PROGRAMIRANJE INTERNET APLIKACIJA

Spring 2026



Elektrotehnički fakultet, Univerzitet u Beogradu
Školska 2026/2027

PREGLED

Sadržaj

01 **Spring i Spring Boot**

Kratak uvod i ciljevi Spring Boot-a

02 **Instalacija**

Java 25, VS Code, Spring Initializr

03 **Prazan projekat**

Struktura praznog projekta

04 **Backend – User API**

User, UserController, Postman

05 **Frontend povezivanje**

Model, HttpClient, servis, komponenta

06 **Baza podataka**

MySQL, JDBC, DB.java

07 **Primeri**

Login, CRUD, datumi

01

Spring i Spring Boot

Kratak uvod pre instalacije i koda.



Spring · Spring Boot · docs.spring.io

Šta je Spring?

Najpopularniji *framework* za razvoj Java aplikacija

Spring Framework

- Osnova ekosistema na `spring.io`
- **IoC / DI** – objekti se ne prave ručno sa `new`, nego ih kontejner „ubrizga“
- **AOP**, MVC, pristup podacima…
- Gradivni blokovi koje **Boot** kasnije spaja

Šta sve pokriva

- Spring Boot
- Spring Security
- Spring Data
- Spring Cloud
- Spring Batch
- Spring Integration
- Spring AI

Zašto nam treba

- Manje „plumbing“ koda
- Standardan način za DI i web
- Velika zajednica i docs
- Boot = brži put do radnog API-ja

Šta je Spring Boot?

Način da brzo napraviš stand-alone Spring aplikaciju.

„Spring Boot helps you to create **stand-alone, production-grade** Spring-based applications that you can run.“ - docs.spring.io/spring-boot

Brži start

- Manje ručne konfiguracije
- Starteri donose zavisnosti
- Ugrađeni Tomcat (web)

Auto-configuration

- `@EnableAutoConfiguration`
- Pogađa podešavanja iz jar-ova
- Kad zatreba – možeš da odstupiš

Bez XML-a

- Nema obaveznog XML-a
- Nema generisanja koda
- `java -jar` ili IDE Run

02

Instalacija

Java 25, VS Code i Spring Initializr.



Java 25 VS Code Initializr

Šta nam treba pre prvog projekta

Na fakultetu: **Java 25**, **VS Code**, projekat preko **Spring Initializr**.

1

Java 25

JDK **25.0.4.1**. na PATH. Boot **4.1.2** radi od Java 17 do 27 – mi koristimo 25.

terminal

```
$ java -version
openjdk version "25" …
```

2

VS Code

Extension Pack for Java + Spring Boot Extension Pack. Projekat = Maven folder.

VS Code

```
Ctrl+Shift+P
→ Command Palette
```

3

Initializr

Ili `start.spring.io`, ili u VS Code: **Ctrl+Shift+P** → `Spring Initializr: Create a Maven Project…`

izbor

```
Boot 4.1.2 · Java 25
+ Spring Web (webmvc)
```

Spring Initializr

Ista verzija kao u `pom.xml` projekta: Boot **4.1.2**, Java **25**.

Polje	Vrednost na vežbama
Project	`Maven`
Language	`Java`
Spring Boot	`4.1.2`
Group / Artifact	`com.example / backend`
Packaging	`Jar`
Java	`25`
Dependency	`Spring Web (webmvc)`

Posle Generate

- Otvoriš kreiran projekat
- Ili koristiš gotov `backend_Spring.7z`
- Pokretanje: Run na `BackendApplication` ili `./mvnw spring-boot:run`

03

Prazan projekat

Šta dobijaš iz Spring Initializr / backend_Spring.7z.



pom.xml **BackendApplication** **struktura**

Struktura novog Spring Boot projekta

Ovo je sadržaj arhive `backend_Spring.7z`

backend_Spring.7z

```
backend/
├── pom.xml
├── mvnw / mvnw.cmd
├── .mvn/wrapper/
├── src/main/java/com/example/backend/
│   └── BackendApplication.java
├── src/main/resources/
│   ├── application.properties
│   ├── static/
│   └── templates/
└── src/test/java/.../BackendApplicationTests.java
```

pom.xml

- Parent: `spring-boot-starter-parent` **4.1.2**
- `java.version` = **25**
- Zavisnost: `spring-boot-starter-webmvc`

BackendApplication

- Ulazna tačka – `main` + `@SpringBootApplication`
- Boot podiže kontekst i ugrađeni server
- Podrazumevani port: **8080**

BackendApplication – ulaz u aplikaciju

`@SpringBootApplication` uključuje auto-configuration i component scan.

