Weather Application

- Where does the meteorology data for the application come from?
- How can the meteorology data be updated every minute as the weather changes constantly
- Does the application itself generate and maintain meteorology data?
- How much will the application cost to perform data management by itself, i.e., to gather, store, and maintain data?

Slide Note

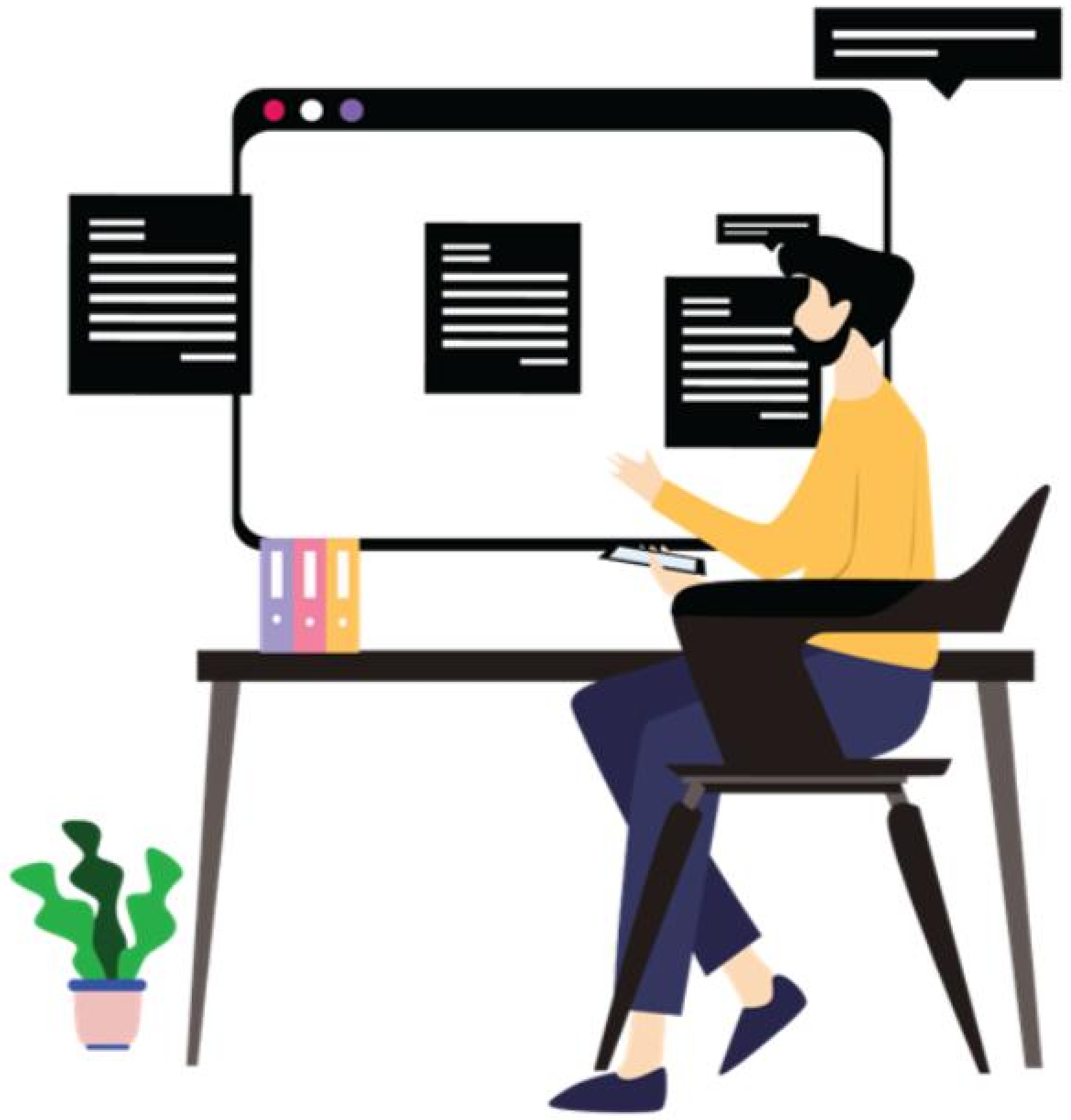
The Application Response

- In a typical web application scenario:
  - The weather application will run on a mobile device or computer.
  - The browser or mobile application is the client that will request the data from a web server.
  - The web server will return a response back, which can be an HTML page or data in the form of XML or JSON.
- A web application will always render an HTML page back to the requesting client.
- If a weather application is completely built, will returning an HTML page be relevant?
- Can the object data for the application be provided independently by another source?

Menu

Slide Note

Implement CRUD by Using JPA Within a RESTful Service



Menu

00:03 00:00:28 03/ 28

Slide Note

Learning Objectives

- Explore JPA
- Explain the Spring Data JPA
- Implement a REST API using MySQL in the Data Layer

Slide Note

Explore JPA

Menu

00:05 00:00:28 05/ 28

Slide Note

Providing Data in the Response to the Weather Application Using a REST API

- The data for the weather application can be read from a REST API available on a remote server.
- The REST API will have an internal mechanism to store and manage the data.
- If `WeatherInformation` is a class that contains `temperature`, `wind speed`, etc., the REST API will persist the data using a persistence mechanism like JDBC if the REST API is built using Java.
- When compared to much bigger web applications like banking or reservation systems, the weather app is pretty small.
- In such large applications, multiple objects will be saved and retrieved from the database; we cannot write multiple queries for each.
- It is where Java provides the programmer with the Java Persistence API, which simplifies the work of reading, saving, and retrieving data from web applications.

Menu

Slide Note

Java Persistence API

- The Java Persistence API (JPA) provides a mechanism for managing persistence and object-relational mapping.
- The JPA specification defines the object-relational mapping internally.
- JPA is based on the Java programming model.
- The relational mapping between classes is also maintained when it is persisted into the database.
- To store and retrieve data, simple SQL queries can be written in the application.
- All the Java classes that are to be persisted in the database are called entities. They are represented using the `@Entity` annotation present in the `javax.persistence` package.
- Each entity has a unique object identifier; the unique identifier, or primary key, of the entity is represented using the `@Id` annotation.

Menu

00:07:00:00:28 07/ 28

Slide Note

Explain Spring Data JPA

Slide Note

Spring Data JPA

- The Spring Data JPA makes it easy to implement a JPA-based data layer, or repositories, in a Spring REST API.
- Spring Data JPA reduces the need to write boilerplate or redundant code to exercise routine queries.
- Spring Data JPA provides built-in implementations for all these operations.
- The programmer can also write custom finder methods, which Spring can automatically implement.

Menu

00:09 00:00:28 09/ 28

Advantages of Using Spring Data JPA

The repository layer of the application can be free of code.

