Spring Boot Back End

Java Development Environment

- We assume that you are already an experienced Spring Boot Developer

- You should have the following items already installed
  - Java Development Kit (JDK)
  - Java IDE (we'll use IntelliJ in the videos, but any Java IDE will work)
  - Maven
  - MySQL Database and MySQL Workbench

About IntelliJ

Super Amazing IDE!!!

- In this course, we will use the free version of IntelliJ
  - Known as **IntelliJ Community Edition**
  - Download from: **https://www.jetbrains.com/idea/download**
  - Select **Community Edition**

- You can also use the Ultimate Edition ($) ... free trial version is available

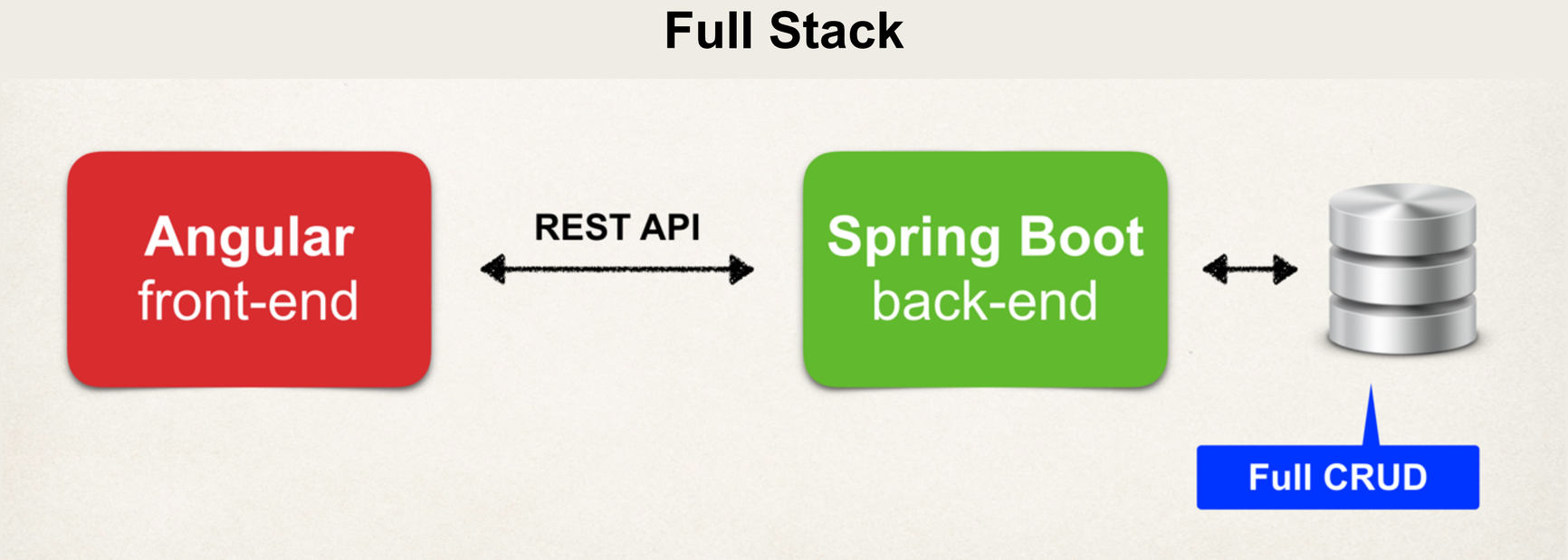
Additional Java IDEs

- You are free to use other Java IDEs such as Eclipse, VS Code, NetBeans
- All you need is a Java IDE that supports Maven ... that's it!
- You can easily follow along with any Java IDE

- We will provide tech support for IntelliJ, Eclipse, VS Code, NetBeans

Spring Boot Back End

- Leverage Spring Data REST for REST API
- Minimizes the coding for Spring Boot back end



Create Repository

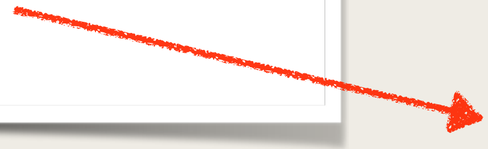
- Spring Data REST will scan your project for **JpaRepository**
- Expose REST APIs for each entity type for your **JpaRepository**

```
public interface ProductRepository extends JpaRepository<Product, Long> {

}
```

REST Endpoints

- By default, Spring Data REST will create endpoints based on entity type
- Simple pluralized form
  - First character of Entity type is lowercase
  - Then just adds an “s” to the entity

```
public interface ProductRepository extends JpaRepository<Product, Long> {

}
```



