Practice Report

Delivery 2

Student: Òscar Meléndez Codina

Student: Jordi Solé

Student: Pol Laguna Soto

Student: Jaime Mariano Servera

Student: Iván García Rodríguez

Escola Tècnica Superior
d'Enginyeria de Telecomunicació de Barcelona

Escola Tècnica Superior
d'**Enginyeria de Telecomunicació de Barcelona**

UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Facultat d'Informàtica de Barcelona

FIB telecos BCN

Contents

Final Design

The project consists of developing a web-based forum platform, where different users can upload images accompanied by metadata (such as publication date, unique identifier, title, description, tags, etc.), and other users can comment on those posts.

The main goal is to create an interactive and secure environment where users can share visual content and engage in related discussions.

The system includes two main roles:

- Administrator (Admin): has full permissions to edit or delete any content on the platform, including posts, comments, and user data.
- Standard User: can create, edit, or delete only their own posts and comments, without access to modify other users' content.

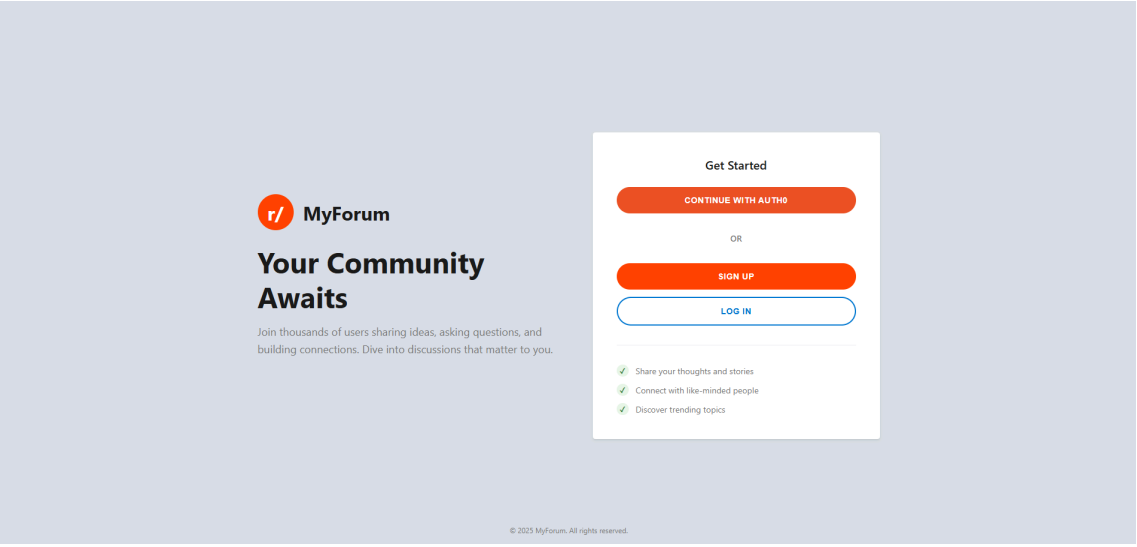
For development, a client–server architecture has been implemented using modern technologies:

- Frontend: built with React.js, providing a dynamic, modular, and responsive user interface.
- Backend: implemented with Node.js and Express, responsible for handling requests, authentication, authorization, and database communication.