No-code repository

Reduced boilerplate code

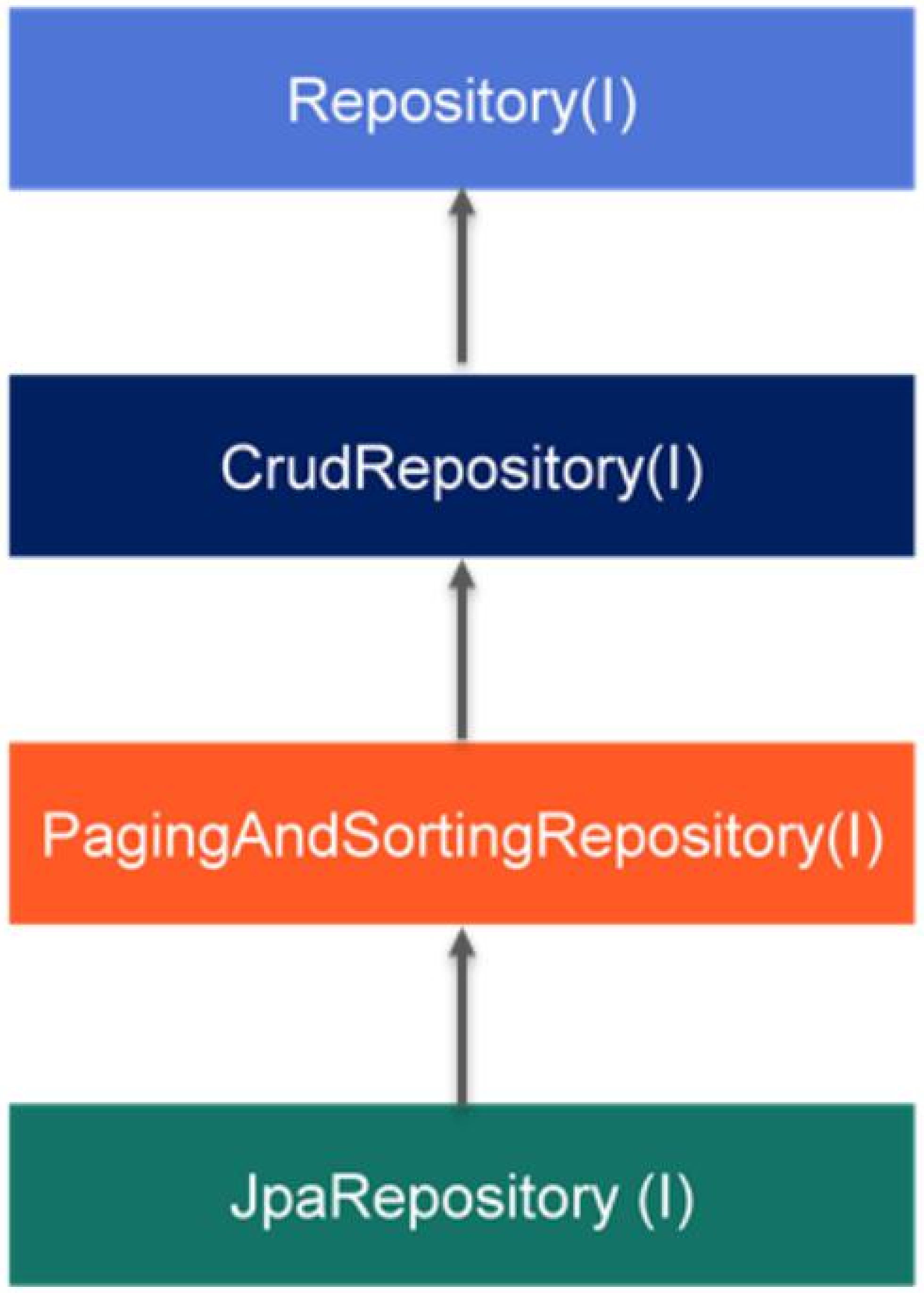
It provides the default implementation for each method of the repository interfaces.

Generates database queries based on the method name. If the query is complex, a method must be defined in the repository interface with a name that begins with findBy. Spring automatically parses the method name and creates a query for it.

Generated Queries

Slide Note

Types of Spring Data JPA Repositories



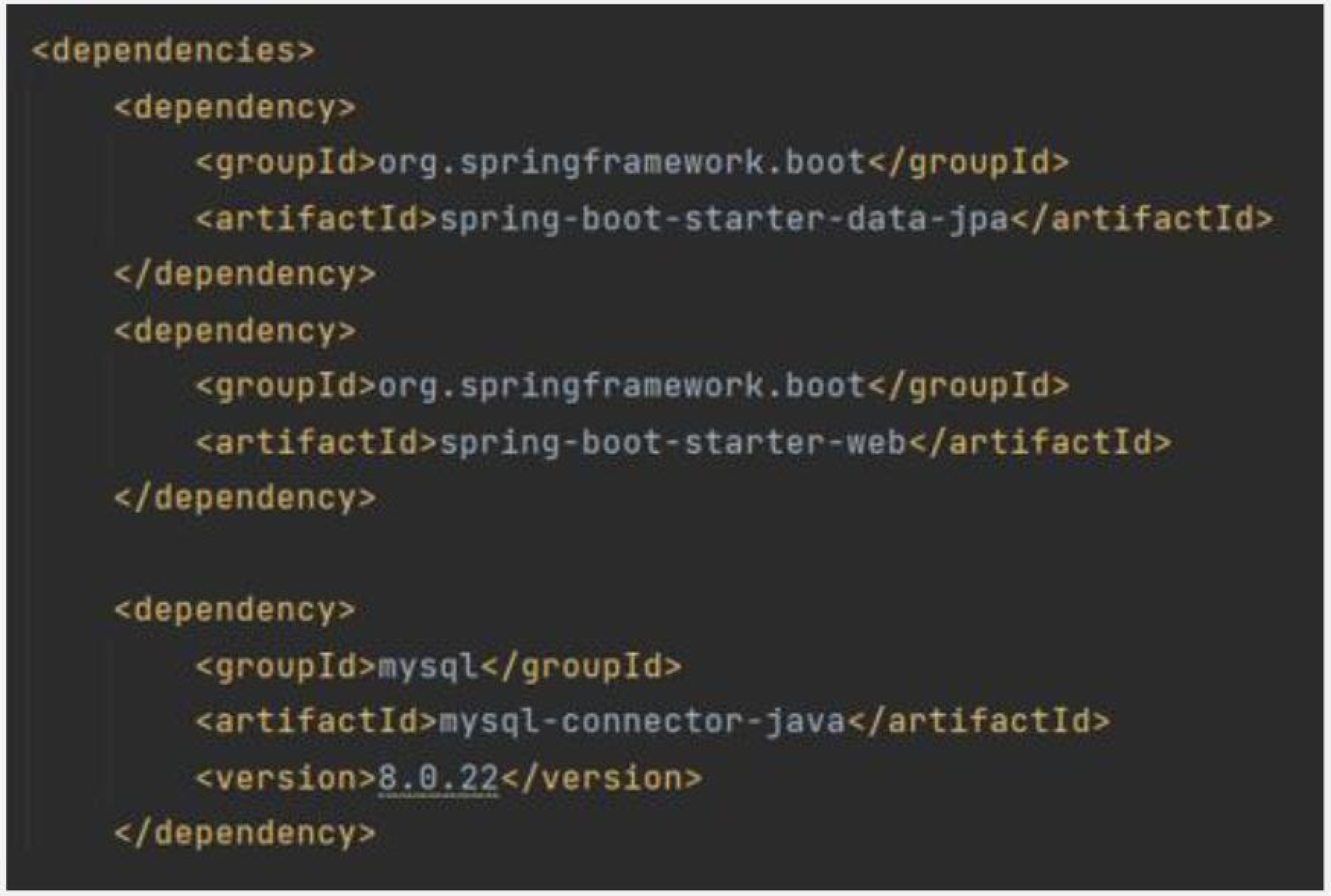
- The `Repository` is the top-level interface.
- `CrudRepository` is the child of Repository and offers standard create, read, update, and delete operations. It contains methods like findOne(), save(), delete(), etc.
- `PagingAndSortingRepository` is a child of `CrudRepository` and adds the findAll methods. It allows data to be sorted and retrieved in a paginated way.
- `JpaRepository` is a child of `PagingAndSortingRepository` and is a JPA-specific repository.
- Each interface provides functionality and can be used depending on requirements.