…/backend/BackendApplication.java

```java
package com.example.backend;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class BackendApplication {

  public static void main(String[] args) {
    SpringApplication.run(
        BackendApplication.class, args);
  }
}
```

`@SpringBootApplication`

- Meta-anotacija: konfiguracija + scan
- Uključuje auto-configuration

Pokretanje

- VS Code: Run na `main`
- Ili: `./mvnw spring-boot:run`
- Log: Started BackendApplication…

Ključni delovi pom.xml

Parent i starteri – Boot bira kompatibilne verzije biblioteka.

pom.xml (skraćeno)

```xml
<parent>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-parent</artifactId>
  <version>4.1.2</version>
</parent>

<properties>
  <java.version>25</java.version>
</properties>

<dependencies>
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-webmvc</artifactId>
  </dependency>
</dependencies>
```

Parent POM

- Maven defaulti + dependency management
- Ne moraš da pišeš verzije za zvanično testirane biblioteke

`starter-webmvc`

- Spring MVC + ugrađeni Tomcat
- Dovoljno za REST kontrolere
- (Boot 4: webmvc umesto starog `web`)

04

Backend – User API

Od modela do REST endpointa i Postmana.



User

UserController

Postman

Model: klasa User

Običan POJO – polja + konstruktor + get/set. Spring ga serijalizuje u JSON.

User.java

```java
package com.example.backend;

public class User {
  private String firstname;
  private String lastname;
  private String username;
  private String password;

  public User(String firstname, String lastname,
              String username, String password) {
    this.firstname = firstname;
    this.lastname = lastname;
    this.username = username;
    this.password = password;
  }
  // getFirstname / setFirstname …
  // getLastname / setLastname …
  // getUsername / setUsername …
  // getPassword / setPassword …
}
```

Zašto get/set?

- JSON čita **gettere**
- Enkapsuliramo privatna polja

Polja u primeru

- `firstname`, `lastname`
- `username`, `password`
- Isti nazivi kasnije u Angular modelu

UserController – anotacije

REST kontroler mapira URL-ove na metode koje vraćaju podatke (JSON).

UserController.java (skraćeno)

```java
@RestController
@RequestMapping("/users")
@CrossOrigin(origins = "http://localhost:4200/")
public class UserController {

  @GetMapping("/hello")
  public String helloWorld() {
    return "Hello world";
  }

  private List<User> allUsers;

  // … initUsers, getAll, getByUsername …
}
```

@RestController = @Controller + @ResponseBody

- Klasa obrađuje HTTP zahteve
- Povratna vrednost ide u telo odgovora

@RequestMapping("/users")

- Zajednički prefiks svih ruta
- npr. `/users/getAll`

@CrossOrigin

- Dozvoljava Angular na `localhost:4200`
- Bez toga browser blokira zahtev (CORS)

Prvi endpoint: /users/hello

Najjednostavniji GET – vraća običan string. Pogodno za prvi test.

UserController.java

```java
@GetMapping("/hello")
public String helloWorld() {
  return "Hello world";
}
```

odgovor

```
GET http://localhost:8080/users/hello

Hello world
```

Test u Postmanu

- Pokreni backend (port **8080**)
- Novi request: metoda **GET**
- URL: `http://localhost:8080/users/hello`
- Send → u Body vidiš `Hello world`
- Isti obrazac kasnije za `/getAll`

Lista korisnika: getAll

`@PostConstruct` puni listu posle kreiranja bean-a; GET vraća JSON niz.

UserController.java

```java
private List<User> allUsers;

@PostConstruct
private void initUsers() {
  allUsers = new ArrayList<>();
  allUsers.add(new User(
      "Petar", "Petrovic", "pera", "pera123"));
  allUsers.add(new User(
      "Mika", "Mikic", "mika", "mika213"));
}

@GetMapping("/getAll")
public List<User> getUsers() {
  return this.allUsers;
}
```

@PostConstruct

- Poziva se jednom posle konstrukcije
- Ovde: početni podaci u memoriji
- (nema baze – lista u RAM-u), za sad ☺

GET /users/getAll

- Vraća `List<User>` → JSON niz
- Postman / browser



Jedan korisnik: getByUsername

Deo putanje `{username}` ulazi u metodu preko `@PathVariable`.