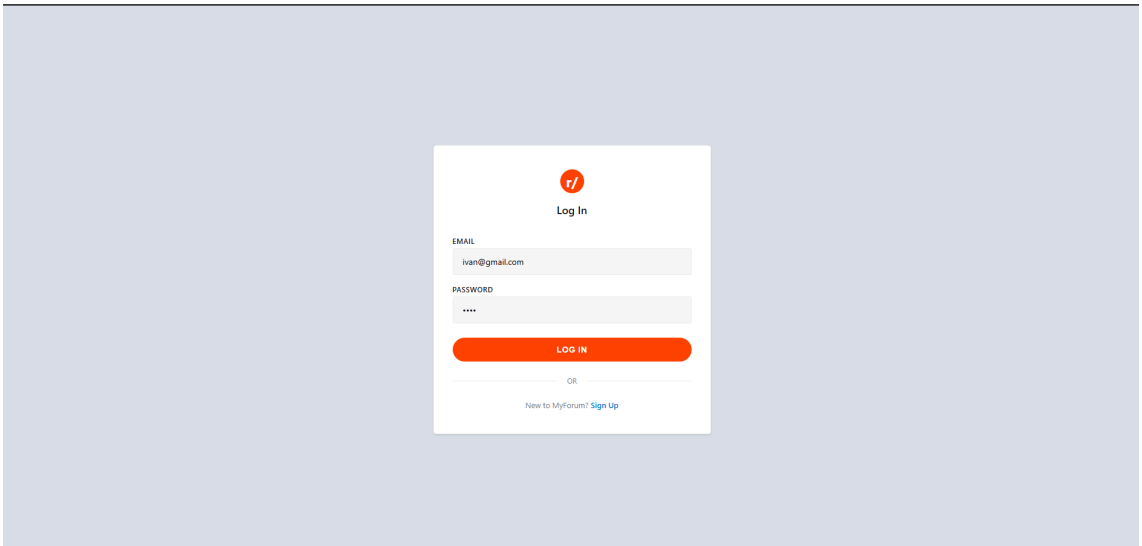
At this moment we have a functional forum where users can post images with some text alongside and comment on their posts or other ones.

We chose this design for the frontend including the following pages:

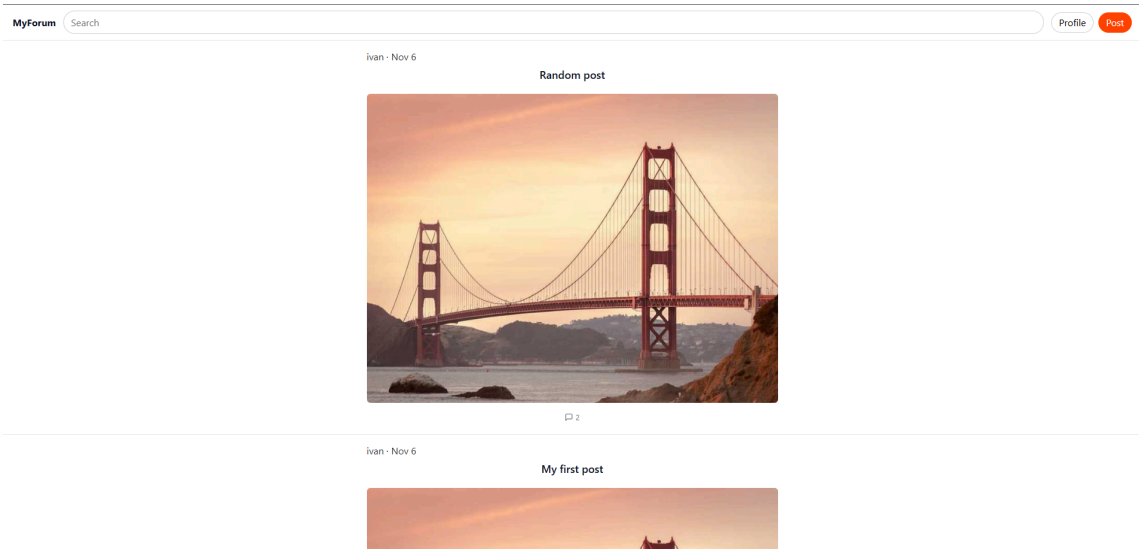
Home:



