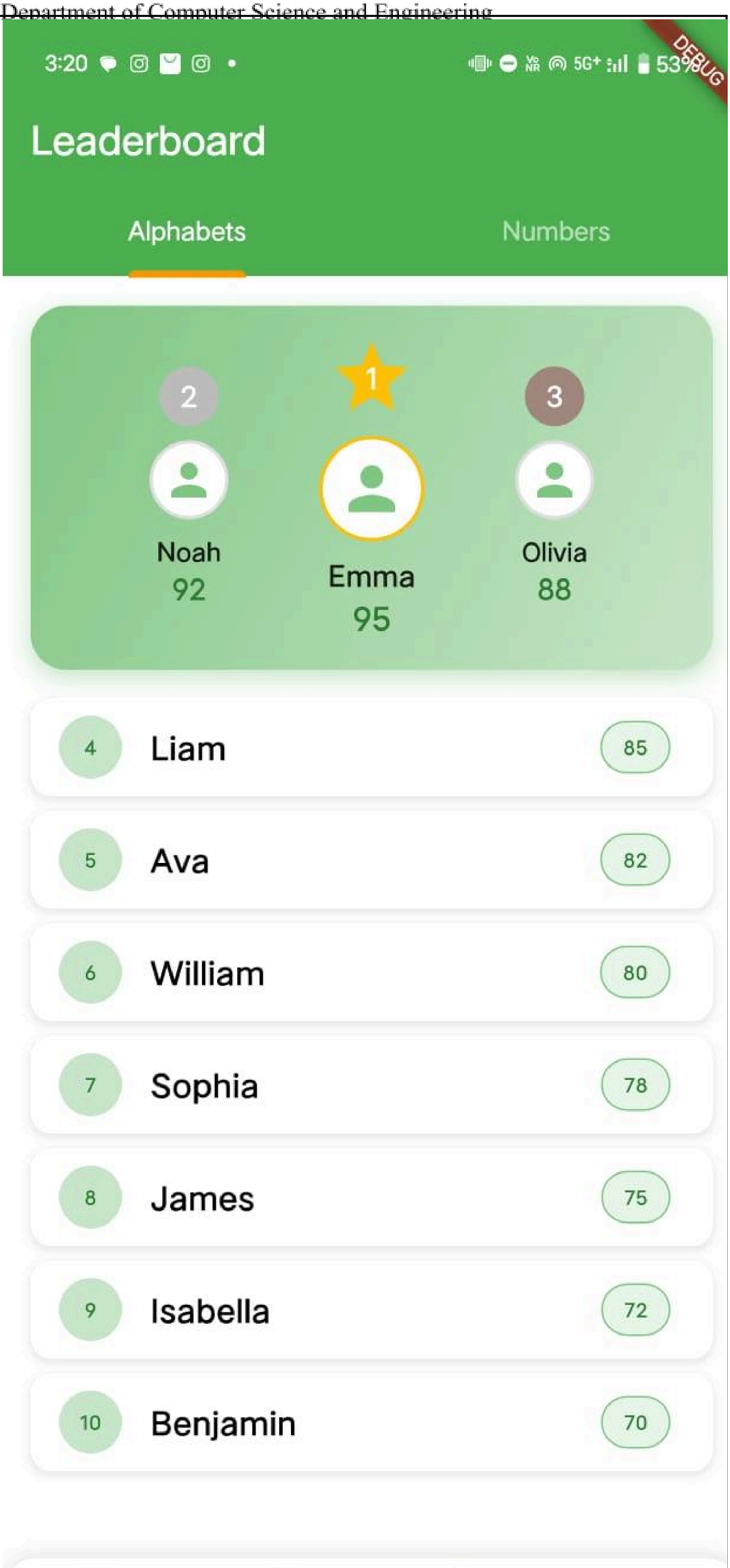
Let's Learn Numbers!



Quiz Time!

Alphabets Quiz

Test your knowledge of the alphabet with fun questions! Identify which letters words begin with.



