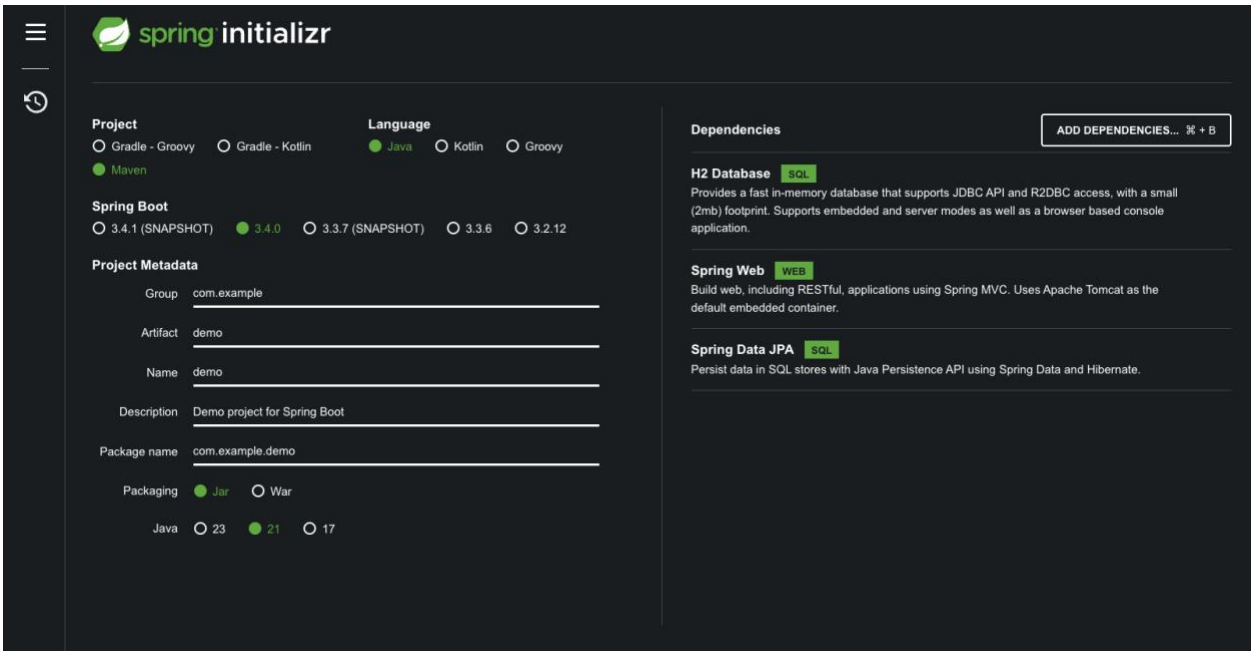


1. Create the Project using Spring Initializr

1. **What is Spring Initializr?**

Spring Initializr is an online tool that helps you quickly generate a Spring Boot project. It sets up the basic structure of your project so you can focus on coding.

2. **Steps**:

- Go to Spring Initializr.
- Configure your project:
    - **Maven**: A tool to manage project dependencies and build your project.
    - **Java**: The language we'll use.
    - **Spring Boot Version**: Use the latest stable version for features and bug fixes.
    - **Dependencies**: Add:
        - **Spring Web**: Allows us to create APIs.
        - **Spring Data JPA**: Provides tools to interact with the database using Java objects.
        - **H2 Database**: An in-memory database for quick and easy testing.

3. **Generate and Extract**:

- Download the project as a .zip file.
- Extract it into a folder and open it in your favorite IDE, such as IntelliJ or Eclipse.

2. Understand the application.properties

What is it?
The application.properties file is where you configure your application's behavior, such as database settings, server port, and other important options.

```
spring.application.name=demo

# H2 Database Configuration
spring.datasource.url=jdbc:h2:mem:testdb
spring.datasource.driverClassName=org.h2.Driver
spring.datasource.username=sa
spring.datasource.password=password

# JPA / Hibernate Configuration
spring.jpa.database-platform=org.hibernate.dialect.H2Dialect
spring.jpa.show-sql=true
spring.jpa.hibernate.ddl-auto=create

# H2 Console Configuration
spring.h2.console.enabled=true
spring.h2.console.path=/h2-console
```

Explanation:

1. **Database Configuration**:
   - spring.datasource.url: Connects to the H2 database. Here, mem:testdb means an in-memory database is used, which gets erased when the app stops.
   - spring.datasource.username/password: Credentials for connecting to the database.
2. **Hibernate Configuration**:
   - spring.jpa.hibernate.ddl-auto=create: Automatically creates database tables based on your Java classes (like Person).
   - spring.jpa.show-sql=true: Logs SQL queries in the console for debugging.
3. **H2 Console**:
   - Enables a web-based UI to interact with the database at http://localhost:8080/h2-console.

3. Create the Entity Class (Person)

What is an Entity?
An entity is a Java class that represents a table in the database. Each instance of the class corresponds to a row in the table.

```java
@Entity
public class Person {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    private String name;
    private String email;

    // Getters and Setters
}
```

Explanation:

1. @Entity:
   - Marks the class as a database table.
2. @Id:
   - Specifies the primary key of the table (used to uniquely identify rows).
3. @GeneratedValue:
   - Automatically generates unique IDs for each new person added to the table.
4. **Fields (**name **and** email**)**:
   - Each field becomes a column in the database.
5. **Getters and Setters**:
   - These allow other parts of the program to access and modify the fields.