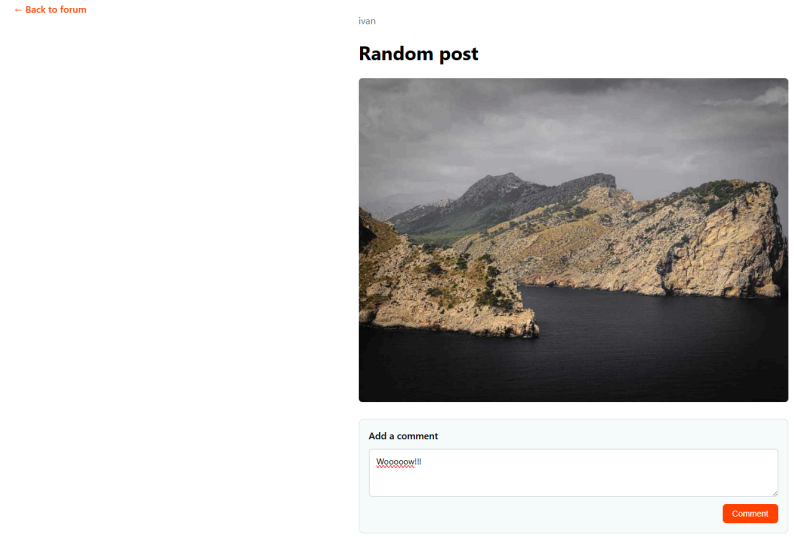
Login:

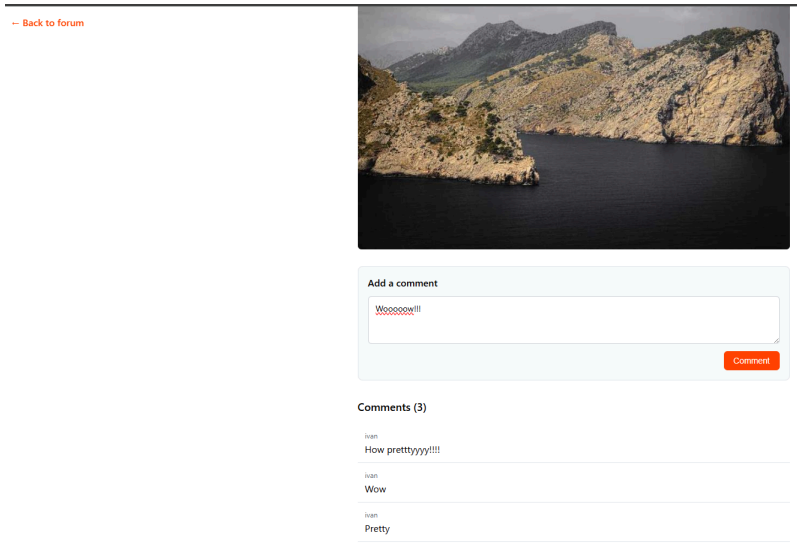

Log In
EMAIL
ivan@gmail.com
PASSWORD
LOG IN
OR
New to MyForum? Sign Up

Forum:

