SASVA Project Scan Report

Git URL: https://github.com/vishalgoyal16444/microservices-architecture | Branch: master

Specification:

Project-description: The DroneDelivery project is a comprehensive system designed to manage and facilitate drone deliveries. It leverages cloud computing services, specifically Azure, to deploy and run its components. The application is built using ASP.NET Core, providing a robust backend for handling delivery requests, processing packages, and scheduling drones. The system includes multiple components such as web applications, function apps, and storage accounts, all orchestrated through an ARM template to ensure consistent deployment. The primary goal of the project is to automate and streamline the process of managing deliveries via drones, from request handling to package processing and drone scheduling. The project uses dependency injection for service management, ensuring modularity and ease of testing. It also incorporates logging and monitoring through Application Insights, allowing for effective tracking of operations and performance. The use of Azure Functions demonstrates a serverless approach to handling specific tasks, enhancing scalability and efficiency. Overall, the DroneDelivery project aims to provide a seamless and efficient solution for drone-based delivery services, leveraging modern technologies to achieve high reliability and performance.

Features:

- **Feature:** Delivery Management
  Feature-details: Manages delivery-related operations through a dedicated API controller, allowing retrieval of delivery details and statuses.
- **Feature:** Delivery Request Processing
  Feature-details: Handles HTTP POST requests to process delivery requests, including logging and route creation for new deliveries.
- **Feature:** Package Creation
  Feature-details: Creates packages by sending HTTP POST requests to an external service, ensuring successful network operations.
- **Feature:** Drone Scheduling
  Feature-details: Simulates accessing a data store to retrieve drone IDs for delivery requests, ensuring efficient drone assignment.
- **Feature:** Azure Function for Package Updates
  Feature-details: Handles package updates via an Azure Function triggered by HTTP PUT requests, utilizing serverless computing capabilities.
- **Feature:** Consistent Deployment
  Feature-details: Uses an ARM template to automate the deployment of web and function apps in Azure, ensuring repeatable and reliable resource creation.

Technical-specification:

- **Platform/Technologies:** Azure, ASP.NET Core, Azure Functions
  Programming Languages: C#
  API Endpoints: api/deliveries for delivery requests, external service endpoint for package creation
  State Management: Dependency injection for service management
  Data Flow: Delivery requests flow through controllers to services, which interact with external APIs and Azure Functions
  Data Persistence: Simulated with utility functions; potential for integration with SQL Azure
  Integration: Azure Function for serverless operations, external package service API
  Validation: Handled within controllers and services, with logging for error tracking
  Routing: ASP.NET Core routing for API endpoints
  Security: Managed through Azure configurations and best practices
  Third-party Components: None specified
  Other Tools and Technologies: Application Insights for monitoring, HttpClient for network operations

File-groups:

- **Filegroup name:** Backend
  Summary: Contains the core logic and services for managing deliveries and processing requests.
  Files: Controllers/DeliveriesController.cs, Controllers/DeliveryRequestsController.cs, Services/PackageServiceCaller.cs, Services/DeliveryRepository.cs, Services/DroneScheduler.cs, Services/RequestProcessor.cs
- **Filegroup name:** Configuration
  Summary: Holds configuration files for application settings and deployment.
  Files: appsettings.Development.json, appsettings.json, deployment/azuredeploy.json
- **Filegroup name:** Entry Point
  Summary: Contains the entry point for the application startup.
  Files: Program.cs, Startup.cs
- **Filegroup name:** Azure Functions
  Summary: Contains the Azure Function for handling package updates.
  Files: PackageService/PackageServiceFunction.cs, PackageService/host.json
- **Filegroup name:** Interfaces
  Summary: Defines interfaces for service contracts.
  Files: Services/IDeliveryRepository.cs, Services/IDroneScheduler.cs, Services/IPackageProcessor.cs, Services/IRequestProcessor.cs

Project-entrypoint:

- **Program.cs:** The global entry point for the ASP.NET Core application, responsible for initializing and running the web server.
- **Startup.cs:** Configures services and the HTTP request pipeline, essential for setting up the application.

Folder-analysis:

- **Folder name:** src
  Summary: The 'src' folder contains the core source code for the DroneDelivery project, organizing the application's main logic, configuration, and startup scripts.
  Purpose: The 'src' folder is the main directory for the DroneDelivery project's source code, housing critical components for application execution and configuration. It includes sub-folders like 'before' and 'after', each playing a role in organizing code for different stages of the application's lifecycle. This folder supports scalability and maintainability by separating configuration, startup logic, and core business logic into distinct areas. It interacts with other components through HTTP requests processed by controllers and services, and integrates with external tools like Application Insights for monitoring. The folder contains configuration files, C# scripts for application startup, and folders for services, controllers, and connected services. Data flows through this folder starting from the entry point in Program.cs,

through configuration in Startup.cs, and into various services and controllers that handle business logic and data processing.

Folder-features:

- **Feature:** Application Configuration
  Feature-details: Manages application settings and environment configuration through files like appsettings.json.
- **Feature:** Startup Logic
  Feature-details: Defines how the application initializes and configures services, including middleware and routing.
- **Feature:** Core Business Logic
  Feature-details: Contains the main logic for processing HTTP requests and executing business rules.

Folder-technical-specification:

- **Platform/Technologies:** ASP.NET Core
  Programming Languages: C#
  API Endpoints: Defined in controllers to handle HTTP requests
  State Management: Managed through services and dependency injection
  Data Flow: Data flows from Program.cs to Startup.cs, then through services and controllers
  Data Persistence: Interacts with databases or external APIs for data storage and retrieval
  Integration: Integrates with external monitoring tools like Application Insights
  Validation: Input validation is handled within controllers and services
  Routing: Configured in Startup.cs for handling HTTP requests
  Security: May include authentication and authorization mechanisms

Category: backend
Folder-tags:
src,services,controllers,data,clients,monitoring,azure,dto,config,utils,logging,serialization,metrics,models,middleware,integration,cloud,repositories,api,telemetry,alerts
Defect-criteria: Defects related to this folder can be identified by checking for issues in application startup, configuration errors, service initialization failures, and incorrect routing or API endpoint handling.
Sub-folders:

- **Folder name:** src/before
  Summary: The 'src/before' folder serves as the main directory for the DroneDelivery application, containing essential components for application configuration, startup, and core business logic services.
  Purpose: The 'src/before' folder is critical to the DroneDelivery project's architecture, providing the main structure for the application. It contains configuration files, the entry point of the application, and core business logic services. This folder organizes the application into distinct areas such as configuration (appsettings.json), startup logic (Startup.cs), and the main program execution (Program.cs). This organization supports scalability and maintainability by clearly separating concerns. The folder interacts with other components via HTTP requests processed by controllers and services, and it integrates with external tools like Application Insights for monitoring. Data flows through this folder starting from the entry point in Program.cs, through configuration in Startup.cs, and into various services and controllers that handle business logic and data processing.
  Folder-features:
  - **Feature:** Configuration Management
    Feature-details: Manages application settings and configurations through appsettings.json, allowing for dynamic configuration adjustments without code changes.
  - **Feature:** Application Startup
    Feature-details: Handles application initialization and dependency injection setup through Startup.cs, ensuring all services are properly configured and available.
  - **Feature:** Main Program Execution
    Feature-details: Defines the entry point of the application in Program.cs, setting up the host and starting the application.
  - **Feature:** HTTP Request Processing
    Feature-details: Processes HTTP requests via controllers and services, enabling interaction with clients and other services.
  - **Feature:** Monitoring Integration
    Feature-details: Integrates with Application Insights for monitoring and logging, providing insights into application performance and errors.

  Folder-technical-specification:
  - **Platform/Technologies:** ASP.NET Core, C#
    Programming Languages: C#
    API Endpoints: Defined in controllers to handle HTTP requests and responses.
    State Management: Managed through dependency injection and service lifetimes.
    Data Flow: Data flows from Program.cs through Startup.cs into services and controllers.
    Data Persistence: Handled via repositories in DroneDelivery.Common.
    Integration: Integrates with external services via HTTP clients and Application Insights.
    Validation: Data validation is likely handled in controllers or services.
    Routing: Configured in Startup.cs to map endpoints to controllers.
    Security: Security configurations would typically be handled in Startup.cs.
    Third-party Components: Application Insights for monitoring.

  Category: backend
  Folder-tags: dto,config,src,services,controllers,repositories,utils,data,models,metrics,api,logging,telemetry,alerts,monitoring,serialization
  Defect-criteria: Defects related to this folder can be identified by monitoring application startup errors, configuration issues, HTTP request failures, and integration problems with Application Insights. Logs and exception handling mechanisms should be reviewed to detect and diagnose issues.
  Sub-folders:
  - **Folder name:** src/before/DroneDelivery.Common
    Summary: This folder contains the common services and models used in the drone delivery system.
    Purpose: The 'DroneDelivery.Common' folder plays a crucial role in the project by encapsulating the core business logic and data structures needed for the drone delivery application. It contributes to the project's modularity and scalability by separating concerns into services and models. This separation allows for easier maintenance and potential future expansion. The folder fits into the system's functionality by providing essential services for processing delivery requests and managing data flow, interacting with other components like data repositories and external services. It includes interfaces and implementations for services such as IDeliveryRepository, IRequestProcessor, IDroneScheduler, and IPackageProcessor, as

well as data models representing user accounts, delivery details, and package information. The data flows through this folder as delivery requests are processed, drones are scheduled, and delivery data is accessed and modified.

Folder-features:

- **Feature:** Service Interfaces and Implementations
  Feature-details: Defines and implements core business logic for processing delivery requests and scheduling drones.
- **Feature:** Data Models
  Feature-details: Contains models representing user accounts, delivery details, package information, and enums for confirmation types.

Folder-technical-specification:

- **Platform/Technologies:** C#
  Programming Languages: C#
  Data Flow: Data flows through services and models to process delivery requests and manage delivery operations.
  Integration: Interacts with data repositories and external services to manage deliveries.

Category: backend
Folder-tags: dto,src,services,repositories,utils,data,models,serialization
Defect-criteria: Defects related to this folder can be identified by testing the functionality of delivery processing and drone scheduling. Issues with data integrity or incorrect business logic implementation can also indicate defects. Additionally, integration tests with data repositories and external services can help identify defects in interactions.
Sub-folders:

- **Folder name:** src/before/DroneDelivery.Common/Services
  Summary: This folder contains the core service implementations and interfaces for the drone delivery system.
  Purpose: The 'Services' folder in the DroneDelivery.Common namespace is integral to the drone delivery system's architecture. It houses interfaces and their implementations that define and execute the core business logic of the application, such as processing delivery requests, scheduling drones, and managing delivery data. This folder contributes to the overall structure by encapsulating the logic needed for operations, ensuring that the system is modular and scalable. It fits into the system's functionality by interacting with other components like data repositories and external services, providing a clear separation of concerns. The folder includes interfaces like IDeliveryRepository, IRequestProcessor, IDroneScheduler, and IPackageProcessor, along with their respective implementations, which handle tasks such as scheduling drones and processing delivery requests. Data flows through this folder as delivery requests are processed, drones are scheduled, and delivery data is accessed and modified.
  Folder-features:

  - **Feature:** Delivery Management
    Feature-details: Defines and implements interfaces for managing delivery data and operations, crucial for scheduling and retrieving deliveries.
  - **Feature:** Drone Scheduling
    Feature-details: Implements drone scheduling logic to assign drones to delivery requests, ensuring efficient resource allocation.
  - **Feature:** Request Processing
    Feature-details: Processes delivery requests through a structured workflow, handling exceptions and logging for robust operation.
  - **Feature:** Package Creation
    Feature-details: Provides interfaces and implementations for creating package objects asynchronously, crucial for non-blocking operations.
  - **Feature:** Utility Operations
    Feature-details: Includes utility functions for computational tasks such as permutation calculations, supporting various operations within the system.

  Folder-technical-specification:

  - **Platform/Technologies:** .NET Core
    Programming Languages: C#
    Data Flow: Asynchronous operations using Tasks for non-blocking execution.
    Integration: Dependency injection for obtaining service instances, ensuring modular and testable code.
    Validation: Interfaces provide contracts for implementing validation and processing logic.
    Security: Basic exception handling and logging for error management.

  Category: backend
  Folder-tags: utils,src,services,repositories
  Defect-criteria: Defects can be identified through unit testing of implemented interfaces, checking for unhandled exceptions, and verifying the correct execution of asynchronous operations. Code reviews and static analysis can also help in identifying potential issues in logic and integration.
  Sub-folders:

  Folder depth: 0
  Files: src/before/DroneDelivery.Common/Services/IDeliveryRepository.cs, src/before/DroneDelivery.Common/Services/DeliveryRepository.cs, src/before/DroneDelivery.Common/Services/Utility.cs, src/before/DroneDelivery.Common/Services/IRequestProcessor.cs, src/before/DroneDelivery.Common/Services/IDroneScheduler.cs, src/before/DroneDelivery.Common/Services/DroneScheduler.cs, src/before/DroneDelivery.Common/Services/RequestProcessor.cs, src/before/DroneDelivery.Common/Services/IPackageProcessor.cs
  Level: 3
- **Folder name:** src/before/DroneDelivery.Common/Models
  Summary: This folder contains all the model classes and enumerations used in the drone delivery system.
  Purpose: The 'Models' folder is a critical component of the drone delivery project, housing the data models and enumerations that define the structure of the application's core entities. It serves as the backbone for representing and managing data throughout the system, ensuring consistency and integrity. This folder organizes the data models that represent various aspects of the drone delivery process, such as user accounts, delivery details, package information, and confirmation types. By centralizing these models, the folder facilitates data interchange between different components, ensuring that the application can efficiently process and manage delivery operations. The

organization of this folder supports future scalability by providing a clear structure for adding new models or updating existing ones as the system evolves. The models interact with other parts of the project, such as services and APIs, to enable seamless data flow and processing. The folder's structure ensures that data is consistently serialized and deserialized, maintaining data integrity across the application.

Folder-features:

- **Feature:** User Account Management
  Feature-details: The UserAccount class models user account information, providing properties for UserId and AccountId, essential for managing user-related operations.
- **Feature:** Package and Delivery Data
  Feature-details: Classes like PackageGen, PackageInfo, and Delivery model the core data entities for packages and deliveries, including properties for size, weight, locations, and status.
- **Feature:** Delivery Tracking
  Feature-details: The DeliveryStatus class allows the system to track and manage the state of deliveries, with properties for stage, location, and estimated times.
- **Feature:** Confirmation and Security
  Feature-details: Enumerations like ConfirmationRequired and ConfirmationType define various methods of delivery confirmation, enhancing security and flexibility in the delivery process.
- **Feature:** Standardized Data Representation
  Feature-details: Enumerations such as ContainerSize and DeliveryStage provide standardized representations for container sizes and delivery stages, ensuring consistency across the application.

Folder-technical-specification:

- **Platform/Technologies:** .NET Framework, C#
  Programming Languages: C#
  Data Format: JSON (using Newtonsoft.Json for serialization)
  Data Flow: Data flows through these models as they are used to serialize and deserialize JSON data, facilitating communication between different components.
  Integration: These models integrate with other parts of the application, such as services and APIs, to enable data exchange and processing.
  Validation: The use of enumerations and read-only properties ensures data integrity and validation at the model level.
  Security: The ConfirmationType and ConfirmationRequired enums enhance security by defining and managing delivery confirmation methods.

