Digital Egypt Pioneers Initiative DEPI

AMIT Learning

Habit Tracking Mobile App

Presented by:

Rehab Nader Galal Ali
Hager Ragaee Abdeltawab
Roqaya Abdelsamia Mahmoud Ibrahim
Maram Basyouni Mohamed El-Ganzouri
Asmaa Ali Abdelwahab Ali

Table of Contents

Chapter 1
Introduction

Introduction:

1.1 Overview

- **Habit Formation and Personal Development**: Habits are essential in shaping our daily lives and achieving long-term goals. Good habits contribute to productivity, health, and personal growth, while breaking bad habits can improve well-being. However, forming new habits or eliminating negative ones can be challenging without a system to track progress and stay motivated.

- **Role of Technology in Habit Tracking**: Mobile apps have become valuable tools in habit formation, providing accessible and interactive ways for users to set and manage goals. With personalized notifications, analytics, and reminders, habit tracking apps make it easier for users to stick to their goals and form lasting routines.

- **Overview of the HabitFlow Project**: The HabitFlow App was developed to assist users in establishing and maintaining habits. It simplifies the habit formation process by providing goal setting, reminders, and progress visualization. Through user-friendly UI and customizable settings, this app caters to individuals who want to enhance their personal and professional lives by building consistent routines.

1.2 Objectives

- **Enable Users to Set and Track Habits**: The app's primary objective is to assist users in setting, tracking, and achieving habit goals across various areas of life, whether fitness, productivity, or personal growth.

- **Provide Visual Progress Tracking**: The app includes visual progress tools like charts, streak counters, and progress bars to help users stay motivated by showing their habit journey over time.

- **Offer Motivational Reminders and Notifications**: The app encourages consistency by providing customizable reminders and notifications that help users perform tasks regularly, reinforcing positive habits.

- **Deliver Behavioral Analytics and Insights**: By collecting data on user habits, the app provides insights into consistency patterns and milestones, helping users adjust goals and stay on track.

- **Support Customization and Flexibility**: The app aims to adapt to individual needs by allowing users to customize their goals, reminders, and habit types, making the app versatile for a range of personal and professional habit-building efforts.

1.3 Purpose

- **Promote Positive Behavior Change**: The HabitFlow's primary purpose is to foster positive behavior changes, helping users establish beneficial routines and overcome barriers to habit formation.

- **Enhance Productivity and Well-being**: By facilitating habit tracking, the app enables users to build routines that can boost productivity, reduce stress, and improve overall mental and physical health.

- **Serve as a Self-Improvement Tool**: Designed to aid personal growth, the app acts as a supportive tool for users who are committed to achieving both short- and long-term goals.

- **Build Accountability and Consistency**: With features like reminders, notifications, and progress tracking, the app serves as a reliable platform for accountability, helping users maintain consistency in their habits.

1.4 Scope

➢ **Feature Set and Functionalities**:

- **Habit Creation and Customization**: Users can create habits based on their unique goals, choosing from a wide range of customizable habit types.

- **Goal Setting and Scheduling**: The app allows users to define daily, weekly, or monthly habits and track them according to personal schedules.

- **Progress Visualization**: Through visual tools like graphs, streaks, and completion counters, the app provides a clear picture of user progress over time.

- **Reminders and Notifications**: Users can set up personalized reminders and notifications to encourage regular action on their goals.

- **Insight and Analytics Dashboard**: The app includes an analytics dashboard where users can view trends, milestones, and statistics on habit consistency.

- **Target Audience**: The app is designed for individuals aiming to improve their productivity, wellness, and personal habits. It is suitable for a wide range of users, including students, professionals, and anyone looking to improve daily routines.

- **Platform Compatibility**: The Habit App is compatible with both iOS and Android platforms, making it accessible to a large audience.

1.5 General Constraints

➢ Technical Constraints:

- **Platform-Specific Limitations:** Some features may vary depending on the platform (iOS or Android), potentially impacting user experience across different devices.

- **Data Storage and Privacy:** The app is designed to ensure user privacy, with considerations for data storage and encryption methods to secure personal information.