UserController.java

```java
@GetMapping("getByUsername/{username}")
public User getUser(
    @PathVariable String username) {
  for (int i = 0; i < this.allUsers.size(); i++) {
    if (username.equals(
        this.allUsers.get(i).getUsername())) {
      return this.allUsers.get(i);
    }
  }
  return null;
}
```

URL primer

- `/users/getByUsername/pera`
- `username = "pera"`
- Odgovor: jedan User (JSON)

Napomena

- Ako nema poklapanja → `null`
- Može HTTP status (404)

Isto, ali sa @RequestParam

Parametar nije u putanji – ide u **query string** (`?username=...`).

Alternativa

```java
@GetMapping("getByUsername")
public User getUser(
    @RequestParam String username) {
  // … ista petlja kroz allUsers …
  return null;
}
```

@RequestParam → /users/getByUsername?username=pera

Na času

```java
@GetMapping("getByUsername/{username}")
public User getUser(
    @PathVariable String username) { … }
```

@PathVariable → /users/getByUsername/pera

Provera u Postmanu

Pre fronta – uveri se da API radi. Tri GET zahteva.

Metoda	URL	Očekivano
GET	`…/users/hello`	Hello world
GET	`…/users/getAll`	JSON niz (2 usera)
GET	`…/users/getByUsername/pera`	jedan User objekat

Baza URL-a: `http://localhost:8080` – backend mora da radi.

Ako front zove sa `4200`, CORS je već rešen sa `@CrossOrigin` na kontroleru.

05

Frontend povezivanje

Angular čita isti API: model → HttpClient → servis → UI.



model

provideHttpClient

UserService

Angular model: User

Ista polja kao na backu – TypeScript klasa u `models/`.

src/app/models/user.ts

```ts
export class User {
  firstname = '';
  lastname = '';
  username = '';
  password = '';
}
```

Zašto model?

- Tip za `HttpClient.get<User[]>`
- Ista imena polja kao JSON sa servera
- Komponenta i servis dele jedan tip

provideHttpClient u app.config

Bez ovoga `HttpClient` nije dostupan za `inject()`.

src/app/app.config.ts

```ts
import { ApplicationConfig,
  provideBrowserGlobalErrorListeners } from '@angular/core';
import { provideRouter } from '@angular/router';
import { routes } from './app.routes';
import { provideHttpClient } from '@angular/common/http';

export const appConfig: ApplicationConfig = {
  providers: [
    provideBrowserGlobalErrorListeners(),
    provideRouter(routes),
    provideHttpClient(),
  ],
};
```

Korak

- Import: `provideHttpClient`
- Sada servis može `inject(HttpClient)`
- Radi uz postojeći `provideRouter`

- Od verzije Angular 21 nije obavezno da se dodaje provideHttpClient u appConfig, a dodaje se samo ukoliko je potrebno definisati dodatne parametre!

UserService – samo getAllUsers

Servis drži signal sa listom i puni ga odgovorom sa backenda.

src/app/services/user-service.ts

```ts
@Service()
export class UserService {
  private _allUsers = signal<User[]>([]);
  allUsers = this._allUsers.asReadonly();

  private http = inject(HttpClient);
  backendUrl = 'http://localhost:8080/users';

  getAllUsers() {
    this.http.get<User[]>(
        `${this.backendUrl}/getAll`).subscribe({
      next: users => this._allUsers.set(users),
      error: err =>
        console.error('Greška pri učitavanju', err),
    });
  }
}
```

Signal

- `_allUsers` – upis iznutra
- `allUsers` – readonly za UI

HTTP

- `get<User[]>` očekuje niz
- `subscribe` upisuje u signal
- URL: `/users/getAll`

Komponenta Users + ruta

Tri fajla: TS zove servis, ruta ga prikaže, HTML čita signal.

src/app/users/users.ts

```ts
export class Users implements OnInit {
  private userService = inject(UserService);
  allUsers = this.userService.allUsers;

  ngOnInit(): void {
    this.userService.getAllUsers();
  }
}
```

