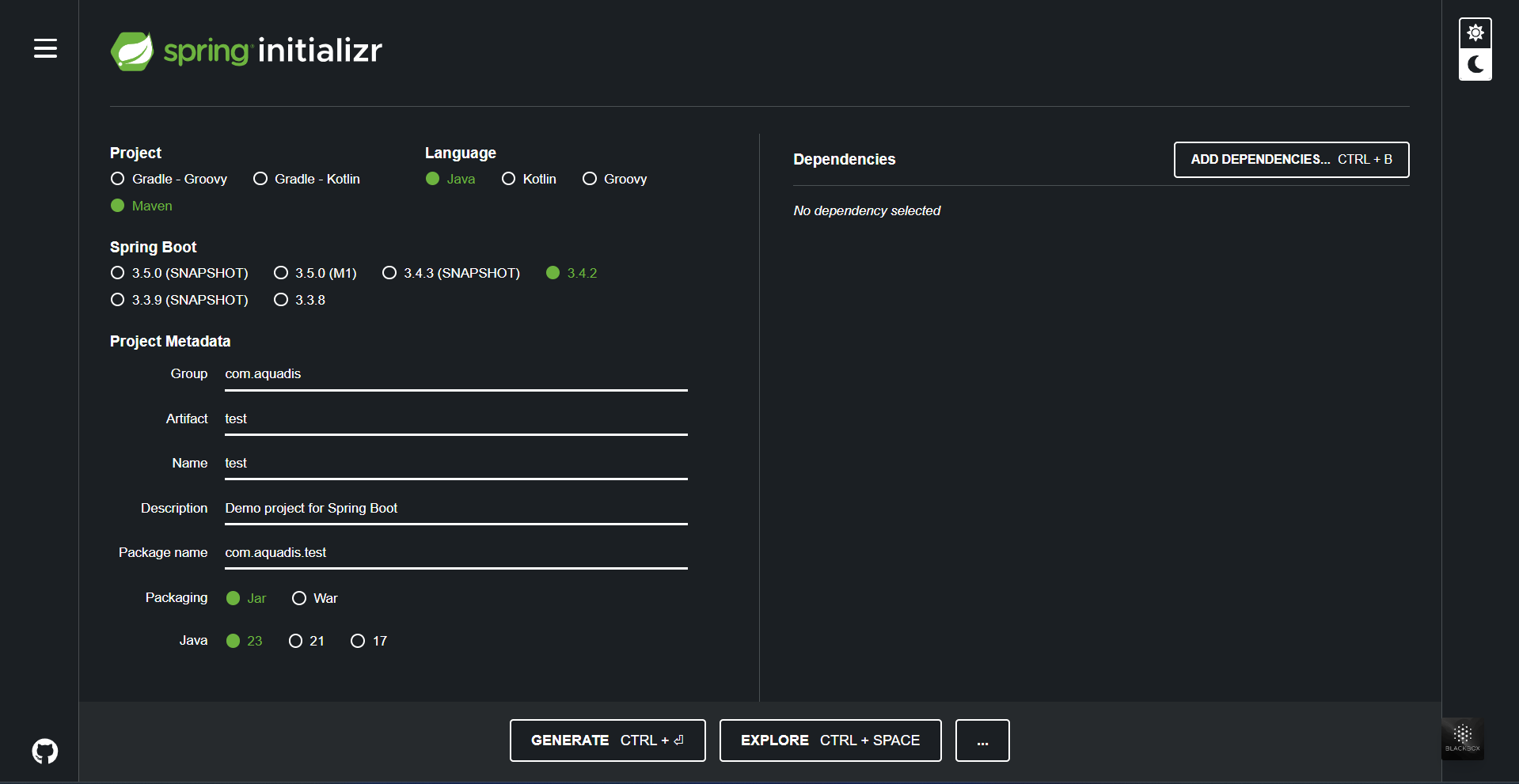
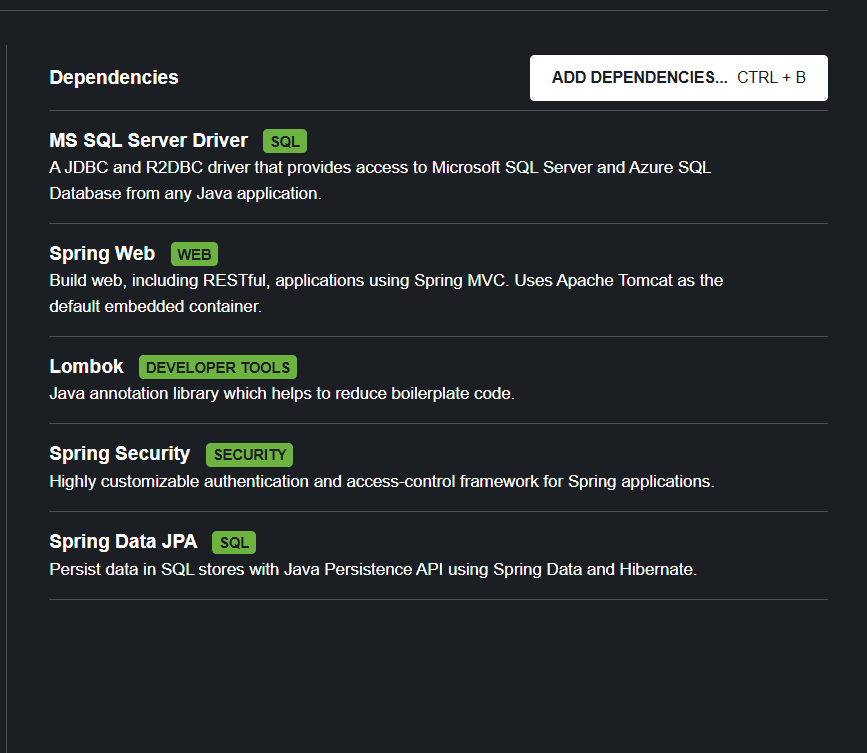
Aquadis Spring Boot + Angular Test Steps

