High-Level Design

Design Restrictions:

- Given that this application is customer-facing, we need to prioritize usability, performance, and scalability.
- We'll employ a microservice architecture to manage the business logic and integrate with external systems (offer API and contracts REST API using “contracts/search” endpoint).
- The backend should provide REST endpoints for communication.

Design:

- The backend will be a Java-based Spring Boot application.
- This microservice will manage operations related to offers, status transitions, and contract retrieval.
- The microservice will interact with the offer API and contracts REST API.

Components:

- Controller Layer: Provides REST endpoints for listing offers and their statuses.
- Service Layer: Implements business rules, including status definition and contract retrieval.
- Repository Layer: AS-IS: Using mocks
  TO-DO: Handles data storage using an in-memory database.
- Integration Layer: Interfaces with external APIs (offer API and contracts REST API).

Business Rules:

- Retrieve offers from the offer API.
- Retrieve contracts based on offer IDs from the contracts REST API.
- Determine offer status (CREATED, IN PROGRESS, REJECTED, CONVERTED) based on creation, rejection dates and existing contract.

Scalability and Maintainability:

- As features are added, consider modularizing the microservice (e.g., separate modules for offers and contracts).
- Implement proper exception handling and logging. (TODO)
- Thoroughly document the code and design decisions. (IN PROGRESS)

Backend Components:

- Controller Layer

  The Controller Layer is crucial for handling client requests, directing them to the appropriate services, performing initial processing, and formatting the responses.

- OfferController
  Maps incoming HTTP requests to appropriate handler methods.
  Uses services like OfferService to perform business logic operations related to offers.
  Ensure that the controller remains thin by delegating business logic to the service layer.

- Service Layer

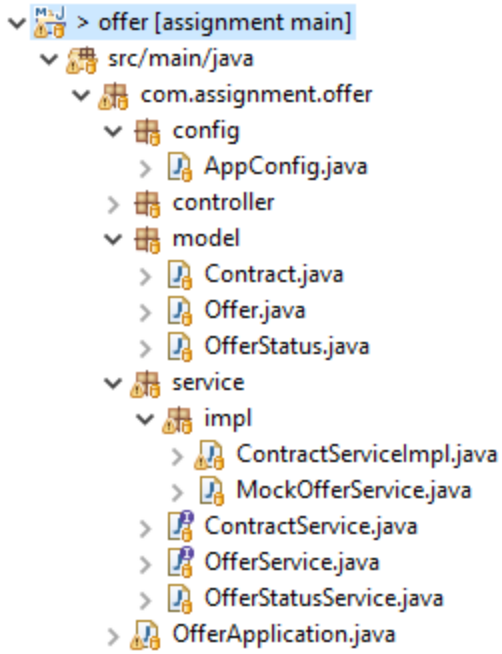
  Implement service classes to handle business logic.

- OfferService:
  As offers are retrieved from an offer API for which the integration mechanism is not yet known, we need to design the system in a way that allows for future integration of the offer API without requiring significant changes to the existing codebase. This can be achieved by abstracting the offer fetching mechanism and providing a placeholder or mock implementation that can be replaced once the details of the offer API are known.

- ContractService:
  This service interacts with the contracts REST API using the “contracts/search” endpoint. It searches for contracts based on the offer ID received from the OfferService.

- StatusService:
  This service uses information from OfferService and ContractService to determine the offer status.