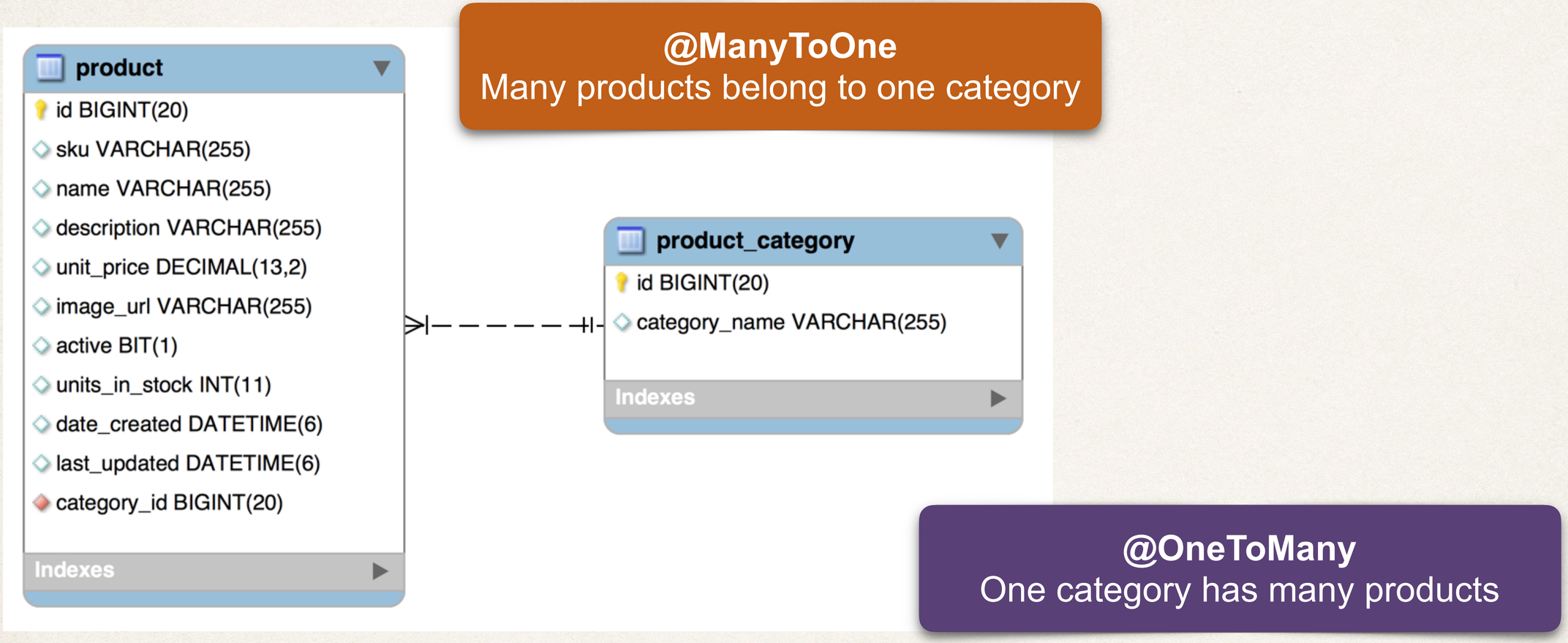
/products

REST API

- Spring Data REST will expose these endpoints for free!

HTTP Method		CRUD Action
POST	/products	**C**reate a new product
GET	/products	**R**ead a list of products
GET	/products/{id}	**R**ead a single product
PUT	/products/{id}	**U**pdate an existing product
DELETE	/products/{id}	**D**elete an existing product