# Spring and DB Initialization:

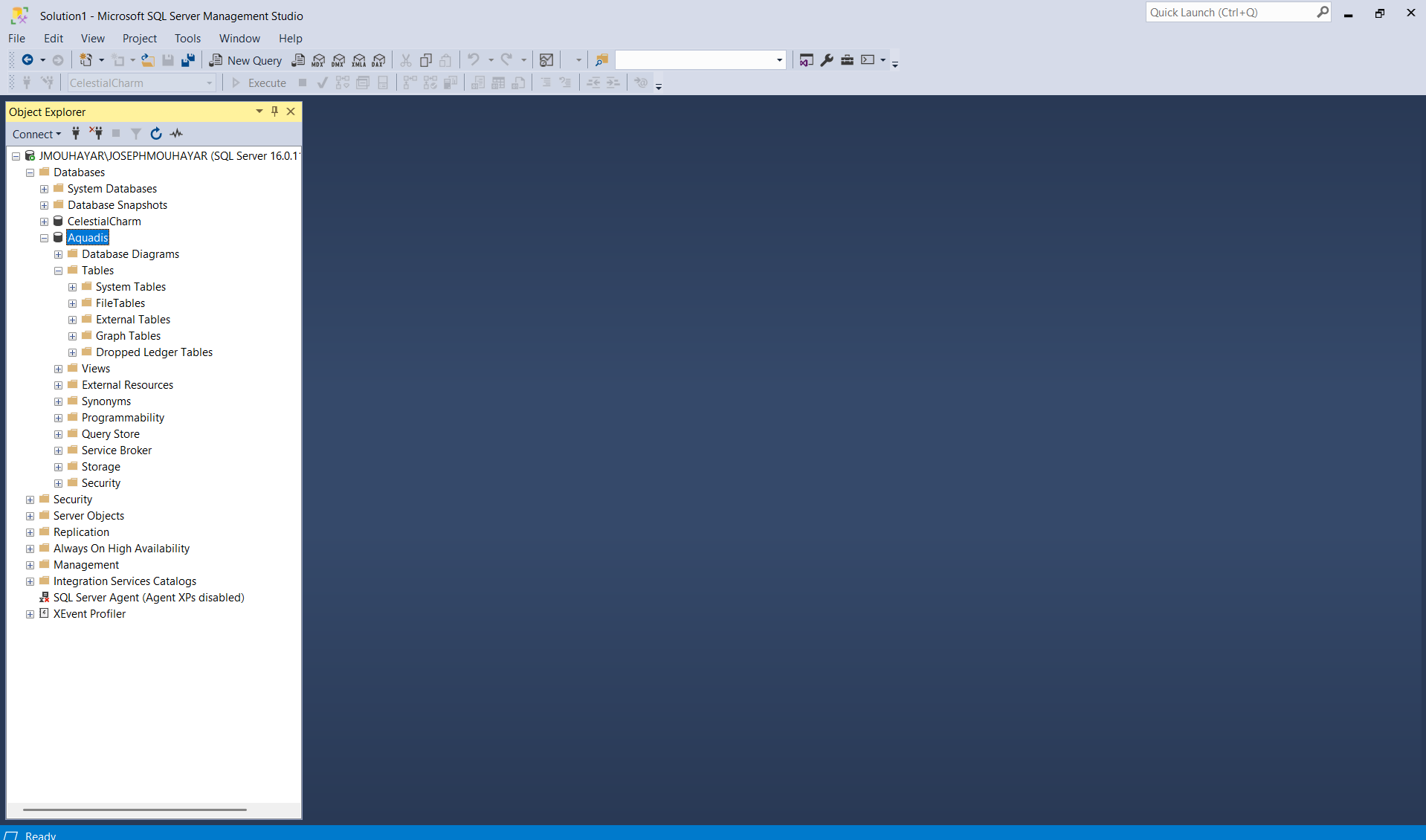
* + Create a spring project in spring initializr