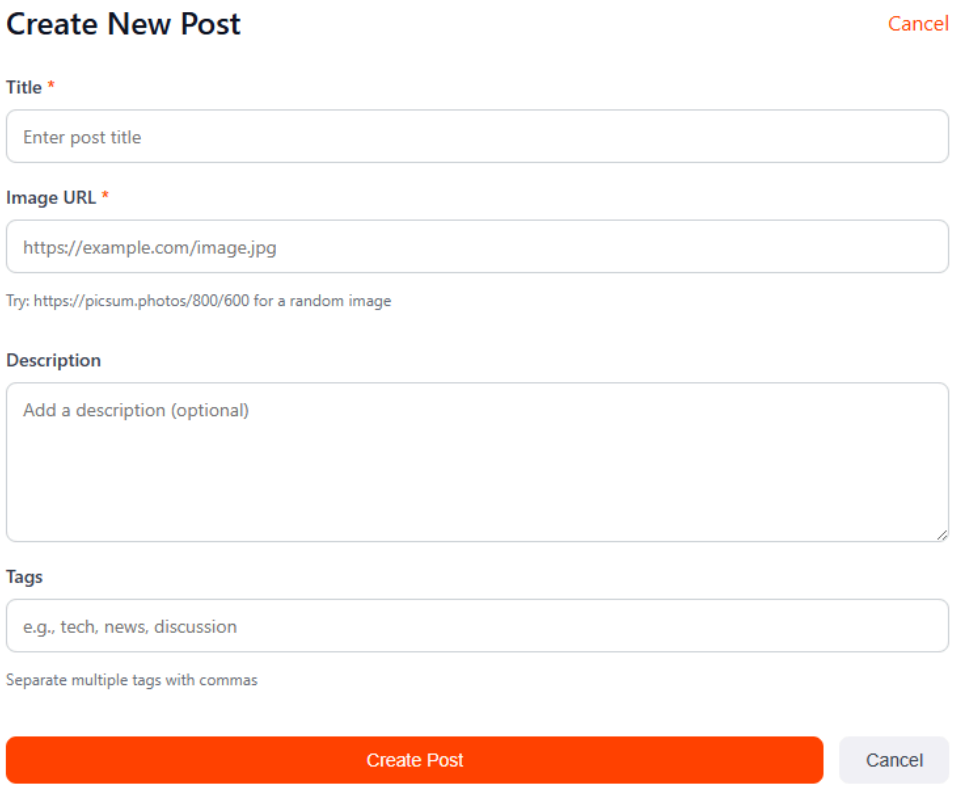

MyForum
Search
Profile
Post
ivan · Nov 6
Random post
ivan · Nov 6
My first post

Comment Section:





Create Post:



For the backend we have developed the necessary endpoints to ensure the proper functioning of the forum, allowing users to create posts and comments, as well as retrieve all posts or a specific one for example.

Security Implementations:

- **Self Signed Certificate**: For this project, a self-signed SSL/TLS certificate will be used to enable secure HTTPS communication between the React frontend and the Node.js backend. We chose this approach for local development or educational environments, as it allows data encryption and testing of secure connections without the need for a registered domain. Although free certificates from trusted authorities such as Let's Encrypt are recommended for production websites, they are unnecessary in this case since the project will not be publicly deployed. Therefore, using a self-signed certificate provides sufficient security and simplicity for the scope of this school project.

- **OAuth 2.0:** In order to log in to their accounts, the users will either use their user/password or their Google/Github account via auth0. Auth0 is a platform that handles authentication and authorization. It implements the OpenID Connect protocol, which is an extension of OAuth 2.0, to manage user authentication. OIDC adds an identity layer to OAuth 2.0. When a user signs up using a username and password, Auth0 stores their credentials in its managed database, applying password hashing. This login will only work with normal users. Admins will have their own local log in mechanism, managed by the backend. So normal users will log in using the Auth0 mechanism, and admin users will log in with the local log in, without using Auth0.

- **Password Protection:** We will not store plaintext passwords; instead the server will hash them with a modern password hashing function (bcrypt) so that what resides in the database is only a slow-to-crack verifier rather than a reversible secret. Every password will be combined with a unique, cryptographically secure random salt before hashing, preventing rainbow-table reuse and ensuring identical passwords are stored as different hashes across users and systems.

- **Database Encryption:** The database has been fully encrypted at rest using InnoDB and a MySQL plugin (called keyring). This allowed us to implement Transparent Data Encryption, which means the app works in a normal way but the disk data is protected (data is decrypted on every query). We also obfuscated the table names, including normal views for our app, so that an attacker won't even know which table does each .idb file contain.

  Additionally the backend will use parameterized queries or ORM-based access to prevent injection attacks. Proper access controls and environment variables will be used to hide database credentials and connection strings. These measures will help ensure that even if the system is targeted by an attacker, the integrity and confidentiality of the stored data remain protected.