Database Schema - Release 1.0



Two Database Scripts

- 01-create-user.sql
- 02-create-products.sql

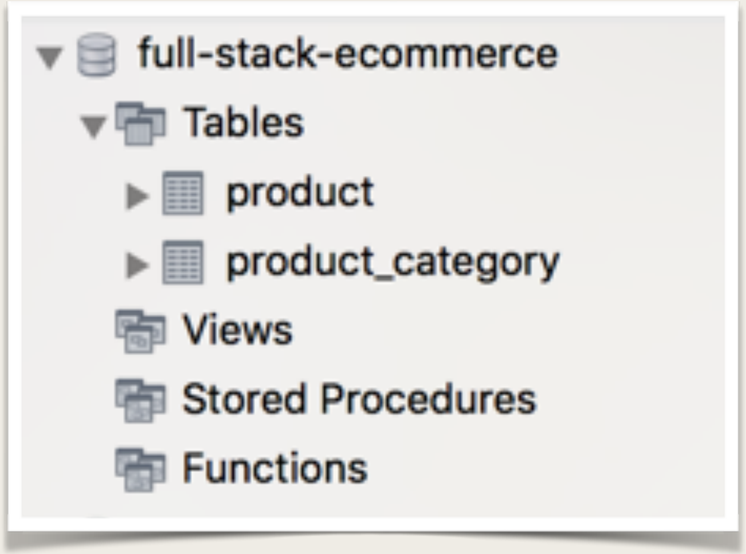
About: 01-create-user.sql

1. Create a new MySQL user for our application
    - user id: **ecommerceapp**
    - password: **ecommerceapp**

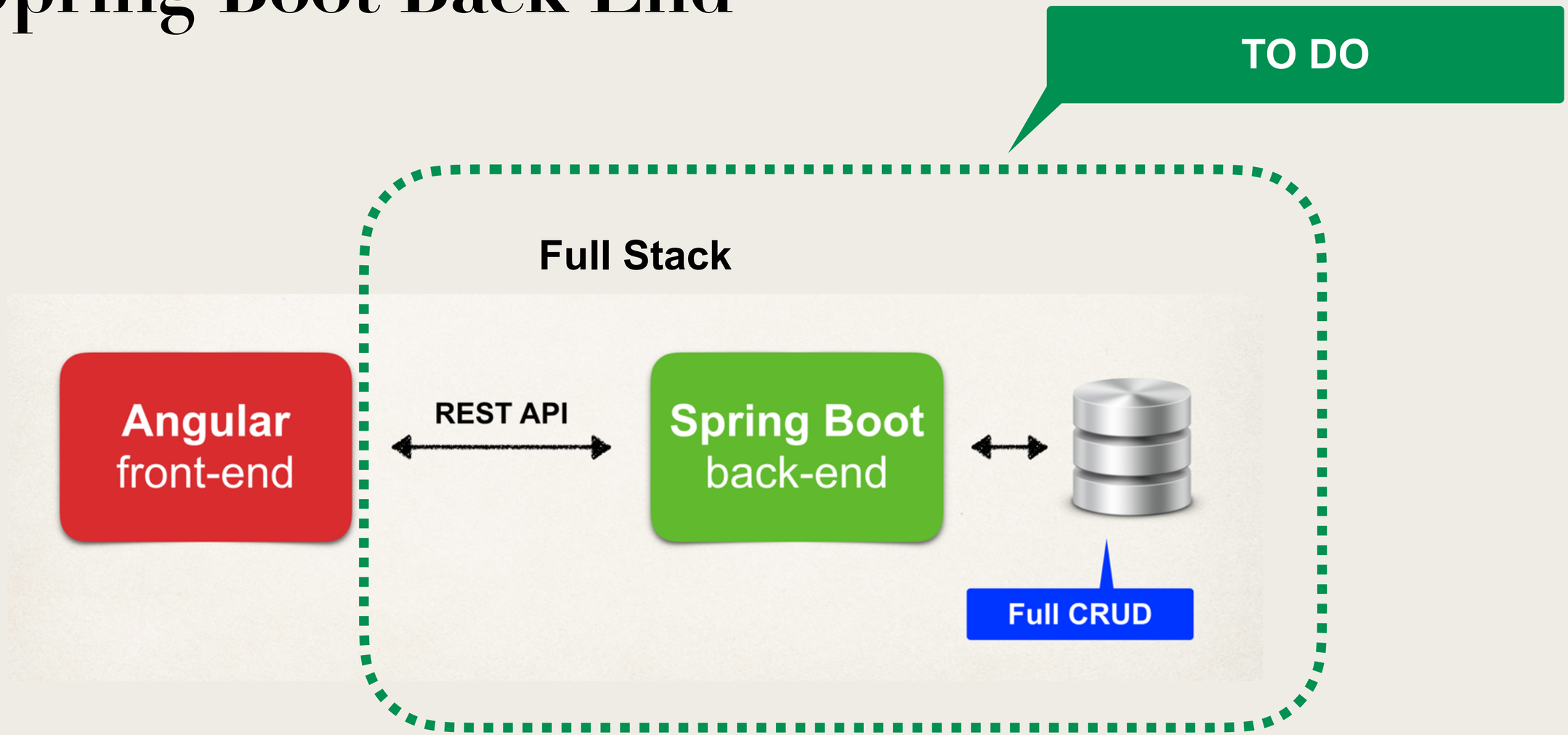
About: 02-create-products.sql

1. Create new database tables: **product, product_category**

2. Load tables with sample data



Spring Boot Back End



Development Process

1. Set up the database tables

2. Create a Spring Boot starter project (start.spring.io)

```
spring-boot-starter-data-jpa
spring-boot-starter-data-rest
mysql-connector-java
lombok
```

3. Develop the Entities: **Product** and **ProductCategory**

4. Create REST APIs with Spring Data JPA Repositories and Spring Data REST

Project Lombok

- Modern Java project
- Lombok automagically generates the getters/setters (behind the scenes)
- No need for the developer to manually define getters/setters, etc ...
- Easy-to-use Annotations to eliminate boilerplate code

http://www.projectlombok.org

Project Lombok

Before Lombok

```
@Entity
@Table(name = "product")
public class Product {

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

    @Column(name = "name")
    @NotBlank
    private String name;

    public Product() {
    }

    public Long getId() {
        return id;
    }

    public void setId(Long id) {
        this.id = id;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null || getClass() != o.getClass())
          return false;
        Product product = (Product) o;
        return Objects.equals(id, product.id) &&
                Objects.equals(name, product.name);
    }

    @Override
    public int hashCode() {
        return Objects.hash(id, name);
    }
}
```

After Lombok

Lombok annotation

```
@Entity
@Table(name = "product")
@Data
public class Product {

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

    @Column(name = "name")
    @NotBlank
    private String name;
}
```

That's It!!!
Absolutely no need to generate getters and setters

Lombok will do this work for you automagically behind the scenes