Slide Note

Implement a REST API Using MySQL in the Data Layer

Menu

00:12 00:00:28 12/ 28



```
<dependencies>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-data-jpa</artifactId>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-web</artifactId>
    </dependency>

    <dependency>
        <groupId>mysql</groupId>
        <artifactId>mysql-connector-java</artifactId>
        <version>8.0.22</version>
    </dependency>
```

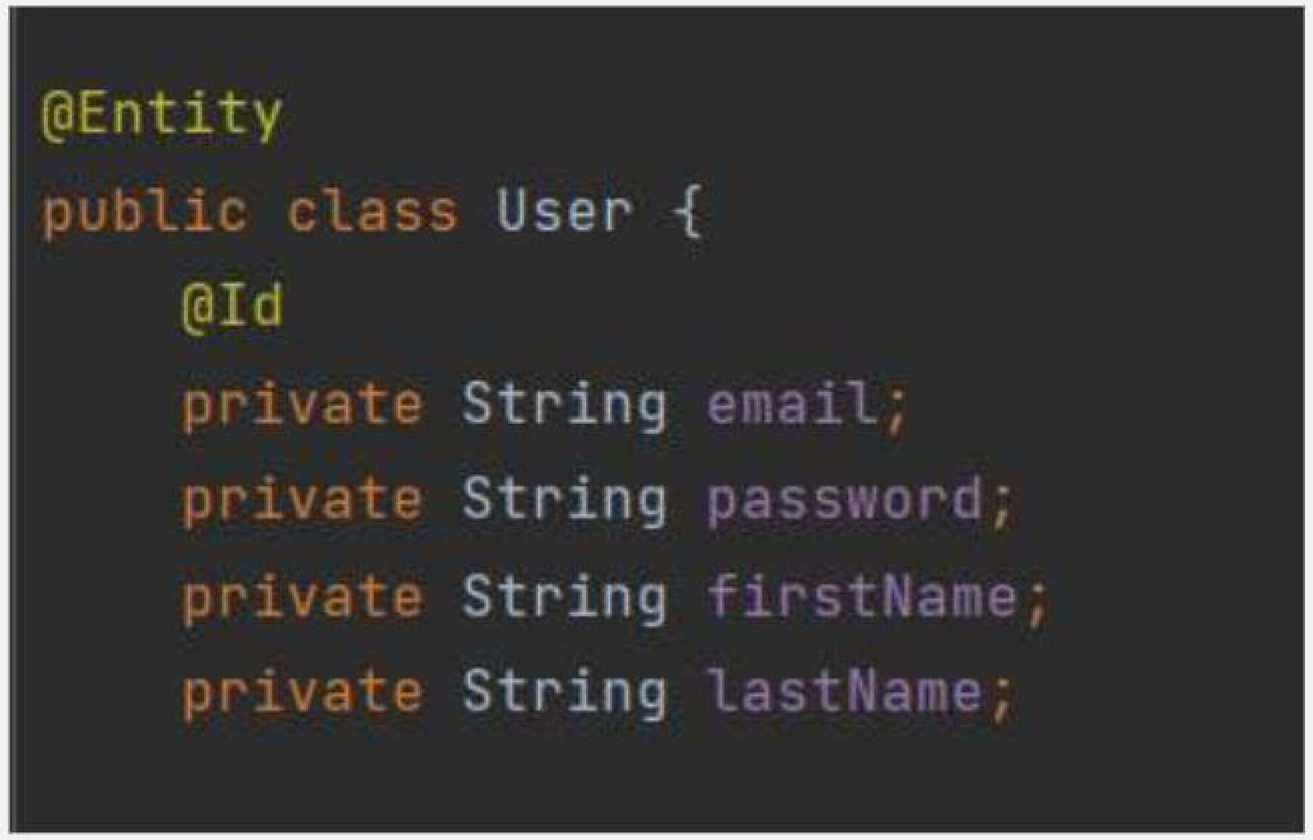
Dependencies

- **Import the necessary dependencies.**
- **The `spring-boot-starter-data-jpa` dependency is used to connect a Spring application with a relational database efficiently.**
- **The relational database used is MySQL, so the `mysql-connector-java` dependency must be imported.**

Slide Note

Domain Layer

- The domain object that needs to be persisted in the database is annotated with `@Entity`.
- The primary key or object identity field is annotated with `@Id`.
- The annotations are part of the `javax.persistence` package.



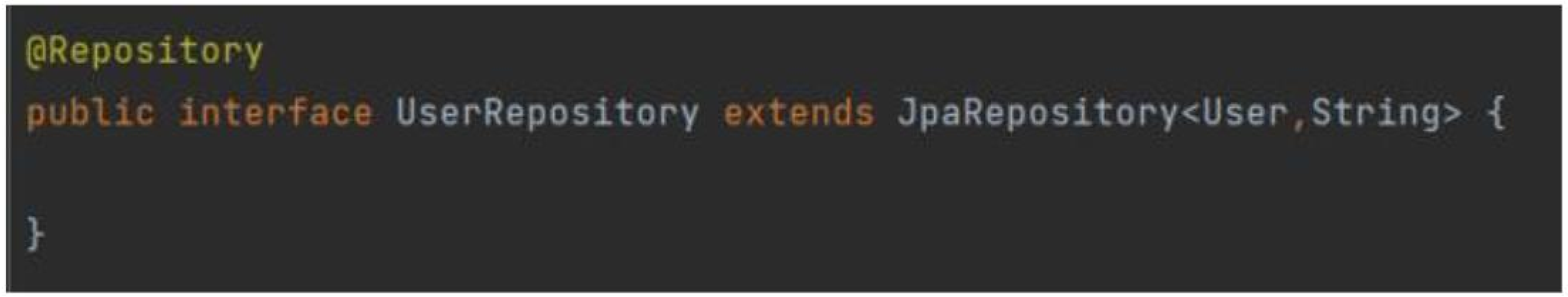
```
@Entity
public class User {
    @Id
    private String email;
    private String password;
    private String firstName;
    private String lastName;
```

Menu

00:14 00:00:28 14/ 28

Repository Layer of the REST API

- The Repository interface declared in the repository layer of the application is the central point of abstraction in Spring Data.
- It takes the domain class to manage and the ID type of the domain class as type arguments.
- `UserRepository` extends the `JpaRepository` interface, providing the CRUD functionality for the managed entity class.