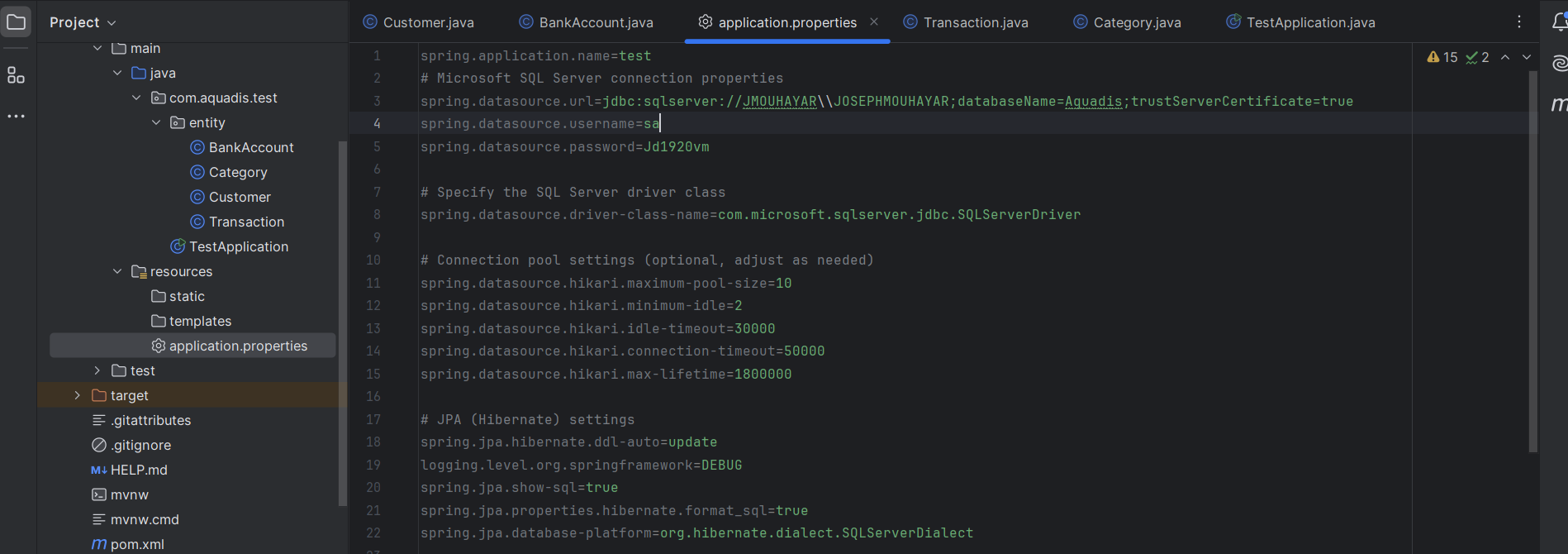
* + Add Dependencies:
    - MS SQL Server Driver (to connect to Microsoft SQL Server)
    - Spring Web (to build restful APIs)
    - Lombok (to create getters and setters)
    - Spring Security (for authentication and security against threats)
    - Spring Data JPA (to allow easy interaction with relational databases using repository interfaces) (similar to entity framework in .NET)