1 · inject

- Servis se ne pravi sa `new`, već se injectuje
- `allUsers` – readonly signal

2 · ngOnInit

- Pri ulasku na stranicu: `getAllUsers()`
- HTTP puni signal → tabela se osveži

app.routes.ts

```ts
export const routes: Routes = [
  { path: '', component: Users },
];
```

users.html

```html
@for (user of allUsers(); track user.username) {
  <td>{{ user.firstname }}</td>
  <td>{{ user.lastname }}</td>
}
```

Tok: komponenta ↔ servis ↔ backend

Jedna putanja podataka – od HTTP odgovora do tabele.

1

ngOnInit

`getAllUsers()` na servisu

2

HttpClient

GET `/users/getAll`

3

signal.set

upis `User[]` u `_allUsers`

4

šablon

`@for (user of allUsers())`

Provera

- Backend na **8080**, Angular na **4200**
- U tabeli: Petar / Mika (iz `initUsers`)
- Greška u konzoli? Proveri CORS, URL i da li je `provideHttpClient` dodat

Šta je Observable?

`HttpClient.get()` ne vraća odmah podatke – vraća **Observable**, tok koji čeka pretplatu.

Observable

- Objekat koji **opisuje** zahtev (npr. GET)
- Ništa se ne šalje dok ne pozoveš `subscribe`
- Kad stigne odgovor → poziva se `next`
- Ako padne mreža / greška → `error`
- Tip: `Observable<User>` ili `Observable<User[]>`

subscribe

- `subscribe({ next, error })` – prijava na tok
- Bez `subscribe` – nema HTTP poziva
- U servisu **ili** u komponenti – biraš gde u zavisnosti od logike

Kod nas

- `getAllUsers` – `subscribe` **u servisu** (puni signal)
- `getUserByUsername` – **vraća** Observable
- Komponenta se pretplati i upisuje `selectedUser, jer je ovo specifično za tu komponentu`

Detalji: servis vraća Observable

Metoda ne radi `subscribe` – samo prosleđuje rezultat `http.get`.

src/app/services/user-service.ts

```
getUserByUsername(username: string) {
  return this.http.get<User>(
    `${this.backendUrl}/getByUsername/${username}`);
}
```

Zašto return?

- Komponenta odlučuje šta radi sa User-om
- Isti obrazac: GET → Observable

Napomena: dok komponenta ne pozove `.subscribe(...)`, zahtev **još nije poslat**. A tek kada stigne odgovor se izvršava kod unutar subscribe. Dok odgovor ne stigne, ostatak koda nastavlja sa izvršavanjem.

Detalji: subscribe + selectedUser

Komponenta se pretplati, upisuje korisnika u signal, šablon ga prikaže sa `@if`.

src/app/users/users.ts

```ts
selectedUser = signal<User | null>(null);

getUserDetails(user: User) {
  this.userService
    .getUserByUsername(user.username)
    .subscribe(user => {
      this.selectedUser.set(user);
    });
}
```

users.html

```html
@if (selectedUser()) {
  <hr />
  <h4>Details:</h4>
  Username: {{ selectedUser()?.username }}
  <br />
  Password: {{ selectedUser()?.password }}
}
```

1 · signal

- `null` dok nema izbora
- posle GET → `set(user)`

2 · subscribe

- `next` dobija `User`
- isti Observable sa servisa

3 · @if

- Prikaz samo kad ima vrednost
- `?.` – optional chaining

06

Baza podataka

MySQL, JDBC komande i čitanje korisnika iz tabele.



MySQL 26.7.0 JDBC DB.java

MySQL Community Server 26.7.0

Na fakultetu koristimo verziju **26.7.0** (Windows, 64-bit).

Preuzimanje

- Zvanična stranica: `https://dev.mysql.com/downloads/mysql/26.7.html`
- Izaberi **Windows (x86, 64-bit), MSI Installer (ili drugi OS)**
- Fajl: `mysql-26.7.0-winx64.msi`
- Posle instalacije: server na portu **3306**

Šta nam treba

- MySQL Server (servis)
- Baza npr. `baza`
- Tabela `korisnici`
- Nalog (kod nas: `root`, prazna lozinka)

download

```
# link za skidanje (Community Server 26.7.0)
https://dev.mysql.com/downloads/mysql/26.7.html
```

Connection i PreparedStatement

Prvo veza ka bazi, zatim pripremljen SQL upit.

