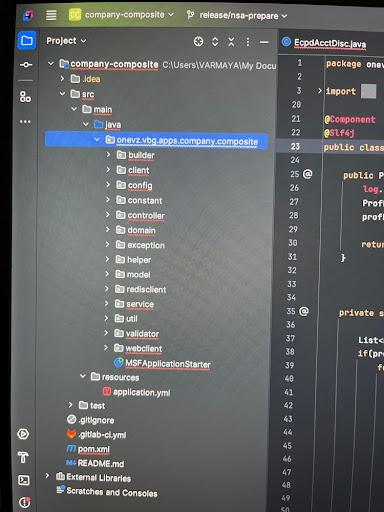
**VBG project code analysis**



**Builder**:

**Overall Functionality:**

The classes in builder for example AccountServiceBuilder acts as a utility class to construct specific response objects based on the input data. It likely serves as a bridge between different services or layers within the application, ensuring that the data is formatted and transformed correctly for further processing or consumption.

**Possible Use Cases:**

* **API Response Formatting:** This class might be used to format responses for API calls, ensuring that the data is structured correctly according to the API specification.
* **Data Transformation:** It could be used to transform data from one format to another, such as converting raw data from a database into a specific response format.
* **Error Handling:** The class handles error scenarios by setting appropriate error messages in the response.

**Client:**

**Purpose:**

The classes in client for example shoppingCarCoreClient serves as a client for interacting with a shopping cart service. It provides methods to retrieve and save cart information, likely using a remote API or service.

The presence of model classes within this package is a common practice in software development, especially when dealing with remote APIs or services. These model classes serve as data transfer objects (DTOs) to represent the structure and content of data exchanged between the client and the server

**Config:**

**Ex:**

The RolePermissionConfiguration class, located in the config folder, is likely used to configure role-based permissions within your application. This is a common practice in enterprise applications to control access to specific features or functionalities based on user roles.

Here's a breakdown of its purpose and how it might be used:

**Purpose:**

* **Centralized Configuration:** This class acts as a centralized location for defining role-based permissions. This makes it easier to manage and modify permissions as needed.
* **Role-Based Access Control (RBAC):** It implements a basic form of RBAC by defining lists of permitted and not-permitted actions for each role.

**How it works:**

1. **Configuration Properties:** The class likely loads configuration properties from a configuration file (e.g., application.properties or a YAML file) or from environment variables.
2. **Role Permissions Mapping:** The configuration properties define mappings between roles and their corresponding permitted and not-permitted actions.
3. **Permission Enforcement:** During runtime, the application can consult this configuration to determine if a user with a specific role is allowed to perform a certain action. This can be implemented using filters, interceptors, or custom annotations.

Like that each config class have a separate use.

The SecurityContextTaskDecorator class is likely designed to ensure that the current security context (e.g., user credentials, roles, permissions) is propagated to asynchronous tasks or threads. This is crucial in multi-threaded applications where the security context of the original thread might not be automatically inherited by child threads

By using SecurityContextTaskDecorator, you can effectively manage security contexts in multi-threaded applications, ensuring that asynchronous operations are executed with the appropriate permissions.

The TaskExecutor class is responsible for managing the execution of asynchronous tasks in your application. It provides a flexible way to handle tasks concurrently, improving performance and responsiveness.

Here's a breakdown of its role and the code you provided:

**Purpose:**

* **Asynchronous Task Execution:** Handles the scheduling and execution of tasks in a non-blocking manner.
* **Thread Pool Management:** Manages a pool of threads to efficiently execute tasks.
* **Security Context Propagation:** Ensures that the security context (e.g., user credentials, roles) is propagated to asynchronous tasks.

**Domain:**

The classes in the domain for example duns are likely used as a data transfer object (DTO) to encapsulate the necessary information required to create an alternative DUNS number. It might be used in conjunction with a service or controller class that handles the actual creation process.

EX:

The DunsBuilder class seems to be responsible for constructing GchDunsRequest and GchCreateAltDunsRequest objects. These objects are likely used to interact with a DUNS service to retrieve or create DUNS numbers.

**Purpose:**

The DunsBuilder class likely serves as a utility class to simplify the construction of GchDunsRequest, GchCreateAltDunsRequest, and DunsResponse objects. By using this builder class, you can avoid manual object creation and reduce the potential for errors.

**How it might be used:**

1. **In a Service Layer:** A service layer might use the DunsBuilder to construct the appropriate request object based on the input data.
2. **In a Controller Layer:** A controller might use the DunsBuilder to create response objects to send back to the client.

Note: Like this, the classes in domain are used the builder to map the values to be in sync with the api parameters etc. They are like mapper classes.

**Helper:**

The helper classes are like utility classes.

Ex: The ShareByteHelper class appears to be a utility class that handles tasks related to encryption, decryption, and generating unique IDs

The ShareByteHelper class seems to be a centralized location for common utility functions related to encryption, decryption, and ID generation. It simplifies these tasks by providing reusable methods and encapsulating the underlying complexity.

Like that each class has some specific functionality and use.

**Utility:**

**Ex:**

The CompanyCompositeUtil class appears to be a utility class that provides various helper methods related to security, user information retrieval, and exception handling.

**In essence,** the CompanyCompositeUtil class enhances the overall functionality and maintainability of the application by providing essential utility methods for security, user information, and error handling.

Like this each utility class has different helper methods and uses.

**Web Client:**

The CompanyCoreClient interface appears to be a web client interface that defines methods for interacting with a Company Core service. This service seems to provide various functionalities related to ECP (Electronic Commerce Platform) data, such as searching profiles, retrieving account information, and fetching equipment details.

**Key Points:**

* **Web Client Interface:** It defines a set of methods for interacting with the Company Core service using HTTP requests.
* **Service Endpoints:** The interface exposes methods for various endpoints, including searching profiles by tax ID or federal tax ID, retrieving profile information by ECP ID, fetching deposit waiver count, equipment term information, account rules, SPOC information, hot remarks, and account discounts.
* **Request and Response Objects:** The methods likely use request and response objects to encapsulate the data being sent to and received from the service.
* **Error Handling:** The methods might throw exceptions to indicate errors during the API calls.