**Architecture Explanation**

**Frontend**

The frontend layer is built using React and provides the user interface for the platform. It interacts with Auth0 for user authentication and authorization. Auth0 is a third-party service that provides secure authentication and authorization for web and mobile applications. It supports a wide range of authentication and authorization protocols, including OAuth2 and OpenID Connect. By using Auth0, the platform can offload the complexities of authentication and authorization to a dedicated service, allowing the platform to focus on its core functionality. More over that will allow us to generate JWT for verification and we can store in it user type, for front end differentiation (source: <https://community.auth0.com/t/how-to-add-roles-and-permissions-to-the-id-token-using-actions/84506>)

**API**

Once the user is authenticated and authorized, the frontend interacts with the backend via the REST API Gateway, which serves as the entry point for all client requests. The REST API Gateway is responsible for exposing the API that the React frontend consumes.

API design:

* Admin:
  + DELETE /delete/user/{id} delete user, requires special admin token
* Authentication
  + POST /auth/register - Register a new user
  + POST /auth/login - Log in a user and get a JWT
  + GET /auth/profile - Get the profile of the logged-in user
* Small Business or User
  + GET /{id}– info about user or organization (JWT in header request)
* Security Tests
  + GET /{id}/tests - Get a list of security tests that allowed for user (JWT in header request)
  + For each test
    - POST /{id}/tests/1…n - Start security test (JWT in header request)
      * In post request we send JSON with link of required web app
        + {‘link’ : ‘https://xxx.com’}
      * As a result we get a response in JSON as well with required key metrics for this test and then represent it on the front end
  + GET {id}/tests/results/1…n - (JWT in header) Get the results of last security test – returns blob object (pdf generates in backend)

**User differentiation**

The platform supports two user types: individual users and small business users. Individual users have a different UI experience but consume the same backend information as small business users. Small business users have access to a web vulnerability scanner in addition to the features available to personal users.

**Django/Python**

The backend layer is built using Python/Django and contains the majority of the business logic. It interacts with the MongoDB database and the Security Test Orchestrator. The backend is responsible for storing the results of security tests in the MongoDB database. The MongoDB database is chosen for its scalability and flexibility as a NoSQL database, and it stores user information, organization information, and security test results.

**Orchestration**

The Security Test Orchestrator is responsible for orchestrating and executing security tests. It can run tests using Docker containers and Python libraries, ensuring proper data sharing between tests. It also handles the scheduling and prioritization of tests. The actual security testing tools and libraries are containerized using Docker for easy deployment and scalability.

**Kubernetes**

The Kubernetes cluster is responsible for managing the Docker containers. It handles tasks such as container deployment, scaling, and management. This ensures that the platform can handle many security tests running concurrently.

**Other**

The platform also integrates with third-party services for additional functionality, such as threat intelligence and identity verification. These integrations are handled by the backend layer.