InfiniMuse

High Level Design

Hello World

Updated - April 27, 2021

Team Leader: Matt Caponi 011161928

Ian Macabulos 016401240

Jakub Koziol 016523856

Shaan Shariff 015741750

Revision History

Date	Version	Description	Author
11/10/20	<0.1>	Started Outline	Jakub Koziol, Matthew Caponi, Shaan Shariff, Ian Macabulos, Wonsuk Seo
11/13/20	<0.2>	First Draft	Matthew Caponi Ian Macabulos Jakub Koziol
11/15/20	<.3>	Second Draft	Matthew Caponi
11/16/20	<.4>	Final Draft	Jakub Koziol, Matthew Caponi, Shaan Shariff, Ian Macabulos, Wonsuk Seo
4/27/21	<1.0>	Updated Authentication and Authorization	Matthew Caponi

Table Of Contents

1. Introduction

1.1. Purpose

The purpose of this document is to provide a high-level design for the architecture of our web application, as well as relevant policies that developers must follow. It will use both text explanations and diagrams to help convey the high-level design in a simplistic way. This document is mainly intended for both developers and system administrators, as well as technically-minded clients.

1.2. Scope

This document covers the following scope:

- Security Policies
- Error Handling Policies
- Design Architecture

2. Policies

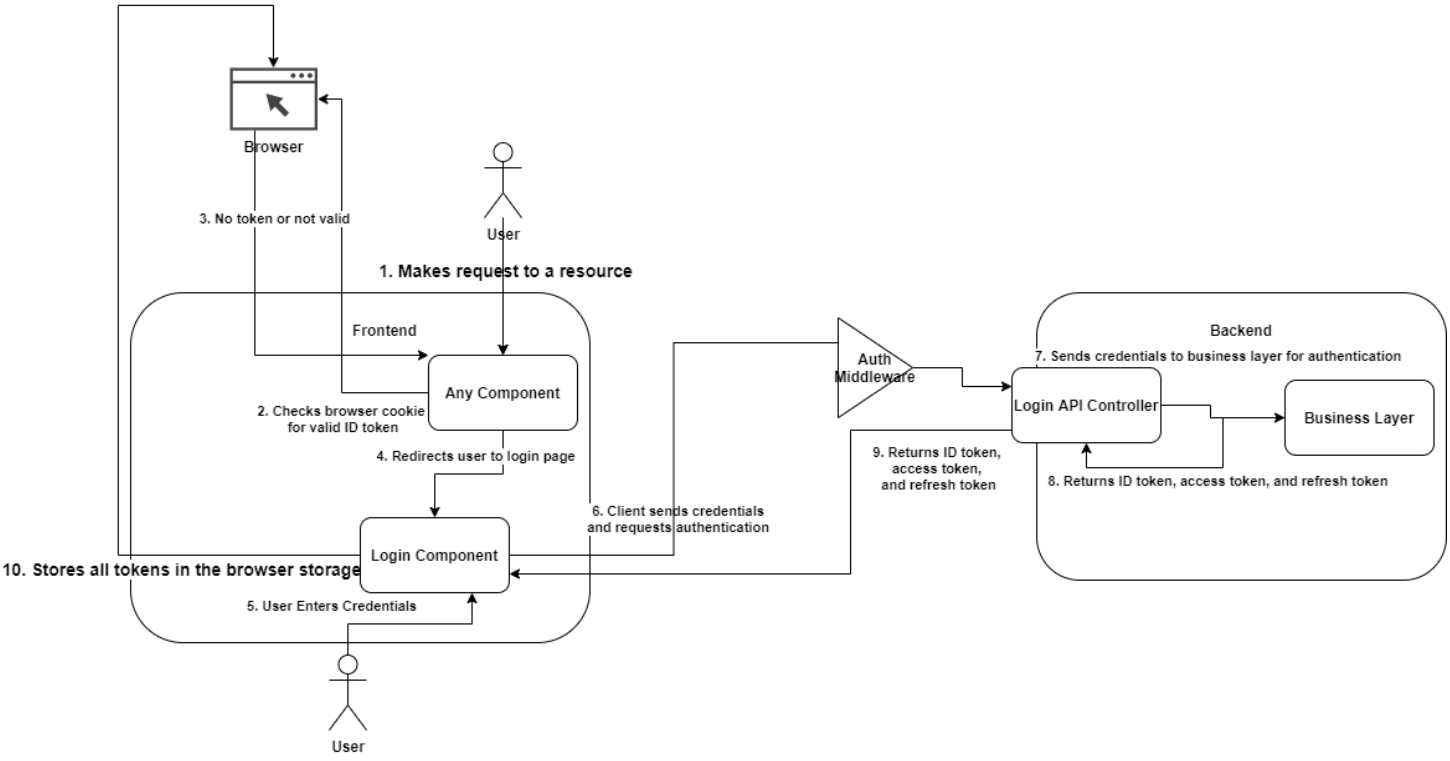
2.1. Security

2.1.1. Encryption

- We will be using HTTPS for all network requests and responses, to ensure secure data information transfer.
- Authentication and Authorization will rely on a token-based system using JWT tokens. Each JWT token will be signed using an RS256 algorithm.
    - The public and private key will both be 3072 bits and base 64 encoded.
    - They will also be stored in a secret manager during development to avoid being committed during source control and a separate server in production.
- The password will be hashed using a SHA256 hash + key, and a randomly generated salt, which will both be stored in the database.
    - Both the hash and the salt will be 256 bits.
    - A new salt will be generated each time the user changes their password.

2.1.2. Authentication

- Token-based Authentication Process
  - Registration
    - Upon registration, the user will be required to enter identification and access information.
    - The password will be immediately encrypted using a SHA256 hash and a salt, and then sent straight to the database.
    - Once registered, they will be sent an email containing a base 64 encoded authorization token appended to the end of a url, which they must click on to confirm their account.
  - Login