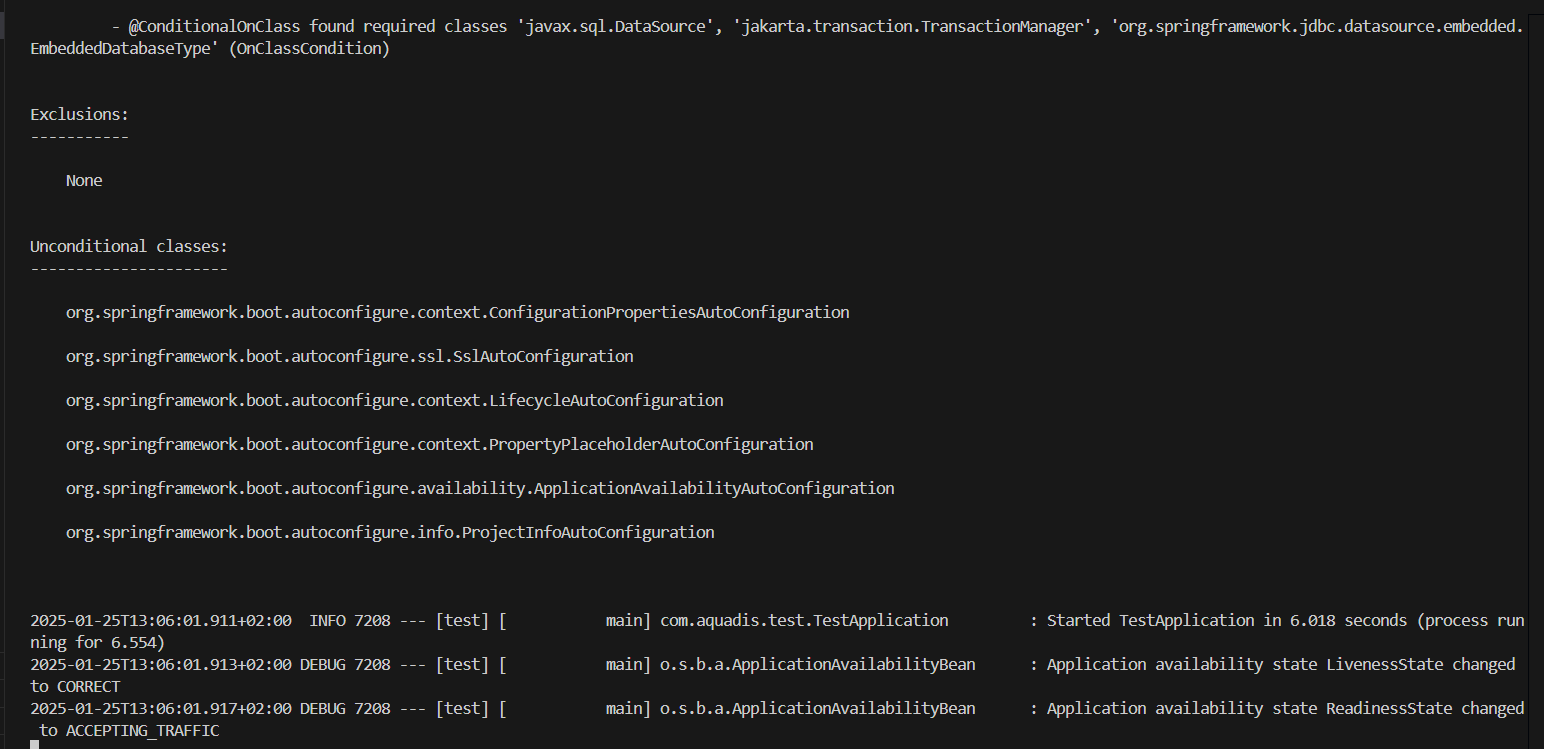
* + Create a new Database
    - Create a new Database in SSMS (Aquadis)



* + - Link the SQL Server database in my spring project (by updating the application.properties)