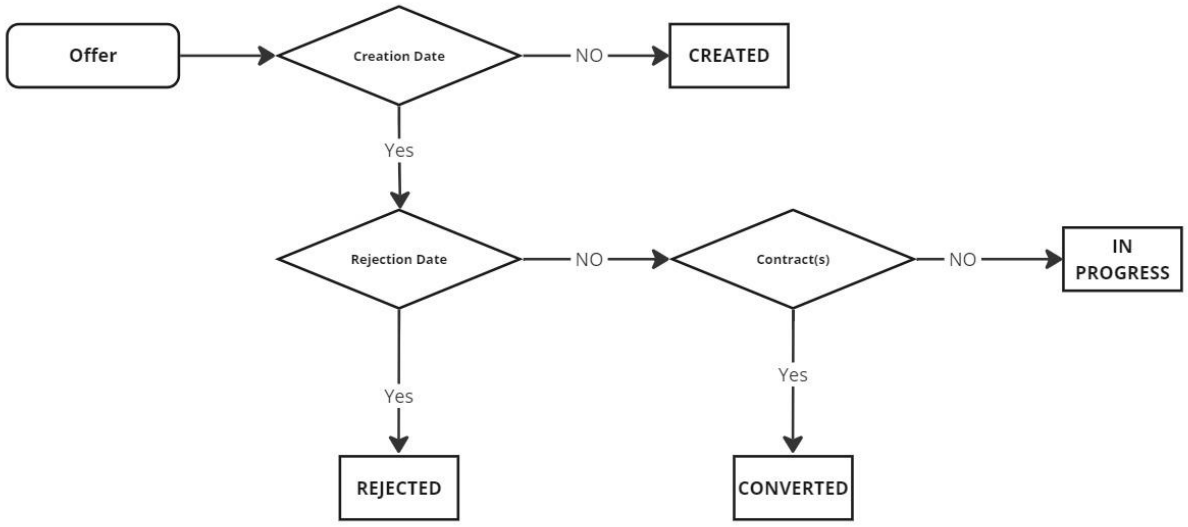
Project Structure



Benefits of this Design:

- Clear separation of concerns: Each service has a well-defined responsibility.
- Loose coupling: Services are independent and can be modified without affecting others.
- Flexibility: New features can be implemented by adding new services.
- Scalability: Services can be scaled independently based on load.

Business Rules



 If an offer exists, has no creation date, the status is "CREATED".
Justification: An offer that exists but has no creation date implies it has been initiated but not yet formally processed or started. Hence, it is in the "CREATED" state.

 If an offer exists, has a creation date, no rejection date, and no contract found linked to this offer, the status is "IN PROGRESS".
Justification: An offer with a creation date signifies that the offer process has begun. If it has no rejection date and no linked contract, it indicates that the offer is currently being worked on or evaluated, thus it is "IN PROGRESS".

 If an offer exists and has a rejection date, the status is "REJECTED".
Justification: The presence of a rejection date clearly indicates that the offer has been reviewed and explicitly rejected. Therefore, its status should be "REJECTED".

 If an offer exists and a contract is found linked to the offer ID, the status is "CONVERTED".
Justification: A linked contract signifies that the offer has been accepted and converted into a contractual agreement. This is a natural progression from an offer to a formal contract, hence the status "CONVERTED".