Category: models
Folder-tags: dto,src,data,models,serialization
Defect-criteria: Defects related to this folder can be identified by checking for issues in data serialization/deserialization, ensuring that all properties are correctly mapped and utilized. Validation of data integrity and consistency across the application can also highlight potential defects. Additionally, ensuring that enumerations cover all necessary cases and are correctly implemented can prevent logical errors.
Sub-folders:

Folder depth: 0
Files: src/before/DroneDelivery.Common/Models/ConfirmationRequired.cs, src/before/DroneDelivery.Common/Models/Location.cs, src/before/DroneDelivery.Common/Models/UserAccount.cs, src/before/DroneDelivery.Common/Models/PackageGen.cs, src/before/DroneDelivery.Common/Models/DeliveryStatus.cs, src/before/DroneDelivery.Common/Models/PackageInfo.cs, src/before/DroneDelivery.Common/Models/ContainerSize.cs, src/before/DroneDelivery.Common/Models/ConfirmationType.cs, src/before/DroneDelivery.Common/Models/Delivery.cs, src/before/DroneDelivery.Common/Models/DeliveryStage.cs
Level: 3

Folder depth: 1
Files:
Level: 2

- **Folder name:** src/before/DroneDelivery-before
  Summary: This folder contains the core components of the DroneDelivery application, including configuration, entry point, and business logic.
  Purpose: The 'src/before/DroneDelivery-before' folder serves as the main directory for the DroneDelivery application. It encompasses essential components like configuration files, the entry point of the application, and core business logic services. This folder is central to the project's architecture, organizing the application into distinct areas such as configuration (appsettings.json), startup logic (Startup.cs), and the main program execution (Program.cs). This organization supports scalability and maintainability by clearly separating concerns. The folder interacts with other components via HTTP requests processed by controllers and services, and it integrates with external tools like Application Insights for monitoring. Data flows through this folder starting from the entry point in Program.cs, through configuration in Startup.cs, and into various services and controllers that handle business logic and data processing.
  Folder-features:
  - **Feature:** Configuration Management
    Feature-details: Handles application settings through appsettings.json and environment-specific configurations.
  - **Feature:** Dependency Injection Setup
    Feature-details: Configures services and dependencies in Startup.cs for use throughout the application.
  - **Feature:** HTTP Request Handling
    Feature-details: Defines controller routes and processes incoming HTTP requests to deliver responses.
  - **Feature:** Business Logic Processing
    Feature-details: Encapsulates business logic in services to handle package processing and delivery scheduling.
  - **Feature:** Telemetry and Monitoring
    Feature-details: Integrates with Application Insights to collect and analyze performance data.

  Folder-technical-specification:
  - **Platform/Technologies:** ASP.NET Core, C#

Programming Languages: C#
API Endpoints: Defined in Controllers for delivery operations
State Management: Managed through Dependency Injection
Data Flow: Data flows from controllers to services and repositories
Data Persistence: Handled by repositories (not detailed in provided files)
Integration: Integrates with Application Insights for monitoring
Validation: Implemented in controllers and services
Routing: Configured in Startup.cs using default controller route
Security: HTTPS redirection and HSTS configured in Startup.cs
Third-party Components: Application Insights for telemetry

Category: backend
Folder-tags: config,src,services,controllers,utils,metrics,api,logging,telemetry,alerts,monitoring
Defect-criteria: Defects in this folder can be identified through issues in application startup, incorrect configurations, failures in dependency injection, HTTP request handling errors, and telemetry data discrepancies. Monitoring logs and debugging can help pinpoint issues in these areas.
Sub-folders:

- **Folder name:** src/before/DroneDelivery-before/Controllers
  Summary: This folder contains controller classes that define the API endpoints for handling delivery-related operations in the ASP.NET Core application.
  Purpose: The Controllers folder plays a crucial role in the ASP.NET Core application by defining the endpoints for handling HTTP requests related to deliveries. It provides the interface between the client and server, processing incoming requests, interacting with services, and returning responses. The folder contributes to the overall structure by organizing the API logic, ensuring a clear separation of concerns. It contains classes like DeliveryRequestsController, DeliveriesController, and HomeController, each responsible for specific HTTP operations. These controllers interact with services such as IRequestProcessor and repositories to handle business logic and data retrieval. The folder's organization supports scalability by allowing easy addition of new endpoints and modifications to existing ones. It fits into the system's functionality by enabling client applications to request delivery operations and receive structured responses. Data flows into this folder through HTTP requests, which are processed and transformed into appropriate responses.
  Folder-features:
  - **Feature:** Delivery Request Handling
    Feature-details: Processes HTTP POST requests to create new delivery requests and returns the created resource's route.
  - **Feature:** Delivery Retrieval
    Feature-details: Handles HTTP GET requests to retrieve delivery details by ID and fetch the status of a delivery.
  - **Feature:** Home Page Interaction
    Feature-details: Manages requests related to the home page and simulates sending multiple delivery requests.

  Folder-technical-specification:
  - **Platform/Technologies:** ASP.NET Core
    Programming Languages: C#
    API Endpoints: /api/deliveries, /api/DeliveryRequests
    State Management: Stateless HTTP requests
    Data Flow: Incoming HTTP requests are processed and transformed into responses
    Data Persistence: Interacts with repositories for data retrieval
    Integration: Uses dependency injection for service integration
    Validation: Basic validation through method parameters
    Routing: Attribute routing for API endpoints
    Security: Relies on ASP.NET Core security features
    Third-party Components: ILogger for logging, IHttpClientFactory for HTTP clients

  Category: backend
  Folder-tags: src,services,controllers,api,logging
  Defect-criteria: Defects related to this folder can be identified by testing the API endpoints for correct functionality, checking logs for errors or exceptions, and ensuring that the controllers interact correctly with services and repositories.
  Sub-folders:

  Folder depth: 0
  Files: src/before/DroneDelivery-before/Controllers/DeliveryRequestsController.cs, src/before/DroneDelivery-before/Controllers/DeliveriesController.cs, src/before/DroneDelivery-before/Controllers/HomeController.cs
  Level: 3
- **Folder name:** src/before/DroneDelivery-before/Connected Services
  Summary: This folder is dedicated to the configuration and integration of Microsoft Application Insights for the project.
  Purpose: The 'Connected Services' folder plays a crucial role in enabling telemetry and monitoring capabilities for the project by integrating Microsoft Application Insights. It facilitates the collection of performance data, which is vital for diagnosing issues and understanding usage patterns. This folder's organization is significant for maintaining application reliability and performance. It primarily handles configuration tasks and ensures that telemetry data is correctly captured and sent to Application Insights. Within this folder, configuration files, such as ConnectedService.json, are used to establish and manage the connection with Application Insights. This folder interacts with other parts of the project by sending runtime telemetry data to Application Insights, where it is processed and visualized for further analysis.
  Folder-features:
  - **Feature:** Telemetry Data Collection
    Feature-details: Configures the project to collect telemetry data, including performance metrics and usage patterns, through Application Insights.
  - **Feature:** Monitoring and Diagnostics
    Feature-details: Enables monitoring of application health and diagnostics by integrating with Application Insights.

  Folder-technical-specification:

  - **Platform/Technologies:** Microsoft Application Insights
    Data Flow: Telemetry data is collected from the application and sent to Application Insights for processing and visualization.
    Integration: ConnectedService.json is used to configure the connection with Application Insights.
    Validation: Ensures that telemetry data is correctly captured and transmitted.

  Category: config
  Folder-tags: config,metrics,telemetry,alerts,monitoring
  Defect-criteria: Defects related to this folder can be identified by checking for misconfigurations in the ConnectedService.json file, ensuring that telemetry data is being correctly captured, transmitted to Application Insights, and visualized without errors.
  Sub-folders:
  - **Folder name:** src/before/DroneDelivery-before/Connected Services/Application Insights
    Summary: This folder contains configuration files necessary for integrating Microsoft Application Insights into the project.
    Purpose: The 'Connected Services/Application Insights' folder serves as a configuration hub for integrating Microsoft Application Insights with the project. Its primary role is to facilitate the monitoring and analysis of application performance by enabling telemetry data collection. This folder is crucial for the project's observability, allowing developers to track application health, usage patterns, and diagnose issues efficiently. It contributes to the overall structure by ensuring that performance metrics are captured and sent to Application Insights for further analysis. The folder's significance lies in its ability to enhance the project's monitoring capabilities, which is essential for maintaining application reliability and performance. Within this folder, the ConnectedService.json file is implemented, which contains configuration settings required to establish a connection with Application Insights. This folder interacts with other parts of the project by sending telemetry data generated during application runtime to the Application Insights service, where it is processed and visualized.
    Folder-features:
    - **Feature:** Telemetry Configuration
      Feature-details: Configures the application to send telemetry data to Microsoft Application Insights, enabling performance monitoring and diagnostics.
    - **Feature:** Service Provider Identification
      Feature-details: Includes provider ID for identifying the Application Insights service provider.
    - **Feature:** Documentation URI
      Feature-details: Contains a URI pointing to the Application Insights getting started documentation to guide developers.

    Folder-technical-specification:
    - **Platform/Technologies:** Microsoft Application Insights
      Configuration Format: JSON
      Integration: The application integrates with Application Insights through the configuration settings provided in ConnectedService.json.
      Data Flow: Telemetry data flows from the application to Application Insights for monitoring and analysis.
      Security: Ensures secure communication between the application and Application Insights using the configuration settings.
      Monitoring: Facilitates real-time monitoring of application performance and user behavior.

    Category: config
    Folder-tags: monitoring,config,alerts,metrics
    Defect-criteria: Defects in this folder can be identified by checking the accuracy and completeness of the configuration settings in ConnectedService.json. Issues may manifest as failures in telemetry data being sent to Application Insights or incorrect monitoring metrics. Logs and error messages should be reviewed to diagnose configuration-related problems.
    Sub-folders:

    Folder depth: 0
    Files: src/before/DroneDelivery-before/Connected Services/Application Insights/ConnectedService.json
    Level: 4

  Folder depth: 1
  Files:
  Level: 3
- **Folder name:** src/before/DroneDelivery-before/Services
  Summary: This folder contains service classes responsible for handling core business logic related to drone delivery operations.
  Purpose: The 'Services' folder plays a crucial role in the project by encapsulating the business logic necessary for drone delivery operations. It contributes to the overall structure by providing a dedicated place for service classes that manage tasks such as package processing. This organization enhances maintainability and scalability by separating business logic from other layers like data access or presentation. The folder's significance lies in its ability to streamline workflow by centralizing logic that can be reused across different components. Within the system's functionality, this folder interacts with other parts by providing services that can be called by controllers or other components needing business logic execution. The folder contains classes like PackageProcessor, which implement interfaces and interact with utilities to perform their tasks. Data flows through this folder as input objects are processed by these services, which then produce output objects or perform actions.
  Folder-features:
  - **Feature:** Package Processing
    Feature-details: Implements logic for creating and managing packages in the drone delivery system, specifically through the PackageProcessor class.

  Folder-technical-specification:
  - **Platform/Technologies:** C#
    Programming Languages: C#
    Data Flow: Data flows into the service classes as input objects and is processed to produce output objects or perform specific actions.
    Integration: Interacts with utility classes for performing work simulations and possibly other service classes within the

application.
State Management: Manages state through objects passed to and from service methods.
Data Persistence: Not directly handled in this folder, but service methods may interact with repositories or data access layers elsewhere in the application.
Validation: Likely performed within service methods to ensure data integrity before processing.

Category: services
Folder-tags: utils,src,services
Defect-criteria: Defects can be identified by testing the service methods for expected output given specific inputs and ensuring that all business logic is correctly implemented. Unit tests should be written to cover all possible scenarios and edge cases. Additionally, integration tests can verify that these services interact correctly with other components.
Sub-folders:

Folder depth: 0
Files: src/before/DroneDelivery-before/Services/PackageProcessor.cs
Level: 3

Folder depth: 2
Files: src/before/DroneDelivery-before/appsettings.json, src/before/DroneDelivery-before/Startup.cs, src/before/DroneDelivery-before/Program.cs, src/before/DroneDelivery-before/appsettings.Development.json
Level: 2

Folder depth: 3
Files:
Level: 1

- **Folder name:** src/after
  Summary: This folder serves as the main directory for the DroneDelivery application, organizing core functionalities and configurations.
  Purpose: The 'src/after' folder encapsulates the main application logic, configuration, and startup scripts for the DroneDelivery system. It plays a crucial role in setting up the ASP.NET Core environment, defining the application's behavior, and organizing essential components such as configuration files, startup scripts, and application logic. This folder contributes to the overall structure by providing a cohesive and maintainable codebase that supports future scalability. It fits into the system's functionality by defining how the application initializes, configures services, and handles HTTP requests, interacting with other components like services, controllers, and external APIs. The folder contains various types of files, including JSON configuration files, C# scripts for application startup and configuration, and folders for services, controllers, and connected services. It interacts with other parts of the project by setting up the environment for the application's operation, configuring service dependencies, and defining routing and middleware for handling requests. Data flows through this folder as it processes configuration settings, initializes services, and routes HTTP requests to appropriate controllers for further processing.
  Folder-features:
  - **Feature:** Application Initialization
    Feature-details: Defines how the ASP.NET Core application initializes and configures its environment.
  - **Feature:** Service Configuration
    Feature-details: Handles the registration and configuration of services required by the application.
  - **Feature:** Request Handling
    Feature-details: Manages HTTP request routing and processing through middleware and controllers.
  - **Feature:** Configuration Management
    Feature-details: Utilizes JSON files to manage application settings and configurations.
  - **Feature:** Scalability Support
    Feature-details: Organizes code in a way that supports future scalability and maintainability.

  Folder-technical-specification:
  - **Platform/Technologies:** ASP.NET Core, C#
    Programming Languages: C#
    API Endpoints: Defined through controllers to handle HTTP requests.
    State Management: Managed through service configurations and middleware.
    Data Flow: HTTP requests are routed to controllers, processed, and responses are returned.
    Data Persistence: Interacts with databases as needed for data storage and retrieval.
    Integration: Integrates with external APIs and services as required.
    Validation: Input validation is performed at the controller level.
    Routing: Configured through middleware to direct requests to appropriate controllers.
    Security: Handled through ASP.NET Core security features and middleware.
    Third-party Components: May include packages from NuGet for additional functionality.

  Category: backend
  Folder-tags: dto,config,src,services,controllers,cloud,repositories,utils,integration,clients,api,models,metrics,telemetry,middleware,alerts,monitoring,azure
  Defect-criteria: Defects in this folder can be identified through failed application startup, incorrect service configurations, routing errors, or improper handling of HTTP requests. Unit and integration tests can help catch defects related to these areas.
  Sub-folders:
  - **Folder name:** src/after/DroneDelivery.Common
    Summary: This folder contains core components of the drone delivery system, including services and models essential for its operation.
    Purpose: The 'src/after/DroneDelivery.Common' folder serves as the backbone of the drone delivery system, containing essential service classes and data models. It organizes the business logic related to drone scheduling, delivery processing, package creation, and repository management. By encapsulating these responsibilities, the folder ensures a clean separation of concerns, promoting scalability and maintainability. The folder's significance lies in its role in handling delivery requests, scheduling drones, and processing packages, all crucial for the project's workflow and functionality. It interacts with other components such as data stores and logging systems, facilitating communication and data flow across the system. The folder contains classes that define and implement core functionalities and data models that represent entities like UserAccount, Package, Delivery, and Location. These models provide structured data representations for business logic and service layers, often serialized to and from JSON for external communication.

Folder-features:

- **Feature:** Drone Scheduling
  Feature-details: Implements logic for scheduling drones for deliveries, ensuring optimal resource utilization.
- **Feature:** Delivery Processing
  Feature-details: Handles the workflow for processing delivery requests and managing the lifecycle of a delivery.
- **Feature:** Package Creation
  Feature-details: Provides functionality to create and manage package information within the system.
- **Feature:** Repository Management
  Feature-details: Manages data storage and retrieval operations, interacting with data stores to persist and access data.
- **Feature:** Data Modeling
  Feature-details: Defines data models for entities like UserAccount, Package, Delivery, and Location, facilitating structured data handling.

Folder-technical-specification:

- **Platform/Technologies:** C#, .NET
  Programming Languages: C#
  Data Flow: Data flows through service classes and models, often serialized to/from JSON for communication.
  Data Persistence: Interacts with data stores for data persistence, likely using repositories.
  Integration: Integrates with external systems via APIs for data exchange and service interactions.
  Validation: Includes validation logic within service classes to ensure data integrity.
  Security: Implements security measures for data access and service interactions.
  Third-party components: May use libraries for JSON serialization/deserialization and logging.

Category: backend
Folder-tags: dto,src,services,repositories,utils,models,middleware
Defect-criteria: Defects can be identified by reviewing service class logic for errors, ensuring data models align with requirements, and verifying integration points with external systems for correct data exchange.
Sub-folders:

- **Folder name:** src/after/DroneDelivery.Common/Services
  Summary: This folder contains the core service implementations and interfaces for managing drone delivery operations.
  Purpose: This folder serves as the backbone of the drone delivery system, containing the service classes and interfaces that define and implement the core functionalities of the project. It plays a crucial role in organizing the business logic related to drone scheduling, delivery processing, package creation, and repository management. By encapsulating these responsibilities, the folder ensures a clean separation of concerns, promoting scalability and maintainability. It interacts with other components such as data stores and logging systems, facilitating communication and data flow across the system. The folder is significant for its role in handling delivery requests, scheduling drones, and processing packages, all of which are essential for the project's workflow and functionality.
  Folder-features:
  - **Feature:** Drone Scheduling
    Feature-details: Implements logic to assign drones to delivery requests efficiently.
  - **Feature:** Delivery Processing
    Feature-details: Processes delivery requests by coordinating between different services and handling exceptions.
  - **Feature:** Package Creation
    Feature-details: Handles the creation of packages asynchronously based on provided package information.
  - **Feature:** Repository Management
    Feature-details: Manages delivery data persistence and retrieval through a common repository interface.
  - **Feature:** Utility Functions
    Feature-details: Provides computational utility functions like permutation calculations to support service operations.

