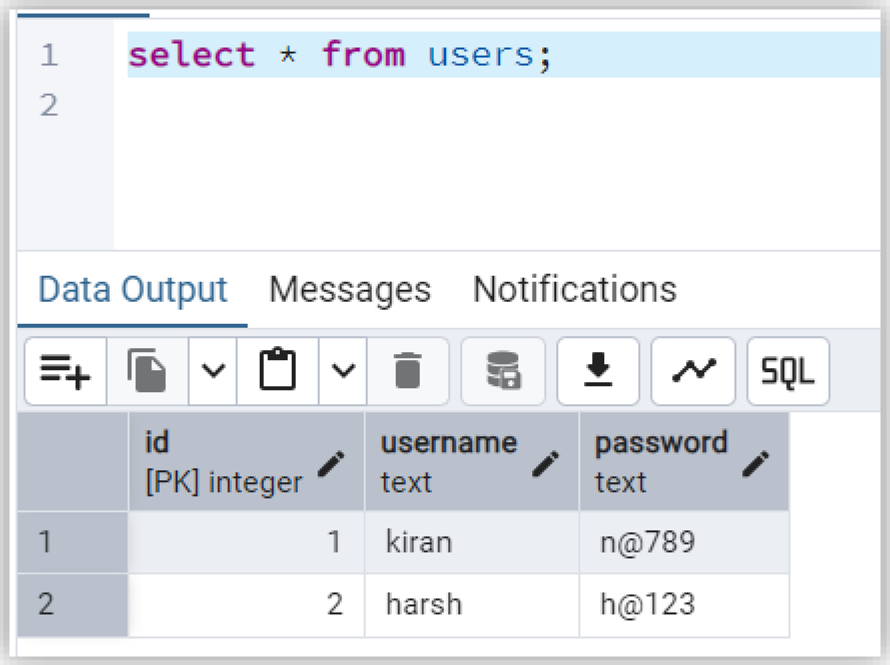
Creating User Table And DB Properties

Let's see how to set up Spring Security with database authentication

- **First, we need a database table to store user information. We'll create a users table with columns for:**
  - id (primary key)
  - username
  - password

