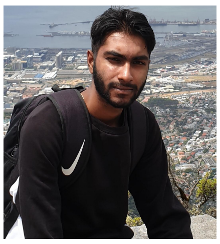
LUNA

Coding Standards

August 1st, 2023

Group Members

				
Priyul Mahabeer	Ashir Butt	Jaimen Govender	Dharshan Pillay	Edwin Sen-Hong Chang
u20421169	u20422475	u20464348	u19027487	u20424575

EPI·USE®

File Structure

- Each component, service, and model should have its own file.
- Group related components, services, and models in separate folders.
- Use meaningful and descriptive names for files.

📁 Frontend <br>

📁 Backend

Import Statements

Import statements should be organized in the following order:

1. 1. Angular core modules
2. 2. Third-party modules
3. 3. Custom modules and components
4. 4. Services and other dependencies

Component Naming

- Component file names should be in kebab-case, e.g., create-account.component.ts.
- Component class names should use PascalCase, e.g., CreateAccountComponent.
- Component selectors should use kebab-case, e.g., app-create-account.

Variable Declarations

- Use const for variables that do not need reassignment. e.g const generateToken =
- Use let for variables that need reassignment. e.g let server =
- Declare variables with meaningful and descriptive names. e.g projectId

- Avoid using single-letter variable names unless they are used in a limited scope (e.g., loop counters).

Constructor

- * Inject dependencies as parameters in the constructor. e.g constructor(private userService: UserService)
- * Declare class properties for injected dependencies. e.g import { UserService } from 'src/services/user.service';

Template and Styles

- Use a separate template and styles file for each component.
- Name the template file with the component name, followed by .component.html.
- Name the styles file with the component name, followed by .component.scss.
- Use descriptive class names in styles.
- Avoid using inline styles unless necessary.

Methods

- Use descriptive method names that reflect their purpose.
- Keep methods short and focused on a single task.
- Use comments to explain complex logic or provide additional context. e.g //handle request abort error

Error Handling

- Use try-catch blocks to handle synchronous errors.
- Use async/await or toPromise() for handling asynchronous operations.
- Handle errors appropriately and provide meaningful error messages.
- Log errors to the console for debugging purposes.

Testing

- Use a separate file for tests related to a component or service.
- Name test files with the component or service name, followed by .spec.ts.
- Write meaningful test descriptions that reflect the test's purpose.
- Organize tests using describe blocks.
- Use beforeEach and afterEach hooks to set up and clean up test data.
- Use descriptive variable and function names in tests.
- Test both positive and negative scenarios.

API Routes

- Use descriptive names for API routes.
- Write tests for each API route to ensure proper functionality.
- Handle different response scenarios (success, errors) and assertions accordingly.

Authentication and Authorization

- Define guidelines for implementing authentication and authorization mechanisms.
- Ensure sensitive user information (e.g., passwords, tokens) is handled securely.
- Follow security best practices for token-based authentication (e.g., JWT).

Version Control and Collaboration

- Define guidelines for using version control systems (e.g., Git) and branching strategies.
- Establish code review practices to ensure code quality and share knowledge among team members.
- Encourage the use of meaningful commit messages and adhere to commit message conventions.

Miscellaneous

- Use consistent indentation and follow the chosen coding style guide (e.g., 2 spaces, 4 spaces, or tabs).
- Write code with readability and maintainability in mind.
- Use meaningful variable and function names to enhance code understanding.
- Follow best practices and guidelines provided by the Angular framework.