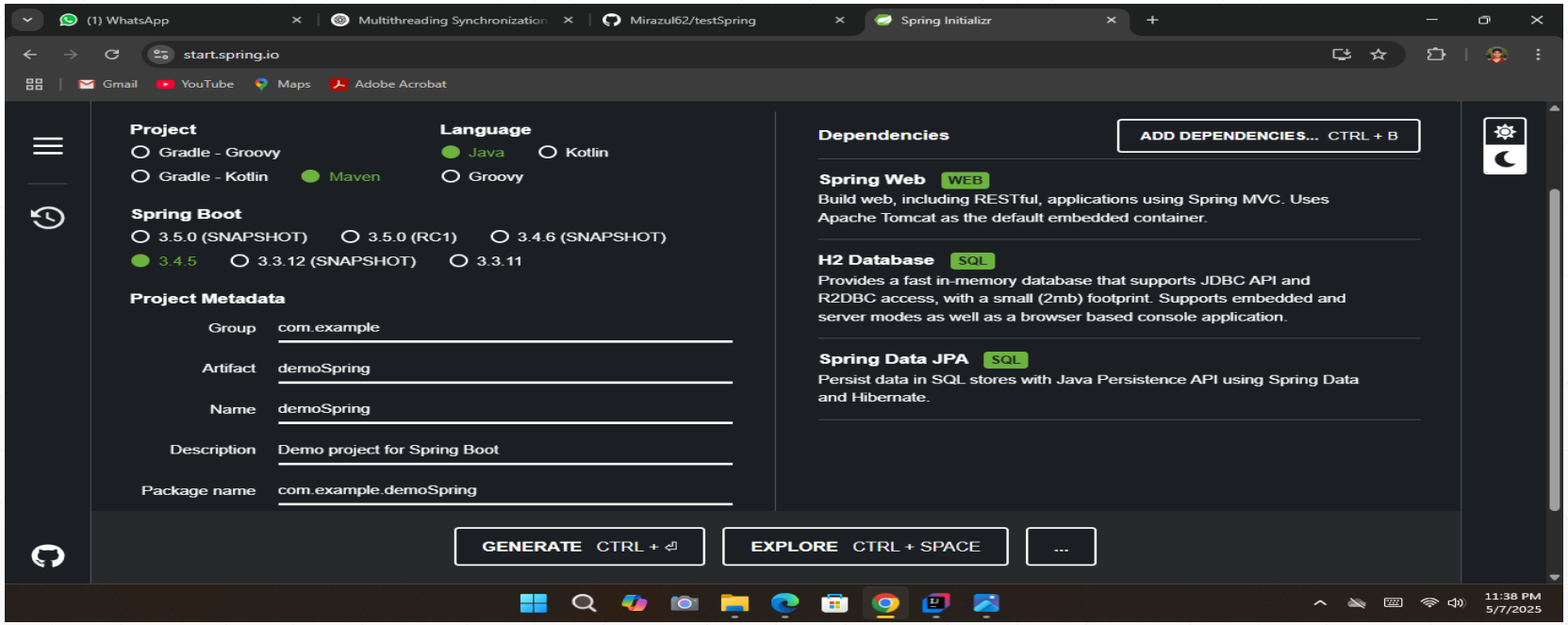
# Spring Boot Essentials: A Practical Introduction

Presented By:
**Mirazul Islam**
**Java Instructor, EU**

# What is Spring Initializr?

- A web-based project generator for Spring Boot
- Helps you quickly set up a new Spring Boot application
- Provides customizable project configuration and dependencies



# H2 Database Overview

- In-memory mode: Data is stored in RAM (disappears after app stops)
- Embedded or Server modes
- SQL-compliant
- Zero installation (jar-based)
- Simple web-based console (h2-console)
- Easy integration with Spring Boot

# Spring Boot + H2 Setup

- Add dependency in pom.xml:

```xml
<dependency>
  <groupId>com.h2database</groupId>
  <artifactId>h2</artifactId>
  <scope>runtime</scope>
</dependency>
```

- Configure in application.properties:

```
spring.datasource.url=jdbc:h2:mem:test
spring.datasource.driverClassName=org.h2.Driver
spring.datasource.username=sa
#spring.datasource.password=password
spring.jpa.database-platform=org.hibernate.dialect.H2Dialect

spring.jpa.defer-datasource-initialization=true
spring.h2.console.enabled=true
```

- Access H2 Console:
- URL: http://localhost:8080/h2-console

# What is pom.xml?

- pom.xml stands for Project Object Model and is the core configuration file used in Maven projects.
- It's an XML file that tells Maven:
- What the project is (name, version, description)
- What dependencies it needs
- How to build and package the project
- What plugins and goals to use
- Project structure and other metadata

# What is JPA?

- JPA (Java Persistence API) is a Java specification for managing relational data in Java applications. It defines how Java objects are mapped to database tables and how to manage their lifecycle (create, read, update, delete).

- ORM (Object-Relational Mapping): Maps Java classes to database tables.
- Annotations: Uses annotations like @Entity, @Table, @Id, @Column to define mappings.
- Entity Manager: Provides API to interact with the database.
- JPQL (Java Persistence Query Language): Object-oriented version of SQL.

# Introduction to REST(Representational State Transfer)

- Architectural style for designing networked applications
- Uses standard HTTP methods for CRUD operations
- Stateless and scalable communication

**HTTP Methods in REST**

| HTTP Method | Action | Example |
|---|---|---|
| GET | Retrieve a resource | `GET /users` |
| POST | Create a new resource | `POST /users` |
| PUT | Update an existing resource | `PUT /users/1` |
| DELETE | Delete a resource | `DELETE /users/1` |

# @Autowired in Spring

- @Autowired is a Spring annotation used for dependency injection. It allows Spring to automatically inject the necessary dependencies into a class at runtime, making the class less dependent on external configuration.

```java
@Service
public class UserService {

    @Autowired
    private UserRepo userRepo;  // Automatically injected by Spring

    public User getUserById(long id) {
        return userRepo.findById(id).orElse(null);
    }
}
```

- In this example, the userRepo dependency is automatically injected into the UserService class by Spring. No explicit instantiation or configuration is required.

# What is MVC?

- MVC stands for Model-View-Controller
- A design pattern that separates application logic into three interconnected components

# Model in Spring

```java
@Entity
public class User {
    @Id
    private Long id;
    private String name;
    private String email;

    // Getters and setters
}
```

# View

- Represents the user interface (UI) elements of the application.
- In Spring, **views** are typically **JSP** or **Thymeleaf** templates that present the data from the model.

```
<h1>User Information</h1>
<p>Name: <span th:text="${user.name}"></span></p>
<p>Email: <span th:text="${user.email}"></span></p>
```

# Controller

- Acts as an intermediary between the **Model** and the **View**.

```java
@Controller
@RequestMapping("/user")
public class UserController {

    @Autowired
    private UserService userService;

    // Mapping for showing user details
    @GetMapping("/{id}")
    public String getUser(@PathVariable Long id, Model model) {
        User user = userService.findUserById(id);
        model.addAttribute("user", user);
        return "userDetails";  // This will render the userDetails.html view
    }

    // Mapping for saving a new user
    @PostMapping("/save")
    public String saveUser(@ModelAttribute User user) {
        userService.saveUser(user);
        return "redirect:/user/all";  // Redirect to list all users
    }
}
```