- **Backend and Server Constraints**: The app relies on a backend infrastructure that must be scalable to accommodate increased data as more users join the platform.

➢ Resource Constraints:

- **Budget Limitations:** Financial constraints may affect the speed of development, availability of certain features, and the frequency of updates.

- **Development Team and Expertise:** The app's design and functionality may be limited by the size and skills of the development team, impacting the inclusion of complex features.

➢ Time Constraints:

- **Development Timeline:** The app is developed according to a timeline with milestones, such as initial release, user testing, and regular updates.

- **User Feedback and Improvement Cycle**: The app will incorporate user feedback in future versions; however, updates may be limited by time and resource constraints.

Chapter 2

System Analysis

System Analysis:

- **Introduction to System Analysis:** System analysis plays a crucial role in understanding the requirements necessary for the HabitFlow App to function effectively. By identifying both functional and non-functional requirements, this section outlines what the app must accomplish and how it should perform to meet user needs.

2.1 Functional Requirements

- **User Registration and Authentication:**
  - **Account Creation:** Users can create an account using email and password.
  - **Login/Logout:** Users are able to log in and log out securely, with authentication methods ensuring data protection.
  - **Password Reset:** Users can reset their password securely in case of loss.
- **Habit Management:**
  - **Habit Creation:** Users can create new habits by specifying the habit name, category, and frequency (e.g., daily, weekly, monthly).
  - **Habit Customization:** Each habit can be personalized with settings such as target days, time of day, and custom tags.
  - **Habit Deletion and Modification:** Users have the option to edit or delete existing habits based on evolving goals.
- **Goal Setting and Scheduling:**
  - **Goal Definition:** Users set measurable goals for each habit, defining a target outcome over time (e.g., exercise 3 times a week).
  - **Scheduling:** Users can set schedules for each habit, including specific days of the week or intervals (daily, weekly, monthly).
- **Progress Tracking and Visualization:**
  - **Streaks and Milestones:** The app tracks consecutive days a habit is completed and shows streaks, offering visual milestones.

  - **Progress Graphs:** Graphical visualizations (e.g., bar charts, progress bars) help users monitor their performance over time.
  - **Completion History:** Users can view a history log showing dates of habit completion, offering a comprehensive look at their consistency.

- **Reminders and Notifications:**
  - **Customizable Reminders:** Users can set reminders for each habit to prompt them to complete it on time.
  - **Push Notifications:** Timely push notifications ensure users are reminded to complete their habits as scheduled.

- **Insights and Analytics:**
  - **Behavioral Insights:** Users gain access to insights about habit performance, including consistency rates and notable trends.
  - **Achievement Tracking:** The app tracks achievement of goals and provides feedback, motivating users to stay consistent.

- **User Profile and Custom Settings:**
  - **Profile Management:** Users can view and edit their profile information, update preferences, and manage notification settings.
  - **Customization:** The app allows users to personalize settings, including theme, privacy options, and habit visibility.

2.2 Non-Functional Requirements

- **Performance Requirements:**
  - **Responsiveness:** The HabitFlow App is designed with efficient, responsive UI elements that load quickly and offer smooth transitions. This responsiveness ensures users can navigate between screens and update habits without delays.
  - **Data Syncing (if applicable):** If the app integrates cloud storage or syncing features, it ensures timely updates across devices, allowing users to access their habit data on multiple platforms seamlessly.

- **Usability Requirements:**

- **User-Friendly Interface:** The app includes a user-centered design with intuitive navigation, making it easy for users to create, edit, and track habits without extensive instructions.
- **Consistency in UI:** The design follows a uniform style and interaction flow across all screens, enhancing usability and reducing user confusion.
- **Accessibility:** Basic accessibility considerations, such as clear icons and labels, are implemented to make the app easy to use for people with various needs.

• Reliability Requirements:

- **Error Handling:** Basic error handling has been implemented to display informative messages in case of issues, helping users understand and correct minor errors.
- **Offline Access (if applicable):** If the app supports offline functionality, users can update their habits without an internet connection, with data syncing once the device reconnects.