Connection

- Otvorena **veza** ka MySQL-u
- `DB.source().getConnection()`
- Preko nje praviš naredbe (`PreparedStatement`)
- Zatvori je kad završiš – **try-with-resources**
- Jedna veza = jedan „razgovor“ sa bazom

PreparedStatement

- Pripremljen SQL: `conn.prepareStatement(sql)`
- Upit se šalje bazi kao **šablon**, pa se popune vrednosti
- Bezbednije od `"..." + username` (SQL injection)
- Isti objekat za SELECT / INSERT / UPDATE / DELETE
- Izvršavanje: `executeQuery` ili `executeUpdate`

executeQuery i executeUpdate

Dve metode na `PreparedStatement` – biraj prema tipu SQL naredbe.

executeQuery()

- Za **SELECT** (čitanje)
- Vraća **`ResultSet`** – skup redova rezultata
- `while (rs.next())` – prolazak kroz redove
- `rs.getString("kolona")` – vrednost ćelije
- Kod nas: `select * from korisnici`

executeUpdate()

- Za **INSERT**, **UPDATE**, **DELETE** (izmena)
- Vraća **`int`** – broj pogođenih redova
- `1` = jedan red ubačen / izmenjen / obrisan
- `0` = nijedan red nije pogođen
- **Ne** vraća `ResultSet`

Redosled JDBC poziva

Uvek isti obrazac: veza → upit → izvrši → čitaj redove.

šablon

```
try (Connection conn = …getConnection();
     PreparedStatement stm =
         conn.prepareStatement("select * from korisnici")) {

  ResultSet rs = stm.executeQuery();
  while (rs.next()) {
    String username = rs.getString("username");
    // … ostala polja → new User(...)
  }
} catch (SQLException e) {
  // obrada greške
}
```

executeQuery

- Za **SELECT**
- Vraća `ResultSet`

try-with-resources

- `Connection` i `PreparedStatement`
- automatski se zatvaraju
- manje curenja veza ka bazi

Zavisnosti u pom.xml

Dva startera/drajvera: JDBC podrška Spring-a + MySQL Connector/J.

pom.xml

```xml
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-jdbc</artifactId>
</dependency>

<dependency>
  <groupId>com.mysql</groupId>
  <artifactId>mysql-connector-j</artifactId>
  <version>9.7.0</version>
</dependency>
```

starter-jdbc

- DataSource, JDBC šablon
- Veza ka bazi u Spring stilu

mysql-connector-j

- Drajver: `com.mysql.cj.jdbc.Driver`
- Verzija u projektu: **9.7.0**

./mvnw spring-boot:run

Klasa DB – DataSource bean

`@Configuration` + `@Bean`: Spring zna kako da otvori vezu ka MySQL-u.

DB.java

```java
@Configuration
public class DB {
  @Bean
  public static DataSource source() {
    DriverManagerDataSource dmds =
        new DriverManagerDataSource();
    dmds.setDriverClassName(
        "com.mysql.cj.jdbc.Driver");
    dmds.setUrl(
        "jdbc:mysql://localhost:3306/baza");
    dmds.setUsername("root");
    dmds.setPassword("");
    return dmds;
  }
}
```

URL

- `localhost:3306` – lokalni MySQL
- Baza: **`baza`**

Poziv

- `DB.source().getConnection()`
- Isti bean koristi kontroler

getAll – čitanje iz tabele korisnici

Umesto liste u memoriji: SELECT iz MySQL-a, pa `User` objekti.