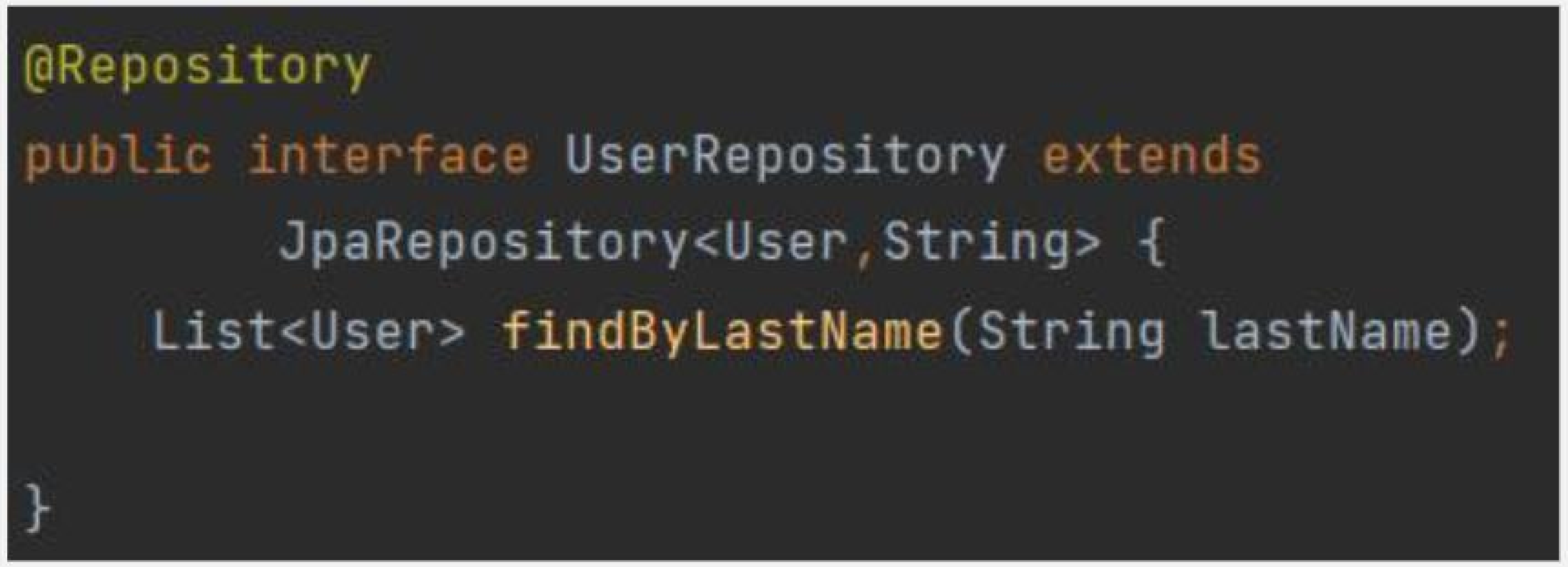
```
@Repository
public interface UserRepository extends JpaRepository<User,String> {

}
```

- In the example shown,
  - `UserRepository` is the central repository for the `User` entity.
- The `User` is the domain class with an `email` of the `String` type as the object identity field.
- Annotate the data layer interface with `@Repository`.

Slide Note

Repository Layer – User-Defined Methods



```
@Repository
public interface UserRepository extends
        JpaRepository<User,String> {
    List<User> findByLastName(String lastName);

}
```

- **User-defined methods can be created in the** `UserRepository`**.**
- **In the code,** `findByLastName()` **is a user-defined method, where** `lastName` **is an attribute of the domain class** `User`**.**
- **Here,** `"findBy"` **is a reserved phrase that Spring Data JPA will identify and provide the implementation for the method.**

Check the link for more information.

Menu

00:16 00:00:28 16/ 28

Slide Note

Quick Check

Which of the following statements is true about the CrudRespository?

1. `save(S entity)` **– saves multiple entities at a time.**

2. `findAllById(Iterable<ID> ids)` **– returns all entities with the given IDs.**

3. `deleteAll()` **– deletes all entities.**

4. `count()` **– It returns the number of entities.**

Menu

Slide Note

Quick Check: Solution

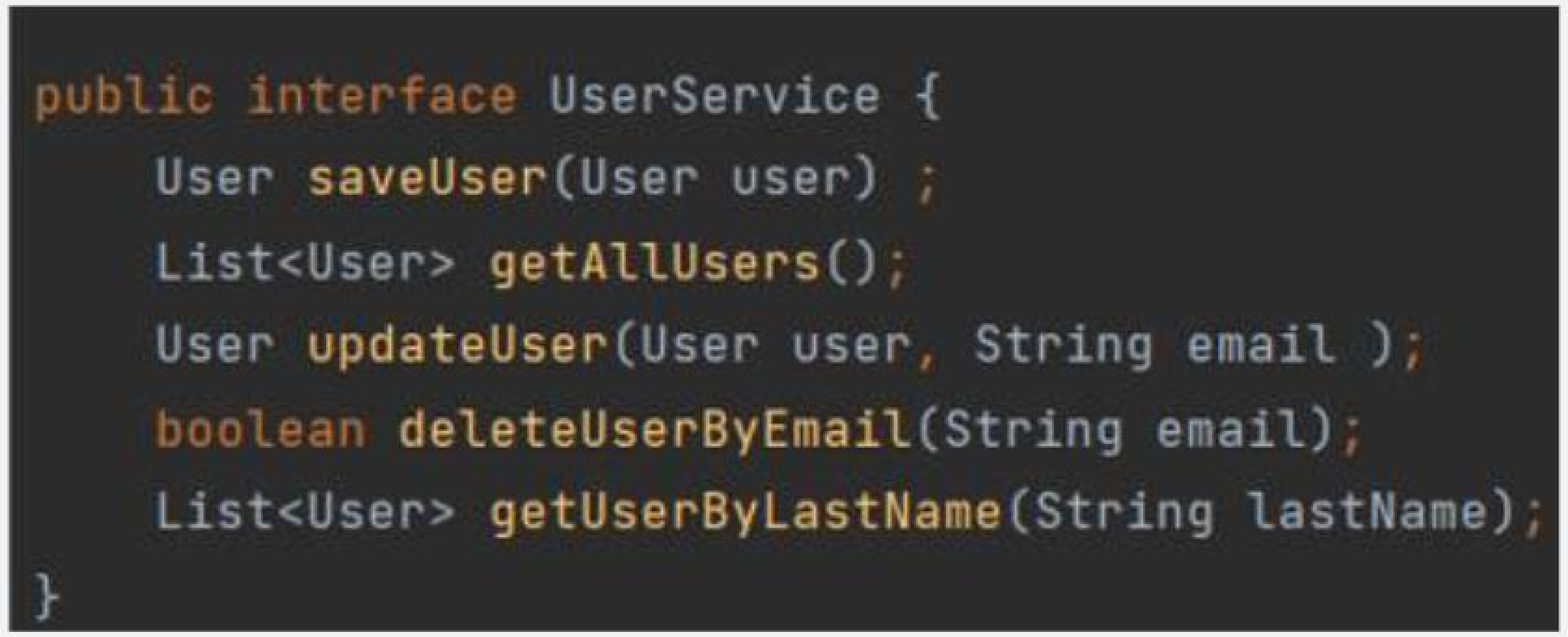
Which of the following statements is true about the CrudRespository?