• Security Requirements:

- **Authentication and Privacy:** The app includes user authentication to ensure account protection, requiring users to securely log in to access their data.
- **Data Privacy Measures:** Basic data privacy protocols, such as protecting user information and limiting access, are implemented to maintain user confidentiality and data security.

• Compatibility Requirements:

- **Platform Compatibility:** The app is compatible with both iOS and Android platforms, making it accessible to a wider audience regardless of device type.
- **Responsive Design:** The UI is designed to perform well across different screen sizes, adapting to both mobile phones and tablets for a consistent experience.

• Maintainability Requirements:

- **Modular Code Structure:** The app uses a modular code base, facilitating easy updates, maintenance, and the potential for new feature additions in the future.
- **Documentation:** Basic documentation of the app's code and functionalities is available, supporting future troubleshooting and modifications by other developers.

Chapter 3
Architecture

Technical Architecture

Flutter Application

- **Structure**:
  - **Presentation Layer**: UI built using Flutter widgets
  - **Data Layer**: interacting with Firebase.

Firebase Services

- **Authentication**:
  - User sign-up, login, and logout functionality.
- **FireStore**:
  - Storage for user habits, progress tracking.

Interaction with Firebase

Initialization

The app initializes Firebase during startup

Authentication

- **Sign-Up**: Users can create an account using email and password.
- **Login**: Users can log in.

Firestore Data Operations

- **Adding a Habit**
- **Retrieve a Habit**

Database Schema (Firestore Structure)

Collections

- **Users Collection**:

  Fields:

  - `email`: String
  - `name`: String
  - `phone`: String
- **Complete Tasks Collection**:
  - Fields:

        - `name`: String
        - `data`: time stamp
- **UnComplete Tasks Collection**:
    - Fields:
        - `name`: String
        - `data`: time stamp
- **Daily Collection**:
    - Fields:
        - `Task name`: String
        - `Color:` Int
        - `completed`: Boolean
        - `Alarm`: Booelan
        - `repeat_days`: Array of int
        - `reminder_time`: map
- **Monthly Collection**:
    - Fields:
        - `Task name`: String
        - `Color:` Int
        - `completed`: Boolean
        - `description`: String
        - `repeat_days`: Array of int
        - `user_id`: String

Chapter 4
Code Explanation

Code Explanation

This section provides an overview of the core functionalities of the "Habit Flow" app, explaining how the key features are implemented and providing relevant code snippets for better understanding.

4.1 Login

Description: The Login page allows existing users to securely sign in to their accounts using their registered email and password.

implementation:

·**signInWithEmailAndPassword** This function is part of Firebase's FirebaseAuth service. It attempts to sign in using the email and password provided by emailController.text and passwordController.text.

- ·If the credentials are valid, the signInWithEmailAndPassword function returns a UserCredential object, which contains user information and authentication details.

```
onPressed: () async {
                        try {
                          final credential = await FirebaseAuth.instance
                              .signInWithEmailAndPassword(
                            email: emailController.text,
                            password: passwordController.text,
                          );
```

·If the sign-in is successful, the app navigates to the "Home" screen. This is done by calling Navigator.of(context).popAndPushNamed("Home"), which clears the current screen and pushes the "Home" screen to the top of the navigation stack.

```
Navigator.of(context).popAndPushNamed("Home");
```

·Error Handling

1) **user-not-found**: Triggered if no user is found with the entered email. A message, No user found for that email, is printed to the console.

2) **wrong-password**: Triggered if the provided password is incorrect. The message Wrong password provided is printed.

3) **invalid-credential**: This error indicates that the authentication credentials are malformed or expired. The message The supplied auth credential is malformed or has expired is printed.

