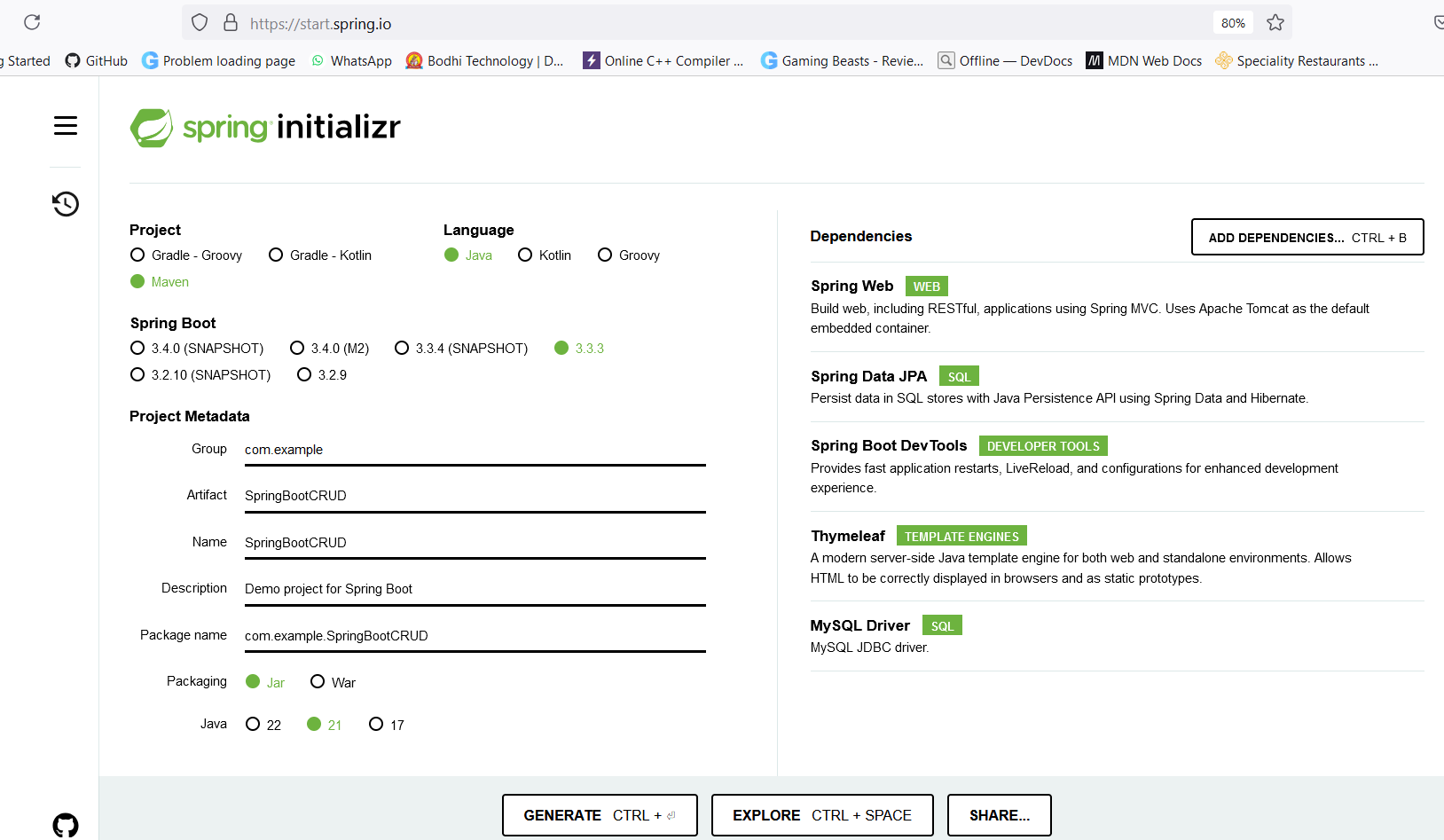
**Spring Boot**



When comparing CRUD applications in Spring Boot and Spring Framework (Spring Core), the core concepts remain similar, but there are notable differences in setup, configuration, and features:

### Spring Framework (Spring Core) vs. Spring Boot

#### 1. Setup and Configuration

- Spring Framework:

- Manual Configuration: In Spring Framework, you often need to manually configure beans, data sources, transaction management, and other components using XML or Java-based configuration classes.