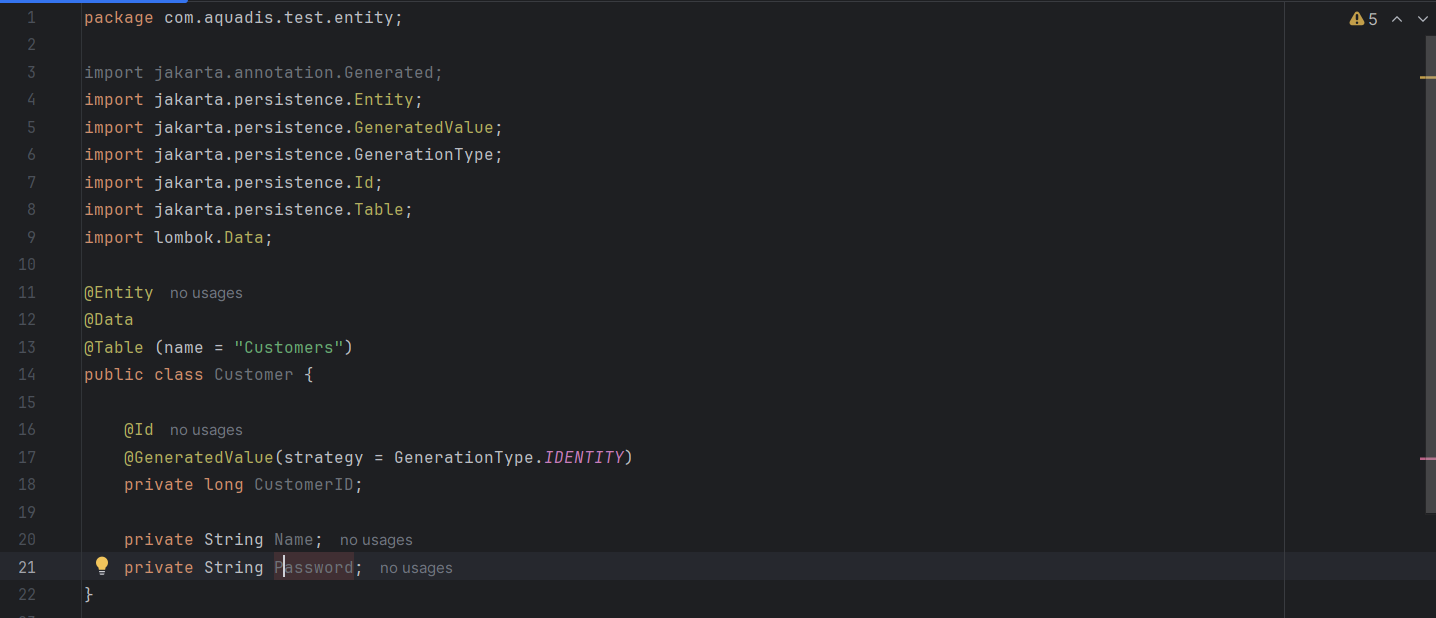


* + - Check connection by running [mvn clean spring-boot:run]

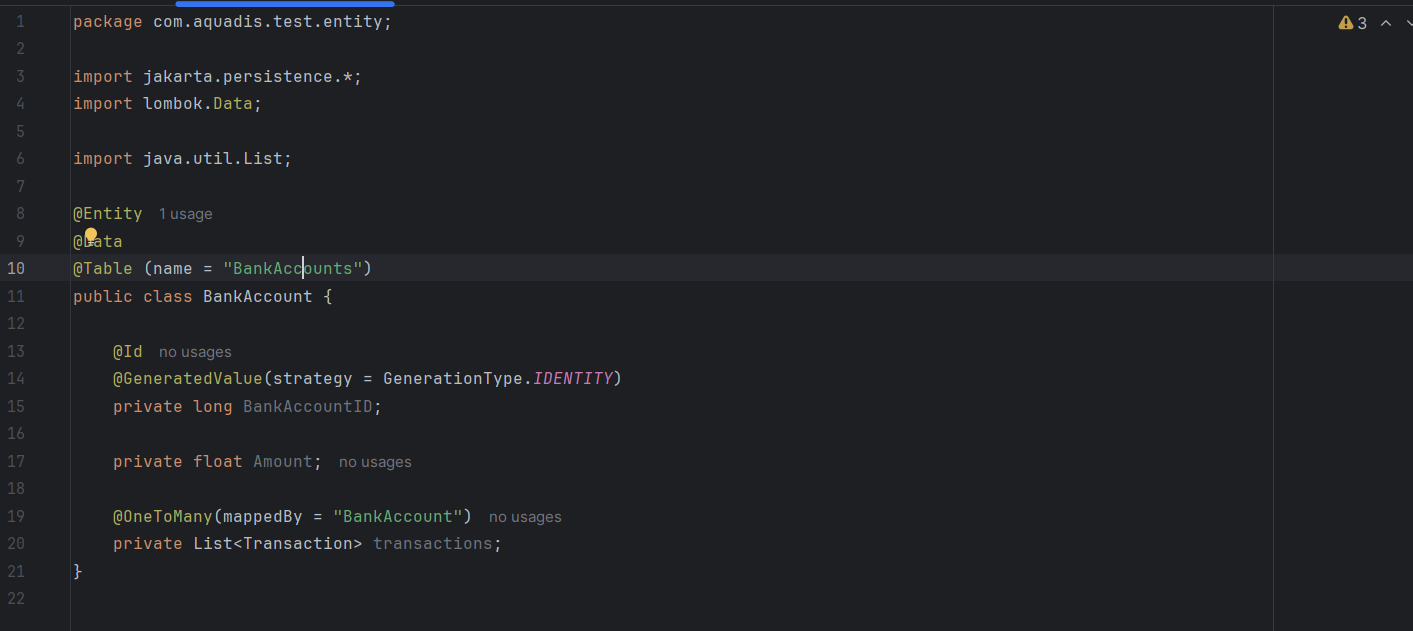


# Database and Entity Creation:

* + Create the database structure using code first JPA entities:
    - Customer: CustomerID (PK), Name, Password