4) **General Error**: Any other errors print Error: ${e.message}, allowing visibility into unexpected issues.

```
on FirebaseAuthException catch (e) {
                        if (e.code == 'user-not-found') {
                          print('No user found for that email.');
                        } else if (e.code == 'wrong-password') {
                          print('Wrong password provided.');
                        } else if (e.code == 'invalid-credential') {
                          print(
                              'The supplied auth credential is malformed or has expired.');
```

4.2 Registration

Description : The _registerUser function handles the Firebase registration and stores user data in Firestore.

implementation:

·Store user data in Firestore

```
  await FirebaseFirestore.instance
      .collection('users')
      .doc(userCredential.user?.uid)
      .set({
    'name': nameController.text.trim(),
    'email': emailController.text.trim(),
    'phone': phoneController.text.trim(),
  });
```

·Store user data locally

```
SharedPreferences prefs = await SharedPreferences.getInstance();
await prefs.setString('userId', userCredential.user?.uid ?? '');
await prefs.setString('name', nameController.text.trim());
await prefs.setString('email', emailController.text.trim());
await prefs.setString('phone', phoneController.text.trim());
```

·error handling like weak passwords or duplicate emails are managed using Firebase's error codes and shown to the user with SnackBar.

```
on FirebaseAuthException catch (e) {
  String message;
  if (e.code == 'weak-password') {
    message = 'The password provided is too weak.';
  } else if (e.code == 'email-already-in-use') {
    message = 'The account already exists for that email.';
  } else {
    message = 'Registration failed. Please try again.';
  }
  ScaffoldMessenger.of(context).showSnackBar(SnackBar(content: Text(message)));
}
```

4.3 Habit Creation and updating

Description: The **CreateHabitScreen** class is a stateful widget that allows users to create or edit a habit. The screen includes input fields for the habit name and description, a color picker for selecting the habit color, and options for setting a repeat schedule (daily or monthly). Users can also set reminders for their habits

Implementation:

· Users can enter the habit name and description through text fields created by the **TextfieldStructrue** widget.

· A color picker allows users to select a color for the habit.

```
TextfieldStructrue(
   controller: nameController,
   hintText: "Habit Name",
   labelText: "Habit Name",
),

ColorPicker(
   color: screenPickerColor,
   onColorChanged: (Color color) => setState(() => screenPickerColor = color),
),
```

· Users can choose between a daily or monthly schedule using buttons that navigate between different views (DailyView and MonthlyView).

```
children: [
   DailyView(onDaysSelected: _updateSelectedDays),
   MonthlyView(onDatesSelected: _updateSelectedDates),
],
```

·When the user presses the save button, the app checks if the required fields are filled and either updates an existing habit or creates a new one using **HabitService**.

```
} else {
   HabitService.saveHabitToFirestore(
      habitName: nameController.text,
      habitDescription: disController.text,
      isDaily: currentPage == 0,
      isMonthly: currentPage == 1,
      color: screenPickerColor,
      selectedDates: _selectedDates,
      selectedDays: _selectedDays,
      reminderEnabled: reminderEnabled,
      reminderTime: reminderTime,
   );
}
```

```
ElevatedButton(
   onPressed: () {
      if (nameController.text.isNotEmpty) {
         if (widget.habitId != null) {
            HabitService.updateHabitInFirestore(
               habitId: widget.habitId!,
               habitName: nameController.text,
               habitDescription: disController.text,
               isDaily: currentPage == 0,
               isMonthly: currentPage == 1,
               color: screenPickerColor,
               selectedDates: _selectedDates,
               selectedDays: _selectedDays,
               reminderEnabled: reminderEnabled,
               reminderTime: reminderTime,
            );
```

4.4 Progress tracking

Description: The LineChartPage class displays a line chart using the fl_chart package. It visualizes data related to user habits over a selected timeframe (like "This Week," "This Month," or "This Year"). The data is retrieved from a Firestore collection named completeTasks, filtered by user ID and habit name.

implementation:

· The constructor requires selectedTimeframe and selectedHabit parameters to determine which data to fetch and display.

