URL SHORTNER

CEP MAD

Ms. Mariam Memon

Submitted by:
Sunjay Kumar (22SW050)
Tufail Ahmed (22SW071)

Date: 26/10/2025

1. Real-World Problem Identification

In today's digital world, long URLs are inconvenient to share, remember, or display, especially on mobile platforms where screen space is limited. Long links often look unprofessional and can break when shared through messages or social media.

Existing URL shorteners (like Bitly or TinyURL) are mostly **web-based and paid** services that do not provide mobile app integration or self-hosted control over data. Users, developers, and businesses need a **simple, free, and mobile-friendly solution** that lets them shorten URLs quickly, track their links, and manage data securely.

2. Proposed Solution

To solve this problem, we developed a **cross-platform URL Shortener App** using **Flutter (frontend)** and **Node.js + MongoDB (backend)**.

The app allows users to log in, shorten any long URL, and view their previously created short links. The backend handles authentication (via JWT), URL storage, and redirection, while the Flutter app focuses on a responsive user interface and secure token handling.

Key Features

- **User Authentication (JWT):** Secure login system using JSON Web Tokens.
- **Instant URL Shortening:** Converts long URLs into unique short links using the **shortId** package.
- **MongoDB Cloud Storage:** Stores URLs and user accounts securely in MongoDB Atlas.
- **Responsive Design:** Optimized layout for both mobile and tablet screens.
- **Persistent Login:** JWT token saved using Shared Preferences for auto-login.
- **Formatted Timestamps:** Uses intl package to show human-readable creation dates.

3. Responsive User Interfaces

The app's UI was built with Flutter's adaptive widgets and Material Design principles to ensure responsiveness across different screen sizes.

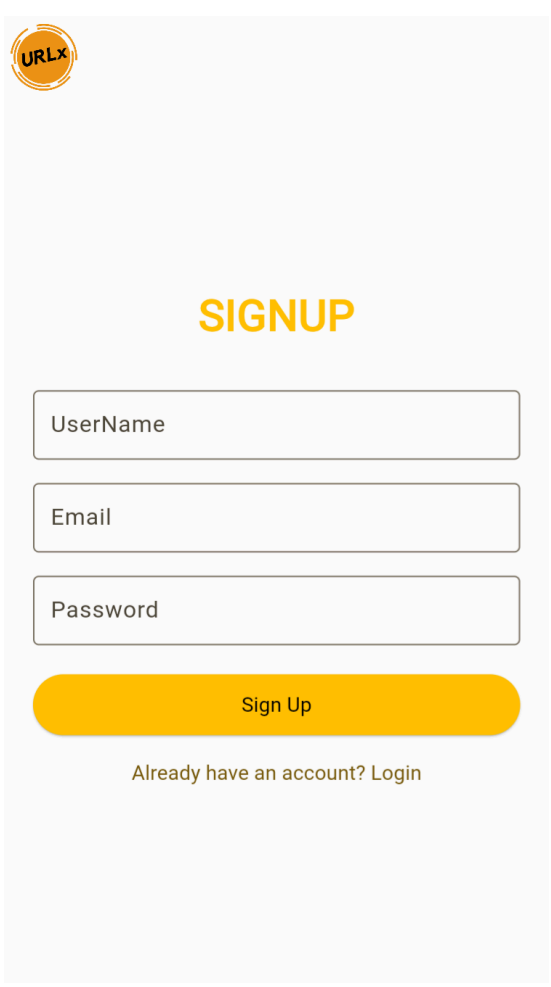
Main Screens:

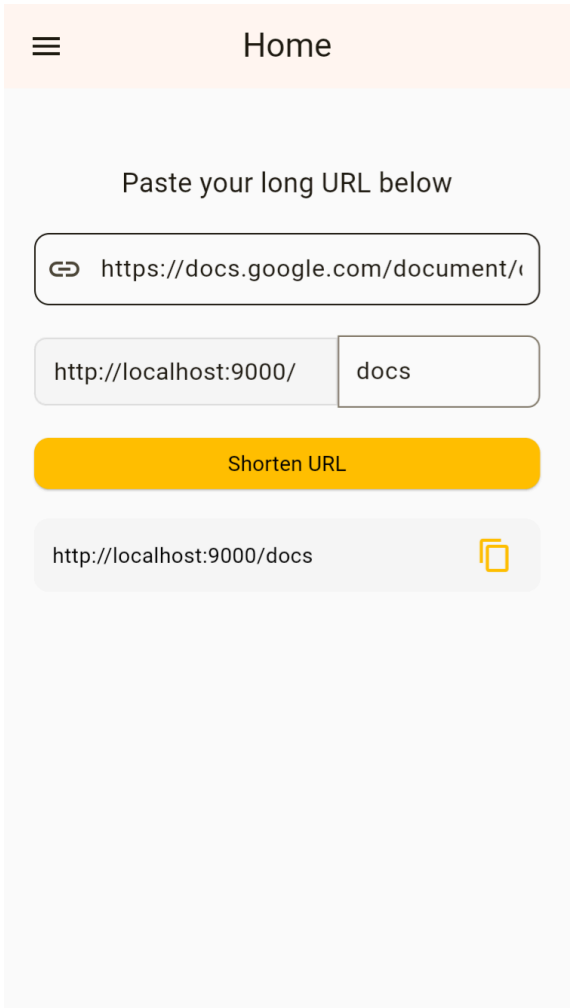
1. **Login / Register Screen:** Allows user authentication via email and password.
2. **Home Screen:** Input field for long URL, “Shorten” button, and list of user’s shortened links.
3. **History Section:** Displays all shortened links fetched from the backend with timestamps and copy options.

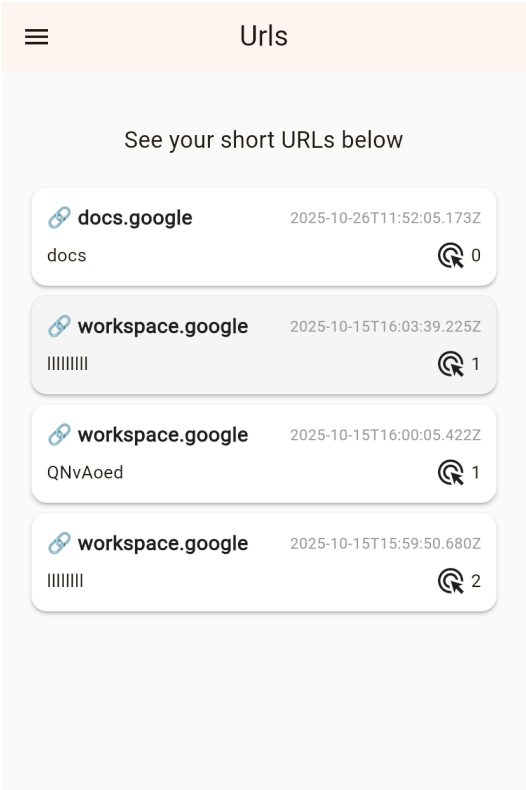
UI Highlights:

- Used MediaQuery and Flexible for dynamic sizing.
- Used ListView Builder for scrollable layouts.
- Consistent color theme using Flutter’s Material design system.