- **We'll add records in users table**

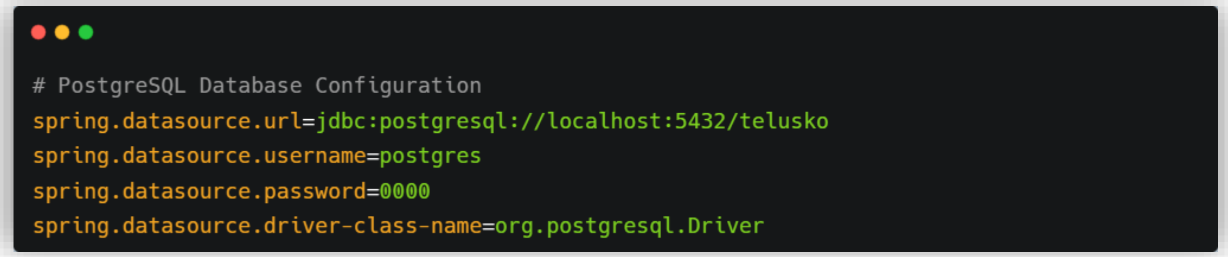


```sql
select * from users;
```

	id [PK] integer	username text	password text
1	1	kiran	n@789
2	2	harsh	h@123

- **Database Configuration in application.properties**

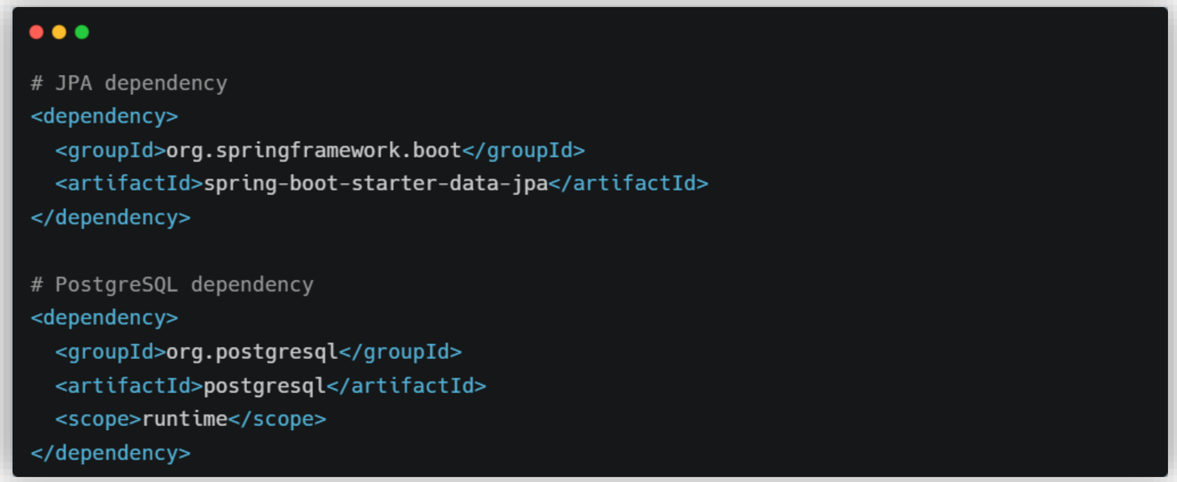
  We need to tell our Spring application how to connect to the PostgreSQL database:



```
# PostgreSQL Database Configuration
spring.datasource.url=jdbc:postgresql://localhost:5432/telusko
spring.datasource.username=postgres
spring.datasource.password=0000
spring.datasource.driver-class-name=org.postgresql.Driver
```

➢ Adding Required Dependencies

To connect our Java application to the database using JPA, we need to add these dependencies to our pom.xml file:



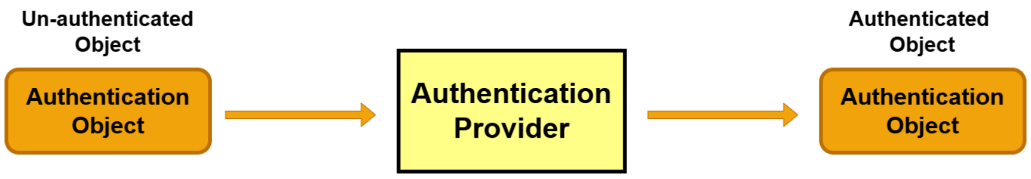
```
# JPA dependency
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-data-jpa</artifactId>
</dependency>

# PostgreSQL dependency
<dependency>
  <groupId>org.postgresql</groupId>
  <artifactId>postgresql</artifactId>
  <scope>runtime</scope>
</dependency>
```

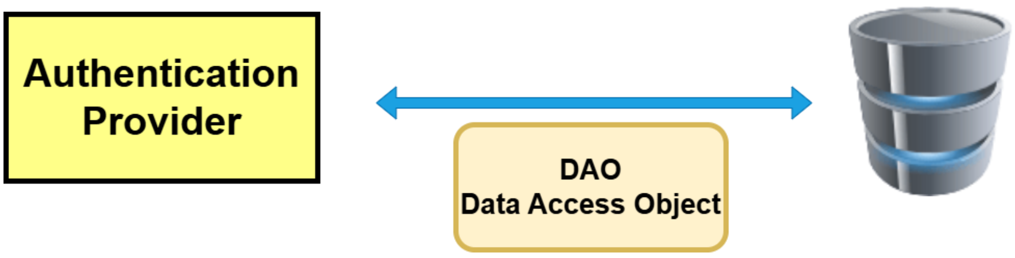
➢ Configuring Spring Security to Use Database Authentication

By default, Spring Security uses hardcoded values for authentication, but we want to use our database instead. We need to:

- Change the default Authentication Provider By default, Spring Security uses a built-in Authentication Provider that works with hardcoded values



- Switch to a database-aware Authentication Provider We need to configure a new Authentication Provider that connects to our database



- Create a Users entity class Finally, we need to specify how our application maps to the database table using a Users entity class. This class will represent our users table in Java code.

- This setup allows our Spring Security to authenticate users against the database instead of using hardcoded credentials. When a user attempts to log in, Spring Security will:

  - Take the provided username and password

  - Use the Authentication Provider to check these credentials against our database

  - Grant or deny access based on whether the credentials match