·A FutureBuilder is used to asynchronously fetch the chart data. It shows a loading spinner while fetching data, and handles errors or empty

```
FutureBuilder<List<FlSpot>>(
  future: _getChartData(selectedTimeframe, selectedHabit),
  builder: (context, snapshot) {
    if (snapshot.connectionState == ConnectionState.waiting) {
      return Center(child: CircularProgressIndicator());
    }
    if (snapshot.hasError) {
      return Center(child: Text("Error: ${snapshot.error}"));
    }
    if (!snapshot.hasData || snapshot.data!.isEmpty) {
      return Center(
        child: Text("No data available for the selected habit/timeframe"),
      );
    }
```

·The methods _getSpecificHabitData and _getAllHabitData query Firestore for completed tasks within the specified timeframe and construct a list of counts for each day (or week/month) in that range.

```
Future<List<double>> _getSpecificHabitData(String timeframe, String habit) async {
  // ... Date range calculation ...

  final querySnapshot = await FirebaseFirestore.instance
      .collection('completeTasks')
      .where('user_id', isEqualTo: userId)
      .where('task_name', isEqualTo: habit.toLowerCase())
      .where('completed_at', isGreaterThanOrEqualTo: startDate)
      .where('completed_at', isLessThanOrEqualTo: endDate)
      .orderBy('completed_at')
      .get();
```

·The chart is configured with titles and appearance using the FlChart package. The _getBottomTitles and _getLeftTitles methods customize the labels on the axes based on the selected timeframe.

4.5 Notification

Description: make reminders for uncompleted tasks (alarm/push Notification)

implementation :

·Define a high-priority channel

```
const AndroidNotificationChannel channel = AndroidNotificationChannel(
  'high_importance_channel', // id
  'High Importance Notifications', // name
  description: 'This channel is used for important notifications.',
  importance: Importance.high,
);
```

·showNotification :Initializes notifications with timezone data, allowing for accurate scheduling across different locales.

```
class NotificationHandler {
  static final FlutterLocalNotificationsPlugin _notification = FlutterLocalNotificationsPlug

  static Future<void> init() async {
    _notification.initialize(const InitializationSettings(
      android: AndroidInitializationSettings('@mipmap/ic_launcher'),
      iOS: DarwinInitializationSettings(),
    ));

    // Initialize timezone data
    tz.initializeTimeZones();
  }
}
```

·schedule periodic notifications

```
void schedulePeriodicNotifications() {
  Timer.periodic(const Duration(hours:3), (timer) {
    NotificationServiceR.checkAndSendUncompletedTaskNotifications();
  });
}
```

·schedule alarm notifications

```
// Schedule the notification based on the time provided
final now = tz.TZDateTime.now(tz.local);
tz.TZDateTime scheduledDateTime = tz.TZDateTime(
  tz.local,
  now.year,
  now.month,
  now.day,
  timeOfDay.hour,
  timeOfDay.minute,
);

// If the scheduled time is in the past, schedule it for the next day
if (scheduledDateTime.isBefore(now)) {
  scheduledDateTime = scheduledDateTime.add(const Duration(days: 1));
}

// Schedule the notification
await _notification.zonedSchedule(
  id, title, body, scheduledDateTime, notificationDetails,
  uiLocalNotificationDateInterpretation:
      UILocalNotificationDateInterpretation.absoluteTime,
  androidScheduleMode: AndroidScheduleMode.exactAllowWhileIdle,
);
```

Chapter 5
Testing

5.1 Testing

The Habit Tracking App has been thoroughly tested across all features and functionalities. All test cases passed successfully, and the app meets all specified requirements, including habit creation, reminders, progress tracking, and cross-platform compatibility. The app is ready for deployment.

- Testing Results:

1. Login Testing:

Functionality: Users can log in with valid credentials, and the session is maintained as expected.

Error Handling: When invalid credentials are entered, the error message "Invalid username or password" appears accurately.

Edge Cases: Special characters in the input fields are handled appropriately, and simultaneous logins on multiple devices are prevented.

Recommendations: No major issues, but consider enhancing the error message for clarity (e.g., add more guidance if the user forgets their password)..

2.Registration Testing:

- **Functionality**: Users can register with valid inputs, and all required fields are validated correctly.
- **Error Handling**: Duplicate emails are correctly flagged, and a "Account already exists" message is displayed.
- **Edge Cases**: Special characters in names and email fields were handled without breaking the form, and the app only accepts valid email formats.
- **Recommendations**: Strengthen the password requirements to ensure better security and provide feedback for weak passwords..

3.Forgot Password Testing:

- **Functionality**: Users can request a password reset and successfully use the reset link to set a new password.
- **Error Handling**: An error message appears correctly if the email is not registered, and empty submissions are blocked.