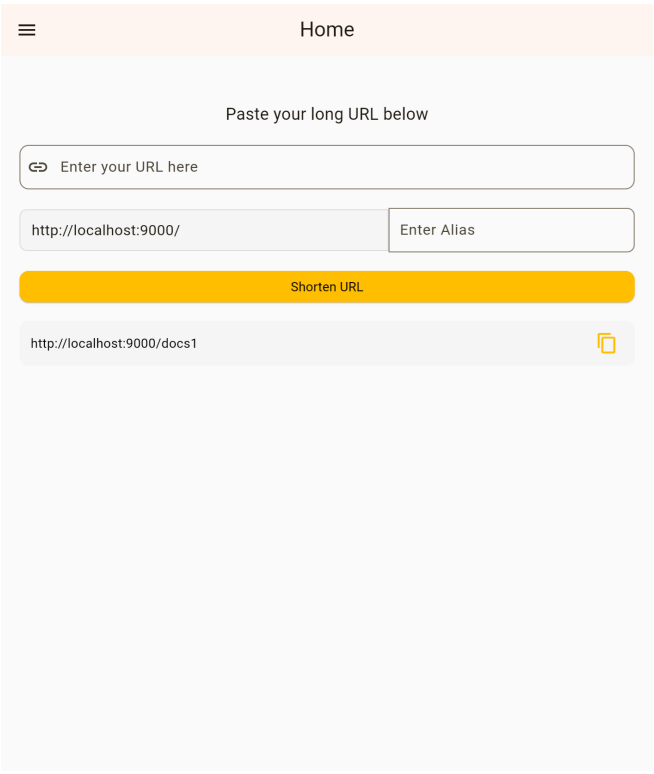
1. Mobile:

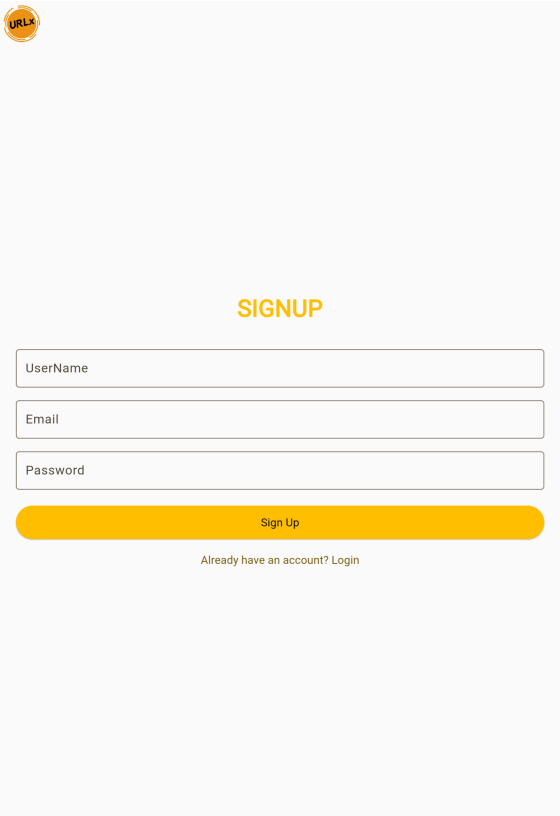


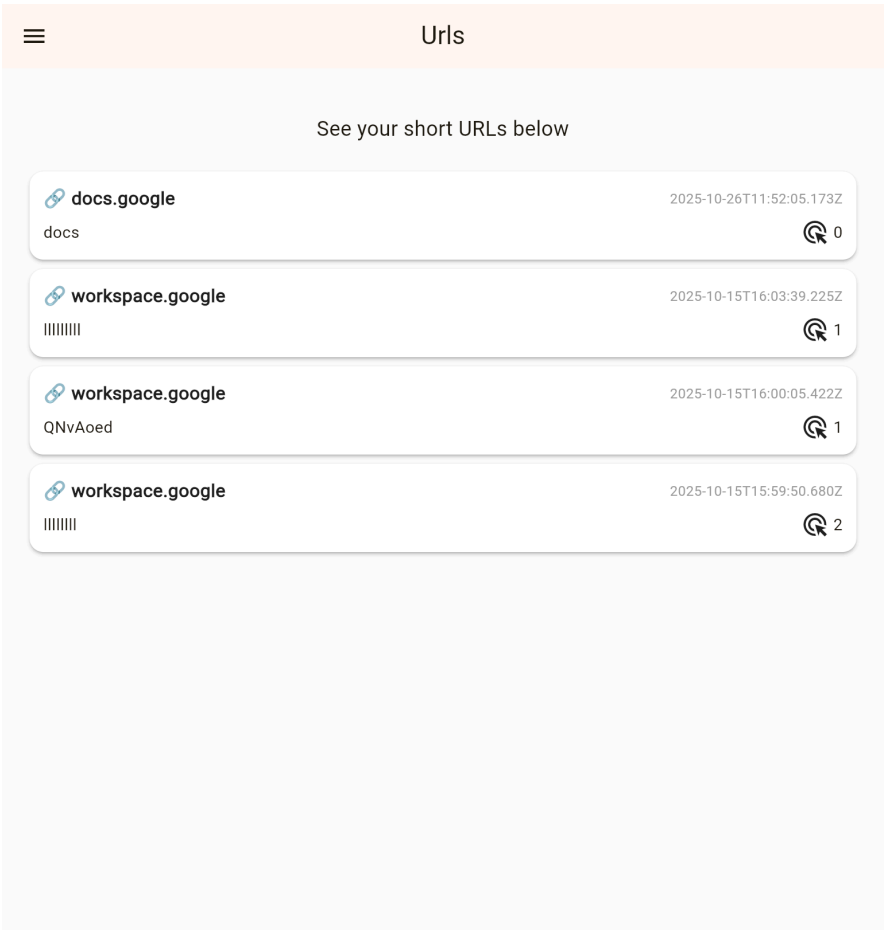



Urls
See your short URLs below
docs.google
2025-10-26T11:52:05.173Z
docs
0
workspace.google
2025-10-15T16:03:39.225Z
lllllllll
1
workspace.google
2025-10-15T16:00:05.422Z
QNvAoed
1
workspace.google
2025-10-15T15:59:50.680Z
lllllllll
2

Tablet:


Home
Paste your long URL below
Enter your URL here
http://localhost:9000/
Enter Alias
Shorten URL
http://localhost:9000/docs1


SIGNUP
UserName
Email
Password
Sign Up
Already have an account? Login



4. Data Storage

Local Storage (Client-Side):

The app uses **Shared Preferences** to store the user's **JWT token** after successful login. This allows automatic login on app restart and smooth API authorization.

Justification:

> Shared Preferences is usually an ideal choice for storing small, secure key-value data like JWTs. It offers persistence without needing external dependencies.

Cloud Storage (Server-Side)

The backend uses **MongoDB Atlas** to store user data and URL mappings.

Collections:

- **Users:** Stores user credentials (email, password hash).
- **URLs:** Stores original and shortened URLs along with their owner's ID.

Example Schema:

User Schema

```
User: {
username: String,
email: String,
password: String
}
```

URL Schema

```
URL: {
  originalUrl: String,
  shortUrl: String,
  userId: ObjectId,
  createdAt: Date,
  visitHistory: Array[]
}
```

5. APIs / Packages / Plug-ins Used

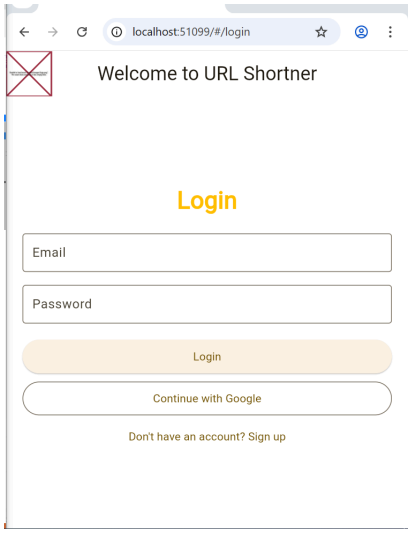
Package / API	Purpose	Justification
http (Flutter)	For sending requests to backend	Connects frontend to Express API
shared_preferences (Flutter)	Store JWT locally	Keeps user logged in between sessions
intl (Flutter)	Format timestamps	Shows human-readable date/time
express (Node.js)	Backend framework	Handles API routes and middleware
mongoose (Node.js)	MongoDB ODM	Simplifies schema and database interaction
shortid (Node.js)	Generate unique short codes	Lightweight, unique ID generator for URLs
jsonwebtoken (Node.js)	Generate and verify JWTs	Ensures secure user authentication
bcryptjs (Node.js)	Hash passwords	Adds security to user credentials
cors (Node.js)	Allow frontend requests	Enables secure API access from Flutter app