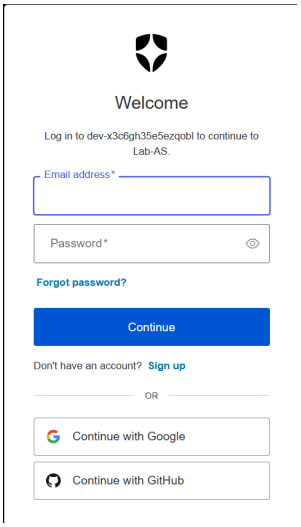
Work in progress

We have started working in order to implement the security features since the beginning, and this is the current state of each feature at the moment:

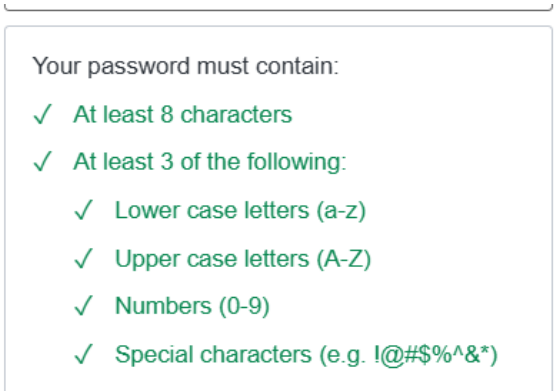
- **OAuth 2.0**:

Auth0 has been implemented correctly, so users can now authenticate and obtain authorization via username/password, Google, or GitHub accounts

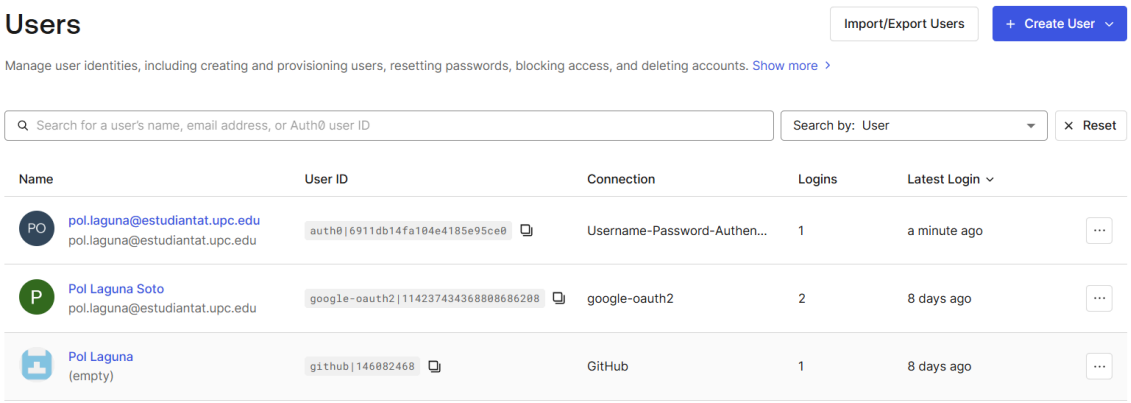
Once the user clicks the Login with Auth0 button on the start screen, they will be automatically redirected to the Auth0 login page, where they can select how they want to authenticate into the app:



If the user signs up and creates his password, the password must contain



Once the user enters the web application via Auth0, they get registered in the Auth0 database, where we can see in this example a user for each Connection: one with username-password, one with google-oauth2, and one with GitHub.