UserController.java

```java
@GetMapping("/getAll")
public List<User> getUsers() {
  try (Connection conn = DB.source().getConnection();
       PreparedStatement stm = conn.prepareStatement(
           "select * from korisnici")) {
    ResultSet rs = stm.executeQuery();
    while (rs.next()) {
      User u = new User(
          rs.getString("firstname"),
          rs.getString("lastname"),
          rs.getString("username"),
          rs.getString("password"));
      this.allUsers.add(u);
    }
  } catch (SQLException e) { }
  return this.allUsers;
}
```

Tabela

- `korisnici`
- Kolone: `firstname`, `lastname`, `username`, `password`

Provera

- MySQL radi, baza `baza` postoji
- Postman: GET `/users/getAll`
- Front ostaje isti

Nova struktura paketa

Podela po ulozi: modeli, baza, DAO, kontroleri.

src/main/java/…/backend/

```
com/example/backend/
├── BackendApplication.java
├── models/
│   └── User.java
├── db/
│   ├── DB.java
│   └── dao/
│       ├── UserRepoInterface.java
│       └── UserRepo.java
└── controllers/
    └── UserController.java
```

models/

- POJO klase – podaci (npr. `User`)

db/

- `DB.java` – DataSource
- `dao/` – pristup tabelama

controllers/

- REST endpointi
- Zovu DAO, ne pišu SQL

DAO pattern – Data Access Object

Sloj koji **enkapsulira** pristup bazi. Kontroler ne zna za SQL ni JDBC.

Zašto DAO?

- Jedno mesto za SQL i JDBC
- Lakše menjati upite / bazu
- Testiranje i čitljivost

Uloga

- **Interface** – šta nudi (npr. `getUsers`)
- **Klasa** – kako (JDBC + SQL)
- Ulaz/izlaz: `User`, `List<User>`
- Ne vraća `ResultSet` napolje

Kod nas

- `UserRepoInterface`
- `UserRepo` + `@Repository`
- Spring ubacuje `DataSource`
- Kontroler zove `userRepo`

Interfejs: UserRepoInterface

Ugovor: koje operacije nad korisnicima postoje – bez JDBC detalja.

db/dao/UserRepoInterface.java

```java
package com.example.backend.db.dao;

import java.util.List;
import com.example.backend.models.User;

public interface UserRepoInterface {
  List<User> getUsers();
}
```

Napomena

- Samo potpis metode
- Implementacija je u `UserRepo`
- Kontroler zavisi od ponašanja, ne od SQL-a

UserRepo – JDBC u DAO sloju

`@Repository` + konstruktor sa `DataSource`.

db/dao/UserRepo.java

```java
@Repository
public class UserRepo implements UserRepoInterface {
  private final DataSource DB;

  UserRepo(DataSource DB) {
    this.DB = DB;
  }

  @Override
  public List<User> getUsers() {
    List<User> allUsers = new ArrayList<>();
    try (Connection conn = DB.getConnection();
         PreparedStatement stm = conn.prepareStatement(
             "select * from korisnici")) {
      ResultSet rs = stm.executeQuery();
      while (rs.next()) {
        allUsers.add(new User(
            rs.getString("firstname"),
            rs.getString("lastname"),
            rs.getString("username"),
            rs.getString("password")));
      }
    } catch (SQLException e) { }
    return allUsers;
  }
}
```

`@Repository`

- Spring bean (komponenta)
- Može se `inject`-ovati

DataSource

- Isti bean iz `DB.java`
- `DB.getConnection()` u try-with-resources

Kontroler zove DAO

UserController više ne otvara Connection – samo prosleđuje rezultat repozitorijuma.

controllers/UserController.java

```java
@RestController
@RequestMapping("/users")
@CrossOrigin(origins = "http://localhost:4200/")
public class UserController {
  private final UserRepo userRepo;

  public UserController(UserRepo userRepo) {
    this.userRepo = userRepo;
  }

  @GetMapping("/getAll")
  public List<User> getUsers() {
    return this.userRepo.getUsers();
  }
}
```

DI

- Konstruktor – Spring ubacuje `UserRepo`
- Bez `new UserRepo()`

Tok

- HTTP → kontroler → DAO → MySQL
- Front i URL ostaju isti

07

Primeri

Login, CRUD i rad sa datumima



Login **CRUD** **LocalDate**

Backend: login u interfejsu

Nova metoda u DAO ugovoru – provera korisnika po username i password.

db/dao/UserRepoInterface.java

```java
public interface UserRepoInterface {
  List<User> getUsers();
  User login(User u);
}
```

Ulaz / izlaz

- Ulaz: `User` sa `username` i `password`
- Izlaz: `User` ako postoji u bazi
- Ili `null` ako podaci nisu tačni

Kontroler i DAO sada imaju i `getUsers` i `login` – ista podela odgovornosti.