- Complex Setup: Setting up a project involves more boilerplate configuration, including setting up `ApplicationContext`, configuring data sources, and handling dependencies manually.

- Spring Boot:

- Auto-Configuration: Spring Boot provides auto-configuration, which means many common configurations are automatically set up based on the dependencies in your project. This reduces the amount of manual configuration needed.

- Convention over Configuration: Spring Boot follows conventions to minimize the need for explicit configuration. For example, it automatically sets up an embedded web server and integrates with various libraries.

#### 2. Dependency Management

- Spring Framework:

- Manual Dependency Management: Dependencies are managed manually, and you often need to configure the version of each library and their interdependencies explicitly.

- No Starter Dependencies: You need to include each library and configuration separately.

- Spring Boot:

- Starter POMs: Spring Boot uses starter POMs (e.g., `spring-boot-starter-web`, `spring-boot-starter-data-jpa`) to bundle commonly used libraries and their dependencies, simplifying dependency management.

- Version Management: Spring Boot manages versions of dependencies automatically, ensuring compatibility and reducing conflicts.

#### 3. Application Entry Point

- Spring Framework:

- Traditional Entry Point: Applications are typically started with a `main` method that initializes the `ApplicationContext` using an `AnnotationConfigApplicationContext` or `ClassPathXmlApplicationContext`.

- Spring Boot:

- Simplified Entry Point: Spring Boot uses a `@SpringBootApplication` annotation, which is a combination of `@Configuration`, `@EnableAutoConfiguration`, and `@ComponentScan`. The application is started with a `main` method that calls `SpringApplication.run()`, which handles context initialization and auto-configuration.

#### 4. Embedded Servers

- Spring Framework:

- External Server: Requires deployment to an external web server like Tomcat, Jetty, or JBoss.

- Spring Boot:

- Embedded Servers: Supports embedded servers (Tomcat, Jetty, or Undertow) out of the box. You can run the application as a standalone Java application without needing an external server.

#### 5. Configuration Management

- Spring Framework:

- Property Files: Configuration is typically managed via `applicationContext.xml` or external property files, and you need to define beans and their properties manually.

- Spring Boot:

- Application Properties: Uses `application.properties` or `application.yml` files for configuration, and provides a unified way to configure application settings. Profiles (e.g., `application-dev.properties`) help manage different environments.

#### 6. Testing

- Spring Framework:

- Manual Test Setup: Testing often requires more manual setup for context initialization and configuration.

- Spring Boot:

- Built-in Testing Support: Provides extensive support for testing with `@SpringBootTest`, which automatically configures the application context and integrates with testing frameworks.

#### Example Comparison: CRUD Operation

Spring Framework Example:

- Configuration: Requires manual configuration of beans, data sources, and transaction management.

- Repository: Typically requires explicit configuration of `JpaRepository` or `JdbcTemplate`.

- Controller and Service: Requires manual wiring of services and repositories.

Spring Boot Example:

- Configuration: Uses auto-configuration to simplify setup. `@SpringBootApplication` handles context setup.

- Repository: Automatically configured using `JpaRepository` with minimal additional configuration.

- Controller and Service: Dependency injection is managed automatically with less manual setup.

In summary, Spring Boot simplifies development by providing automatic configurations, embedded servers, and streamlined dependency management, while Spring Framework (Spring Core) requires more manual configuration and setup. Both frameworks use similar core concepts, but Spring Boot enhances productivity and ease of use.

Dependency Injection, Autowire, Web App using spring boot, Spring

boot AOP, spring boot Database, Spring Rest.