1. `save(S entity)` **– saves multiple entities at a time.**
2. `findAllById(Iterable<ID> ids)` **– returns all entities with the given IDs.**
3. `deleteAll()` **– deletes all entities.**
4. `count()` **– It returns the number of entities.**

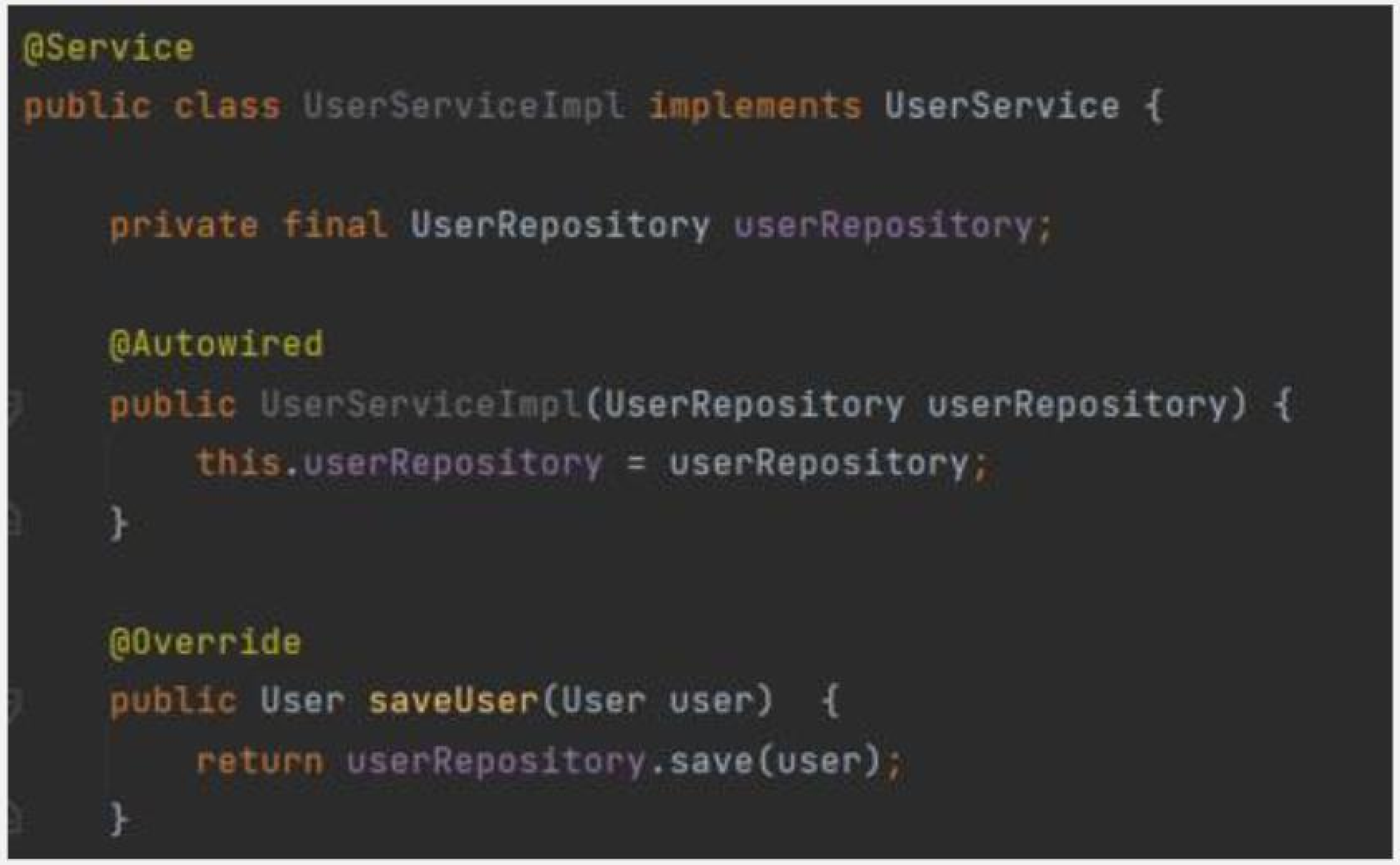
Menu

00:18:00:00:28

Slide Note



```java
public interface UserService {
    User saveUser(User user) ;
    List<User> getAllUsers();
    User updateUser(User user, String email );
    boolean deleteUserByEmail(String email);
    List<User> getUserByLastName(String lastName);
}
```



```java
@Service
public class UserServiceImpl implements UserService {

    private final UserRepository userRepository;

    @Autowired
    public UserServiceImpl(UserRepository userRepository) {
        this.userRepository = userRepository;
    }

    @Override
    public User saveUser(User user)  {
        return userRepository.save(user);
    }
```

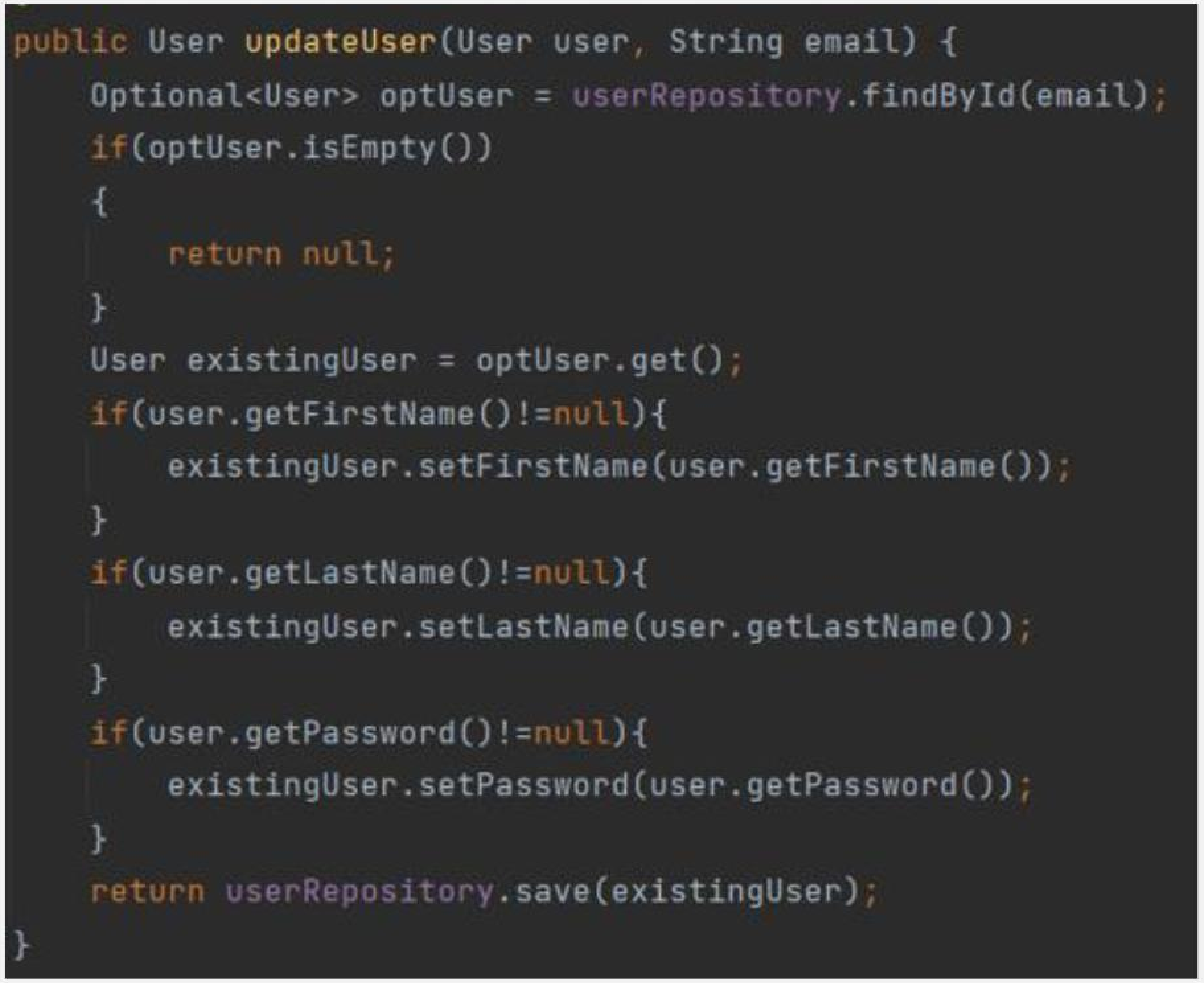
The Service Layer

- The service layer performs the business logic necessary to provide functionality for the application.
- If a new user is registered, the information must be saved in the database, and the service layer utilizes the methods of the Repository layer to perform the save functionality.
- The programmer has no code to save the user object in the database explicitly. Still, Spring Data JPA implicitly provides functionality when the Repository layer extends the JpaRepository.