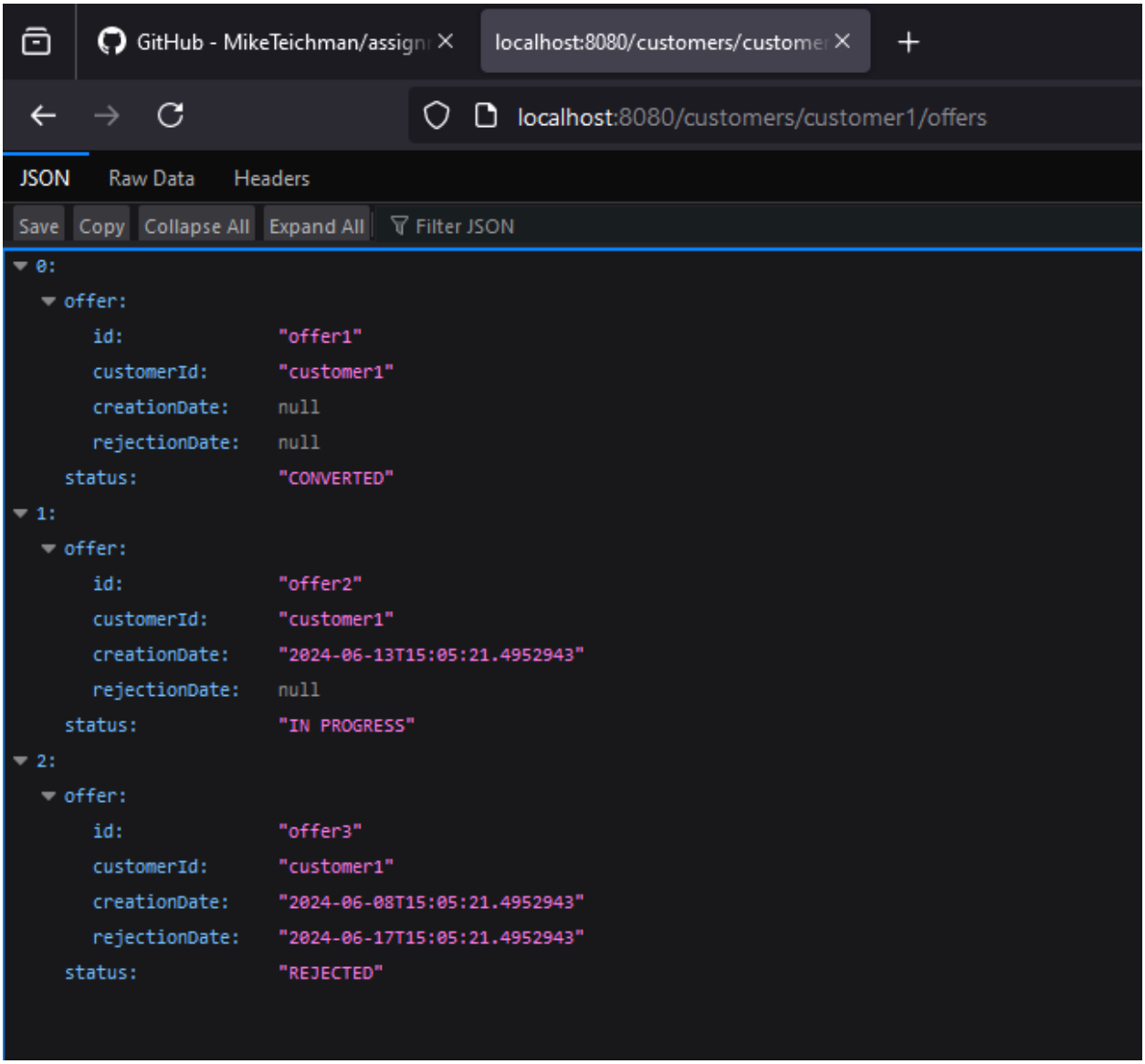
Related to business rules, here are open points to be discussed with business:

- How to make a distinction between CREATED and IN PROGRESS status?
- How to handle exceptions or errors like offers with creation date, rejection date and a contract linked to the offer.

Running the application

- Request: http://localhost:8080/customers/customer1/offers
- Response:


```
[{ "offer": { "id": "offer1",
              "customerId": "customer1",
              "creationDate": null,
              "rejectionDate": null},
  "status": "CONVERTED"},
{

  "offer": { “id": "offer2",
              "customerId": "customer1",
              “creationDate": "2024-06-13T14:56:17.2348707",
              "rejectionDate": null      },
  "status": "IN PROGRESS"  },

{

  "offer": { "id": "offer3",
              "customerId": "customer1",
              "creationDate": "2024-06-08T14:56:17.2348707",
              "rejectionDate": "2024-06-17T14:56:17.2348707"  },

  "status": "REJECTED"

}

]
```


GitHub - MikeTeichman/assign
localhost:8080/customers/custome
localhost:8080/customers/customer1/offers
JSON
Raw Data
Headers
Save
Copy
Collapse All
Expand All
Filter JSON
0:
offer:
id: "offer1"
customerId: "customer1"
creationDate: null
rejectionDate: null
status: "CONVERTED"
1:
offer:
id: "offer2"
customerId: "customer1"
creationDate: "2024-06-13T15:05:21.4952943"
rejectionDate: null
status: "IN PROGRESS"
2:
offer:
id: "offer3"
customerId: "customer1"
creationDate: "2024-06-08T15:05:21.4952943"
rejectionDate: "2024-06-17T15:05:21.4952943"
status: "REJECTED"