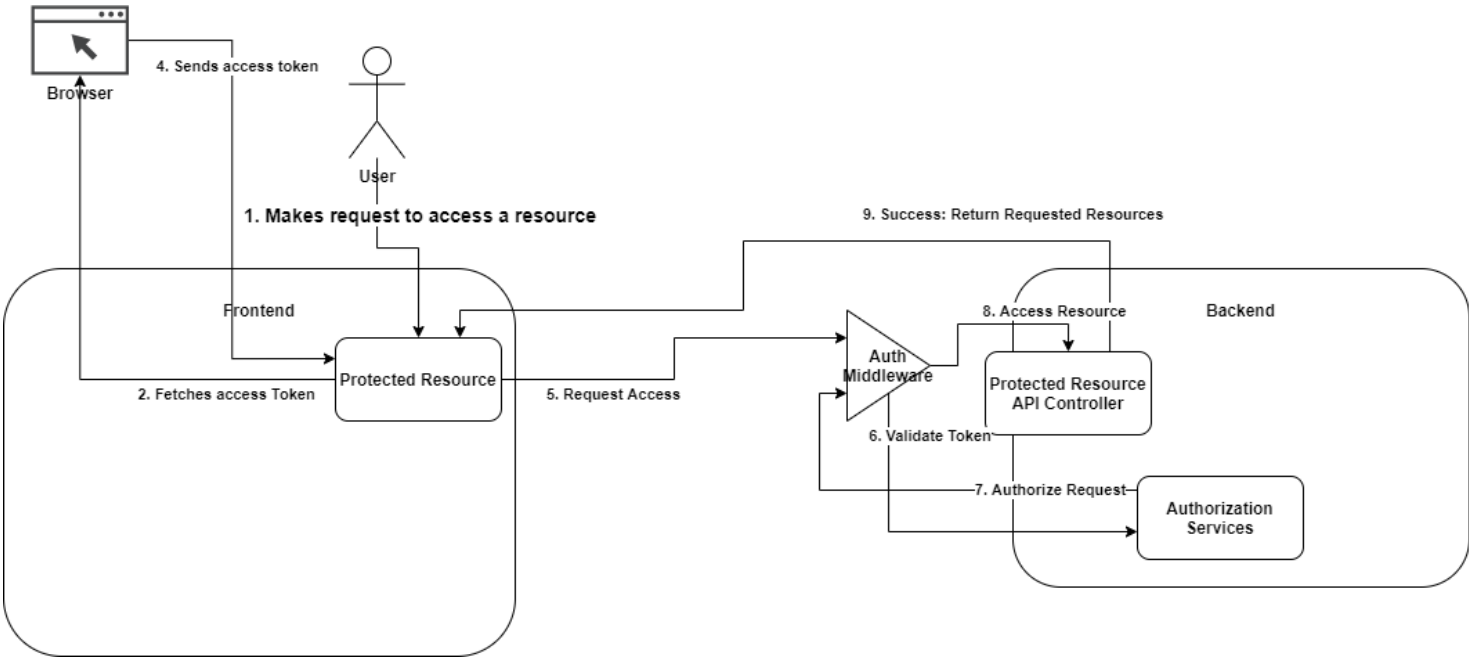


    - Whenever a user attempts to access any resource, the client will check if they have a valid ID token.
      - In the absence of a valid ID token, the user will be redirected the login page, where they will be required to enter their username and password.

        - The client will then send a request to the login API endpoint with ‘ID, to obtain a new ID token, using the username and password for backend authentication.
        - The login API endpoint will pass the username and password to a login manager, which will perform validation checks.
        - The login manager will use an authentication service to hash the password and compare it with the stored, hashed password, for the user with that username.
        - Upon successful authentication, the client will then be issued an ID token using the JWT structure, signed with an RS256 algorithm.
            - In the event that the user authentication fails, the endpoint will return a 401 Not Authorized http error.
- ID Token Policies
    - ID Tokens
        - The ID token will contain default claims that identify the user and provide metadata.
        - The ID token will be stored in a browser cookie.
        - The ID token will be set to expire after a designated period of time.
            - The user will then be automatically logged out after the next attempted request, and will be brought back to the login screen.
    - Refresh Tokens
        - The user will be issued a refresh token that must be passed back to the API endpoint anytime the client requests a new ID token or access token.
        - A new refresh token is generated each time the user exchanges it for an access token or ID token.