References

- Flutter Official Documentation
- Node.js and Express.js Docs
- MongoDB Atlas Documentation
- Flutter Packages: http, shared_preferences, intl

Issues and Bugs Encountered and Resolved during Development:

Issue #1: The logo image stored in the assets folder was not displayed in the app even though it was correctly referenced in the pubspec.yaml file.

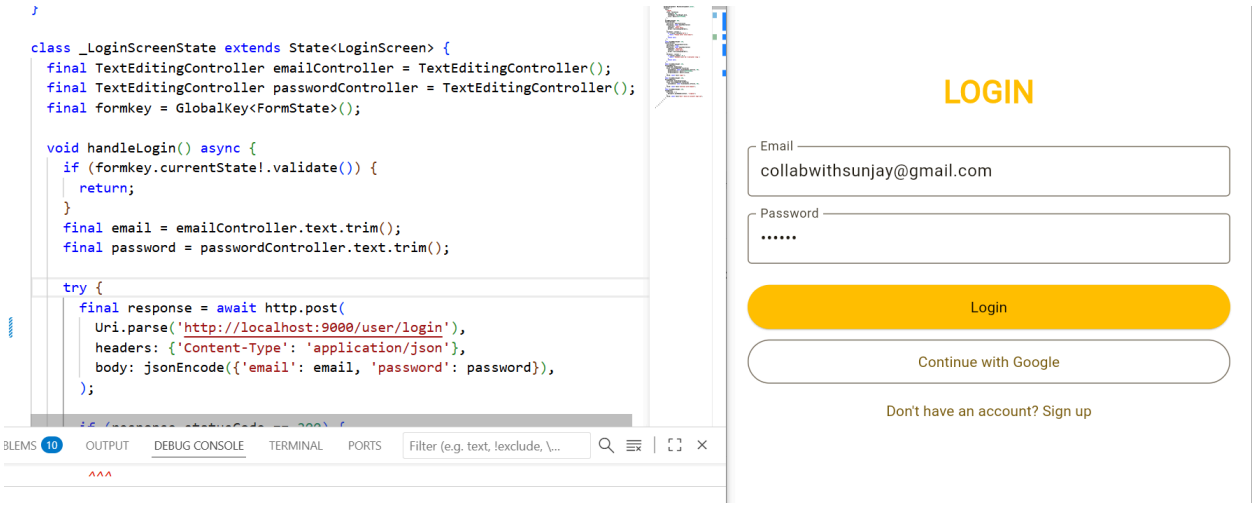



```dart
@override
Widget build(BuildContext context) {
  return Scaffold(
    appBar: AppBar(
      leading: Image.asset("assets/images/logo.png", height: 40),
      title: const Text(
        'Welcome to URL Shortner',
        style: TextStyle(
          fontSize: 24,
          fontWeight: FontWeight.w500,
        ), // TextStyle
      ), // Text
      centerTitle: true,
      backgroundColor: Colors.white,
    ), // AppBar
    body: Padding(
      padding: EdgeInsets.all(20),
      child: Column(
```

Solution: Initially, We tried loading it using the **Image.network()** method to test if the image itself was valid. After confirming that, We restarted **VS Code**, and the local asset image started loading properly using **Image.asset()**.