- **Edge Cases**: Expired reset links prompt the user to request a new one, ensuring smooth handling of expired requests.
- **Recommendations**: Ensure reset links expire within a reasonable timeframe for security , and add feedback if multiple reset requests are submitted.

4.Functional Testing:

- Habit Management: Users can create, update, and delete habits without any issues. Progress tracking works correctly, displaying accurate habit streaks and progress charts.
- UI and Navigation: The app's interface is responsive and user-friendly on all tested devices.

5.Notification System:

- Push notifications were successfully delivered on all devices, following the habit schedule set by the user. Firebase Cloud Messaging worked flawlessly, ensuring timely reminders.

6.Compatibility Testing:

- The app was tested on various devices across Android and iOS platforms and worked seamlessly without any compatibility issues.
- It performed well on different network types (3G, 4G, Wi-Fi), ensuring consistent notifications and data synchronization.

7.Performance Testing:

- The app demonstrated efficient resource usage and smooth transitions, with minimal impact on battery consumption.

8.Security Testing:

- User data is securely stored, meeting safety requirements with no security vulnerabilities detected.

9.User Feedback:

- End-user testing showed positive feedback on the user interface and ease of navigation. No usability issues were reported.

Chapter 6
Conclusion & Future work

6.1 Conclusion

The HabitFlow App was developed as a user-friendly tool for individuals aiming to build and maintain positive habits. With an intuitive design and essential habit-tracking features, the app supports users in setting achievable goals, scheduling reminders, and monitoring progress over time. By providing visual feedback on streaks and offering customizable reminders, the app empowers users to stay consistent and motivated in their personal growth journey.

Throughout the project, significant emphasis was placed on creating a reliable, accessible, and responsive application that can meet a wide range of user needs. However, the app's potential for growth remains vast. Future work could include integrating social features, enhanced analytics, and additional accessibility improvements to create a more inclusive and engaging user experience.

In conclusion, the Habit App serves as a strong foundation for promoting productivity and self-improvement. By focusing on both functionality and user experience, this project not only helps users build positive habits but also has the potential to evolve into a comprehensive personal development platform.

6.2 Future work

- **Advanced Data Analytics and Insights:**
    - Introduce detailed insights, such as habit success rates, weekly/monthly summaries, and behavioral trends over time, to provide users with a deeper understanding of their progress and consistency.
- **Social Features and Habit Sharing:**
    - Implement social sharing features, allowing users to share their habit progress with friends or on social media platforms. Adding a community aspect, where users can join groups, compete, or encourage each other, could further enhance motivation.
- **Gamification Elements:**
    - Introduce gamification features, such as earning badges, rewards, or levels for habit completion streaks. This could make habit tracking more engaging and motivating for users.
- **Customizable Reminder Notifications:**

    - Offer more customization options for notifications, allowing users to set different reminder frequencies or choose from various notification sounds, adding personalization to the experience.

- **Multiple Language Support:**
    - Expand language support to reach a global audience by including more language options, along with culturally relevant formats for dates, times, and metrics.
- **Enhanced Security Features:**
    - Add two-factor authentication (2FA) for an added layer of account security, ensuring that user data remains protected and private.
- **Offline Mode with Automatic Syncing:**
    - Develop a fully functional offline mode, allowing users to track and update their habits without an internet connection. Data would sync automatically when the device reconnects to the internet.
- **Integration with Wearable Devices:**
    - Consider integrating the app with popular wearable devices like Apple Watch, Fitbit, or other fitness trackers, enabling users to track habits and receive notifications directly from their wearable devices.
- **AI-Based Habit Suggestions:**
    - Leverage AI to suggest new habits based on user preferences, past habits, and trends, helping users expand their goals and stay engaged with their personal development journey.
- **Personalized Habit Recommendations and Tips:**
    - Include habit recommendations and practical tips for habit formation, creating a more personalized experience based on the user's habit history and progress.
- **Improved Accessibility Features:**
    - Add enhanced accessibility features, such as voice commands or a high-contrast mode, to make the app more inclusive for users with disabilities.