Starting from the Firebase framework, we've implemented various design patterns in the development of our web application. These patterns are not only foundational in maintaining the structure and cleanliness of our codebase but also ensure that we can adapt and scale our application with ease. Below is an elaboration of how we applied specific design patterns, leveraging Firebase for authentication and data storage, and ensuring our application remains both organized and efficient.

Pure Fabrication:

This is when we make up an object that doesn't represent something from the real world but helps us keep our code clean. The `firebaseConfig` object is a perfect example. It doesn't represent anything in terms of the app's functionality; it's just there to hold all the Firebase setup details in one place. This way, we don't clutter other parts of our code with setup configurations that are only relevant when we're conducting Firebase setup.

Indirection:

This pattern is all about playing middleman to keep two pieces of our app from getting too chummy. It provides a buffer to maintain order and flexibility, and reduce coupling. Our `signUp()` and `logIn()` functions are the middlemen between the user hitting a button and Firebase doing its thing. These functions take care of talking to Firebase on behalf of the user, so the rest of the app can stay out of it. This makes it easier to change things later on without affecting the user experience.

Creator:

This one's about figuring out who gets the job of making new stuff. In our case, we're looking at how we handle downloading files from Firebase storage. The `setDownloadLink()` function sets up everything needed to grab a file from Firebase. It decides how and where to get the link, making it the "creator" of that link. This keeps our file-handling neat and means we don't have random bits of code trying to access Firebase storage all over the place.

Low Coupling:

This approach reduces dependencies between different parts of our system, ensuring smooth flow and easier maintenance. By minimizing the connection between the user interface and Firebase's underlying logic, we isolate changes, making debugging and troubleshooting much easier. Segregating Firebase-related code into specific modules or classes simplifies the overall codebase and enhances clarity over the project. This promotes modularity and reusability, as each component can be managed independently. Overall, adhering to the Low Coupling principle has allowed us to build a flexible and robust architecture, capable of adapting to evolving user needs and changes over time.

Controller:

We use the Controller design pattern to streamline user interactions and manage system operations effectively. When users sign up or log in, dedicated controllers handle these tasks seamlessly, ensuring a smooth and coordinated experience. By using controller design patterns, we establish clear boundaries between the Firebase authentication services and the user interface layer, allowing for easier maintenance and updates. This separation of concerns enables us to modify authentication processes without impacting other parts of the application. Because of this we maintain a strong and clear architecture while seamlessly integrating Firebase functionalities into our web app.

By sticking to these patterns, we've managed to keep our project organized and prepared for future changes. It's like putting each piece of code in its right place, making sure our app is not just a bunch of code thrown together but a well-structured project that's easy to update and maintain.

Pseudocode:
```
// Initialize Firebase configuration (Pure Fabrication)
initializeFirebaseConfig():
  // Firebase configuration details
  const firebaseConfig = {
    apiKey: "YourAPIKey",
    authDomain: "YourAuthDomain",
    projectId: "YourProjectID",
    storageBucket: "YourStorageBucket",
    messagingSenderId: "YourMessagingSenderID",
    appId: "YourAppID",
    measurementId: "YourMeasurementID"
  }
  // Initialize Firebase with the provided configuration
  initialize Firebase with firebaseConfig

// User sign up function (Indirection)
signUp(email, password):
  // Validate form inputs
  if form is valid:
    // Create a new user with provided email and password
    createUserWithEmailAndPassword(email, password)
      .then(() => {
        // On successful sign up, redirect to the confirmation page
        redirect to confirmation page
      })
      .catch(error => {
        // On failure, display error message
        show error message
      })

// User login function (Indirection)
logIn(email, password):
  // Validate form inputs
  if form is valid:
    // Authenticate user with provided email and password
    signInWithEmailAndPassword(email, password)
      .then(() => {
        // On successful login, redirect to datasets page
        redirect to datasets page
      })
      .catch(error => {
        // On failure, display error message
        show error message
      })

// Set download link for files (Creator)
setDownloadLink():
  // Reference to the file in Firebase storage
  const fileReference = firebase.storage().ref('dataSet.csv')
  // Get download URL for the file
  fileReference.getDownloadURL()
    .then(downloadURL => {
      // On success, set download URL for each download button
      for each download button:
        set URL as data attribute
        add click event listener for downloading
    })
    .catch(error => {
      // On failure, display error message
      show error message
    })

// Main program execution on DOMContentLoaded
on DOMContentLoaded:
  // Initialize Firebase configuration
  initializeFirebaseConfig()

  // Attach sign up function to form submit event if sign up form exists
  if signUpForm exists:
    attach signUp() to form submit event

  // Attach log in function to form submit event if login form exists
  if logInForm exists:
```