Slide Note

Service Layer – Update Operation

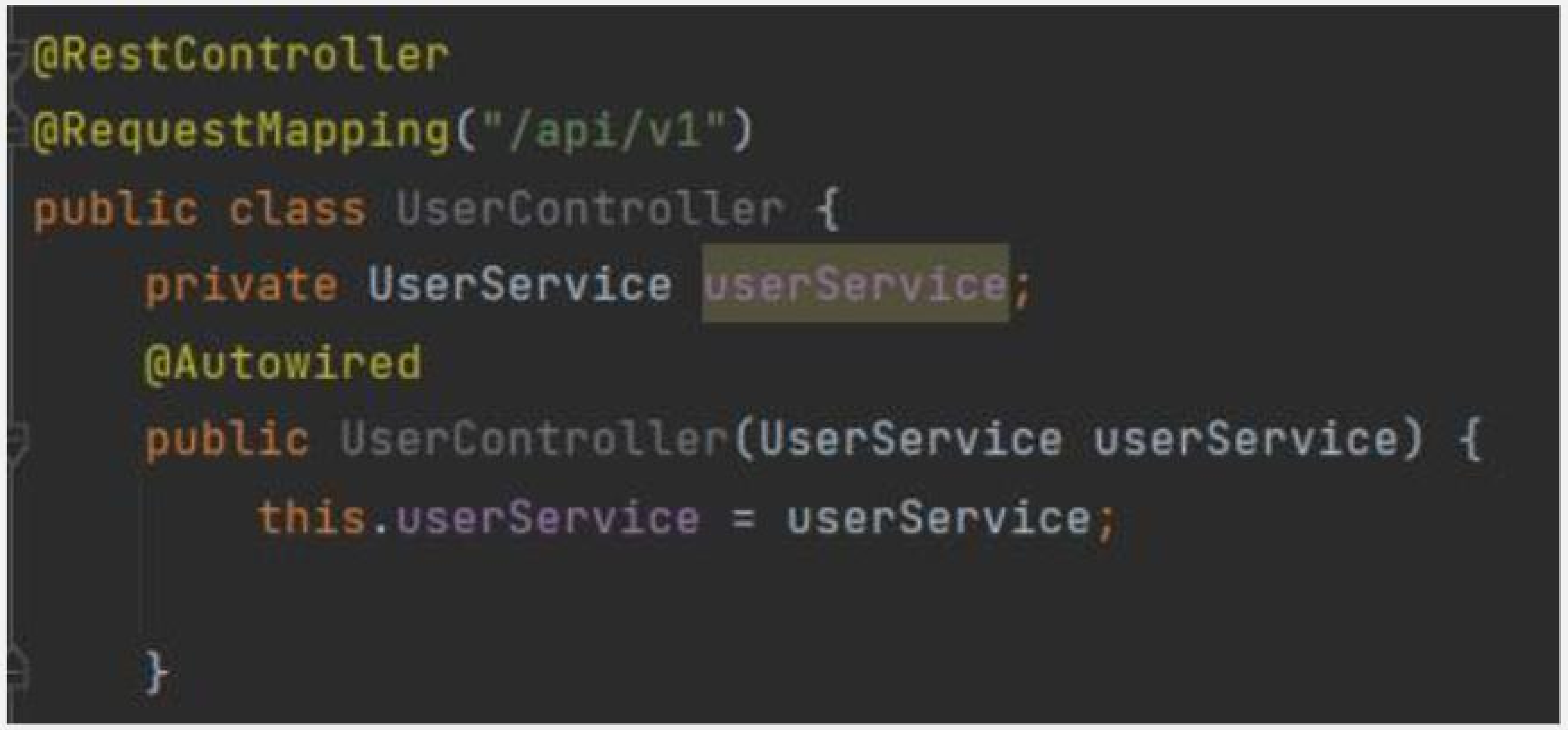
- When the details of a user need to be updated in the database, Spring Data JPA does not provide an update method.
- The programmer can write the required conditions, utilize the save method, and update the details in the database.



```
public User updateUser(User user, String email) {
    Optional<User> optUser = userRepository.findById(email);
    if(optUser.isEmpty())
    {
        return null;
    }
    User existingUser = optUser.get();
    if(user.getFirstName()!=null){
        existingUser.setFirstName(user.getFirstName());
    }
    if(user.getLastName()!=null){
        existingUser.setLastName(user.getLastName());
    }
    if(user.getPassword()!=null){
        existingUser.setPassword(user.getPassword());
    }
    return userRepository.save(existingUser);
}
```

Menu

Slide Note



```java
@RestController
@RequestMapping("/api/v1")
public class UserController {
    private UserService userService;
    @Autowired
    public UserController(UserService userService) {
        this.userService = userService;

    }
```

Controller Layer

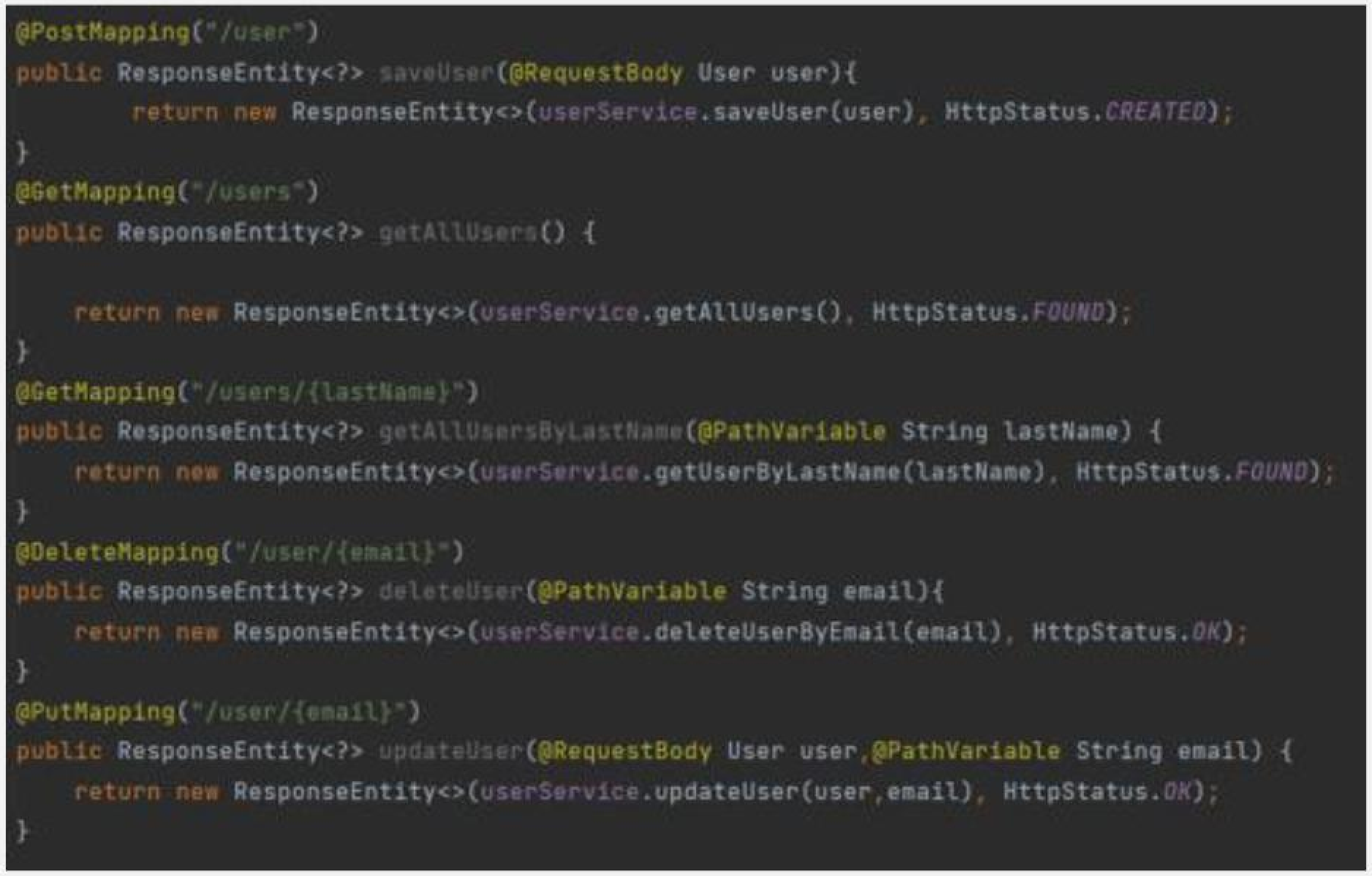
- The controller layer is responsible for handling requests and sending a response back.
- The controller layer interacts with the service layer to create and send the response back.
- The service layer object is autowired in the controller layer.