* + - BankAccount: BankAccountID (PK), Amount



* Category: CategoryID (PK), Name

A screenshot of a computer

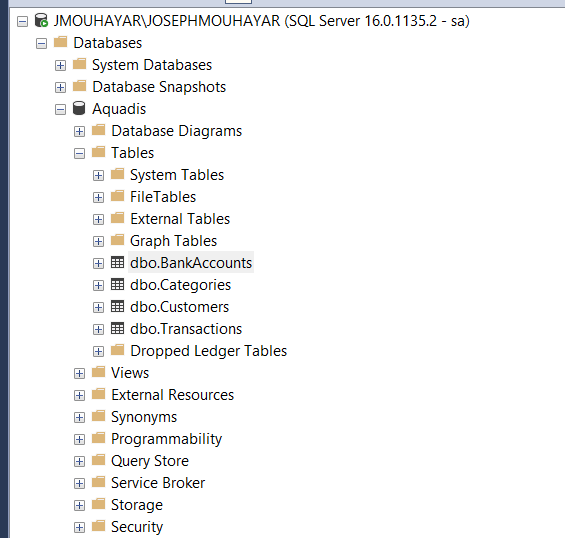
Description automatically generated

* + - Transaction: TransactionID (PK), Type (income or expenses), Amount, createdAt (date time that is directly generated on creation), BankAccountID (FK), CategoryID (FK)

A screenshot of a computer program

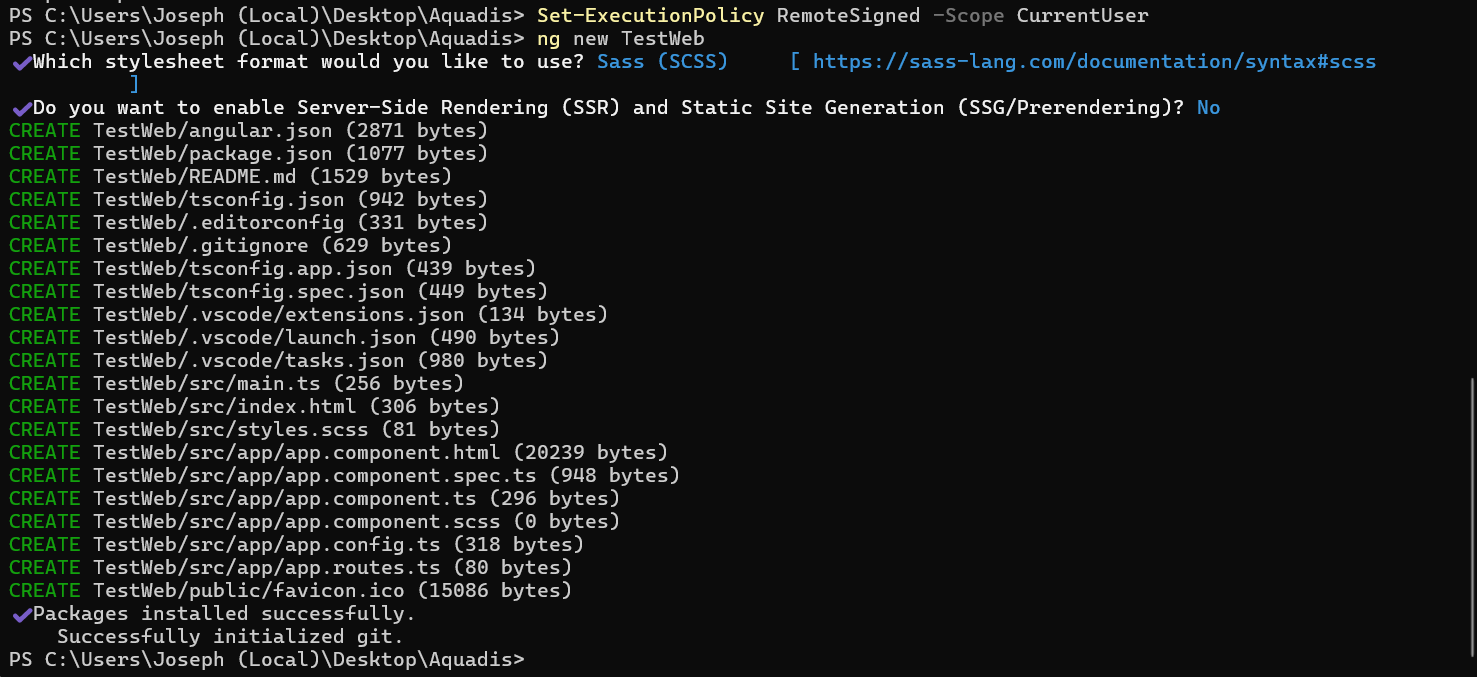
Description automatically generated

* + - Database Structure:

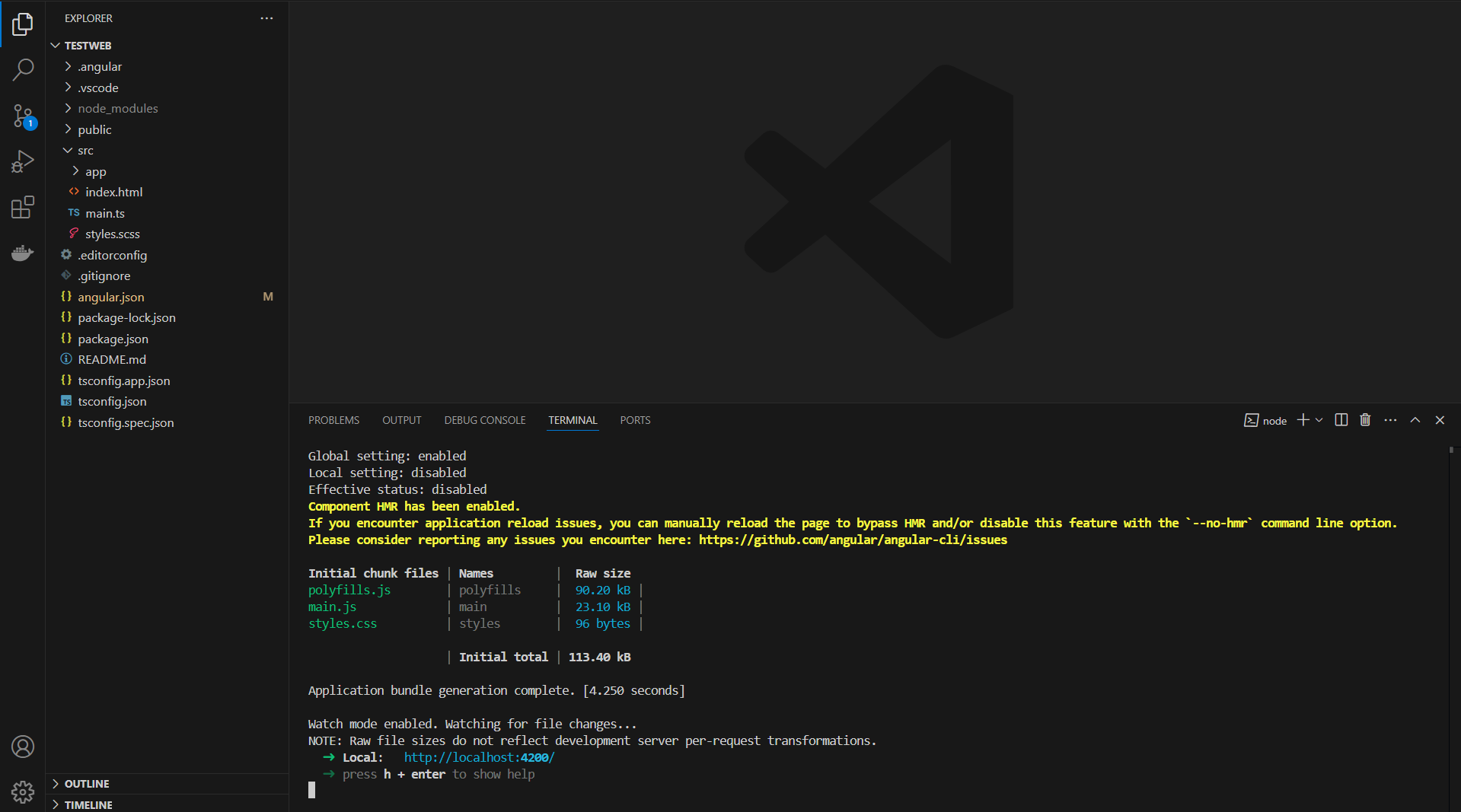


# Angular Initialization:

* PowerShell script



* VsCode View + ng serve:



Ng serve

# Login/SignUp pages:

* Spring-Boot:
  + Creation of JwtUserDetails:
    - Custom Implementation of UserDetails: The JwtUserDetails class implements the UserDetails interface from Spring Security, which is used to represent user authentication and authorization details, but in this case, it's based on the Customer entity.
    - Customer Information Mapping: The class maps customer data (such as password, username, and customerID) to the required UserDetails methods. It provides the customer's password and username, while also exposing the getCustomerID() method for customer-specific data.
    - Default Account Settings: It overrides the standard account status checks (isAccountNonExpired, isAccountNonLocked, etc.) to always return true, assuming the account is always active and valid in this example.

A screenshot of a computer program

Description automatically generated

* + Creation of JwtUtil:
* WT Token Generation: The class manually constructs a JWT token by creating a header, payload (with claims such as customerID, name, and expiration time), and signature using HMAC SHA-256 and a secret key (SECRET\_KEY). It then combines these parts to form a valid JWT.
* Token Parsing and Claims Extraction: The class provides methods to decode a JWT, parse its payload, and extract claims like username (subject), customerID, and expiration. It uses Base64 URL decoding and simple string manipulation to extract and return these values.
* Token Validation: It validates a JWT by checking if the username in the token matches the provided UserDetails and if the token has not expired, using the expiration date in the payload and comparing it with the current date.