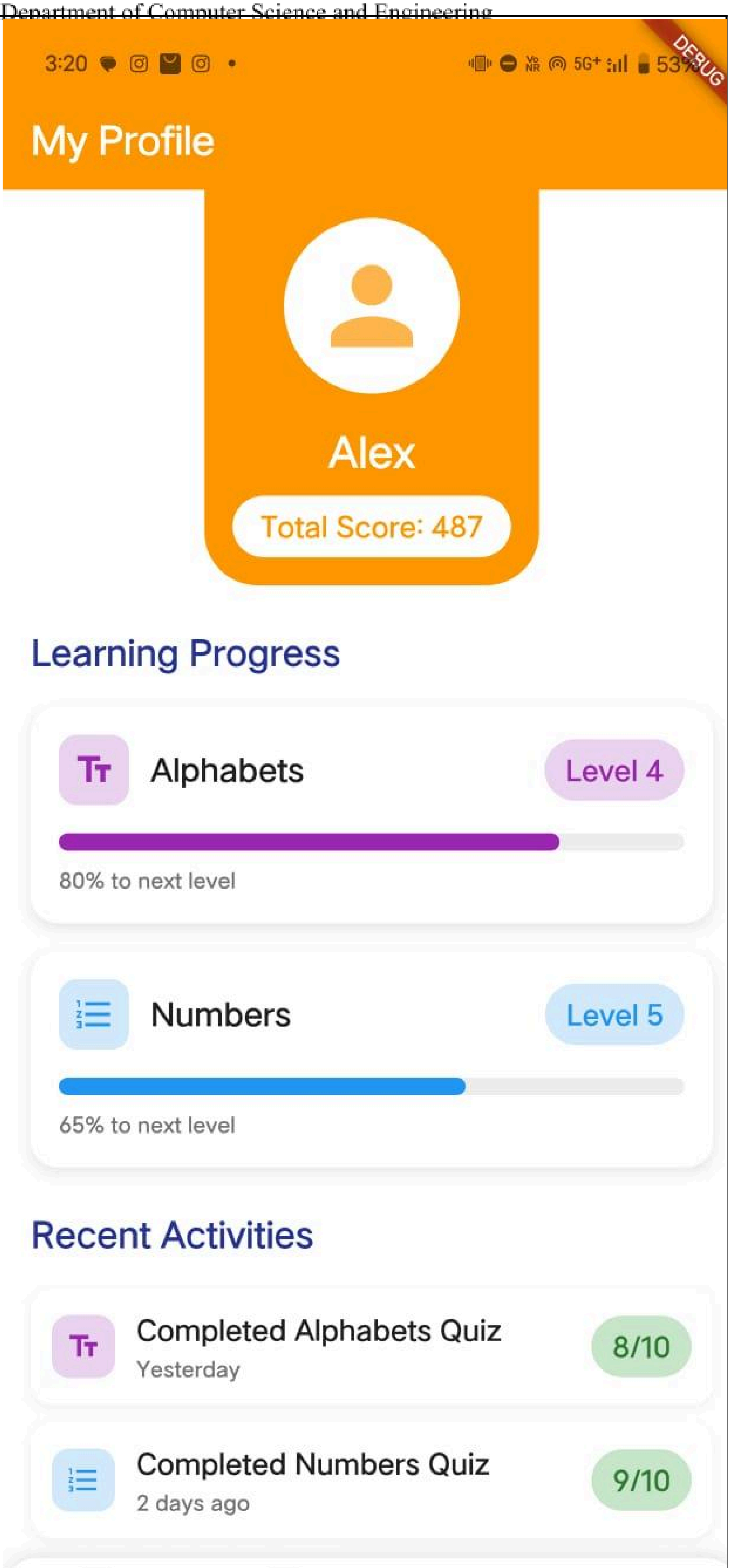
Leaderboard

2 Noah 92

1 Emma 95

3 Olivia 88

4	Liam	85
5	Ava	82
6	William	80
7	Sophia	78
8	James	75
9	Isabella	72
10	Benjamin	70



My Profile

Alex

Total Score: 487

Learning Progress

Alphabets Level 4

80% to next level

Numbers Level 5

65% to next level

Recent Activities

Completed Alphabets Quiz
Yesterday
8/10

Completed Numbers Quiz
2 days ago
9/10

Question and Answers	Answer the following Questions: 1. How to create Upper Tabs in Flutter? In the Kids Learning App, the upper tabs are created by pairing a TabController with a TabBar placed in the AppBar and a corresponding TabBarView in the body of the screen. The TabController is initialized with the number of tabs and requires a vsync provider, which is supplied by mixing in SingleTickerProviderStateMixin. Once the controller is set up, the TabBar displays the tab labels (for example, “Alphabets” and “Numbers”) in the AppBar’s bottom slot, and the TabBarView hosts the associated content screens, allowing the user to switch between sections by tapping on the tabs 2. How did you use 60-30-10 rule in your application? The 60-30-10 color rule is applied through the app’s theme. Approximately 60 percent of the interface uses the primary blue swatch for major surfaces like the AppBar and tab indicators, 30 percent uses a neutral white background on cards and screens, and the remaining 10 percent uses an accent orange for buttons, selected tab underlines, and progress indicators. This distribution creates a visually balanced and harmonious design that guides the user’s attention and maintains consistency throughout the app 3. Which new elements did you use for creating UI components? Several newer Flutter widgets and techniques enhance the UI components. The splash screen leverages AnimationController and ScaleTransition to animate the logo on startup. GridView.builder with a SliverGridDelegate arranges alphabet and number cards in a responsive grid. InkWell wrapped around Card widgets provides tappable feedback with elevation and ripples. The bottom navigation bar is styled with ClipRRect and BoxDecoration to achieve rounded corners and shadows. AlertDialog widgets present detailed information and quiz results, while CircularProgressIndicator and LinearProgressIndicator convey loading states and quiz progress 4. In pubspec.yaml file, what dependencies need to be there? To support data persistence and additional icons, the pubspec.yaml file should include dependencies on flutter (via sdk), shared_preferences for storing user profiles and quiz history, and optionally cupertino_icons for extra icon options. The shared_preferences package enables saving and retrieving user data such as quiz scores and levels, ensuring the app remembers progress between sessions 5. What is the use of Splash Screen? The splash screen serves as the app’s animated entry point, displaying a branded logo that scales into view over two seconds and shows a progress indicator. After a set delay of three seconds, it automatically navigates to the main screen. This approach masks any initial loading time, reinforces the app’s identity, and provides a smooth transition into the learning experience

Conclusion	In this experiment, you have successfully implemented a cohesive children's learning app that combines engaging animations, intuitive navigation, and consistent design principles. By setting up upper tabs with TabController and TabBar, you enabled seamless section switching for lessons, quizzes, leaderboards, and profiles. Applying the 60-30-10 color rule within ThemeData ensured a balanced and visually appealing interface, while the SplashScreen's animated logo and progress indicator created a smooth and branded app entry. Leveraging modern Flutter widgets—such as GridView.builder for content layouts, InkWell-wrapped Cards for interactive elements, and AlertDialogs for user feedback—added both functionality and polish. Including the shared_preferences dependency in pubspec.yaml allowed persistent storage of user progress, tying together the app's dynamic features. Overall, this exercise reinforced best practices in Flutter UI design, stateful widget management, and responsive theming, laying a solid foundation for building rich, user-friendly mobile applications.