Auth0 has been implemented using the following tutorial:
https://www.youtube.com/watch?v=sTJaHQINpTc&t=107s

The auth0 provider is added in index.js:

```JavaScript
            <Auth0Provider    domain="dev-x3c6gh35e5ezqobl.eu.auth0.com"
clientId="zfvgGXlKFquTnJpHbi2LByXH8k0ixIMJ"
redirectUri={window.location.origin}>
```

Then in Home.jsx, we can check if the user is authenticated via Auth0 to navigate to /forum, the main page:

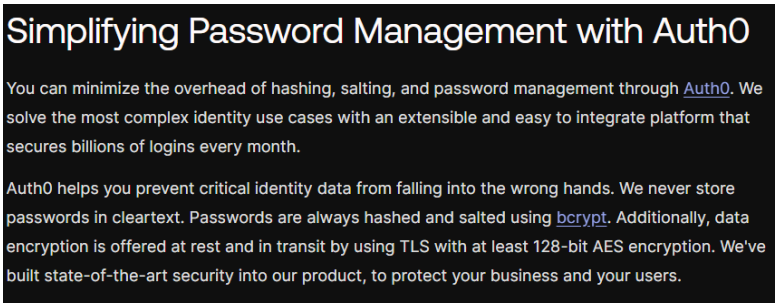
```JavaScript
 const { loginWithRedirect, isAuthenticated, isLoading } = useAuth0();
  const navigate = useNavigate();

  useEffect(() => {
    if (!isLoading && isAuthenticated) {
      navigate("/forum");
    }
  }, [isAuthenticated, isLoading, navigate]);

  const handleAuth0Login = () => {
    loginWithRedirect({
      appState: { returnTo: "/forum" }
    });
  };
```

Thanks to the button with onClick handleAuth0Login, where the user can authenticate:

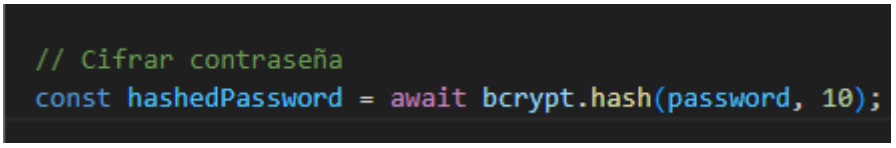
```
JavaScript
          <div style={styles.buttonGroup}>
             <button
               onClick={handleAuth0Login}
               style={{ ...styles.button, ...styles.auth0Button }}
                    onMouseOver={(e) => e.target.style.backgroundColor =
'#D14820'}
                     onMouseOut={(e) => e.target.style.backgroundColor =
'#EB5424'}
             >
               Continue with Auth0
             </button>
```

-Password protection:

Auth0, the service used for normal users to log in, uses password hashing, concretely with bcrypt, a hashing method.



In addition to that, if an administrator logs in, their passwords don't get stored hashed by ***Auth2.0***, it gets stored automatically. Using ***bcrypt*** also as the hashing mechanism



Work to do

Although most of the core functionalities and security mechanisms are already implemented, there are still two main tasks pending before the final delivery:

1. SSL Certificate Implementation

We plan to generate and configure a self-signed SSL/TLS certificate to secure the communication between the frontend (React) and the backend (Node.js + Express).
This step will allow us to enable HTTPS in the local environment, ensuring that all transmitted data — such as login credentials and profile information — is encrypted.

The purpose of this implementation is to:

- Demonstrate how HTTPS improves confidentiality and integrity during data transmission.
- Simulate a secure production environment without relying on an external Certificate Authority.
- Test that both frontend and backend correctly handle secure requests (https://).

Although in real deployments a trusted CA (e.g., Let's Encrypt) would be used, for this project a self-signed certificate is sufficient to validate SSL functionality and secure communication.

2. Frontend Improvements — Preventing XSS and SSTI

We will also finalize a new Profile Page that allows each user to modify their display name.
This feature will be used to demonstrate that our frontend properly escapes all user input and is resistant to Cross-Site Scripting (XSS) and Server-Side Template Injection (SSTI) attacks.

The page will:

- Include an input field to change the username.
- Sanitize and validate all user input before sending it to the backend.
- Safely render the updated name across the interface (e.g., header, posts, or comments).

This enhancement will serve as proof that our frontend templates correctly escape user data, and that any potentially malicious code (like <script> tags) is displayed as plain text rather than executed.
It will also improve the user experience by providing basic profile customization while maintaining strict security controls.