2.1.3. Authorization

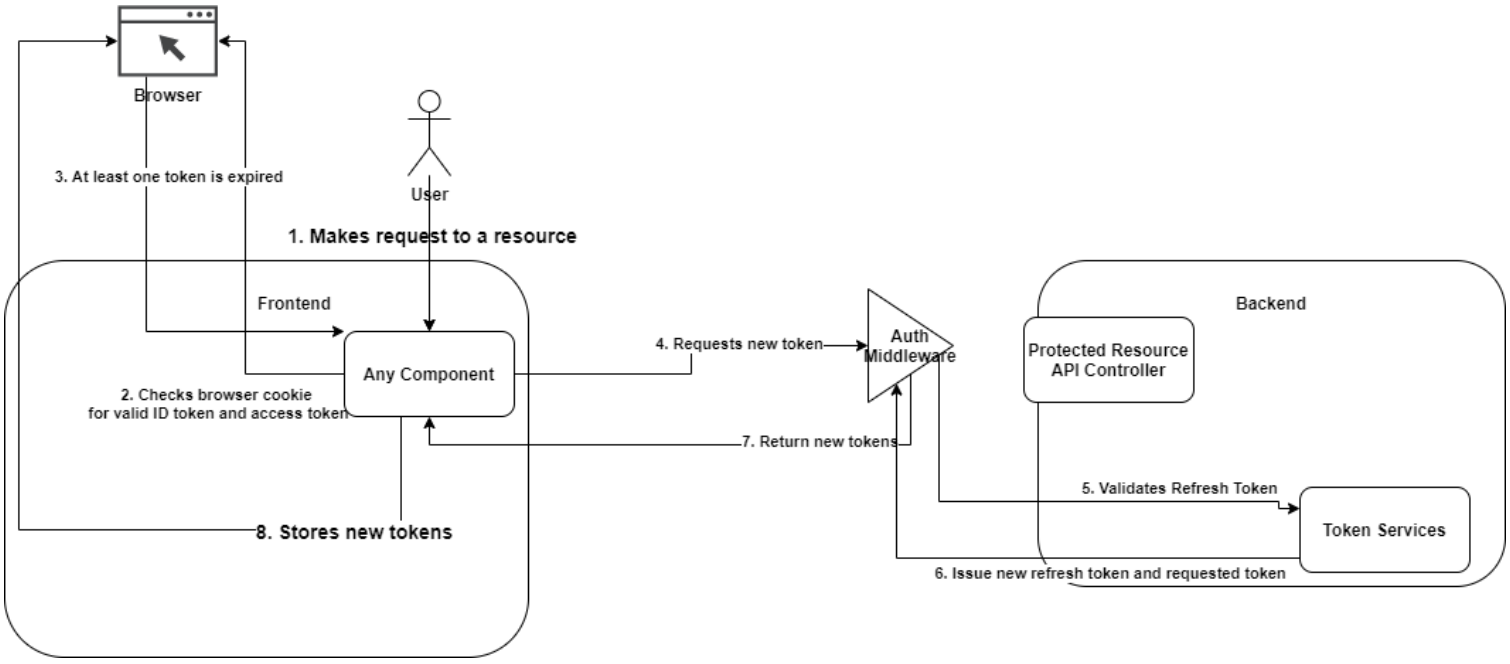
- We will be using attribute-based access control to implement authorization on this system.
- This will consist of token authorization, containing scopes for rough-grained access control, and claims for fine-grained access control.
- Resource Access



- Upon login, a user will be issued an ID token, access token, and refresh token. The ID token is described in 2.1.2.
- The access token will contain claims that detail the access information for that user, specified with:
  - Scopes
  - Roles
  - Operations
- Whenever a user wants to access a protected resource, the client will make a request to the respective API endpoint by sending the access token of the user.
- The API endpoint will then validate that the user has the proper scopes and claims to access that resource and perform the desired operation.
  - In the event that the user does not have the required permissions, the API endpoint will return a 403 Forbidden http error.
- The client should also be developed so that resources that the user is not authorized to access do not even display to them in the first place.

- Access Token Policies
  - Access Tokens
    - The access token will have a specified expiration date, after which time it will use a refresh taken to reissue the access token.

- Refresh Tokens



- The user will be issued a refresh token that must be passed back to the API endpoint anytime the client requests a new ID token or access token.
- A new refresh token is generated each time the user exchanges it for an access token or ID token.

2.1.4. Privacy

- To protect the privacy of the users, Personally Identifiable Information (PII), will not be stored in any logs.
- The passing around of PII will also be minimized, to reduce the chance of that PII being leaked in the unlikely event of a system breach.
- Proper security encryption methods will be implemented to protect application security, and minimize the chance of any data being stolen.
- See 2.1.1 for more information on our encryption policies.

2.1.5. DDoS Mitigation

- The web application will be equipped with a DDoS mitigation system that will limit the amount of connection possible, if an attack is detected.
- The web server will be secured with a DDoS mitigation system.

2.2. Logging

- Any activity performed by the user will be logged into files stored on the server, for the developers and admin to view.
- Logs will consist of a timestamp and the details of the activity including who performed the event and what the event was.
- Logging will take place at the API Controller level of the system, logging transactions based on incoming requests and outgoing responses.