4. Create the Repository

What is a Repository?
The repository is an interface that handles database operations such as saving, updating, deleting, and retrieving data.

```java
package com.example.demo.repository;

import com.example.demo.entity.Person;
import org.springframework.data.jpa.repository.JpaRepository;

public interface PersonRepository extends JpaRepository<Person, Long> {  no usages
}
```

Explanation:

1. JpaRepository:
   - Provides methods like findAll(), save(), deleteById(), etc., so you don't have to write SQL queries manually.
   - Person: The entity this repository will manage.
   - Long: The type of the entity's primary key.
2. **Why is it Important?**
   - It simplifies database operations, allowing you to focus on the logic rather than writing repetitive SQL.

5. Create the Service Layer

What is a Service Layer?
The service layer contains the **business logic**. It acts as a bridge between the controller (API) and the repository (database).

Explanation:

1. **Why is it Important?**
   - Centralizes business logic, keeping the controller and repository clean.
   - Handles data validation and ensures proper interaction between components.
2. **Methods**:
   - getAllPersons: Fetches all persons from the database.
   - savePerson: Adds a new person to the database.
   - updatePerson: Modifies an existing person's details.
   - deletePerson: Removes a person from the database.

6. Create the Controller

What is a Controller?
The controller defines REST endpoints for interacting with the application (e.g., adding, retrieving, or deleting a person).

Explanation:

1. **Annotations**:
   - @RestController: Tells Spring this class handles HTTP requests.
   - @RequestMapping: Sets the base path for all endpoints (/api/persons).
   - @GetMapping, @PostMapping, etc.: Define HTTP methods.
2. **Why is it Important?**
   - Exposes APIs that clients (like Postman or a frontend application) can use to interact with your app.

7. Testing the Application

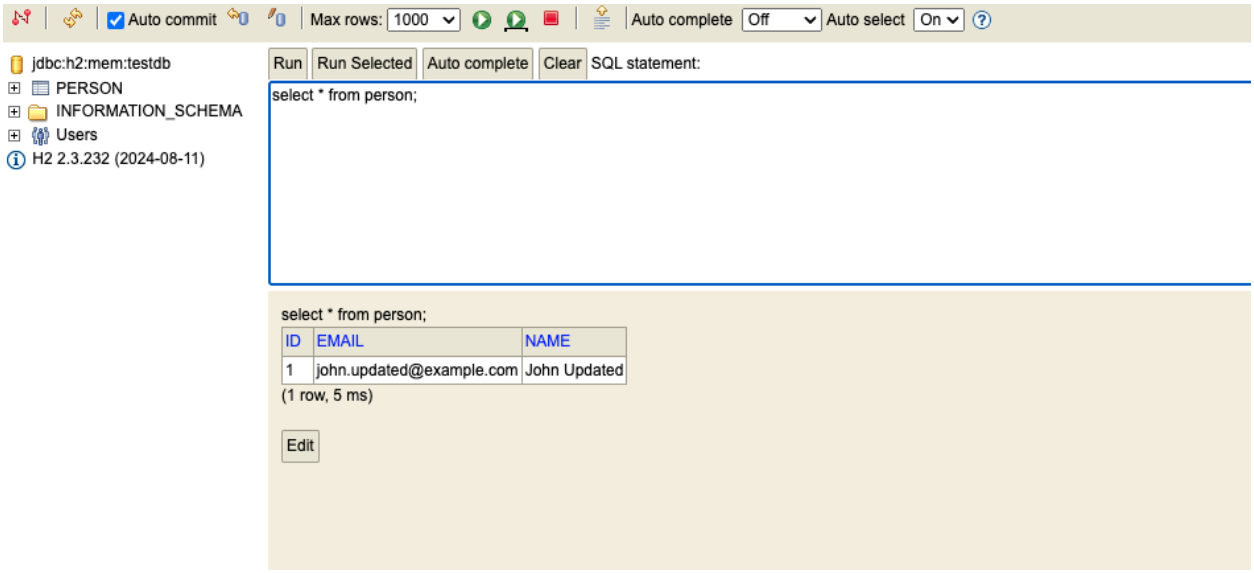
Run the Application

1. Right-click the main class (DemoApplication.java) and run it.
2. Ensure the Spring Boot server starts without errors.

8. Access the H2 Console

1. Visit http://localhost:8080/h2-console.
2. Login using:
   - **JDBC URL**: jdbc:h2:mem:testdb
   - **Username**: sa
   - **Password**: check application.properties file in resources.
3. Query the database:

```
SELECT * FROM PERSON;
```



Why Each Component Matters

1. **Entity (**Person**)**: Represents the structure of your data.
2. **Repository (**PersonRepository**)**: Simplifies database interactions.
3. **Service (**PersonService**)**: Keeps business logic organized and reusable.
4. **Controller (**PersonController**)**: Connects the application to the outside world.
5. **H2 Database**: Provides a quick, in-memory database for testing and development.
6. application.properties: Configures the application's behavior.