A screenshot of a computer program

Description automatically generated

A screenshot of a computer program

Description automatically generated

A screenshot of a computer program

Description automatically generated

* + Dtos:
    - CustomerDto:

A black rectangular object with a black strip

Description automatically generated

* SignUpRequest:

A black rectangular object with a black line

Description automatically generated

* AuthenticationRequest:

A black rectangular object with a blue stripe

Description automatically generated

* + UserRepository: The UserRepository interface extends JpaRepository to provide CRUD operations for the Customer entity and includes a custom method findByName to retrieve a Customer based on their name, returning an Optional<Customer>.

A screenshot of a computer

Description automatically generated

* + Services
    - AuthService: The AuthService interface defines methods for creating a customer from a sign-up request and checking if a customer with a specific name exists.

A screenshot of a computer program

Description automatically generated

* AuthServiceImpl:
  + **Hashing Password**: The hashPassword method hashes a given password using SHA-256 to store it securely.
  + **Create Customer**: The createCustomer method accepts a SignUpRequest, creates a new Customer entity, hashes the password, saves it to the repository, and returns a CustomerDto.
  + **Check Customer Existence**: The hasCustomerWithName method checks if a customer with the given name exists in the repository.

A screen shot of a computer program

Description automatically generated

* UserDetailsServiceImpl: The UserDetailsServiceImpl class implements UserDetailsService to load user details from the repository based on the username, returning a JwtUserDetails object for the authenticated user.

A screen shot of a computer program

Description automatically generated

* + JwtRequestFilter:
* Extracts Token and Customer Details: Checks the Authorization header for a JWT token, extracting customerName and customerID using JwtUtil.
* Validates the Token: Verifies the token's validity using JwtUtil and loads user details via UserDetailsServiceImpl if authentication is not yet set.
* Sets Authentication: Creates an authenticated UsernamePasswordAuthenticationToken and stores it in SecurityContextHolder, including the customerID in the authentication details.
* Continues Request Processing: Passes the request along the filter chain by calling filterChain.doFilter(), ensuring the customer’s ID is available for subsequent processing.

A screenshot of a computer program

Description automatically generated

* + Config:
    - SimpleCorsFilter:
* Handles CORS: It sets the necessary headers to allow cross-origin requests, including Access-Control-Allow-Origin, Access-Control-Allow-Methods, and Access-Control-Allow-Headers.
* Dynamic Origin Handling: The origin of the incoming request is dynamically set in the Access-Control-Allow-Origin header based on the Origin header from the request.
* Supports Preflight Requests: If the HTTP method is OPTIONS, which is used for preflight CORS requests, it responds with status 200 OK to allow the subsequent actual request.
* Filter Execution Order: The filter is configured to run at the highest precedence (Ordered.HIGHEST\_PRECEDENCE), ensuring it is applied first in the filter chain.

A screen shot of a computer

Description automatically generated

* + - WebSecurityConfiguration:
* API Security: It applies security settings only to endpoints under /api/\*\*, disabling CSRF protection and allowing specific endpoints like /authenticate, /signup, and /api/customer/\*\* to be accessed without authentication.
* Stateless Sessions: Configures stateless session management (no session is created or stored) by using SessionCreationPolicy.STATELESS, which is ideal for token-based authentication (e.g., JWT).
* JWT Filter Integration: It adds a JwtRequestFilter to the filter chain before the UsernamePasswordAuthenticationFilter to handle JWT validation in incoming requests.
* Password Encoding and Authentication Manager: It defines a BCryptPasswordEncoder bean for password encoding and provides an AuthenticationManager bean for managing authentication.

A screen shot of a computer

Description automatically generated

* Angular:
  + Creation of Components (Login and SignUp) and Customer Service

A screenshot of a computer

Description automatically generated

* Customer Service:
  + User Data Management: Uses BehaviorSubject to manage and update the current user data (name and password), with currentUserData observable for real-time updates.
  + Sign Up: signUpCustomer method sends customer data to the backend for creating a new customer.
  + Login: loginCustomer method sends login credentials (name and password) to authenticate the user and returns the full response, including headers (for token handling).
  + JWT Decoding: decodeToken method decodes the JWT token using the jwt-decode library to extract user claims.

A screen shot of a computer program

Description automatically generated

* Login Component:
  + HTML:

A screen shot of a computer program

Description automatically generated

* + Component:

A screen shot of a computer program

Description automatically generated

* SignUp Component:
  + HTML:

A screenshot of a computer program

Description automatically generated

* + Component:

A screen shot of a computer program

Description automatically generated