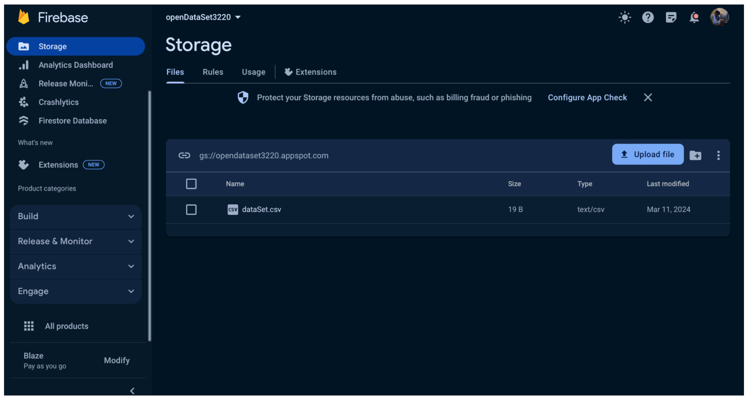
```
  attach logIn() to form submit event

// Redirect to login page if page requires authentication and user not authenticated
// Redirect to datasets page if page doesn't require authentication and user authenticated
handleAuthentication()

// Attach logout function to click event if logout button exists
if logoutButton exists:
  attach logout() to click event

// Set download link for files
setDownloadLink()

// Setup dynamic UI elements (e.g., section links, search functionality)
setupDynamicUIElements()
```

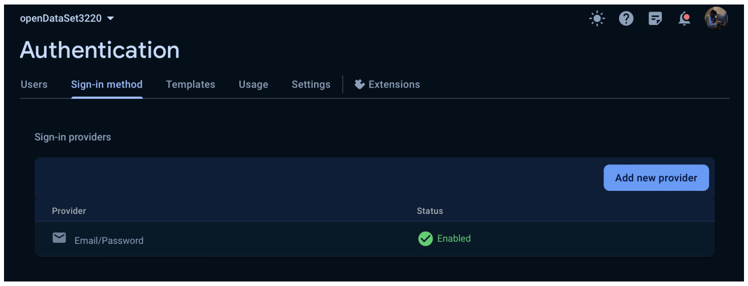


Storage

Protect your Storage resources from abuse, such as billing fraud or phishing

gs://opendataset3220.appspot.com

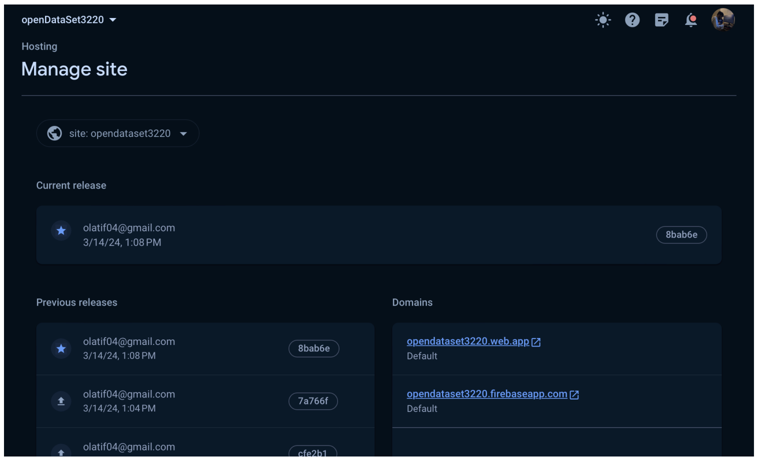
Name	Size	Type	Last modified
dataSet.csv	19 B	text/csv	Mar 11, 2024



Authentication

Sign-in providers

Provider	Status
Email/Password	Enabled


openDataSet3220

Hosting

Manage site

site: opendataset3220

Current release

olatif04@gmail.com
3/14/24, 1:08 PM
8bab6e

Previous releases

olatif04@gmail.com
3/14/24, 1:08 PM
8bab6e

olatif04@gmail.com
3/14/24, 1:04 PM
7a766f

olatif04@gmail.com
cfe2b1

Domains

opendataset3220.web.app
Default

opendataset3220.firebaseapp.com
Default