  Folder-technical-specification:
  - **Platform/Technologies:** C#, .NET Core
    Programming Languages: C#
    Data Persistence: Interacts with SQL Azure for data storage and retrieval.
    Integration: Uses dependency injection to manage service dependencies and logging.
    State Management: Asynchronous operations using Task-based programming.
    Security: Handles exceptions and logs errors to ensure reliable operation.
    Other Tools: Simulated data store access for testing purposes.

  Category: services
  Folder-tags: src,services,repositories,utils,middleware
  Defect-criteria: Defects can be identified by testing the asynchronous methods for correct task completion and error handling. Unit tests should verify that drones are correctly scheduled, packages are created as expected, and delivery requests are processed without exceptions. Logging should be checked to ensure that all steps are recorded accurately, and any integration with SQL Azure should be validated for data consistency and reliability.
  Sub-folders:

  Folder depth: 0
  Files: src/after/DroneDelivery.Common/Services/DeliveryRepository.cs, src/after/DroneDelivery.Common/Services/Utility.cs, src/after/DroneDelivery.Common/Services/IDroneScheduler.cs, src/after/DroneDelivery.Common/Services/DroneScheduler.cs, src/after/DroneDelivery.Common/Services/IPackageProcessor.cs, src/after/DroneDelivery.Common/Services/RequestProcessor.cs, src/after/DroneDelivery.Common/Services/IDeliveryRepository.cs, src/after/DroneDelivery.Common/Services/IRequestProcessor.cs
  Level: 3
- **Folder name:** src/after/DroneDelivery.Common/Models
  Summary: This folder contains the data models and enumerations used by the drone delivery system.
  Purpose: The 'src/after/DroneDelivery.Common/Models' folder is designed to encapsulate all the data models and enumerations that define the structure and behavior of data within the drone delivery application. These models represent various entities like UserAccount, Package, Delivery, and Location, which are crucial for managing the application's core functionalities. By organizing these models in a

dedicated folder, the project ensures a clear separation of concerns, promoting maintainability and scalability. This folder interacts primarily with the business logic and service layers of the application, providing them with structured data representations. Data flows through this folder as it is transformed into and out of these models, often serialized to and from JSON for external communication.
Folder-features:

- **Feature:** User Account Management
  Feature-details: Implements a UserAccount class to manage user identifiers and account information.
- **Feature:** Package Details Representation
  Feature-details: Defines PackageGen and PackageInfo classes to model package attributes like size, weight, and tag.
- **Feature:** Delivery Order Management
  Feature-details: Includes a Delivery class to encapsulate delivery order details and a DeliveryStatus class for tracking progress.
- **Feature:** Geographical Location Modeling
  Feature-details: Provides a Location class to represent geographical coordinates for navigation purposes.
- **Feature:** Delivery Confirmation and Stages
  Feature-details: Uses enumerations like ConfirmationType and DeliveryStage to manage delivery confirmation methods and track delivery progress.

Folder-technical-specification:

- **Platform/Technologies:** C#, .NET
  Programming Languages: C#
  Data Serialization: JSON serialization and deserialization using Newtonsoft.Json
  Data Flow: Data is structured into classes and enums, serialized to JSON for communication, and deserialized back into objects for processing.
  Integration: Models are likely used by service layers and possibly interact with external APIs for data exchange.

Category: models
Folder-tags: dto,src,models
Defect-criteria: Defects in this folder can be identified by testing the integrity of data transformations and ensuring correct serialization/deserialization of model classes. Unit tests should cover each model's properties and methods, checking for correct initialization and data handling.
Sub-folders:

Folder depth: 0
Files: src/after/DroneDelivery.Common/Models/UserAccount.cs, src/after/DroneDelivery.Common/Models/PackageGen.cs, src/after/DroneDelivery.Common/Models/PackageInfo.cs, src/after/DroneDelivery.Common/Models/Delivery.cs, src/after/DroneDelivery.Common/Models/ConfirmationType.cs, src/after/DroneDelivery.Common/Models/DeliveryStage.cs, src/after/DroneDelivery.Common/Models/DeliveryStatus.cs, src/after/DroneDelivery.Common/Models/ContainerSize.cs, src/after/DroneDelivery.Common/Models/Location.cs, src/after/DroneDelivery.Common/Models/ConfirmationRequired.cs
Level: 3

Folder depth: 1
Files:
Level: 2

- **Folder name:** src/after/PackageService
  Summary: This folder contains the implementation of an Azure Function designed to handle package updates via HTTP PUT requests.
  Purpose: The 'src/after/PackageService' folder plays a crucial role in the project by implementing serverless computing capabilities using Azure Functions. Its primary purpose is to manage package updates through an HTTP-triggered function, allowing for scalable and efficient processing of update requests. This folder's significance lies in its ability to provide a backend service that can be easily scaled and managed within the Azure cloud environment. It fits into the system's functionality by acting as a microservice that responds to HTTP requests, processes them, and returns appropriate responses. The folder contains the core logic for handling package updates and is integral to the project's workflow, ensuring that package data is consistently and reliably updated. The main file in this folder, 'PackageServiceFunction.cs', defines the function logic, while 'host.json' configures the Azure Functions runtime. This folder interacts with other parts of the project by serving as an endpoint for package update requests, potentially interacting with databases or other services as needed.
  Folder-features:

  - **Feature:** HTTP Triggered Package Update
    Feature-details: The Azure Function is triggered by HTTP PUT requests to handle package updates, providing a scalable and efficient way to process these requests.
  - **Feature:** Azure Functions Runtime Configuration
    Feature-details: The 'host.json' file configures the Azure Functions runtime, ensuring the function app operates with the correct version and settings.

  Folder-technical-specification:

  - **Platform/Technologies:** Azure Functions, .NET
    Programming Languages: C#
    API Endpoints: HTTP PUT endpoint for package updates
    Data Flow: Incoming HTTP requests are processed by the Azure Function, which simulates a task and returns a result.
    Integration: The function integrates with Azure's serverless architecture to handle HTTP requests.
    Routing: Configured to respond to HTTP PUT requests for package updates.
    Security: Relies on Azure's security features for managing HTTP requests.
    Third-party components: Azure Functions runtime
    Other tools and technologies: Azure cloud services

  Category: backend
  Folder-tags: src,cloud,services,api,azure
  Defect-criteria: Defects related to this folder can be identified by testing the HTTP PUT endpoint for package updates, ensuring that requests are correctly processed and that the function returns the expected results. Additionally, reviewing the 'host.json' configuration for errors or

misconfigurations that could affect the runtime behavior is essential.
Sub-folders:

Folder depth: 0
Files: src/after/PackageService/PackageServiceFunction.cs, src/after/PackageService/host.json
Level: 2

- **Folder name:** src/after/DroneDelivery-after
  Summary: This folder contains the core components of the DroneDelivery application, including configuration files, startup scripts, and main application logic.
  Purpose: The 'src/after/DroneDelivery-after' folder serves as the main directory for the DroneDelivery application, encapsulating its configuration, startup logic, and core functionalities. - It houses essential components such as configuration files (appsettings.json), startup scripts (Startup.cs, Program.cs), and the main application logic, which set up the ASP.NET Core environment and define the application's behavior. - This folder contributes to the overall structure by organizing the application's entry point, configuration settings, and service registration, ensuring a cohesive and maintainable codebase. - The folder's significance lies in its role as the backbone of the application, providing a structured approach to managing configuration, service registration, and request handling, crucial for future scalability. - It fits into the system's functionality by defining how the application initializes, configures services, and handles HTTP requests, interacting with other components like services, controllers, and external APIs. - The folder contains various types of files, including JSON configuration files, C# scripts for application startup and configuration, and folders for services, controllers, and connected services. - It interacts with other parts of the project by setting up the environment for the application's operation, configuring service dependencies, and defining routing and middleware for handling requests. - Data flows through this folder as it processes configuration settings, initializes services, and routes HTTP requests to appropriate controllers for further processing.
  Folder-features:
  - **Feature:** Configuration Management
    Feature-details: Handles environment-specific settings through appsettings.json and appsettings.Development.json, defining logging levels and service URIs.
  - **Feature:** Application Startup
    Feature-details: Defines application startup logic in Startup.cs and Program.cs, setting up dependency injection, middleware, and routing.
  - **Feature:** Service Registration
    Feature-details: Registers services such as IDeliveryRepository, IDroneScheduler, and others for dependency injection.
  - **Feature:** HTTP Request Handling
    Feature-details: Sets up routing and middleware for handling HTTP requests and responses, integrating with controllers.
  - **Feature:** External Service Communication
    Feature-details: Manages interactions with external services through the Services folder, enabling HTTP requests to package APIs.

  Folder-technical-specification:
  - **Platform/Technologies:** ASP.NET Core
    Programming Languages: C#
    API Endpoints: Defined in controllers, accessed via mapped routes
    State Management: Handled through dependency injection and service registration
    Data Flow: Data flows from configuration files to services and controllers, processed via HTTP requests
    Data Persistence: Configuration data persisted in JSON files
    Integration: Integrates with external APIs via HTTP clients
    Validation: Configured through middleware and service logic
    Routing: Defined in Startup.cs, mapping to controller actions
    Security: Managed via HTTPS redirection and potential middleware
    Third-party Components: Swagger for API documentation

  Category: backend
  Folder-tags: integration,config,src,services,controllers,clients,api,metrics,telemetry,alerts,monitoring
  Defect-criteria: Defects related to this folder can be identified through issues in application startup, incorrect configuration settings, improper service registration, middleware failures, and routing errors. Logs and error messages during application runtime can help pinpoint these defects.
  Sub-folders:
  - **Folder name:** src/after/DroneDelivery-after/Services
    Summary: This folder contains service classes responsible for handling external service communications.
    Purpose: The 'Services' folder is crucial for managing interactions with external systems, specifically for creating packages through HTTP requests. It ensures that the application can communicate effectively with other services, which is essential for its operation. - The role of this folder is to encapsulate all logic related to external service calls, making it easier to manage and modify these interactions. - It contributes to the overall structure by isolating service communication logic, improving maintainability and scalability. - This folder is significant for organizing the codebase, as it separates concerns and allows for easier testing and debugging. - It fits into the system's functionality by acting as a bridge between the application and external services, ensuring data is correctly sent and received. - The folder contains classes like 'PackageServiceCaller', which implements interfaces and manages HTTP requests. - It interacts with other parts of the project by being called upon wherever external service communication is required, ensuring seamless data flow between the application and external endpoints.
    Folder-features:
    - **Feature:** HTTP Service Communication
      Feature-details: Implements functionality to send HTTP POST requests to external services for package creation.
    - **Feature:** Interface Implementation
      Feature-details: Implements the IPackageProcessor interface, ensuring a contract for processing packages.
    - **Feature:** JSON Serialization
      Feature-details: Serializes package information into JSON format before sending it over the network.

    Folder-technical-specification:
    - **Platform/Technologies:** C#, .NET
      Programming Languages: C#
      API Endpoints: Utilizes HTTP POST requests to communicate with external service endpoints.
      Data Flow: Data is serialized into JSON and sent via HTTP requests, with responses handled to ensure successful communication.

        Integration: Integrates with external services via HTTP, using HttpClient for network operations.
        Security: Ensures secure communication with external services, likely using HTTPS.

    Category: backend
    Folder-tags: src,services,clients,api
    Defect-criteria: Defects can be identified through failed HTTP requests, incorrect JSON serialization, or interface implementation issues. Logging and monitoring of service interactions can help detect anomalies.
    Sub-folders:

    Folder depth: 0
    Files: src/after/DroneDelivery-after/Services/PackageServiceCaller.cs
    Level: 3

- **Folder name:** src/after/DroneDelivery-after/Controllers
  Summary: This folder contains the controller classes for handling HTTP requests in the DroneDelivery ASP.NET Core application.
  Purpose: The 'Controllers' folder plays a crucial role in the ASP.NET Core MVC architecture by managing the incoming HTTP requests and returning appropriate responses. It acts as the intermediary between the model and view components of the application. The folder contains three main controller classes: DeliveryRequestsController, HomeController, and DeliveriesController, each responsible for handling specific routes and actions related to delivery operations. These controllers utilize dependency injection to interact with services such as IRequestProcessor, IHttpClientFactory, and a delivery repository, ensuring a modular and testable codebase. The folder's organization supports scalability by allowing easy addition of new controllers or modification of existing ones. By defining clear routes and actions, this folder facilitates seamless integration with other parts of the application, including services and external APIs, ensuring efficient data flow and processing.
  Folder-features:
    - **Feature:** Delivery Request Handling
      Feature-details: The DeliveryRequestsController processes incoming delivery requests, logs the details, and creates new delivery resources.
    - **Feature:** Home Page Management
      Feature-details: The HomeController manages the home page requests and sends delivery requests to an API endpoint.
    - **Feature:** Delivery Retrieval and Status Check
      Feature-details: The DeliveriesController provides endpoints to retrieve delivery details by ID and check delivery status.

  Folder-technical-specification:
    - **Platform/Technologies:** ASP.NET Core
      Programming Languages: C#
      API Endpoints: Defined in each controller for handling specific routes like 'api/deliveries'.
      State Management: Managed through dependency injection and services.
      Data Flow: Data flows from HTTP requests to controllers, then to services or repositories for processing.
      Integration: Integrates with services like IRequestProcessor and IHttpClientFactory.
      Routing: Uses ASP.NET Core routing to map HTTP requests to controller actions.
      Security: Relies on ASP.NET Core's built-in security features for handling HTTP requests safely.

  Category: backend
  Folder-tags: src,services,controllers,api
  Defect-criteria: Defects in this folder can be identified by testing the API endpoints for correct request handling, response status codes, and data integrity. Logging and exception handling should be reviewed to ensure errors are captured and managed appropriately.
  Sub-folders:

  Folder depth: 0
  Files: src/after/DroneDelivery-after/Controllers/DeliveryRequestsController.cs, src/after/DroneDelivery-after/Controllers/HomeController.cs, src/after/DroneDelivery-after/Controllers/DeliveriesController.cs
  Level: 3
- **Folder name:** src/after/DroneDelivery-after/Connected Services
  Summary: This folder manages the configuration necessary for connecting the project to Microsoft Application Insights for monitoring and analyzing application performance.
  Purpose: The folder's role in the project is to enable telemetry data collection, which is crucial for diagnosing performance issues and understanding user behavior. It contributes to the overall structure by providing the necessary setup to integrate Application Insights, ensuring that the application can send data for monitoring. The folder is significant in terms of organization as it centralizes the configuration needed for performance monitoring, which is vital for maintaining and scaling the application. It fits into the systemâ€™s functionality by allowing the application to communicate with the Application Insights service and send telemetry data. The types of files primarily include configuration files, such as JSON files, that specify the settings for connecting to Application Insights. This folder interacts with other parts of the project by providing configuration details that are utilized by the application code to establish a connection with Application Insights. Data flows through this folder in the form of configuration settings, which the application uses to send telemetry data to the Application Insights service.
  Folder-features:
    - **Feature:** Telemetry Data Collection
      Feature-details: Enables the collection of telemetry data to monitor application performance and user behavior.
    - **Feature:** Performance Monitoring
      Feature-details: Facilitates the monitoring of application performance metrics through Application Insights integration.

  Folder-technical-specification:
    - **Platform/Technologies:** Microsoft Application Insights
      Configuration Files: JSON
      Data Flow: Configuration settings are used by the application to send telemetry data to the Application Insights service.
      Integration: The folder provides the necessary configuration for the application to integrate with Application Insights.

Category: config
Folder-tags: integration,config,metrics,telemetry,alerts,monitoring
Defect-criteria: Defects related to this folder can be identified by checking for incorrect or missing configuration settings that prevent successful connection to Application Insights, resulting in lack of telemetry data or incorrect performance metrics.
Sub-folders:

- **Folder name:** src/after/DroneDelivery-after/Connected Services/Application Insights
  Summary: This folder contains configuration files for integrating Microsoft Application Insights into the project.
  Purpose: The purpose of this folder is to manage the configuration necessary for connecting the project to Microsoft Application Insights, a service used for monitoring and analyzing application performance. - This folder plays a crucial role in the project by enabling telemetry data collection, which is essential for diagnosing performance issues and understanding user behavior. - It contributes to the overall structure by providing the necessary setup to integrate Application Insights, ensuring that the application can send data for monitoring. - The folder is significant in terms of organization as it centralizes the configuration needed for performance monitoring, which is vital for maintaining and scaling the application. - This folder fits into the system's functionality by allowing the application to communicate with the Application Insights service and send telemetry data. - It primarily contains configuration files, such as JSON files, that specify the settings for connecting to Application Insights. - The folder interacts with other parts of the project by providing configuration details that are utilized by the application code to establish a connection with Application Insights. - Data flows through this folder in the form of configuration settings, which the application uses to send telemetry data to the Application Insights service.
  Folder-features:
  - **Feature:** Application Monitoring Setup
    Feature-details: Configures the connection to Microsoft Application Insights to enable application performance monitoring.
  - **Feature:** Telemetry Data Configuration
    Feature-details: Defines settings for collecting telemetry data, which helps in diagnosing performance issues and analyzing user behavior.