Issue #2: The form validation logic was working in the opposite way, invalid data was being accepted while valid data was being rejected.

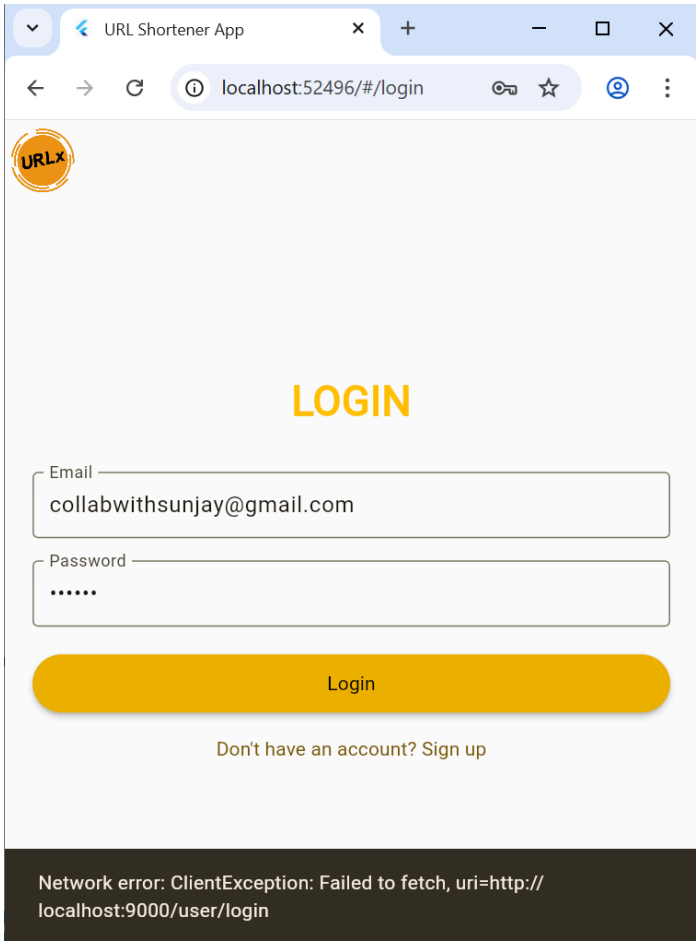
We mistakenly used if (formKey.currentState!.validate()) instead of if (!formKey.currentState!.validate()), which reversed the expected validation flow.



Solution: Corrected the condition by adding the missing negation operator (!), ensuring that the form only proceeds when all inputs are valid.

Issue #3: When connecting the Flutter frontend with the Node.js backend API, the app failed to fetch data and displayed an error.

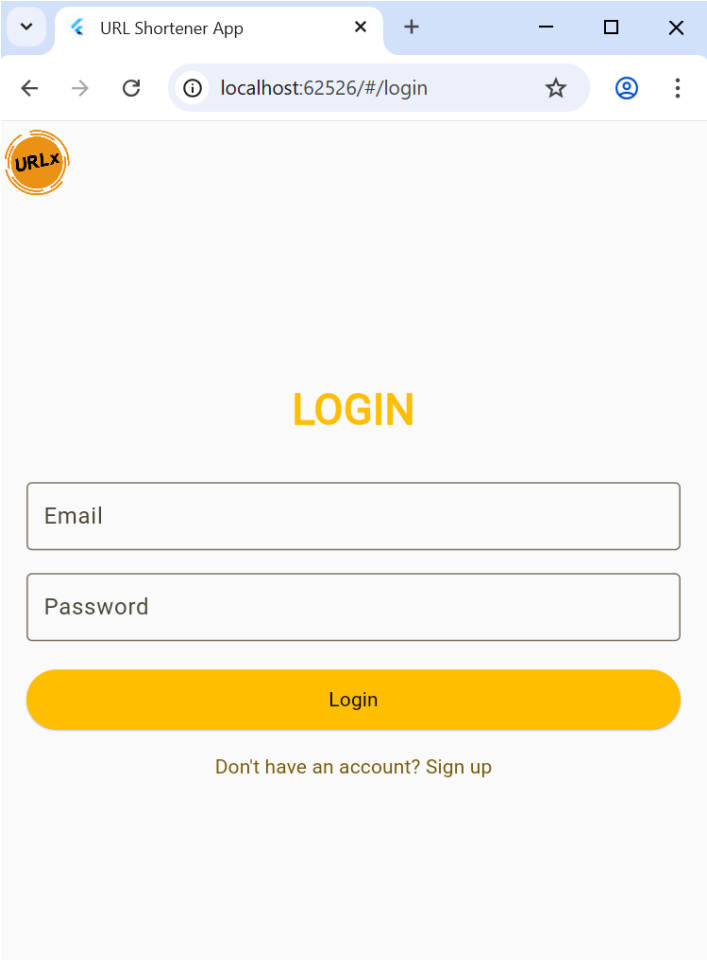
The backend did not include the CORS configuration, which blocked requests coming from the Flutter application.