Menu

00:21 00:00:28 21/ 28

Controller Layer (Cont'd)

- The adjacent image shows all the handler methods to process the GET, POST, PUT, and DELETE requests.



```
@PostMapping("/user")
public ResponseEntity<?> saveUser(@RequestBody User user){
        return new ResponseEntity<>(userService.saveUser(user), HttpStatus.CREATED);
}
@GetMapping("/users")
public ResponseEntity<?> getAllUsers() {

    return new ResponseEntity<>(userService.getAllUsers(), HttpStatus.FOUND);
}
@GetMapping("/users/{lastName}")
public ResponseEntity<?> getAllUsersByLastName(@PathVariable String lastName) {
    return new ResponseEntity<>(userService.getUserByLastName(lastName), HttpStatus.FOUND);
}
@DeleteMapping("/user/{email}")
public ResponseEntity<?> deleteUser(@PathVariable String email){
    return new ResponseEntity<>(userService.deleteUserByEmail(email), HttpStatus.OK);
}
@PutMapping("/user/{email}")
public ResponseEntity<?> updateUser(@RequestBody User user,@PathVariable String email) {
    return new ResponseEntity<>(userService.updateUser(user,email), HttpStatus.OK);
}
```

Slide Note

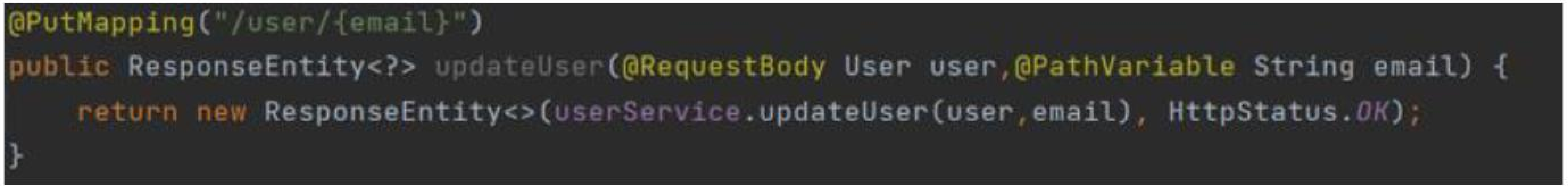
@PathVariable and @RequestBody

@PathVariable

- `@PathVariable` **is a Spring annotation that indicates a method parameter should be bound to a URI template variable.**

@RequestBody

- **The client sends data along with the request; this data is present in the request body.**
- **On the application side, the data in the request body must be read and deserialized into domain objects. This is done using the** `@RequestBody` **annotation.**



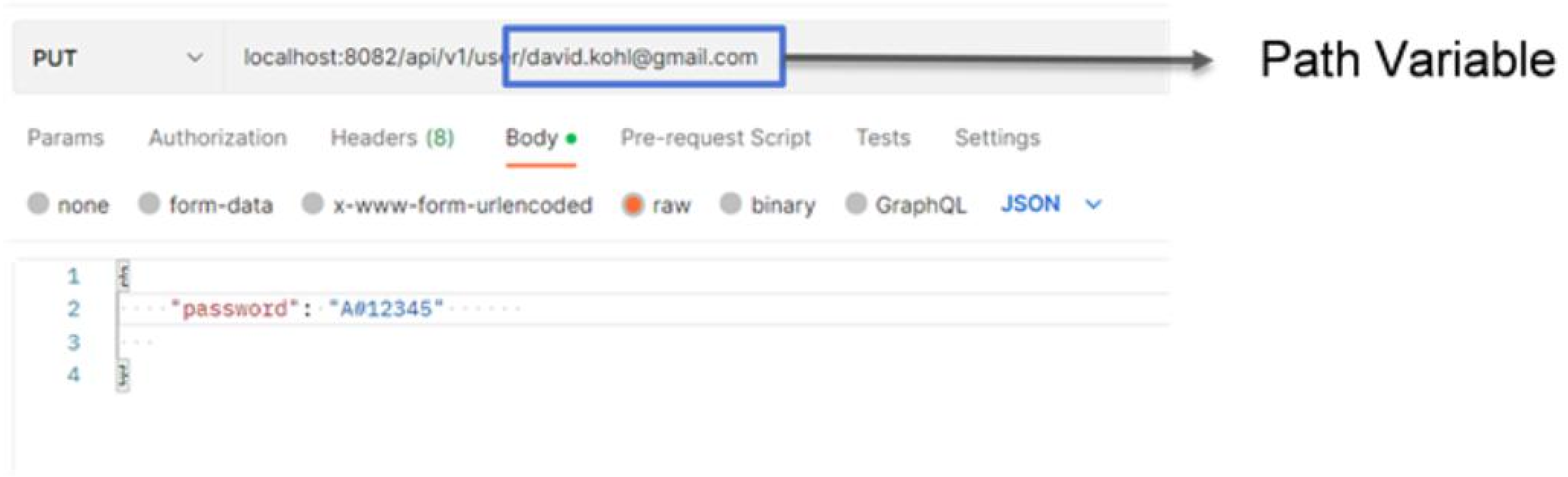
```
@PutMapping("/user/{email}")
public ResponseEntity<?> updateUser(@RequestBody User user,@PathVariable String email) {
    return new ResponseEntity<>(userService.updateUser(user,email), HttpStatus.OK);
}
```

- **To update the user details, the user data must be fetched based on email and then updated; thus, the email is passed in the URI as a path variable.**
- **The client passes the user data to be updated in the request body.**

Slide Note

Path Variable – Client

- The email address of the user David Kohl is passed in the URI of the request.
- In the application, the user data is retrieved based on the email.