  Folder-technical-specification:
  - **Platform/Technologies:** Microsoft Application Insights
    Programming Languages: JSON for configuration
    Integration: Integrates Application Insights for monitoring and analyzing application performance
    Data Flow: Configuration data flows from this folder to the application to enable telemetry data collection
    Security: Ensures secure connection to Application Insights using provider ID and version
    Third-party Components: Uses Microsoft Application Insights as a third-party service

  Category: config
  Folder-tags: monitoring,config,alerts,metrics
  Defect-criteria: Defects related to this folder can be identified by monitoring the application's ability to send telemetry data to Application Insights. Issues may arise if the configuration is incorrect or if there are connectivity problems, which can be diagnosed by checking the settings in the ConnectedService.json file.
  Sub-folders:

  Folder depth: 0
  Files: src/after/DroneDelivery-after/Connected Services/Application Insights/ConnectedService.json
  Level: 4

Folder depth: 1
Files:
Level: 3

Folder depth: 2
Files: src/after/DroneDelivery-after/appsettings.Development.json, src/after/DroneDelivery-after/Startup.cs, src/after/DroneDelivery-after/Program.cs, src/after/DroneDelivery-after/appsettings.json
Level: 2

Folder depth: 3
Files:
Level: 1

Folder depth: 4
Files:
Level: 0

- **Folder name:** deployment
  Summary: Contains deployment configuration files for Azure resources.
  Purpose: The 'deployment' folder is crucial for automating the deployment process of the project on Azure. It contains configuration files that define the infrastructure setup, including web apps, function apps, and other resources. This folder contributes to the project's structure by providing a centralized location for deployment scripts, ensuring consistency and repeatability in setting up environments. It plays a significant role in the project's organization by streamlining the deployment workflow and enabling scalability through automated resource management. The folder interacts with Azure services to provision and configure the necessary resources for the application. It primarily contains ARM templates and possibly scripts for deployment automation, which interact with Azure APIs to manage resources.
  Folder-features:
  - **Feature:** Automated Resource Deployment
    Feature-details: Uses ARM templates to automate the creation and configuration of Azure resources, such as web apps and function apps.
  - **Feature:** Dynamic Resource Configuration
    Feature-details: Defines parameters for resource names, locations, and configurations to allow dynamic resource creation.
  - **Feature:** Dependency Management

Feature-details: Specifies dependencies between resources to ensure proper deployment order and configuration.

- **Feature:** Application Settings Configuration
  Feature-details: Configures application settings like instrumentation keys and runtime versions for deployed resources.
- **Feature:** Monitoring Integration
  Feature-details: Includes Application Insights for monitoring deployed applications.

Folder-technical-specification:

- **Platform/Technologies:** Azure Resource Manager (ARM), JSON
  Programming Languages: JSON for ARM templates
  Integration: Interacts with Azure APIs to deploy and manage cloud resources.
  Data Flow: Data flows from the ARM templates to Azure services, creating and configuring resources.
  Security: Relies on Azure's security features for resource access and management.

Category: config
Folder-tags: ci-cd,config,templates,azure
Defect-criteria: Defects can be identified by deployment failures, incorrect resource configurations, or missing dependencies in the ARM templates.
Sub-folders:

Folder depth: 0
Files: deployment/azuredeploy.json
Level: 0

File Descriptions:

Files:

- **Name:** deployment/azuredeploy.json
  Details:
  Filename: deployment/azuredeploy.json
  Description: This file is an Azure Resource Manager (ARM) template for deploying a web application, function app, and associated resources like Application Insights and storage accounts in Azure.
  Detailedsummary: The azuredeploy.json file is an ARM template used to automate the deployment of a web application and a function app in Azure. It defines parameters for resource names, locations, and configurations, allowing for dynamic resource creation. The template includes resources such as a web app, a hosting plan, Application Insights for monitoring, a function app, and a storage account. It specifies dependencies between resources and configures application settings, such as instrumentation keys and runtime versions. This template facilitates consistent and repeatable deployments of the specified Azure resources.
  Importance: High
  References:
- **Name:** src/after/DroneDelivery-after/Connected Services/Application Insights/ConnectedService.json
  Details:
  Filename: src/after/DroneDelivery-after/Connected Services/Application Insights/ConnectedService.json
  Description: This file configures the connection to Microsoft Application Insights for the project, specifying the provider ID, version, and a link to the getting started documentation.
  Detailedsummary: The ConnectedService.json file is a configuration file used to set up and manage the connection to Microsoft Application Insights, a service for monitoring and analyzing application performance. It includes the provider ID, which identifies the service provider, and the version of the connected service. Additionally, it provides a URI link to the getting started documentation, which helps developers understand how to integrate and utilize Application Insights within their project. This file is crucial for ensuring that the application can send telemetry data to Application Insights for monitoring purposes.
  Importance: High
  References:
- **Name:** src/after/DroneDelivery-after/Controllers/DeliveryRequestsController.cs
  Details:
  Filename: src/after/DroneDelivery-after/Controllers/DeliveryRequestsController.cs
  Description: This file defines a controller for handling delivery requests in a drone delivery system, processing incoming delivery data and creating new delivery records.
  Detailedsummary: The DeliveryRequestsController.cs file is part of an ASP.NET Core application, specifically within the DroneDelivery_after namespace. It defines a controller class, DeliveryRequestsController, which handles HTTP POST requests to the 'api/deliveries' endpoint. The controller uses dependency injection to obtain an IRequestProcessor for processing delivery requests and an ILogger for logging. The Post method is an asynchronous action that receives a Delivery object from the request body, logs the delivery information, and processes the delivery request using the IRequestProcessor. If successful, it returns a 201 Created response with a route to the newly created delivery resource.
  Importance: High
  References:
    - DroneDelivery.Common.Models/Delivery.cs
    - DroneDelivery.Common.Services/IRequestProcessor.cs
- **Name:** src/after/DroneDelivery-after/Controllers/DeliveriesController.cs
  Details:
  Filename: src/after/DroneDelivery-after/Controllers/DeliveriesController.cs
  Description: This file implements a controller for handling delivery-related API requests, providing endpoints to retrieve delivery details and status.
  Detailedsummary: The DeliveriesController class in this file is an ASP.NET Core API controller that manages delivery-related operations. It uses dependency injection to access a delivery repository and a logger. The controller provides two main endpoints: one to retrieve delivery details by ID and another to get the status of a delivery. The Get method fetches a delivery object from the repository and returns it, while the GetStatus method returns a hardcoded delivery status. Both methods log relevant information and handle cases where the delivery is not found.
  Importance: High
  References:
    - DroneDelivery.Common.Models
    - DroneDelivery.Common.Services
- **Name:** src/after/DroneDelivery-after/Controllers/HomeController.cs

Details:
Filename: src/after/DroneDelivery-after/Controllers/HomeController.cs
Description: The HomeController.cs file defines a controller in an ASP.NET Core application responsible for handling HTTP requests related to the home page and sending multiple delivery requests asynchronously.
Detailedsummary: The HomeController.cs file is part of an ASP.NET Core MVC application. It defines the HomeController class, which inherits from the Controller base class. The controller is responsible for handling HTTP requests to the home page and a specific route for sending delivery requests. It uses dependency injection to obtain an IHttpClientFactory instance for creating HTTP clients. The SendRequests method is a POST action that sends a specified number of asynchronous HTTP POST requests to an API endpoint, using a predefined payload representing a delivery. The method measures the time taken to send all requests and displays the result on the view. The Index method simply returns the default view for the home page.
Importance: High
References:
  - DroneDelivery.Common.Models/Delivery.cs
  - DroneDelivery.Common.Models/PackageInfo.cs

- **Name:** src/after/DroneDelivery-after/Program.cs
  Details:
  Filename: src/after/DroneDelivery-after/Program.cs
  Description: This file is the entry point for the DroneDelivery application, setting up and running the web host using ASP.NET Core.
  Detailedsummary: The Program.cs file serves as the entry point for the DroneDelivery application. It defines a Program class with a Main method, which is the standard entry point for C# applications. The Main method calls CreateWebHostBuilder to configure and build the web host. This method uses the Host.CreateDefaultBuilder to set up a default host and configures it to use a Startup class for further configuration. The web host is then built and run, starting the application. This setup is typical for ASP.NET Core applications, where the Program class is responsible for initializing and running the web server.
  Importance: High
  References:
    - src/after/DroneDelivery-after/Startup.cs
- **Name:** src/after/DroneDelivery-after/Services/PackageServiceCaller.cs
  Details:
  Filename: src/after/DroneDelivery-after/Services/PackageServiceCaller.cs
  Description: This file implements a service caller for creating packages in a drone delivery system, utilizing HTTP requests to communicate with an external service.
  Detailedsummary: The PackageServiceCaller class in this file is responsible for creating packages by sending HTTP POST requests to an external service. It implements the IPackageProcessor interface and uses an HttpClient to perform the network operations. The CreatePackageAsync method serializes package information into JSON, sends it to a specified endpoint, and ensures the request's success. The class also includes a static property, FunctionCode, used as a query parameter in the request URL.
  Importance: High
  References:
    - DroneDelivery.Common.Models
    - DroneDelivery.Common.Services
- **Name:** src/after/DroneDelivery-after/Startup.cs
  Details:
  Filename: src/after/DroneDelivery-after/Startup.cs
  Description: This file configures the services and middleware for the DroneDelivery-after application, including dependency injection, HTTP client setup, and Swagger for API documentation.
  Detailedsummary: The Startup.cs file is crucial for setting up the ASP.NET Core application. It defines the Startup class, which includes two main methods: ConfigureServices and Configure. ConfigureServices is used to register services with the dependency injection container, such as HTTP clients and custom services like IDeliveryRepository, IDroneScheduler, IPackageProcessor, and IRequestProcessor. It also configures Swagger for API documentation. The Configure method sets up the HTTP request pipeline, enabling middleware for development exceptions, HTTPS redirection, and Swagger UI. It also maps the default controller route for handling incoming requests.
  Importance: High
  References:
    - DroneDelivery.Common.Services
    - DroneDelivery_after.Services
- **Name:** src/after/DroneDelivery-after/appsettings.json
  Details:
  Filename: src/after/DroneDelivery-after/appsettings.json
  Description: This file contains configuration settings for the DroneDelivery application, including logging levels, allowed hosts, and service URIs.
  Detailedsummary: The appsettings.json file is a configuration file for the DroneDelivery application. It specifies the logging configuration, setting the default log level to 'Warning'. It also defines the allowed hosts for the application, which is set to allow all hosts ('*'). Additionally, it includes the URI for the PackageService, which is an Azure Function endpoint, and a placeholder for the function code. This file is crucial for setting up the application's environment-specific settings and ensuring that the application can connect to necessary services.
  Importance: High
  References:
- **Name:** src/after/DroneDelivery-after/appsettings.Development.json
  Details:
  Filename: src/after/DroneDelivery-after/appsettings.Development.json
  Description: This file configures logging levels and specifies the URI for the package service in a development environment for the DroneDelivery application.
  Detailedsummary: The appsettings.Development.json file is a configuration file used in the DroneDelivery application to set up environment-specific settings for development. It defines logging levels for different components, such as 'Default', 'System', and 'Microsoft', allowing developers to control the verbosity of logs during development. Additionally, it specifies the URI for the PackageService, which is used to interact with the package API hosted locally at 'http://localhost:7071/api/packages/'. This setup is crucial for testing and debugging the application in a development environment, ensuring that developers can monitor application behavior and interact with local services effectively.
  Importance: High
  References:
- **Name:** src/after/DroneDelivery.Common/Models/ConfirmationType.cs
  Details:
  Filename: src/after/DroneDelivery.Common/Models/ConfirmationType.cs

    Description: Defines an enumeration for different types of delivery confirmation methods in a drone delivery system.
    Detailedsummary: The file defines an enumeration named 'ConfirmationType' within the namespace 'DroneDelivery.Common.Models'. This enumeration lists various methods of confirming a delivery, including 'FingerPrint', 'Picture', 'Voice', and 'None'. This allows the system to specify how a delivery should be confirmed, providing flexibility in handling different confirmation scenarios.
    Importance: Medium
    References:

- **Name:** src/after/DroneDelivery.Common/Models/ConfirmationRequired.cs
  Details:
    Filename: src/after/DroneDelivery.Common/Models/ConfirmationRequired.cs
    Description: Defines an enumeration for different types of confirmation methods required for drone delivery, including fingerprint, picture, voice, and none.
    Detailedsummary: The file defines an enumeration named 'ConfirmationRequired' within the namespace 'DroneDelivery.Common.Models'. This enumeration specifies four possible values that represent different methods of confirmation that might be required in a drone delivery system: 'FingerPrint', 'Picture', 'Voice', and 'None'. This allows the system to easily manage and check the type of confirmation needed for a delivery, enhancing flexibility and scalability in handling various delivery scenarios. The use of an enumeration provides a clear and type-safe way to handle these predefined constants throughout the application.
    Importance: Medium
    References:
- **Name:** src/after/DroneDelivery.Common/Models/ContainerSize.cs
  Details:
    Filename: src/after/DroneDelivery.Common/Models/ContainerSize.cs
    Description: Defines an enumeration for container sizes used in the drone delivery system, categorizing them into Small, Medium, and Large.
    Detailedsummary: This file defines an enumeration named 'ContainerSize' within the namespace 'DroneDelivery.Common.Models'. The enumeration specifies three possible sizes for containers: Small, Medium, and Large. This is likely used to standardize the size categories for containers that drones can carry, facilitating consistent handling and processing of container size data across the application. Enumerations like this are crucial for ensuring that only valid, predefined values are used, reducing errors and improving code readability.
    Importance: Medium
    References:
- **Name:** src/after/DroneDelivery.Common/Models/Delivery.cs
  Details:
    Filename: src/after/DroneDelivery.Common/Models/Delivery.cs
    Description: Defines the Delivery class, representing a delivery order with properties like delivery ID, owner ID, locations, deadline, and package information.
    Detailedsummary: The Delivery.cs file defines a class named Delivery within the DroneDelivery.Common.Models namespace. This class models a delivery order, encapsulating various properties such as DeliveryId, OwnerId, PickupLocation, DropoffLocation, Deadline, and Expedited status. It also includes a ConfirmationRequired property to indicate if delivery confirmation is needed, a PickupTime to specify when the package should be picked up, and a PackageInfo object to hold details about the package being delivered. This class serves as a data structure to manage and transfer delivery-related information within the application.
    Importance: High
    References:
    - src/after/DroneDelivery.Common/Models/ConfirmationRequired.cs
    - src/after/DroneDelivery.Common/Models/PackageInfo.cs
- **Name:** src/after/DroneDelivery.Common/Models/DeliveryStage.cs
  Details:
    Filename: src/after/DroneDelivery.Common/Models/DeliveryStage.cs
    Description: Defines an enumeration representing the various stages of a delivery process in a drone delivery system.
    Detailedsummary: The file defines an enumeration named 'DeliveryStage' within the namespace 'DroneDelivery.Common.Models'. This enumeration represents the different stages a delivery can go through in a drone delivery system. The stages include 'Created', 'Rescheduled', 'HeadedToPickup', 'HeadedToDropoff', 'Completed', and 'Cancelled'. Each stage represents a specific point in the delivery lifecycle, providing a clear and structured way to track the progress of a delivery. This enum is likely used throughout the application to manage and update the status of deliveries as they progress through these stages.
    Importance: Medium
    References:
- **Name:** src/after/DroneDelivery.Common/Models/DeliveryStatus.cs
  Details:
    Filename: src/after/DroneDelivery.Common/Models/DeliveryStatus.cs
    Description: Defines the DeliveryStatus class, which encapsulates the current status of a delivery, including its stage, last known location, and estimated times for pickup and delivery.
    Detailedsummary: The DeliveryStatus.cs file defines a class named DeliveryStatus within the DroneDelivery.Common.Models namespace. This class is designed to represent the current status of a delivery in a drone delivery system. It includes properties for the delivery stage (Stage), the last known location of the delivery (LastKnownLocation), and estimated times for both pickup (PickupETA) and delivery (DeliveryETA). The class constructor initializes these properties, ensuring that each instance of DeliveryStatus is created with complete and relevant information about a delivery's progress. This class is likely used throughout the system to track and display the status of deliveries in real-time.
    Importance: High
    References:
    - src/after/DroneDelivery.Common/Models/DeliveryStage.cs
    - src/after/DroneDelivery.Common/Models/Location.cs
- **Name:** src/after/DroneDelivery.Common/Models/Location.cs
  Details:
    Filename: src/after/DroneDelivery.Common/Models/Location.cs
    Description: Defines a Location class representing geographical coordinates with altitude, latitude, and longitude properties.
    Detailedsummary: The file defines a Location class within the DroneDelivery.Common.Models namespace. This class models a geographical location using three properties: Altitude, Latitude, and Longitude, all of which are double precision floating-point numbers. The class provides a constructor to initialize these properties, ensuring that each instance of Location has a defined set of coordinates. This class is likely used to represent the position of drones or delivery points in a drone delivery system, providing essential data for navigation and logistics.
    Importance: Medium
    References:
- **Name:** src/after/DroneDelivery.Common/Models/PackageGen.cs
  Details:

Filename: src/after/DroneDelivery.Common/Models/PackageGen.cs
Description: Defines a C# class representing a package with properties for ID, size, tag, and weight, utilizing JSON serialization attributes.
Detailedsummary: The file defines a C# class named 'PackageGen' within the 'DroneDelivery.Common.Models' namespace. This class models a package entity with four properties: 'Id', 'Size', 'Tag', and 'Weight'. Each property is decorated with 'JsonProperty' attributes to specify their JSON representation. The 'Size' property uses a 'JsonConverter' with 'StringEnumConverter' to handle enum serialization. This setup facilitates the conversion of 'PackageGen' objects to and from JSON, making it suitable for data exchange in applications, particularly in scenarios involving drone delivery systems where package details need to be serialized and deserialized.
Importance: Medium
References:

- src/after/DroneDelivery.Common/Models/ContainerSize.cs

- **Name:** src/after/DroneDelivery.Common/Models/PackageInfo.cs
  Details:
  Filename: src/after/DroneDelivery.Common/Models/PackageInfo.cs
  Description: Defines the PackageInfo class, which represents information about a package, including its ID, size, weight, and tag, with JSON serialization attributes.
  Detailedsummary: The PackageInfo.cs file defines a C# class named PackageInfo within the DroneDelivery.Common.Models namespace. This class models the essential details of a package, including its unique identifier (PackageId), size (Size), weight (Weight), and an optional tag (Tag). The class properties are decorated with Newtonsoft.Json attributes to facilitate JSON serialization and deserialization. The Size property uses a JSON converter to handle enumeration values as strings, ensuring compatibility with JSON data formats. This class is likely used in the context of a drone delivery system to encapsulate package details for processing and communication between different system components.
  Importance: High
  References:
- **Name:** src/after/DroneDelivery.Common/Models/UserAccount.cs
  Details:
  Filename: src/after/DroneDelivery.Common/Models/UserAccount.cs
  Description: Defines a UserAccount class with properties for user and account identifiers, encapsulating user account information.
  Detailedsummary: The file defines a UserAccount class within the DroneDelivery.Common.Models namespace. This class is a simple data model with two read-only properties: UserId and AccountId. These properties are initialized through the constructor, which takes two string parameters. The class is designed to encapsulate user account information, providing a structured way to handle user-related data within the application. The use of read-only properties ensures that once a UserAccount object is created, its UserId and AccountId cannot be modified, promoting immutability and data integrity.
  Importance: Medium
  References:
- **Name:** src/after/DroneDelivery.Common/Services/DeliveryRepository.cs
  Details:
  Filename: src/after/DroneDelivery.Common/Services/DeliveryRepository.cs
  Description: This file implements the DeliveryRepository class, which provides methods for retrieving and scheduling delivery data in a drone delivery system.
  Detailedsummary: The DeliveryRepository.cs file defines the DeliveryRepository class, which implements the IDeliveryRepository interface. It contains two primary methods: GetAsync, which is intended to retrieve a Delivery object by its ID, and ScheduleDeliveryAsync, which schedules a delivery request for a specific drone. The ScheduleDeliveryAsync method simulates work by calling a utility function and returns a successful result. The GetAsync method is not yet implemented. This class is likely part of a larger system that manages drone deliveries, interacting with a common datastore, such as SQL Azure, to persist and retrieve delivery information.
  Importance: High
  References:
    - DroneDelivery.Common/Models/Delivery.cs
    - DroneDelivery.Common/Services/IDeliveryRepository.cs
    - Utility.cs
- **Name:** src/after/DroneDelivery.Common/Services/DroneScheduler.cs
  Details:
  Filename: src/after/DroneDelivery.Common/Services/DroneScheduler.cs
  Description: This file implements the DroneScheduler class, which provides a method to asynchronously retrieve a drone ID for a given delivery request.
  Detailedsummary: The DroneScheduler.cs file defines the DroneScheduler class, which implements the IDroneScheduler interface. It contains a method GetDroneIdAsync that takes a Delivery object as a parameter and simulates accessing a data store to retrieve a drone ID. The method uses a utility function to simulate work and returns a test drone ID asynchronously. This class is likely part of a larger system responsible for managing drone deliveries, and it abstracts the logic for scheduling or assigning drones to delivery requests.
  Importance: Medium
  References:
    - DroneDelivery.Common/Models/Delivery.cs
- **Name:** src/after/DroneDelivery.Common/Services/IDeliveryRepository.cs
  Details:
  Filename: src/after/DroneDelivery.Common/Services/IDeliveryRepository.cs
  Description: This file defines an interface for a delivery repository, specifying methods for retrieving and scheduling deliveries in a drone delivery system.
  Detailedsummary: The IDeliveryRepository.cs file defines an interface named IDeliveryRepository within the DroneDelivery.Common.Services namespace. This interface outlines two asynchronous methods: GetAsync, which retrieves a Delivery object based on a given ID, and ScheduleDeliveryAsync, which schedules a delivery using a Delivery object and a drone ID. These methods are crucial for managing delivery operations in a drone delivery system, providing a contract for implementing classes to interact with delivery data.
  Importance: High
  References:
    - DroneDelivery.Common.Models/Delivery.cs
- **Name:** src/after/DroneDelivery.Common/Services/IDroneScheduler.cs
  Details:
  Filename: src/after/DroneDelivery.Common/Services/IDroneScheduler.cs
  Description: This file defines an interface for scheduling drones, specifying a method to asynchronously retrieve a drone ID based on a delivery request.
  Detailedsummary: The IDroneScheduler.cs file defines an interface named IDroneScheduler within the DroneDelivery.Common.Services namespace. This interface declares a single asynchronous method, GetDroneIdAsync, which takes a Delivery object as a parameter and returns a Task containing a string. The purpose of this interface is to provide a contract for implementing classes to schedule drones by obtaining a drone ID for a given delivery request. This is a crucial

part of the drone delivery system, ensuring that delivery requests are matched with available drones efficiently.
Importance: High
References:
  - DroneDelivery.Common.Models/Delivery.cs

- **Name:** src/after/DroneDelivery.Common/Services/IPackageProcessor.cs
  Details:
  Filename: src/after/DroneDelivery.Common/Services/IPackageProcessor.cs
  Description: Defines an interface for processing packages asynchronously in a drone delivery system.
  Detailedsummary: The file defines the IPackageProcessor interface within the DroneDelivery.Common.Services namespace. This interface declares a single method, CreatePackageAsync, which takes a PackageInfo object as a parameter and returns a Task of type PackageGen. This method is intended to be implemented by classes that handle the creation of packages in an asynchronous manner, facilitating the processing of package data in a drone delivery system. The interface promotes a contract for package processing, ensuring that any implementing class will provide the necessary functionality to create packages based on the provided package information.
  Importance: High
  References:
    - DroneDelivery.Common.Models/PackageGen.cs
    - DroneDelivery.Common.Models/PackageInfo.cs
- **Name:** src/after/DroneDelivery.Common/Services/IRequestProcessor.cs
  Details:
  Filename: src/after/DroneDelivery.Common/Services/IRequestProcessor.cs
  Description: This file defines an interface for processing delivery requests asynchronously within the DroneDelivery.Common.Services namespace.
  Detailedsummary: The IRequestProcessor.cs file defines an interface named IRequestProcessor within the DroneDelivery.Common.Services namespace. This interface declares a single method, ProcessDeliveryRequestAsync, which takes a Delivery object as a parameter and returns a Task of type bool. The method is intended to process delivery requests asynchronously, indicating success or failure through the returned boolean value. This interface is crucial for implementing the logic to handle delivery requests in a decoupled and testable manner, allowing different implementations to be swapped as needed.
  Importance: High
  References:
    - DroneDelivery.Common.Models/Delivery.cs
- **Name:** src/after/DroneDelivery.Common/Services/RequestProcessor.cs
  Details:
  Filename: src/after/DroneDelivery.Common/Services/RequestProcessor.cs
  Description: This file implements the RequestProcessor class, which processes delivery requests by coordinating package creation, drone scheduling, and delivery scheduling.
  Detailedsummary: The RequestProcessor class in this file is responsible for handling delivery requests. It uses dependency injection to obtain instances of ILogger, IPackageProcessor, IDroneScheduler, and IDeliveryRepository. The main method, ProcessDeliveryRequestAsync, logs the processing of a delivery request, creates a package, assigns a drone, and schedules the delivery. It handles exceptions and logs errors if any step fails. The class ensures that each delivery request is processed in a structured manner, coordinating between different components to complete the delivery process.
  Importance: High
  References:
    - DroneDelivery.Common/Models/Delivery.cs
- **Name:** src/after/DroneDelivery.Common/Services/Utility.cs
  Details:
  Filename: src/after/DroneDelivery.Common/Services/Utility.cs
  Description: This file contains a Utility class with a static method DoWork that calculates the number of permutations for a given integer, using nested loops.
  Detailedsummary: The Utility.cs file defines a Utility class within the DroneDelivery.Common.Services namespace. It includes a static method named DoWork, which takes an integer parameter 'permutations'. The method uses four nested loops to iterate over the range of the permutations parameter, effectively counting the number of permutations possible. The result is stored in a long variable 'count', which is returned at the end of the method. This method is likely used for computational tasks that require permutation calculations.
  Importance: Medium
  References:
- **Name:** src/after/PackageService/PackageServiceFunction.cs
  Details:
  Filename: src/after/PackageService/PackageServiceFunction.cs
  Description: This file implements an Azure Function that handles HTTP PUT requests to update package information, logging the request and performing a simulated work task.
  Detailedsummary: The PackageServiceFunction.cs file defines a static class PackageServiceFunction within the PackageService namespace. It contains a single Azure Function named 'PackageServiceFunction' that is triggered by HTTP PUT requests. The function takes an HttpRequest and a package ID as parameters, logs the request, and simulates a task using a utility method. It returns a CreatedResult indicating successful processing. The function is designed to handle package updates, leveraging Azure's serverless computing capabilities.
  Importance: High
  References:
    - DroneDelivery.Common/Services/Utility.cs
- **Name:** src/after/PackageService/host.json
  Details:
  Filename: src/after/PackageService/host.json
  Description: This file is a configuration file for an Azure Functions app, specifying the version of the Azure Functions runtime to use.
  Detailedsummary: The host.json file is a configuration file used in Azure Functions projects. It defines global configuration options for the function app. In this specific file, the version is set to '2.0', indicating that the function app should use version 2.0 of the Azure Functions runtime. This file is crucial for ensuring that the function app runs with the correct runtime version, which can affect the behavior and compatibility of the functions within the app.
  Importance: High
  References:
- **Name:** src/before/DroneDelivery-before/Connected Services/Application Insights/ConnectedService.json
  Details:
  Filename: src/before/DroneDelivery-before/Connected Services/Application Insights/ConnectedService.json
  Description: This file configures the connection to Microsoft Application Insights for the project, specifying the provider ID, version, and a link to the getting

started documentation.
Detailedsummary: The ConnectedService.json file is a configuration file used to set up and manage the connection to Microsoft Application Insights, a service for monitoring and analyzing application performance. It includes the provider ID, which identifies the service provider, and the version of the connected service. Additionally, it contains a URI pointing to the getting started documentation, which provides guidance on how to use Application Insights effectively. This file is crucial for ensuring that the application can send telemetry data to Application Insights for monitoring purposes.
Importance: Medium
References:

- **Name:** src/before/DroneDelivery-before/Controllers/DeliveriesController.cs
  Details:
  Filename: src/before/DroneDelivery-before/Controllers/DeliveriesController.cs
  Description: This file defines a controller for handling delivery-related API requests, including fetching delivery details and status.
  Detailedsummary: The DeliveriesController class in this file is an ASP.NET Core API controller responsible for handling HTTP GET requests related to deliveries. It provides two main endpoints: one for retrieving delivery details by ID and another for fetching the status of a delivery. The controller uses dependency injection to access a delivery repository and a logger. The Get method retrieves a delivery by its ID and returns it if found, otherwise returns a 404 Not Found response. The GetStatus method retrieves the delivery status, simulating a response with a hardcoded status and location. Logging is used to track the flow and any issues encountered during the execution of these methods.
  Importance: High
  References:
    - DroneDelivery.Common.Models/Delivery.cs
    - DroneDelivery.Common.Models/DeliveryStatus.cs
    - DroneDelivery.Common.Models/DeliveryStage.cs
    - DroneDelivery.Common.Models/Location.cs
    - DroneDelivery.Common.Services/IDeliveryRepository.cs
- **Name:** src/before/DroneDelivery-before/Controllers/DeliveryRequestsController.cs
  Details:
  Filename: src/before/DroneDelivery-before/Controllers/DeliveryRequestsController.cs
  Description: This file defines a controller for handling delivery requests in a drone delivery system, processing incoming delivery data and creating new delivery records.
  Detailedsummary: The DeliveryRequestsController.cs file is part of an ASP.NET Core application, specifically within the DroneDelivery_before namespace. It defines a controller class, DeliveryRequestsController, which inherits from ControllerBase. This controller is responsible for handling HTTP POST requests to the 'api/deliveries' endpoint. It uses dependency injection to obtain an IRequestProcessor service for processing delivery requests and an ILogger for logging. The Post method accepts a Delivery object from the request body, logs the delivery information, and processes the delivery request asynchronously. Upon successful processing, it returns a 201 Created response with a route to the newly created delivery resource.
  Importance: High
  References:
    - DroneDelivery.Common.Models/Delivery.cs
    - DroneDelivery.Common.Services/IRequestProcessor.cs
- **Name:** src/before/DroneDelivery-before/Controllers/HomeController.cs
  Details:
  Filename: src/before/DroneDelivery-before/Controllers/HomeController.cs
  Description: The HomeController.cs file handles HTTP requests for the home page and sending multiple delivery requests asynchronously using an HTTP client.
  Detailedsummary: The HomeController.cs file is part of an ASP.NET Core MVC application. It defines a HomeController class that inherits from Controller. The class is responsible for handling HTTP requests related to the home page and sending delivery requests. It uses an IHttpClientFactory to create HTTP clients and sends 100 asynchronous POST requests to the '/api/DeliveryRequests' endpoint with a serialized Delivery object as JSON. The Index method returns the default view, while the SendRequests method measures the time taken to send all requests and displays a message with the result.
  Importance: High
  References:
    - DroneDelivery.Common.Models/Delivery.cs
    - DroneDelivery.Common.Models/PackageInfo.cs
- **Name:** src/before/DroneDelivery-before/Services/PackageProcessor.cs
  Details:
  Filename: src/before/DroneDelivery-before/Services/PackageProcessor.cs
  Description: This file implements the PackageProcessor class, which is responsible for creating package instances asynchronously using provided package information.
  Detailedsummary: The PackageProcessor.cs file defines the PackageProcessor class, which implements the IPackageProcessor interface. It contains a method CreatePackageAsync that takes a PackageInfo object as input and returns a Task of type PackageGen. The method simulates work by calling Utility.DoWork and then creates a new PackageGen object with the provided package ID. This class is part of the DroneDelivery_before.Services namespace and is likely used to handle package creation logic in a drone delivery system.
  Importance: Medium
  References:
    - DroneDelivery.Common.Models
    - DroneDelivery.Common.Services
- **Name:** src/before/DroneDelivery-before/Program.cs
  Details:
  Filename: src/before/DroneDelivery-before/Program.cs
  Description: This file is the entry point for the DroneDelivery application, setting up and running the web host using ASP.NET Core.
  Detailedsummary: The Program.cs file serves as the entry point for the DroneDelivery application. It defines a Program class with a Main method, which is the standard entry point for C# applications. The Main method calls CreateWebHostBuilder to configure and build the web host. This method uses ASP.NET Core's Host.CreateDefaultBuilder to set up the default host configuration and then configures the web server to use a Startup class, which is responsible for configuring services and the app's request pipeline. The application is then run by calling Build().Run() on the host builder.
  Importance: High
  References:
    - src/before/DroneDelivery-before/Startup.cs
- **Name:** src/before/DroneDelivery-before/Startup.cs
  Details:

Filename: src/before/DroneDelivery-before/Startup.cs
Description: This file configures the services and middleware for the DroneDelivery-before application, setting up dependency injection, HTTP request pipeline, and Swagger for API documentation.
Detailedsummary: The Startup.cs file is crucial for setting up the DroneDelivery-before application. It initializes the configuration through dependency injection, adding services like HttpClient and various application-specific services such as DeliveryRepository, DroneScheduler, PackageProcessor, and RequestProcessor. It configures the HTTP request pipeline, enabling HTTPS redirection, and sets up Swagger for API documentation. The file also configures the application to use developer exception pages in development and HSTS in production. It maps the default controller route for handling incoming requests.
Importance: High
References:

- DroneDelivery.Common.Services
- DroneDelivery_before.Services

- **Name:** src/before/DroneDelivery-before/appsettings.Development.json
  Details:
  Filename: src/before/DroneDelivery-before/appsettings.Development.json
  Description: This file configures logging levels for different components in a development environment for the DroneDelivery application.
  Detailedsummary: The appsettings.Development.json file is a configuration file used in the DroneDelivery application to set logging levels specifically for the development environment. It defines the log level for different categories, such as 'Default', 'System', and 'Microsoft'. The 'Default' log level is set to 'Debug', which provides detailed logging information useful during development. The 'System' and 'Microsoft' log levels are set to 'Information', which provides informational messages that highlight the progress of the application at a coarse-grained level. This configuration helps developers monitor and debug the application effectively during the development phase by controlling the verbosity of the logs.
  Importance: Medium
  References:
- **Name:** src/before/DroneDelivery-before/appsettings.json
  Details:
  Filename: src/before/DroneDelivery-before/appsettings.json
  Description: This file is a configuration file for an application, primarily setting logging levels and allowed hosts.
  Detailedsummary: The appsettings.json file is a configuration file used in .NET applications to define various settings. In this file, the 'Logging' section specifies the logging level, with 'Default' set to 'Warning', meaning only warnings and more severe messages will be logged. The 'AllowedHosts' setting is set to '*', which allows the application to accept requests from any host. This file is crucial for managing application behavior without changing the code.
  Importance: High
  References:
- **Name:** src/before/DroneDelivery.Common/Models/ConfirmationRequired.cs
  Details:
  Filename: src/before/DroneDelivery.Common/Models/ConfirmationRequired.cs
  Description: Defines an enumeration for different types of confirmation methods required for drone delivery.
  Detailedsummary: The file defines an enumeration named 'ConfirmationRequired' within the namespace 'DroneDelivery.Common.Models'. This enumeration lists various methods of confirmation that might be required for a drone delivery service, including FingerPrint, Picture, Voice, and None. This allows the system to specify what type of confirmation is needed for a delivery, providing flexibility and security in the delivery process. The use of an enum makes it easy to manage and extend the types of confirmations as needed.
  Importance: Medium
  References:
- **Name:** src/before/DroneDelivery.Common/Models/ConfirmationType.cs
  Details:
  Filename: src/before/DroneDelivery.Common/Models/ConfirmationType.cs
  Description: Defines an enumeration for different types of delivery confirmation methods in a drone delivery system.
  Detailedsummary: The file defines an enumeration named ConfirmationType within the namespace DroneDelivery.Common.Models. This enumeration lists various methods for confirming a delivery, including FingerPrint, Picture, Voice, and None. It is likely used to specify or check the type of confirmation required or provided during a delivery process in the drone delivery system.
  Importance: Medium
  References:
- **Name:** src/before/DroneDelivery.Common/Models/ContainerSize.cs
  Details:
  Filename: src/before/DroneDelivery.Common/Models/ContainerSize.cs
  Description: Defines an enumeration for container sizes used in the drone delivery system, categorizing them as Small, Medium, or Large.
  Detailedsummary: This file defines an enumeration named 'ContainerSize' within the namespace 'DroneDelivery.Common.Models'. The enumeration specifies three possible sizes for containers: Small, Medium, and Large. This is likely used to standardize the size categories for containers that drones can carry, ensuring consistency across the application when dealing with container sizes. Enumerations like this are useful for defining a set of named constants, improving code readability and reducing errors associated with using arbitrary values.
  Importance: Medium
  References:
- **Name:** src/before/DroneDelivery.Common/Models/Delivery.cs
  Details:
  Filename: src/before/DroneDelivery.Common/Models/Delivery.cs
  Description: Defines the Delivery class, representing a delivery order with properties like DeliveryId, OwnerId, locations, deadline, and package information.
  Detailedsummary: The Delivery.cs file defines a Delivery class within the DroneDelivery.Common.Models namespace. This class models a delivery order, encapsulating details such as DeliveryId, OwnerId, PickupLocation, DropoffLocation, Deadline, and whether the delivery is expedited. It also includes a ConfirmationRequired property, a DateTime for PickupTime, and a PackageInfo object to hold package-specific details. This class is crucial for managing delivery data within the application, serving as a data structure for delivery-related operations.
  Importance: High
  References:
    - src/before/DroneDelivery.Common/Models/ConfirmationRequired.cs
    - src/before/DroneDelivery.Common/Models/PackageInfo.cs
- **Name:** src/before/DroneDelivery.Common/Models/DeliveryStage.cs
  Details:
  Filename: src/before/DroneDelivery.Common/Models/DeliveryStage.cs

Description: Defines an enumeration for the various stages of a delivery process in a drone delivery system.
Detailedsummary: The file defines an enumeration named DeliveryStage within the namespace DroneDelivery.Common.Models. This enumeration represents the different stages a delivery can go through in a drone delivery system. The stages include Created, Rescheduled, HeadedToPickup, HeadedToDropoff, Completed, and Cancelled. This enum is likely used throughout the application to track and manage the state of deliveries, providing a standardized way to refer to each stage of the delivery process.
Importance: Medium
References:

- **Name:** src/before/DroneDelivery.Common/Models/Location.cs
  Details:
  Filename: src/before/DroneDelivery.Common/Models/Location.cs
  Description: Defines a Location class representing geographical coordinates with altitude, latitude, and longitude properties.
  Detailedsummary: The file defines a Location class within the DroneDelivery.Common.Models namespace. This class models a geographical location using three properties: Altitude, Latitude, and Longitude, all of which are of type double. The class includes a constructor that initializes these properties, making it useful for representing precise geographical points in applications such as drone delivery systems.
  Importance: Medium
  References:
- **Name:** src/before/DroneDelivery.Common/Models/DeliveryStatus.cs
  Details:
  Filename: src/before/DroneDelivery.Common/Models/DeliveryStatus.cs
  Description: Defines the DeliveryStatus class, which encapsulates the current status of a delivery, including its stage, last known location, and estimated times for pickup and delivery.
  Detailedsummary: The DeliveryStatus.cs file defines a class named DeliveryStatus within the DroneDelivery.Common.Models namespace. This class is designed to represent the current status of a delivery in a drone delivery system. It includes properties for the delivery stage (Stage), the last known location of the delivery (LastKnownLocation), and estimated times for both pickup (PickupETA) and delivery (DeliveryETA). The class constructor initializes these properties, ensuring that each instance of DeliveryStatus is created with complete and relevant information about a delivery's progress. This class is likely used throughout the system to track and manage the state of deliveries.
  Importance: High
  References:
    - src/before/DroneDelivery.Common/Models/DeliveryStage.cs
    - src/before/DroneDelivery.Common/Models/Location.cs
- **Name:** src/before/DroneDelivery.Common/Models/PackageGen.cs
  Details:
  Filename: src/before/DroneDelivery.Common/Models/PackageGen.cs
  Description: Defines the PackageGen class, which represents a package with properties like ID, size, tag, and weight, using JSON serialization attributes.
  Detailedsummary: The PackageGen.cs file defines a class named PackageGen within the DroneDelivery.Common.Models namespace. This class models a package entity with four properties: Id, Size, Tag, and Weight. The properties are decorated with JSON serialization attributes from the Newtonsoft.Json library, allowing them to be serialized and deserialized to and from JSON format. The Size property uses a JSON converter to handle enumeration values as strings. This class is likely used to represent package data in a drone delivery system, facilitating data exchange between different components or services.
  Importance: Medium
  References:
    - src/before/DroneDelivery.Common/Models/ContainerSize.cs
- **Name:** src/before/DroneDelivery.Common/Models/PackageInfo.cs
  Details:
  Filename: src/before/DroneDelivery.Common/Models/PackageInfo.cs
  Description: Defines the PackageInfo class, which represents information about a package, including its ID, size, weight, and tag, with JSON serialization attributes.
  Detailedsummary: The PackageInfo.cs file defines a C# class named PackageInfo within the DroneDelivery.Common.Models namespace. This class models the essential details of a package in a drone delivery system. It includes properties for PackageId, Size, Weight, and Tag, each decorated with JsonProperty attributes to specify their JSON representation. The Size property uses a JsonConverter with StringEnumConverter to handle enum serialization. This class is crucial for data interchange, ensuring that package information is correctly serialized and deserialized in JSON format, facilitating communication between different components of the system.
  Importance: High
  References:
    - src/before/DroneDelivery.Common/Models/ContainerSize.cs
- **Name:** src/before/DroneDelivery.Common/Models/UserAccount.cs
  Details:
  Filename: src/before/DroneDelivery.Common/Models/UserAccount.cs
  Description: This file defines a UserAccount class that models a user account with a user ID and an account ID, providing a constructor for initialization.
  Detailedsummary: The UserAccount.cs file is part of the DroneDelivery.Common.Models namespace and defines a UserAccount class. This class models a user account with two properties: UserId and AccountId, both of which are strings. The class provides a constructor that takes two parameters, userid and accountid, to initialize these properties. The properties are read-only, meaning they can only be set during object construction and cannot be modified afterwards. This class is likely used to represent user account information within the application, serving as a data model for user-related operations.
  Importance: Medium
  References:
- **Name:** src/before/DroneDelivery.Common/Services/DeliveryRepository.cs
  Details:
  Filename: src/before/DroneDelivery.Common/Services/DeliveryRepository.cs
  Description: This file implements the DeliveryRepository class, which provides methods for retrieving and scheduling delivery operations in a drone delivery system.
  Detailedsummary: The DeliveryRepository.cs file defines the DeliveryRepository class, which implements the IDeliveryRepository interface. It provides two main asynchronous methods: GetAsync, which is intended to retrieve a Delivery object by its ID, and ScheduleDeliveryAsync, which schedules a delivery request for a specific drone. The ScheduleDeliveryAsync method simulates accessing a common datastore and returns a successful scheduling result. The GetAsync method is not yet implemented and throws a NotImplementedException. This class is crucial for managing delivery operations within the drone delivery system.
  Importance: High
  References:

    - DroneDelivery.Common/Models/Delivery.cs
    - DroneDelivery.Common/Services/IDeliveryRepository.cs
- **Name:** src/before/DroneDelivery.Common/Services/DroneScheduler.cs
  Details:
  Filename: src/before/DroneDelivery.Common/Services/DroneScheduler.cs
  Description: This file implements the DroneScheduler class, which provides functionality to retrieve a drone ID asynchronously for a given delivery request.
  Detailedsummary: The DroneScheduler.cs file defines the DroneScheduler class, which implements the IDroneScheduler interface. It contains a method GetDroneIdAsync that takes a Delivery object as a parameter and simulates accessing a common data store to retrieve a drone ID. The method uses a utility function to simulate work and returns a test drone ID asynchronously. This class is likely part of a larger system responsible for managing drone deliveries, where scheduling and assigning drones to delivery requests is a key functionality.
  Importance: Medium
  References:
    - DroneDelivery.Common/Models/Delivery.cs
    - DroneDelivery.Common/Services/IDroneScheduler.cs
    - DroneDelivery.Common/Utility.cs
- **Name:** src/before/DroneDelivery.Common/Services/IDeliveryRepository.cs
  Details:
  Filename: src/before/DroneDelivery.Common/Services/IDeliveryRepository.cs
  Description: This file defines an interface for a delivery repository, specifying methods for retrieving and scheduling deliveries in a drone delivery system.
  Detailedsummary: The IDeliveryRepository.cs file defines an interface named IDeliveryRepository within the DroneDelivery.Common.Services namespace. This interface outlines two asynchronous methods: GetAsync, which retrieves a Delivery object based on a given ID, and ScheduleDeliveryAsync, which schedules a delivery using a Delivery object and a drone ID. These methods are crucial for managing delivery operations in a drone delivery system, providing a contract for implementing classes to handle data access and manipulation related to deliveries.
  Importance: High
  References:
    - DroneDelivery.Common.Models/Delivery.cs
- **Name:** src/before/DroneDelivery.Common/Services/IPackageProcessor.cs
  Details:
  Filename: src/before/DroneDelivery.Common/Services/IPackageProcessor.cs
  Description: Defines an interface for processing packages asynchronously in a drone delivery system.
  Detailedsummary: The IPackageProcessor.cs file defines an interface named IPackageProcessor within the DroneDelivery.Common.Services namespace. This interface declares a single asynchronous method, CreatePackageAsync, which takes a PackageInfo object as a parameter and returns a Task of type PackageGen. The purpose of this interface is to provide a contract for implementing classes to handle the creation of package objects in a drone delivery system, ensuring that the process is performed asynchronously. This is crucial for systems that require non-blocking operations, such as those involving network or I/O-bound tasks.
  Importance: High
  References:
    - DroneDelivery.Common.Models/PackageInfo.cs
    - DroneDelivery.Common.Models/PackageGen.cs
- **Name:** src/before/DroneDelivery.Common/Services/IDroneScheduler.cs
  Details:
  Filename: src/before/DroneDelivery.Common/Services/IDroneScheduler.cs
  Description: This file defines an interface for scheduling drones, specifying a method to asynchronously retrieve a drone ID for a given delivery request.
  Detailedsummary: The file contains the definition of the IDroneScheduler interface within the DroneDelivery.Common.Services namespace. This interface declares a single asynchronous method, GetDroneIdAsync, which takes a Delivery object as a parameter and returns a Task containing a string. The string represents the ID of a drone that is scheduled to handle the delivery request. This interface is likely used to standardize how drone scheduling is implemented across different parts of the application, ensuring that any class implementing this interface will provide a consistent method for obtaining a drone ID based on a delivery request.
  Importance: High
  References:
    - DroneDelivery.Common.Models/Delivery.cs
- **Name:** src/before/DroneDelivery.Common/Services/IRequestProcessor.cs
  Details:
  Filename: src/before/DroneDelivery.Common/Services/IRequestProcessor.cs
  Description: Defines an interface for processing delivery requests asynchronously in a drone delivery system.
  Detailedsummary: The file defines an interface named IRequestProcessor within the DroneDelivery.Common.Services namespace. This interface declares a single method, ProcessDeliveryRequestAsync, which takes a Delivery object as a parameter and returns a Task of type bool. This method is intended to process delivery requests asynchronously, indicating success or failure of the operation. The interface is crucial for implementing different strategies or services that handle delivery requests in a drone delivery system, promoting a decoupled and testable architecture.
  Importance: High
  References:
    - DroneDelivery.Common.Models/Delivery.cs
- **Name:** src/before/DroneDelivery.Common/Services/RequestProcessor.cs
  Details:
  Filename: src/before/DroneDelivery.Common/Services/RequestProcessor.cs
  Description: This file implements the RequestProcessor class, which processes delivery requests by coordinating package creation, drone scheduling, and delivery scheduling.
  Detailedsummary: The RequestProcessor class in this file is responsible for handling delivery requests. It uses dependency injection to obtain instances of ILogger, IPackageProcessor, IDroneScheduler, and IDeliveryRepository. The main method, ProcessDeliveryRequestAsync, logs the processing steps, creates a package, assigns a drone, and schedules the delivery. It handles exceptions and logs errors if any step fails, ensuring robust processing of delivery requests.
  Importance: High
  References:
    - DroneDelivery.Common/Models/Delivery.cs
    - DroneDelivery.Common/Services/IPackageProcessor.cs
    - DroneDelivery.Common/Services/IDroneScheduler.cs
    - DroneDelivery.Common/Services/IDeliveryRepository.cs