Solution: Installed and configured the **cors** middleware in the Express.js backend using “cors” library.

Issue #4: Whenever the app was closed or restarted, the token got deleted and the user had to log in again.

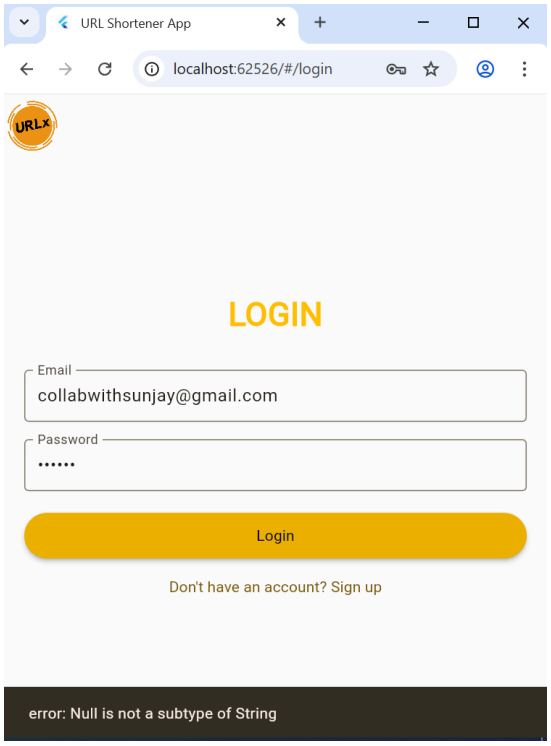
Because the token was stored in a normal variable, which gets deleted after app restart.



Solution: We learned about SharedPreferences and used it to save the JWT token locally. After implementing it, the token persisted even after the app was restarted, allowing users to stay logged in.