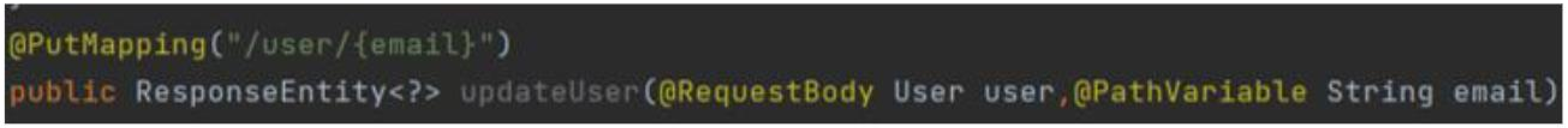


```
PUT  localhost:8082/api/v1/user/david.kohl@gmail.com
```

```
{
    "password": "A@12345"

}
```

- In the `@PutMapping`, the path variable is passed from the client URI and captured with `{<variable name>}` and the same is passed as a parameter to the controller method.



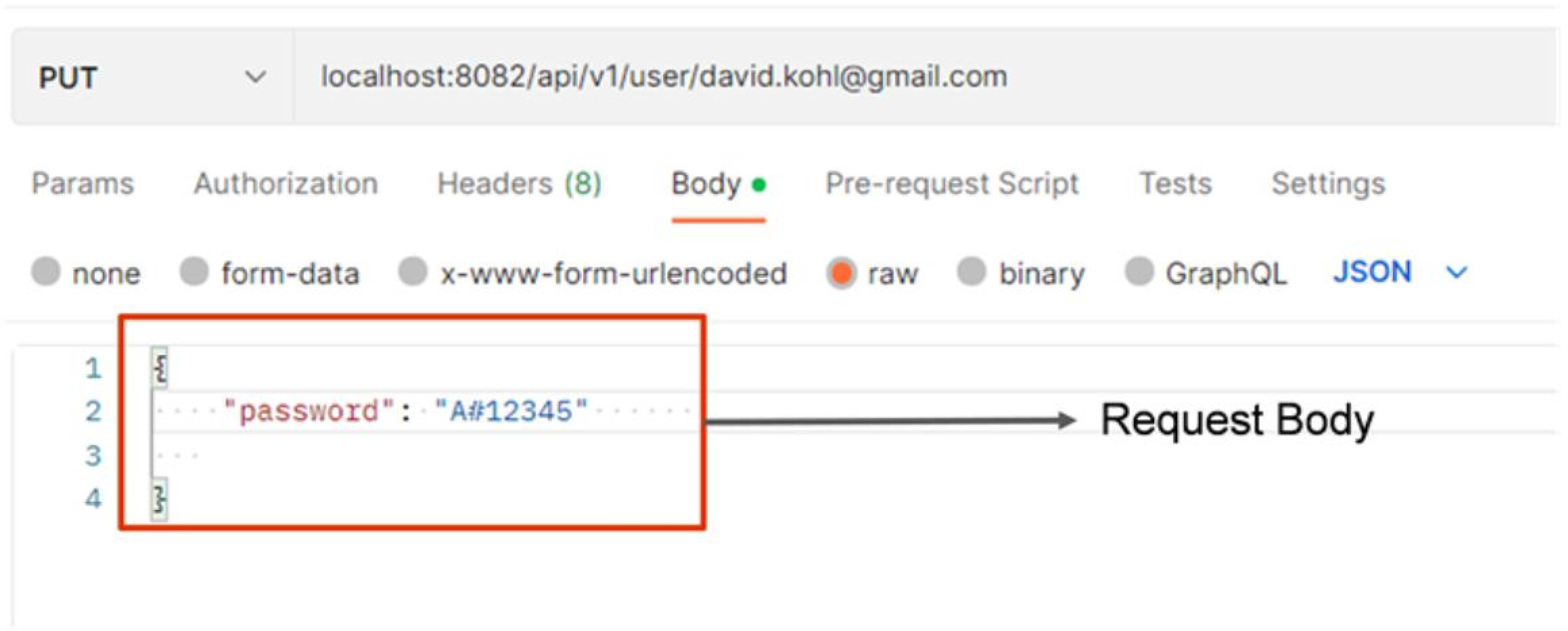
```
@PutMapping("/user/{email}")
public ResponseEntity<?> updateUser(@RequestBody User user,@PathVariable String email)
```

Menu

00:24 00:00:28 24/ 28

Request Body – Client

- The password of the user David Kohl is to be updated.
- To update the password in the database, the password is passed in the request body.



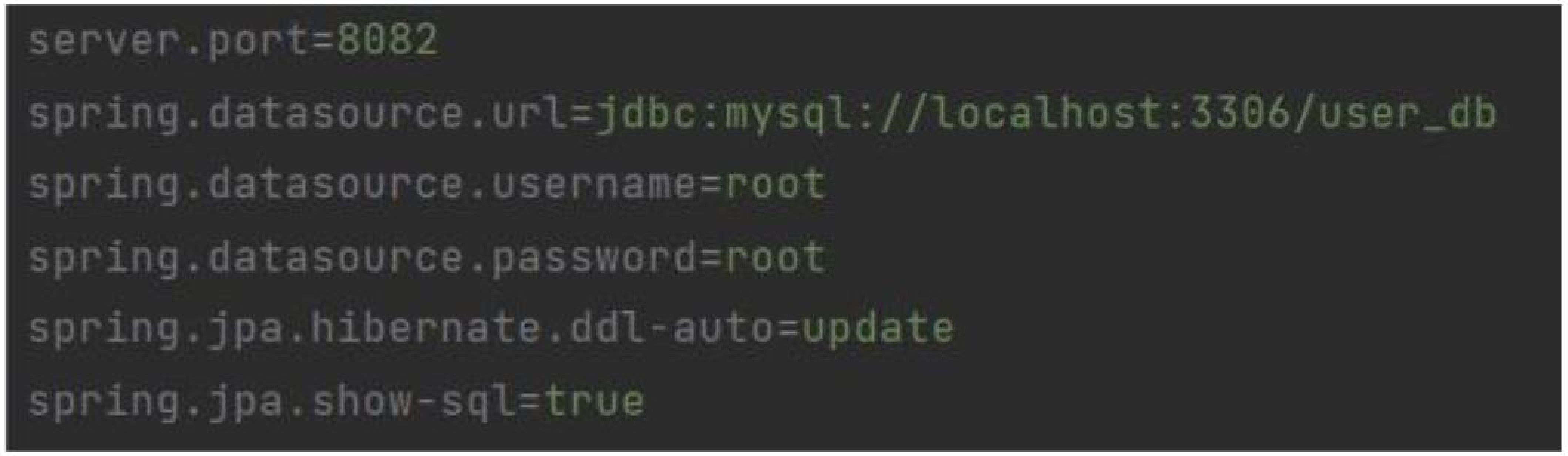
PUT localhost:8082/api/v1/user/david.kohl@gmail.com

Body: [✓ raw] JSON

```
{
    "password": "A#12345"
}
```

Slide Note

Setting the Properties for the Application



```
server.port=8082
spring.datasource.url=jdbc:mysql://localhost:3306/user_db
spring.datasource.username=root
spring.datasource.password=root
spring.jpa.hibernate.ddl-auto=update
spring.jpa.show-sql=true
```

Slide Note

User Details

Basic user details like email, password, first name, and last name need to be saved for a web application.

- Create a REST API that will manage the user information
- Use Spring Data JPA to s ave the user object in a MySQL database, update, and delete a user object
- List users by a specific last name
- List all the users in the system

Click here for the solution.

DEMO

Menu

00:27:00:00:28 27/ 28