- **Name:** src/before/DroneDelivery.Common/Services/Utility.cs
  Details:
    Filename: src/before/DroneDelivery.Common/Services/Utility.cs
    Description: This file contains a Utility class with a static method DoWork that calculates the number of permutations for a given integer, using nested loops.
    Detailedsummary: The Utility.cs file defines a Utility class within the DroneDelivery.Common.Services namespace. It includes a static method named DoWork, which takes an integer parameter 'permutations'. The method uses four nested loops to iterate over the range of the permutations parameter, effectively counting the number of permutations possible. The result is stored in a long variable 'count', which is returned at the end of the method. This method is likely used for computational tasks that require permutation calculations.
    Importance: Medium
    References:

Project Details:

Key	Value
Id	927000385
Node id	R_kgDON0DnQQ
Name	microservices-architecture
Full name	vishalgoyal16444/microservices-architecture
Private	false
Owner	<table><tr><th>Key</th><th>Value</th></tr><tr><td>**Login**</td><td>vishalgoyal16444</td></tr><tr><td>**Id**</td><td>73927117</td></tr><tr><td>**Node id**</td><td>MDQ6VXNlcjczOTI3MTE3</td></tr><tr><td>**Avatar url**</td><td>https://avatars.githubusercontent.com/u/73927117?v=4</td></tr><tr><td>**Gravatar id**</td><td></td></tr><tr><td>**Url**</td><td>https://api.github.com/users/vishalgoyal16444</td></tr><tr><td>**Html url**</td><td>https://github.com/vishalgoyal16444</td></tr><tr><td>**Followers url**</td><td>https://api.github.com/users/vishalgoyal16444/followers</td></tr><tr><td>**Following url**</td><td>https://api.github.com/users/vishalgoyal16444/following{/other_user}</td></tr><tr><td>**Gists url**</td><td>https://api.github.com/users/vishalgoyal16444/gists{/gist_id}</td></tr><tr><td>**Starred url**</td><td>https://api.github.com/users/vishalgoyal16444/starred{/owner}{/repo}</td></tr><tr><td>**Subscriptions url**</td><td>https://api.github.com/users/vishalgoyal16444/subscriptions</td></tr><tr><td>**Organizations url**</td><td>https://api.github.com/users/vishalgoyal16444/orgs</td></tr><tr><td>**Repos url**</td><td>https://api.github.com/users/vishalgoyal16444/repos</td></tr><tr><td>**Events url**</td><td>https://api.github.com/users/vishalgoyal16444/events{/privacy}</td></tr><tr><td>**Received events url**</td><td>https://api.github.com/users/vishalgoyal16444/received_events</td></tr><tr><td>**Type**</td><td>User</td></tr><tr><td>**User view type**</td><td>public</td></tr><tr><td>**Site admin**</td><td>false</td></tr></table>
Html url	https://github.com/vishalgoyal16444/microservices-architecture
Description	Sample code for the "Decompose a monolithic application into a microservices architecture" Microsoft Learn module
Fork	true
Url	https://api.github.com/repos/vishalgoyal16444/microservices-architecture
Forks url	https://api.github.com/repos/vishalgoyal16444/microservices-architecture/forks
Keys url	https://api.github.com/repos/vishalgoyal16444/microservices-architecture/keys{/key_id}
Collaborators url	https://api.github.com/repos/vishalgoyal16444/microservices-architecture/collaborators{/collaborator}
Teams url	https://api.github.com/repos/vishalgoyal16444/microservices-architecture/teams
Hooks url	https://api.github.com/repos/vishalgoyal16444/microservices-architecture/hooks
Issue events url	https://api.github.com/repos/vishalgoyal16444/microservices-architecture/issues/events{/number}
Events url	https://api.github.com/repos/vishalgoyal16444/microservices-architecture/events
Assignees url	https://api.github.com/repos/vishalgoyal16444/microservices-architecture/assignees{/user}
Branches url	https://api.github.com/repos/vishalgoyal16444/microservices-architecture/branches{/branch}
Tags url	https://api.github.com/repos/vishalgoyal16444/microservices-architecture/tags
Blobs url	https://api.github.com/repos/vishalgoyal16444/microservices-architecture/git/blobs{/sha}
Git tags url	https://api.github.com/repos/vishalgoyal16444/microservices-architecture/git/tags{/sha}
Git refs url	https://api.github.com/repos/vishalgoyal16444/microservices-architecture/git/refs{/sha}
Trees url	https://api.github.com/repos/vishalgoyal16444/microservices-architecture/git/trees{/sha}
Statuses url	https://api.github.com/repos/vishalgoyal16444/microservices-architecture/statuses/{sha}

<table>
<tr><th>Key</th><th>Value</th></tr>
<tr><td><b>Languages url</b></td><td>https://api.github.com/repos/vishalgoyal16444/microservices-architecture/languages</td></tr>
<tr><td><b>Stargazers url</b></td><td>https://api.github.com/repos/vishalgoyal16444/microservices-architecture/stargazers</td></tr>
<tr><td><b>Contributors url</b></td><td>https://api.github.com/repos/vishalgoyal16444/microservices-architecture/contributors</td></tr>
<tr><td><b>Subscribers url</b></td><td>https://api.github.com/repos/vishalgoyal16444/microservices-architecture/subscribers</td></tr>
<tr><td><b>Subscription url</b></td><td>https://api.github.com/repos/vishalgoyal16444/microservices-architecture/subscription</td></tr>
<tr><td><b>Commits url</b></td><td>https://api.github.com/repos/vishalgoyal16444/microservices-architecture/commits{/sha}</td></tr>
<tr><td><b>Git commits url</b></td><td>https://api.github.com/repos/vishalgoyal16444/microservices-architecture/git/commits{/sha}</td></tr>
<tr><td><b>Comments url</b></td><td>https://api.github.com/repos/vishalgoyal16444/microservices-architecture/comments{/number}</td></tr>
<tr><td><b>Issue comment url</b></td><td>https://api.github.com/repos/vishalgoyal16444/microservices-architecture/issues/comments{/number}</td></tr>
<tr><td><b>Contents url</b></td><td>https://api.github.com/repos/vishalgoyal16444/microservices-architecture/contents/{+path}</td></tr>
<tr><td><b>Compare url</b></td><td>https://api.github.com/repos/vishalgoyal16444/microservices-architecture/compare/{base}...{head}</td></tr>
<tr><td><b>Merges url</b></td><td>https://api.github.com/repos/vishalgoyal16444/microservices-architecture/merges</td></tr>
<tr><td><b>Archive url</b></td><td>https://api.github.com/repos/vishalgoyal16444/microservices-architecture/{archive_format}{/ref}</td></tr>
<tr><td><b>Downloads url</b></td><td>https://api.github.com/repos/vishalgoyal16444/microservices-architecture/downloads</td></tr>
<tr><td><b>Issues url</b></td><td>https://api.github.com/repos/vishalgoyal16444/microservices-architecture/issues{/number}</td></tr>
<tr><td><b>Pulls url</b></td><td>https://api.github.com/repos/vishalgoyal16444/microservices-architecture/pulls{/number}</td></tr>
<tr><td><b>Milestones url</b></td><td>https://api.github.com/repos/vishalgoyal16444/microservices-architecture/milestones{/number}</td></tr>
<tr><td><b>Notifications url</b></td><td>https://api.github.com/repos/vishalgoyal16444/microservices-architecture/notifications{?since,all,participating}</td></tr>
<tr><td><b>Labels url</b></td><td>https://api.github.com/repos/vishalgoyal16444/microservices-architecture/labels{/name}</td></tr>
<tr><td><b>Releases url</b></td><td>https://api.github.com/repos/vishalgoyal16444/microservices-architecture/releases{/id}</td></tr>
<tr><td><b>Deployments url</b></td><td>https://api.github.com/repos/vishalgoyal16444/microservices-architecture/deployments</td></tr>
<tr><td><b>Created at</b></td><td>2025-02-04T08:26:52Z</td></tr>
<tr><td><b>Updated at</b></td><td>2025-02-04T08:26:52Z</td></tr>
<tr><td><b>Pushed at</b></td><td>2022-12-08T08:20:27Z</td></tr>
<tr><td><b>Git url</b></td><td>git://github.com/vishalgoyal16444/microservices-architecture.git</td></tr>
<tr><td><b>Ssh url</b></td><td>git@github.com:vishalgoyal16444/microservices-architecture.git</td></tr>
<tr><td><b>Clone url</b></td><td>https://github.com/vishalgoyal16444/microservices-architecture.git</td></tr>
<tr><td><b>Svn url</b></td><td>https://github.com/vishalgoyal16444/microservices-architecture</td></tr>
<tr><td><b>Homepage</b></td><td>null</td></tr>
<tr><td><b>Size</b></td><td>60</td></tr>
<tr><td><b>Stargazers count</b></td><td>0</td></tr>
<tr><td><b>Watchers count</b></td><td>0</td></tr>
<tr><td><b>Language</b></td><td>null</td></tr>
<tr><td><b>Has issues</b></td><td>false</td></tr>
<tr><td><b>Has projects</b></td><td>true</td></tr>
<tr><td><b>Has downloads</b></td><td>true</td></tr>
<tr><td><b>Has wiki</b></td><td>true</td></tr>
<tr><td><b>Has pages</b></td><td>false</td></tr>
<tr><td><b>Has discussions</b></td><td>false</td></tr>
<tr><td><b>Forks count</b></td><td>2</td></tr>
<tr><td><b>Mirror url</b></td><td>null</td></tr>
<tr><td><b>Archived</b></td><td>false</td></tr>
<tr><td><b>Disabled</b></td><td>false</td></tr>
<tr><td><b>Open issues count</b></td><td>0</td></tr>
<tr><td><b>License</b></td><td><table>
<tr><th>Key</th><th>Value</th></tr>
<tr><td><b>Key</b></td><td>cc-by-4.0</td></tr>
<tr><td><b>Name</b></td><td>Creative Commons Attribution 4.0 International</td></tr>
<tr><td><b>Spdx id</b></td><td>CC-BY-4.0</td></tr>
<tr><td><b>Url</b></td><td>https://api.github.com/licenses/cc-by-4.0</td></tr>
<tr><td><b>Node id</b></td><td>MDc6TGljZW5zZTI1</td></tr>
</table></td></tr>
<tr><td><b>Allow forking</b></td><td>true</td></tr>
<tr><td><b>Is template</b></td><td>false</td></tr>
<tr><td><b>Web commit signoff required</b></td><td>false</td></tr>
</table>

<table>
<tr><th>Key</th><th>Value</th></tr>
<tr><td><b>Topics</b></td><td>null</td></tr>
<tr><td><b>Visibility</b></td><td>public</td></tr>
<tr><td><b>Forks</b></td><td>2</td></tr>
<tr><td><b>Open issues</b></td><td>0</td></tr>
<tr><td><b>Watchers</b></td><td>0</td></tr>
<tr><td><b>Default branch</b></td><td>master</td></tr>
<tr><td><b>Permissions</b></td><td><table>
<tr><th>Key</th><th>Value</th></tr>
<tr><td><b>Admin</b></td><td>true</td></tr>
<tr><td><b>Maintain</b></td><td>true</td></tr>
<tr><td><b>Push</b></td><td>true</td></tr>
<tr><td><b>Triage</b></td><td>true</td></tr>
<tr><td><b>Pull</b></td><td>true</td></tr>
</table></td></tr>
<tr><td><b>Temp clone token</b></td><td></td></tr>
<tr><td><b>Allow squash merge</b></td><td>true</td></tr>
<tr><td><b>Allow merge commit</b></td><td>true</td></tr>
<tr><td><b>Allow rebase merge</b></td><td>true</td></tr>
<tr><td><b>Allow auto merge</b></td><td>false</td></tr>
<tr><td><b>Delete branch on merge</b></td><td>false</td></tr>
<tr><td><b>Allow update branch</b></td><td>false</td></tr>
<tr><td><b>Use squash pr title as default</b></td><td>false</td></tr>
<tr><td><b>Squash merge commit message</b></td><td>COMMIT_MESSAGES</td></tr>
<tr><td><b>Squash merge commit title</b></td><td>COMMIT_OR_PR_TITLE</td></tr>
<tr><td><b>Merge commit message</b></td><td>PR_TITLE</td></tr>
<tr><td><b>Merge commit title</b></td><td>MERGE_MESSAGE</td></tr>
<tr><td></td><td><table>
<tr><th>Key</th><th>Value</th></tr>
<tr><td><b>Id</b></td><td>187660505</td></tr>
<tr><td><b>Node id</b></td><td>MDEwOlJlcG9zaXRvcnkxODc2NjA1MDU=</td></tr>
<tr><td><b>Name</b></td><td>mslearn-microservices-architecture</td></tr>
<tr><td><b>Full name</b></td><td>MicrosoftDocs/mslearn-microservices-architecture</td></tr>
<tr><td><b>Private</b></td><td>false</td></tr>
<tr><td><b>Owner</b></td><td><table>
<tr><th>Key</th><th>Value</th></tr>
<tr><td><b>Login</b></td><td>MicrosoftDocs</td></tr>
<tr><td><b>Id</b></td><td>22479449</td></tr>
<tr><td><b>Node id</b></td><td>MDEyOk9yZ2FuaXphdGlvbjIyNDc5NDQ5</td></tr>
<tr><td><b>Avatar url</b></td><td>https://avatars.githubusercontent.com/u/22479449?v=4</td></tr>
<tr><td><b>Gravatar id</b></td><td></td></tr>
<tr><td><b>Url</b></td><td>https://api.github.com/users/MicrosoftDocs</td></tr>
<tr><td><b>Html url</b></td><td>https://github.com/MicrosoftDocs</td></tr>
<tr><td><b>Followers url</b></td><td>https://api.github.com/users/MicrosoftDocs/followers</td></tr>
<tr><td><b>Following url</b></td><td>https://api.github.com/users/MicrosoftDocs/following{/other_user}</td></tr>
<tr><td><b>Gists url</b></td><td>https://api.github.com/users/MicrosoftDocs/gists{/gist_id}</td></tr>
<tr><td><b>Starred url</b></td><td>https://api.github.com/users/MicrosoftDocs/starred{/owner}{/repo}</td></tr>
<tr><td><b>Subscriptions url</b></td><td>https://api.github.com/users/MicrosoftDocs/subscriptions</td></tr>
<tr><td><b>Organizations url</b></td><td>https://api.github.com/users/MicrosoftDocs/orgs</td></tr>
<tr><td><b>Repos url</b></td><td>https://api.github.com/users/MicrosoftDocs/repos</td></tr>
<tr><td><b>Events url</b></td><td>https://api.github.com/users/MicrosoftDocs/events{/privacy}</td></tr>
<tr><td><b>Received events url</b></td><td>https://api.github.com/users/MicrosoftDocs/received_events</td></tr>
<tr><td><b>Type</b></td><td>Organization</td></tr>
<tr><td><b>User view type</b></td><td>public</td></tr>
<tr><td><b>Site admin</b></td><td>false</td></tr>
</table></td></tr>
<tr><td><b>Html url</b></td><td>https://github.com/MicrosoftDocs/mslearn-microservices-architecture</td></tr>
</table></td></tr>
</table>