Issue #5: Null is not a subtype of String

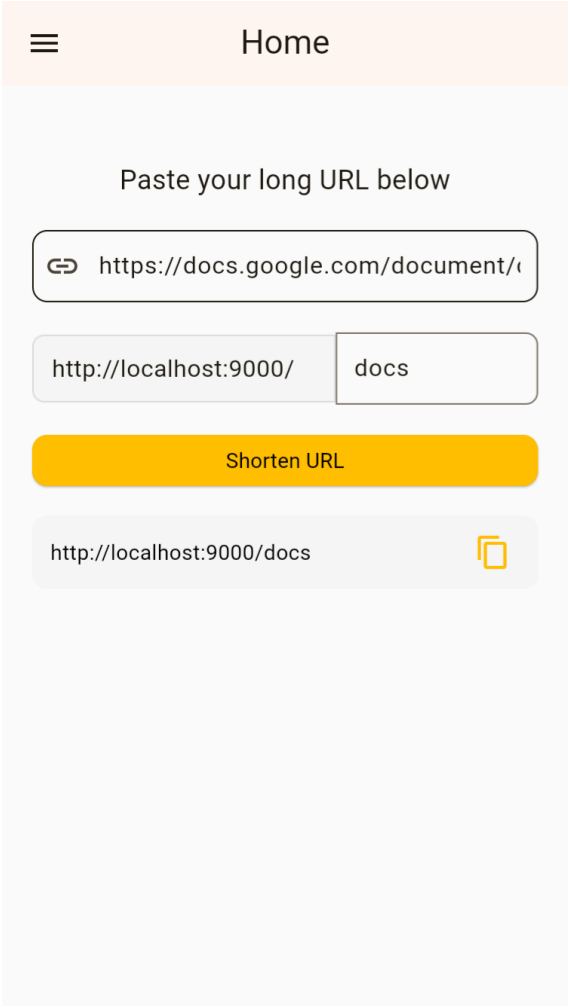
While testing the login functionality, an error appeared stating **“Null is not a subtype of String.”** This occurred because the app was trying to read a JWT token from SharedPreferences before it was actually stored. When the key 'token' did not exist, the function getString('token') returned null, which caused the app to crash.



Solution:
The issue was resolved by adding a null check before accessing the stored token. If the token was not found, a **Snackbar** message was displayed instead of crashing the app. This ensured that the login process handled missing data safely and improved app stability.

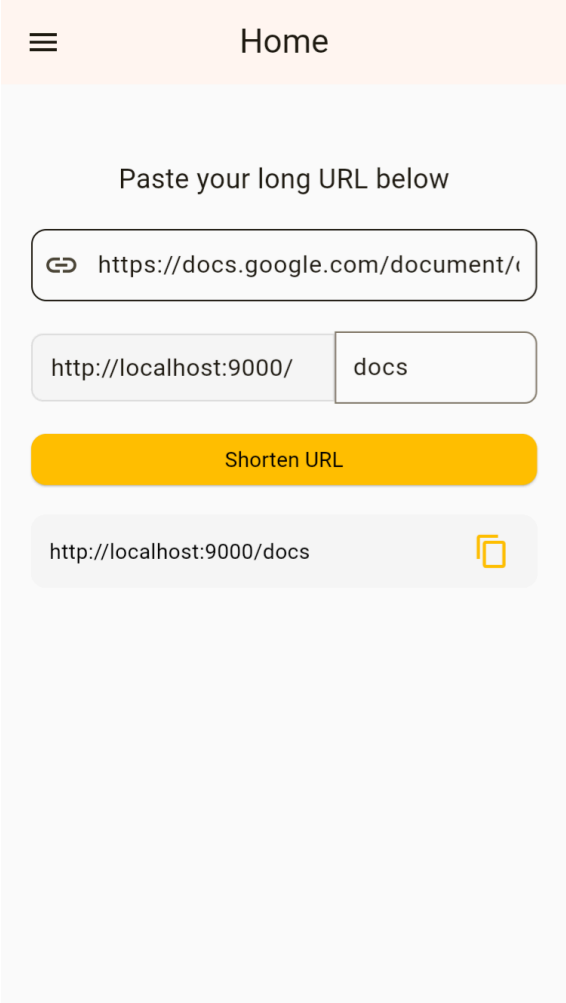
Issue #6: URL Not Validating Properly

While testing the URL shortener form, invalid or incomplete URLs were being accepted without showing any error. The app allowed users to shorten links even when the entered text was not a proper URL.



Solution: Added a regular expression–based validation in the form to ensure that only valid URLs starting with “http” or “https” were accepted before sending them to the backend.

Issue #7: Copy Button Not Working
The “Copy” button in the app did not copy the shortened link to the clipboard, and no confirmation message appeared.



Solution: The issue was fixed by implementing the Clipboard.setData() function properly and showing a Snackbar message after the copy action to confirm success.