UserRepo.login – SQL sa parametrima

`?` markeri + `setString` – bez spajanja stringova u upit, zaštita od SQLInject.

db/dao/UserRepo.java

```java
@Override
public User login(User u) {
  try (Connection conn = DB.getConnection();
       PreparedStatement stm = conn.prepareStatement(
           "select * from korisnici "
           + "where username=? and password=?")) {
    stm.setString(1, u.getUsername());
    stm.setString(2, u.getPassword());
    ResultSet rs = stm.executeQuery();
    if (rs.next()) {
      return new User(
          rs.getString("firstname"),
          rs.getString("lastname"),
          rs.getString("username"),
          rs.getString("password"));
    }
  } catch (SQLException e) { }
  return null;
}
```

Parametri

- `setString(1, …)` → prvi `?`
- `setString(2, …)` → drugi `?`

Rezultat

- `rs.next()` true → nađen korisnik
- inače → `null`

POST /users/login

`@RequestBody` – JSON telo zahteva postaje `User` objekat.

controllers/UserController.java

```java
@PostMapping("/login")
public User login(@RequestBody User u) {
  return this.userRepo.login(u);
}
```

HTTP

- Metoda: **POST** (ne GET)
- URL: `/users/login`
- Body (JSON): `{ "username", "password" }`
- Odgovor: User ili prazno/`null`

Test u Postmanu: Body → raw → JSON, pa Send. Front šalje isto preko `HttpClient.post`.

Frontend: UserService.login

Šalje objekat u telu POST zahteva – vraća Observable kao i detalji.

src/app/services/user-service.ts

```
login(username: string, password: string) {
  const data = {
    username: username,
    password: password,
  };
  return this.http.post<User>(
      `${this.backendUrl}/login`, data);
}
```

Bitno

- `post`, ne `get`
- Drugi argument = **body**
- Tip odgovora: `User`
- Komponenta radi `subscribe`

Komponenta Login + navigacija

Pri uspehu: `router.navigate(['user'])`

src/app/login/login.ts

```ts
export class Login {
  username = signal('');
  password = signal('');
  private userService = inject(UserService);
  private router = inject(Router);

  login() {
    this.userService
      .login(this.username(), this.password())
      .subscribe(user => {
        if (user) {
          this.router.navigate(['user']);
        } else {
          alert('Bad data');
        }
      });
  }
}
```

Router

- `inject(Router)`
- `navigate(['user'])` → ruta `/user`

Ishodi

- User postoji → Users stranica
- `null` → `alert('Bad data')`

Rute i tok posle prijave

Početna je Login; uspešan login vodi na `/user`.

src/app/app.routes.ts

```ts
export const routes: Routes = [
  { path: '', component: Login },
  { path: 'user', component: Users },
];
```

login.ts (subscribe)

```ts
if (user) {
  this.router.navigate(['user']);
} else {
  alert('Bad data');
}
```

URL-ovi

- `/` → Login
- `/user` → tabela korisnika

Tok

- POST `/users/login` → ako User → `navigate`
- Front vidi Users nakon uspešnog login-a

Dodavanje korisnika – backend

`insert` + `executeUpdate` u DAO; kontroler samo prosleđuje JSON.

UserRepo.java

```java
public int addUser(User u) {
  try (Connection conn = DB.getConnection();
       PreparedStatement stm = conn.prepareStatement(
           "insert into korisnici values (?,?,?,?)")) {
    stm.setString(1, u.getUsername());
    stm.setString(2, u.getPassword());
    stm.setString(3, u.getFirstname());
    stm.setString(4, u.getLastname());
    return stm.executeUpdate();
  } catch (SQLException e) { }
  return 0;
}
```

```java
public interface UserRepoInterface {
  public int addUser(User u);
}
```

UserController.java

```java
@PostMapping("/addUser")
public int addUser(@RequestBody User u) {
  return this.userRepo.addUser(u);
}
```

Šta vraća

- `executeUpdate` → **`int`**
- `1` = red ubačen
- `0` = nije uspelo

Dodavanje korisnika – frontend

Akcija menja stanje u servisu: posle uspeha (`num > 0`) ponovo `getAllUsers()`.