Key	Key	Value Value
Parent	**Description**	Sample code for the "Decompose a monolithic application into a microservices architecture" Microsoft Learn module
	Fork	false
	Url	https://api.github.com/repos/MicrosoftDocs/mslearn-microservices-architecture
	Forks url	https://api.github.com/repos/MicrosoftDocs/mslearn-microservices-architecture/forks
	Keys url	https://api.github.com/repos/MicrosoftDocs/mslearn-microservices-architecture/keys{/key_id}
	Collaborators url	https://api.github.com/repos/MicrosoftDocs/mslearn-microservices-architecture/collaborators{/collaborator}
	Teams url	https://api.github.com/repos/MicrosoftDocs/mslearn-microservices-architecture/teams
	Hooks url	https://api.github.com/repos/MicrosoftDocs/mslearn-microservices-architecture/hooks
	Issue events url	https://api.github.com/repos/MicrosoftDocs/mslearn-microservices-architecture/issues/events{/number}
	Events url	https://api.github.com/repos/MicrosoftDocs/mslearn-microservices-architecture/events
	Assignees url	https://api.github.com/repos/MicrosoftDocs/mslearn-microservices-architecture/assignees{/user}
	Branches url	https://api.github.com/repos/MicrosoftDocs/mslearn-microservices-architecture/branches{/branch}
	Tags url	https://api.github.com/repos/MicrosoftDocs/mslearn-microservices-architecture/tags
	Blobs url	https://api.github.com/repos/MicrosoftDocs/mslearn-microservices-architecture/git/blobs{/sha}
	Git tags url	https://api.github.com/repos/MicrosoftDocs/mslearn-microservices-architecture/git/tags{/sha}
	Git refs url	https://api.github.com/repos/MicrosoftDocs/mslearn-microservices-architecture/git/refs{/sha}
	Trees url	https://api.github.com/repos/MicrosoftDocs/mslearn-microservices-architecture/git/trees{/sha}
	Statuses url	https://api.github.com/repos/MicrosoftDocs/mslearn-microservices-architecture/statuses/{sha}
	Languages url	https://api.github.com/repos/MicrosoftDocs/mslearn-microservices-architecture/languages
	Stargazers url	https://api.github.com/repos/MicrosoftDocs/mslearn-microservices-architecture/stargazers
	Contributors url	https://api.github.com/repos/MicrosoftDocs/mslearn-microservices-architecture/contributors
	Subscribers url	https://api.github.com/repos/MicrosoftDocs/mslearn-microservices-architecture/subscribers
	Subscription url	https://api.github.com/repos/MicrosoftDocs/mslearn-microservices-architecture/subscription
	Commits url	https://api.github.com/repos/MicrosoftDocs/mslearn-microservices-architecture/commits{/sha}
	Git commits url	https://api.github.com/repos/MicrosoftDocs/mslearn-microservices-architecture/git/commits{/sha}
	Comments url	https://api.github.com/repos/MicrosoftDocs/mslearn-microservices-architecture/comments{/number}
	Issue comment url	https://api.github.com/repos/MicrosoftDocs/mslearn-microservices-architecture/issues/comments{/number}
	Contents url	https://api.github.com/repos/MicrosoftDocs/mslearn-microservices-architecture/contents/{+path}
	Compare url	https://api.github.com/repos/MicrosoftDocs/mslearn-microservices-architecture/compare/{base}...{head}
	Merges url	https://api.github.com/repos/MicrosoftDocs/mslearn-microservices-architecture/merges
	Archive url	https://api.github.com/repos/MicrosoftDocs/mslearn-microservices-architecture/{archive_format}{/ref}
	Downloads url	https://api.github.com/repos/MicrosoftDocs/mslearn-microservices-architecture/downloads
	Issues url	https://api.github.com/repos/MicrosoftDocs/mslearn-microservices-architecture/issues{/number}
	Pulls url	https://api.github.com/repos/MicrosoftDocs/mslearn-microservices-architecture/pulls{/number}
	Milestones url	https://api.github.com/repos/MicrosoftDocs/mslearn-microservices-architecture/milestones{/number}
	Notifications url	https://api.github.com/repos/MicrosoftDocs/mslearn-microservices-architecture/notifications{?since,all,participating}
	Labels url	https://api.github.com/repos/MicrosoftDocs/mslearn-microservices-architecture/labels{/name}
	Releases url	https://api.github.com/repos/MicrosoftDocs/mslearn-microservices-architecture/releases{/id}
	Deployments url	https://api.github.com/repos/MicrosoftDocs/mslearn-microservices-architecture/deployments
	Created at	2019-05-20T14:49:57Z
	Updated at	2024-05-25T11:19:54Z
	Pushed at	2022-12-08T08:20:27Z
	Git url	git://github.com/MicrosoftDocs/mslearn-microservices-architecture.git
	Ssh url	git@github.com:MicrosoftDocs/mslearn-microservices-architecture.git
	Clone url	https://github.com/MicrosoftDocs/mslearn-microservices-architecture.git
	Svn url	https://github.com/MicrosoftDocs/mslearn-microservices-architecture
	Homepage	null
	Size	60
	Stargazers count	15
	Watchers count	15

<table>
<tr><th>Key</th><th>Key</th><th>Value</th><th>Value</th></tr>
<tr><td></td><td><b>Language</b></td><td colspan="2">C#</td></tr>
<tr><td></td><td><b>Has issues</b></td><td colspan="2">true</td></tr>
<tr><td></td><td><b>Has projects</b></td><td colspan="2">true</td></tr>
<tr><td></td><td><b>Has downloads</b></td><td colspan="2">true</td></tr>
<tr><td></td><td><b>Has wiki</b></td><td colspan="2">true</td></tr>
<tr><td></td><td><b>Has pages</b></td><td colspan="2">false</td></tr>
<tr><td></td><td><b>Has discussions</b></td><td colspan="2">false</td></tr>
<tr><td></td><td><b>Forks count</b></td><td colspan="2">21</td></tr>
<tr><td></td><td><b>Mirror url</b></td><td colspan="2">null</td></tr>
<tr><td></td><td><b>Archived</b></td><td colspan="2">false</td></tr>
<tr><td></td><td><b>Disabled</b></td><td colspan="2">false</td></tr>
<tr><td></td><td><b>Open issues count</b></td><td colspan="2">3</td></tr>
<tr><td></td><td><b>License</b></td><td colspan="2"><table>
<tr><th>Key</th><th>Value</th></tr>
<tr><td><b>Key</b></td><td>cc-by-4.0</td></tr>
<tr><td><b>Name</b></td><td>Creative Commons Attribution 4.0 International</td></tr>
<tr><td><b>Spdx id</b></td><td>CC-BY-4.0</td></tr>
<tr><td><b>Url</b></td><td>https://api.github.com/licenses/cc-by-4.0</td></tr>
<tr><td><b>Node id</b></td><td>MDc6TGljZW5zZTI1</td></tr>
</table></td></tr>
<tr><td></td><td><b>Allow forking</b></td><td colspan="2">true</td></tr>
<tr><td></td><td><b>Is template</b></td><td colspan="2">false</td></tr>
<tr><td></td><td><b>Web commit signoff required</b></td><td colspan="2">false</td></tr>
<tr><td></td><td><b>Topics</b></td><td colspan="2">null</td></tr>
<tr><td></td><td><b>Visibility</b></td><td colspan="2">public</td></tr>
<tr><td></td><td><b>Forks</b></td><td colspan="2">21</td></tr>
<tr><td></td><td><b>Open issues</b></td><td colspan="2">3</td></tr>
<tr><td></td><td><b>Watchers</b></td><td colspan="2">15</td></tr>
<tr><td></td><td><b>Default branch</b></td><td colspan="2">master</td></tr>
</table>

Key	Value
Id	187660505
Node id	MDEwOlJlcG9zaXRvcnkxODc2NjA1MDU=
Name	mslearn-microservices-architecture
Full name	MicrosoftDocs/mslearn-microservices-architecture
Private	false

<table>
<tr><th>Key</th><th></th><th>Key</th><th>Value Value</th><th></th></tr>
<tr><td>Source</td><td></td><td>Owner</td><td><table>
<tr><th>Key</th><th>Value</th></tr>
<tr><td>Login</td><td>MicrosoftDocs</td></tr>
<tr><td>Id</td><td>22479449</td></tr>
<tr><td>Node id</td><td>MDEyOk9yZ2FuaXphdGlvbjIyNDc5NDQ5</td></tr>
<tr><td>Avatar url</td><td>https://avatars.githubusercontent.com/u/22479449?v=4</td></tr>
<tr><td>Gravatar id</td><td></td></tr>
<tr><td>Url</td><td>https://api.github.com/users/MicrosoftDocs</td></tr>
<tr><td>Html url</td><td>https://github.com/MicrosoftDocs</td></tr>
<tr><td>Followers url</td><td>https://api.github.com/users/MicrosoftDocs/followers</td></tr>
<tr><td>Following url</td><td>https://api.github.com/users/MicrosoftDocs/following{/other_user}</td></tr>
<tr><td>Gists url</td><td>https://api.github.com/users/MicrosoftDocs/gists{/gist_id}</td></tr>
<tr><td>Starred url</td><td>https://api.github.com/users/MicrosoftDocs/starred{/owner}{/repo}</td></tr>
<tr><td>Subscriptions url</td><td>https://api.github.com/users/MicrosoftDocs/subscriptions</td></tr>
<tr><td>Organizations url</td><td>https://api.github.com/users/MicrosoftDocs/orgs</td></tr>
<tr><td>Repos url</td><td>https://api.github.com/users/MicrosoftDocs/repos</td></tr>
<tr><td>Events url</td><td>https://api.github.com/users/MicrosoftDocs/events{/privacy}</td></tr>
<tr><td>Received events url</td><td>https://api.github.com/users/MicrosoftDocs/received_events</td></tr>
<tr><td>Type</td><td>Organization</td></tr>
<tr><td>User view type</td><td>public</td></tr>
<tr><td>Site admin</td><td>false</td></tr>
</table></td><td></td></tr>
<tr><td></td><td></td><td>Html url</td><td>https://github.com/MicrosoftDocs/mslearn-microservices-architecture</td><td></td></tr>
<tr><td></td><td></td><td>Description</td><td>Sample code for the "Decompose a monolithic application into a microservices architecture" Microsoft Learn module</td><td></td></tr>
<tr><td></td><td></td><td>Fork</td><td>false</td><td></td></tr>
<tr><td></td><td></td><td>Url</td><td>https://api.github.com/repos/MicrosoftDocs/mslearn-microservices-architecture</td><td></td></tr>
<tr><td></td><td></td><td>Forks url</td><td>https://api.github.com/repos/MicrosoftDocs/mslearn-microservices-architecture/forks</td><td></td></tr>
<tr><td></td><td></td><td>Keys url</td><td>https://api.github.com/repos/MicrosoftDocs/mslearn-microservices-architecture/keys{/key_id}</td><td></td></tr>
<tr><td></td><td></td><td>Collaborators url</td><td>https://api.github.com/repos/MicrosoftDocs/mslearn-microservices-architecture/collaborators{/collaborator}</td><td></td></tr>
<tr><td></td><td></td><td>Teams url</td><td>https://api.github.com/repos/MicrosoftDocs/mslearn-microservices-architecture/teams</td><td></td></tr>
<tr><td></td><td></td><td>Hooks url</td><td>https://api.github.com/repos/MicrosoftDocs/mslearn-microservices-architecture/hooks</td><td></td></tr>
<tr><td></td><td></td><td>Issue events url</td><td>https://api.github.com/repos/MicrosoftDocs/mslearn-microservices-architecture/issues/events{/number}</td><td></td></tr>
<tr><td></td><td></td><td>Events url</td><td>https://api.github.com/repos/MicrosoftDocs/mslearn-microservices-architecture/events</td><td></td></tr>
<tr><td></td><td></td><td>Assignees url</td><td>https://api.github.com/repos/MicrosoftDocs/mslearn-microservices-architecture/assignees{/user}</td><td></td></tr>
<tr><td></td><td></td><td>Branches url</td><td>https://api.github.com/repos/MicrosoftDocs/mslearn-microservices-architecture/branches{/branch}</td><td></td></tr>
<tr><td></td><td></td><td>Tags url</td><td>https://api.github.com/repos/MicrosoftDocs/mslearn-microservices-architecture/tags</td><td></td></tr>
<tr><td></td><td></td><td>Blobs url</td><td>https://api.github.com/repos/MicrosoftDocs/mslearn-microservices-architecture/git/blobs{/sha}</td><td></td></tr>
<tr><td></td><td></td><td>Git tags url</td><td>https://api.github.com/repos/MicrosoftDocs/mslearn-microservices-architecture/git/tags{/sha}</td><td></td></tr>
<tr><td></td><td></td><td>Git refs url</td><td>https://api.github.com/repos/MicrosoftDocs/mslearn-microservices-architecture/git/refs{/sha}</td><td></td></tr>
<tr><td></td><td></td><td>Trees url</td><td>https://api.github.com/repos/MicrosoftDocs/mslearn-microservices-architecture/git/trees{/sha}</td><td></td></tr>
<tr><td></td><td></td><td>Statuses url</td><td>https://api.github.com/repos/MicrosoftDocs/mslearn-microservices-architecture/statuses/{sha}</td><td></td></tr>
<tr><td></td><td></td><td>Languages url</td><td>https://api.github.com/repos/MicrosoftDocs/mslearn-microservices-architecture/languages</td><td></td></tr>
<tr><td></td><td></td><td>Stargazers url</td><td>https://api.github.com/repos/MicrosoftDocs/mslearn-microservices-architecture/stargazers</td><td></td></tr>
<tr><td></td><td></td><td>Contributors url</td><td>https://api.github.com/repos/MicrosoftDocs/mslearn-microservices-architecture/contributors</td><td></td></tr>
<tr><td></td><td></td><td>Subscribers url</td><td>https://api.github.com/repos/MicrosoftDocs/mslearn-microservices-architecture/subscribers</td><td></td></tr>
<tr><td></td><td></td><td>Subscription url</td><td>https://api.github.com/repos/MicrosoftDocs/mslearn-microservices-architecture/subscription</td><td></td></tr>
<tr><td></td><td></td><td>Commits url</td><td>https://api.github.com/repos/MicrosoftDocs/mslearn-microservices-architecture/commits{/sha}</td><td></td></tr>
<tr><td></td><td></td><td>Git commits url</td><td>https://api.github.com/repos/MicrosoftDocs/mslearn-microservices-architecture/git/commits{/sha}</td><td></td></tr>
<tr><td></td><td></td><td>Comments url</td><td>https://api.github.com/repos/MicrosoftDocs/mslearn-microservices-architecture/comments{/number}</td><td></td></tr>
<tr><td></td><td></td><td>Issue comment url</td><td>https://api.github.com/repos/MicrosoftDocs/mslearn-microservices-architecture/issues/comments{/number}</td><td></td></tr>
<tr><td></td><td></td><td>Contents url</td><td>https://api.github.com/repos/MicrosoftDocs/mslearn-microservices-architecture/contents/{+path}</td><td></td></tr>
<tr><td></td><td></td><td>Compare url</td><td>https://api.github.com/repos/MicrosoftDocs/mslearn-microservices-architecture/compare/{base}...{head}</td><td></td></tr>
<tr><td></td><td></td><td>Merges url</td><td>https://api.github.com/repos/MicrosoftDocs/mslearn-microservices-architecture/merges</td><td></td></tr>
</table>

Key	Value
Archive url	https://api.github.com/repos/MicrosoftDocs/mslearn-microservices-architecture/{archive_format}{/ref}
Downloads url	https://api.github.com/repos/MicrosoftDocs/mslearn-microservices-architecture/downloads
Issues url	https://api.github.com/repos/MicrosoftDocs/mslearn-microservices-architecture/issues{/number}
Pulls url	https://api.github.com/repos/MicrosoftDocs/mslearn-microservices-architecture/pulls{/number}
Milestones url	https://api.github.com/repos/MicrosoftDocs/mslearn-microservices-architecture/milestones{/number}
Notifications url	https://api.github.com/repos/MicrosoftDocs/mslearn-microservices-architecture/notifications{?since,all,participating}
Labels url	https://api.github.com/repos/MicrosoftDocs/mslearn-microservices-architecture/labels{/name}
Releases url	https://api.github.com/repos/MicrosoftDocs/mslearn-microservices-architecture/releases{/id}
Deployments url	https://api.github.com/repos/MicrosoftDocs/mslearn-microservices-architecture/deployments
Created at	2019-05-20T14:49:57Z
Updated at	2024-05-25T11:19:54Z
Pushed at	2022-12-08T08:20:27Z
Git url	git://github.com/MicrosoftDocs/mslearn-microservices-architecture.git
Ssh url	git@github.com:MicrosoftDocs/mslearn-microservices-architecture.git
Clone url	https://github.com/MicrosoftDocs/mslearn-microservices-architecture.git
Svn url	https://github.com/MicrosoftDocs/mslearn-microservices-architecture
Homepage	null
Size	60
Stargazers count	15
Watchers count	15
Language	C#
Has issues	true
Has projects	true
Has downloads	true
Has wiki	true
Has pages	false
Has discussions	false
Forks count	21
Mirror url	null
Archived	false
Disabled	false
Open issues count	3
License	
Allow forking	true
Is template	false
Web commit signoff required	false
Topics	null
Visibility	public
Forks	21
Open issues	3
Watchers	15
Default branch	master

Key	Value
Key	cc-by-4.0
Name	Creative Commons Attribution 4.0 International
Spdx id	CC-BY-4.0
Url	https://api.github.com/licenses/cc-by-4.0
Node id	MDc6TGljZW5zZTI1

Key	Value
Security and analysis	
Network count	21
Subscribers count	0

Key	Value
Secret scanning	
Secret scanning push protection	
Dependabot security updates	
Secret scanning non provider patterns	
Secret scanning validity checks	

Key	Value
Status	enabled

Key	Value
Status	enabled

Key	Value
Status	disabled

Key	Value
Status	disabled

Key	Value
Status	disabled