- Categories of Logs:
    - Trace
        - Sensitive info just for debugging
    - Debug
        - Where
            - All layers
    - Information
        - Where
            - Controllers
        - What
            - Flow of application.
            - Requests made by the user.
            - User interactions with system
    - Warning
        - Where
            - Controllers
        - What
            - Security alerts
    - Error
        - Where
            - All layers that have error handling
        - What
            - Errors handled that don't crash the system
    - Critical

        - Where
            -
        - What
            - Crashes

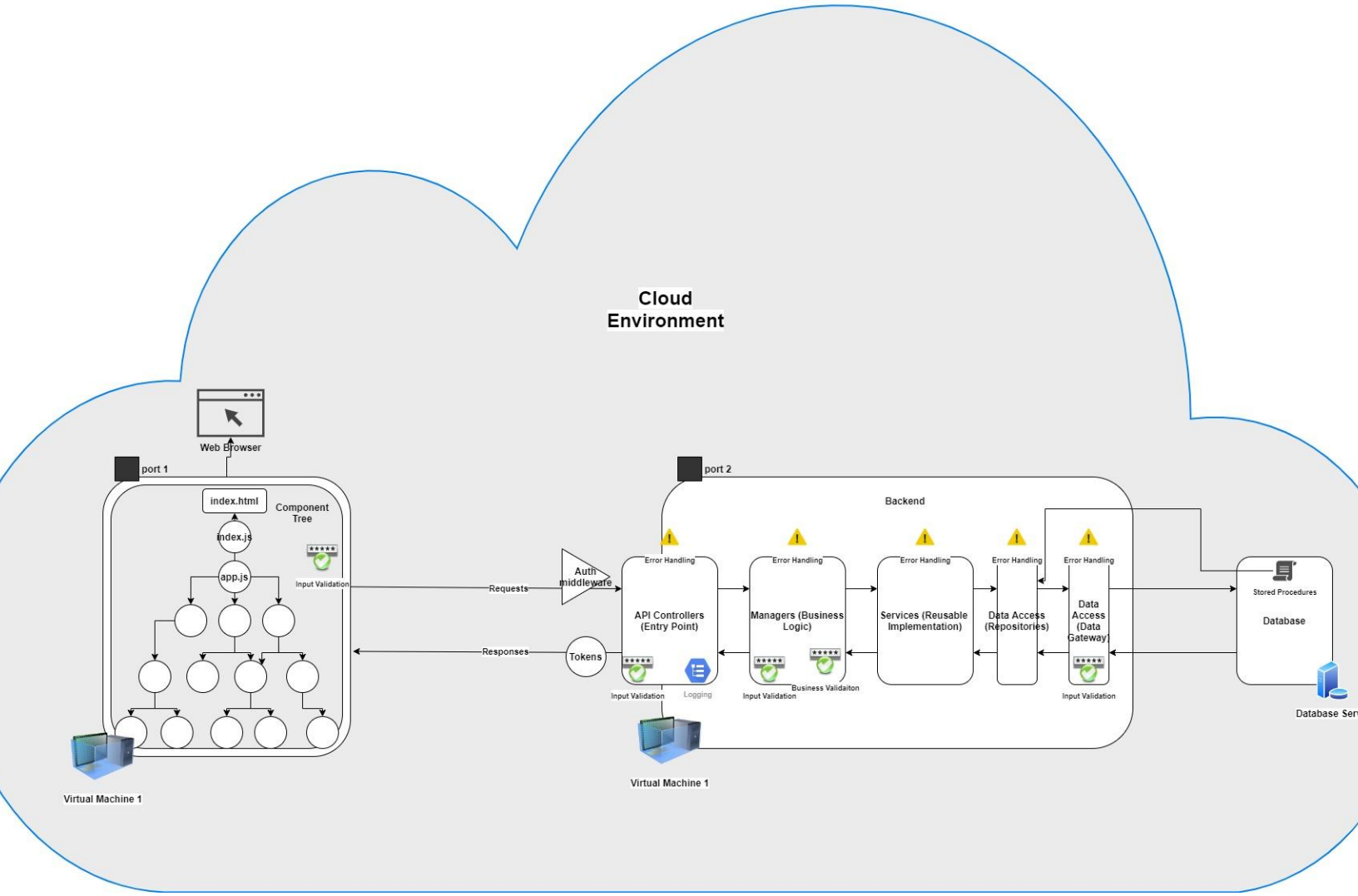
2.3. Error Handling

- Exception Handling
    - Every layer of the backend will handle exceptions, except for managers.
    - Managers will only need to handle exceptions for any work that is not delegated to a service (this shouldn't happen often, if at all).
- Business Rule Validation
    - Managers will handle the validation of business rules.
    - Managers are the sole backend layer that will handle business rule validation.
- Input Validation
    - The frontend will handle input validation to enforce input-related business requirements.
    - The API layer will handle input validation to check for null values or objects.
    - The manager layer will handle input validation to check for null values or objects.
    - The data gateway will handle input validation to check for null values or objects.

3. Architecture

3.1. Description

The application will follow a 3-tiered architecture, consisting of a frontend, backend, and data store, that will be deployed to a cloud environment.



3.2. Front End

- The frontend will be implemented using a separate runtime environment from the backend.
- Architecture
    - Single Page
        - Will follow a Single Page Application (SPA) architecture.
        - This will consist of a single index.html file that is to be served to the client.

        - This single index.html page will be composed of a single, root level component.
    - Component Tree
        - The single, root level component serves as the root for the component tree.
        - The component tree consists of a series of page-components, which each consist of their own component sub-trees that make up the individual page.
        - Components may be used for any of the following:
            - Pages
            - Specific functionality
            - UI elements
            - Groups of similar UI elements
    - Routing
        - Routing will be used to specify URL routes for each page component.
    - Communication
        - The frontend will communicate with the backend by sending http requests and receiving http responses.