user-service.ts

```ts
addUser(user: User) {
  this.http.post<number>(
      `${this.backendUrl}/addUser`, user)
    .subscribe({
      next: num => { if (num > 0) this.getAllUsers(); },
      error: err => console.error(
          'Greška pri dodavanju korisnika', err)
    });
}
```

users.ts

```ts
addedUser = signal<User>(new User());

addUser() {
  this.userService.addUser(this.addedUser());
}
```

users.html

```html
Firstname: <input [(ngModel)]="addedUser().firstname">
Lastname:  <input [(ngModel)]="addedUser().lastname">
Username:  <input [(ngModel)]="addedUser().username">
Password:  <input [(ngModel)]="addedUser().password">
<button (click)="addUser()">Add user</button>
```

Brisanje korisnika – backend

`IN (?, ?, …)` dinamički; `executeUpdate` vraća broj obrisanih redova.

UserRepo.java

```java
public int deleteUsers(User[] users) {
  String usernames = String.join(",",
      Collections.nCopies(users.length, "?"));
  try (Connection conn = DB.getConnection();
       PreparedStatement stm = conn.prepareStatement(
           "delete from korisnici where username in ("
           + usernames + ")")) {
    for (int i = 0; i < users.length; i++)
      stm.setString(i + 1, users[i].getUsername());
    return stm.executeUpdate();
  } catch (SQLException e) { }
  return 0;
}
```

```java
public interface UserRepoInterface {
  public int deleteUsers(User[] users);
}
```

UserController.java

```java
@PostMapping("/deleteUsers")
public int deleteUsers(
    @RequestBody User[] users) {
  return
this.userRepo.deleteUsers(users);
}
```

Napomena

- Niz `?` = broj korisnika
- Collections.nCopies(users.length, "?") – kreira listu sa onoliko "?" koliko ima korisnika.
- String.join(",", ...) – spaja ih zarezima, npr. "?,?,?".
- Vraća `int` (koliko redova)

Brisanje korisnika – frontend

`deleteAction` na modelu; servis šalje niz i osvežava listu.

user-service.ts

```ts
deleteUsers(users: User[]) {
  this.http.post<number>(
      `${this.backendUrl}/deleteUsers`, users)
    .subscribe({
      next: num => { if (num > 0) this.getAllUsers(); },
      error: err => console.error(err)
    });
}
```

user.ts

```ts
export class User {
  firstname = "";
  lastname = "";
  username = "";
  password = "";
  deleteAction = false;
}
```

users.html

```html
@for (user of allUsers(); track user.username) {
  <input type="checkbox"
         [(ngModel)]="user.deleteAction">
}
<button (click)="deleteUsers()">Delete selected</button>
```

users.ts

```ts
deleteUsers() {
  const selected: User[] = [];
  for (const u of this.allUsers())
    if (u.deleteAction) selected.push(u);
  this.userService.deleteUsers(selected);
}
```

LocalDate i LocalDateTime

MySQL kolona	Java (model)	SQL tip (JDBC)
DATE	LocalDate	java.sql.Date
DATETIME	LocalDateTime	java.sql.Timestamp

Model – polja

```
private LocalDate birthDate;       // DATE
private LocalDateTime createdAt;   // DATETIME
```

Upis u bazu

DAO – upis

```
// LocalDate  →  DATE
LocalDate birth = user.getBirthDate();
java.sql.Date sqlDate = java.sql.Date.valueOf(birth);
stm.setDate(1, sqlDate);

// LocalDateTime  →  DATETIME
LocalDateTime created = user.getCreatedAt();
java.sql.Timestamp ts = java.sql.Timestamp.valueOf(created);
stm.setTimestamp(2, ts);

// kraće (isti efekat)
stm.setObject(1, birth);     // LocalDate
stm.setObject(2, created);   // LocalDateTime
```

Čitanje iz baze

DAO – čitanje

```java
// DATE  →  LocalDate
java.sql.Date d = rs.getDate("birthDate");
LocalDate birth = (d == null) ? null : d.toLocalDate();

// DATETIME  →  LocalDateTime
java.sql.Timestamp t = rs.getTimestamp("createdAt");
LocalDateTime created = (t == null) ? null : t.toLocalDateTime();

// kraće (JDBC 4.2 / Connector/J)
LocalDate birth2 = rs.getObject("birthDate", LocalDate.class);
LocalDateTime created2 = rs.getObject("createdAt", LocalDateTime.class);
```

Hvala na pažnji!