    - Authentication and Authorization
        - The frontend will receive the following types of tokens:
            - Id tokens
                - Used to authenticate the user and keep them logged in.
                - Validated by the frontend.
            - Access tokens
                - Used to request access to backend API's for a given protected resource.
            - Refresh tokens.
                - Used to retrieve replacement ID and access tokens upon token expiration, to avoid having to make the user login again.

3.3. Backend

The backend will consist of the following layers:

- API Controller Layer

    - Receive requests from the frontend.
    - Authorize the requests.
    - Send the information to the appropriate managers for processing.
    - Return back a response from the managers with the appropriate, processed data, to the frontend.
    - Handles exceptions.
    - Handles business-specific errors.
    - Uses http error codes to communicate errors to the client.
- Business Layer
    - Managers
        - Controlling the flow of action of a given resource or operation.
        - Validating business rules.
        - Implementing business logic.
        - Delegates any reusable, business-agnostic implementation to the appropriate services.
    - Services
        - Perform a reusable operation that is business-agnostic.
        - Generally will be used at the behest of managers, although services will sometimes be called by other services that want to delegate work that is outside of its scope.
        - Will send data to the data access layer repositories if the database access is needed.
        - Handles errors.
- Data Access Layer
    - Repositories
        - Retrieve queries from stored procedures stored in the database.
        - These stored procedures to the appropriate Data Gateway method, with the appropriate parameters.
        - Handles database errors.
    - Data Gateways
        - Receives parameterized queries, stored in procedures, from a given repository.
        - Sends that query to the database for execution and returns back the appropriate data.
        - Performs last minute input validation.
        - Handles database errors.

3.4. Database

- Contains stored procedures that hold the specific, parameterized queries for the application.
- Contains tables to relevant user data.
- Uses third-normal form to separate data across tables, linked by foreign ids.

3.5. Cloud

- The entire application will be deployed to the cloud for production.
- The database will be running on a separate server.
- The backend and frontend will run on the same virtual machine, but on separate ports.