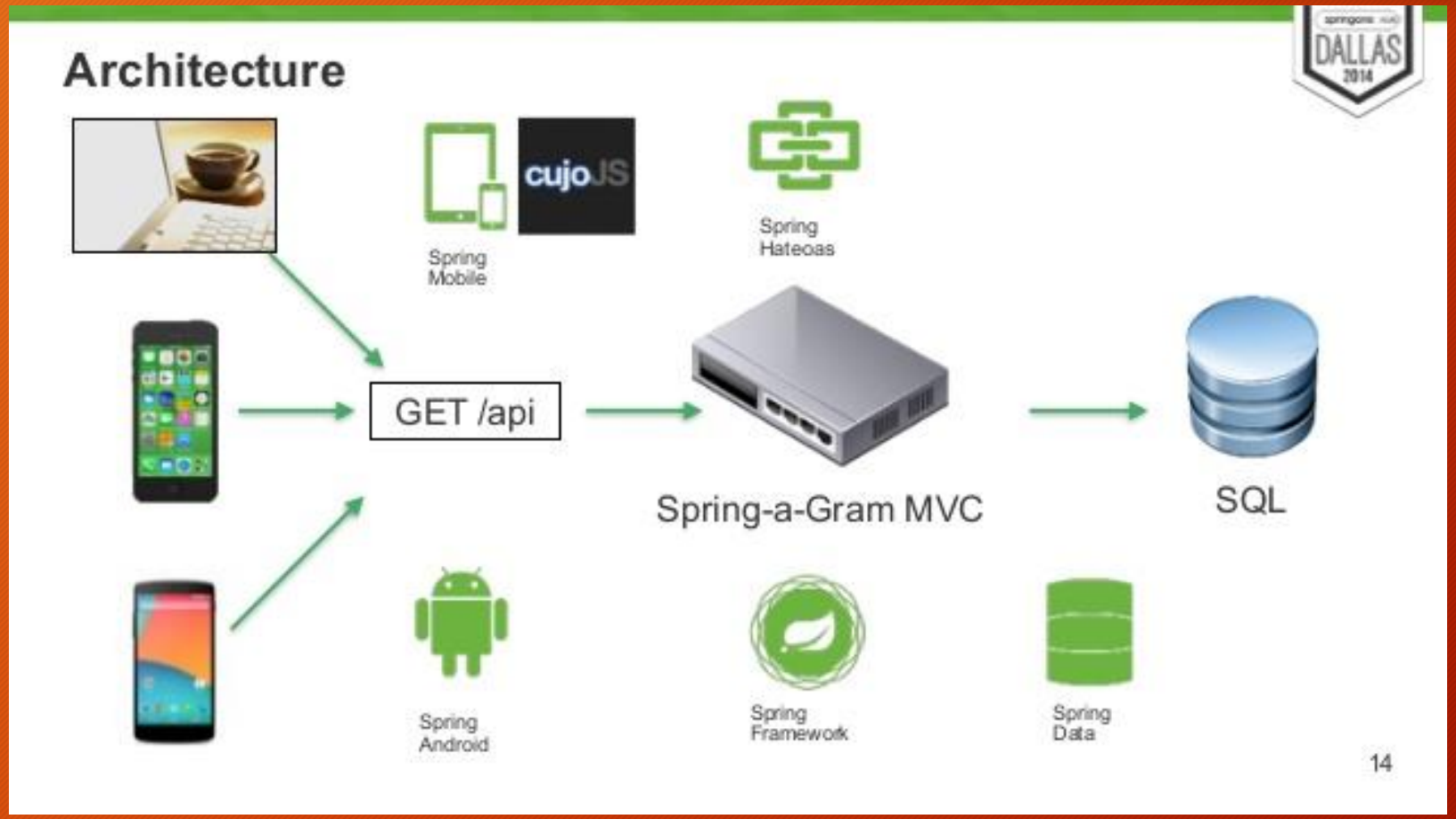
Spring MVC REST servisi

Web aplikacije u Javi

REST

- Engl. *Representational State Transfer*
- Arhitekturalni stil za dizajniranje web servisa koje mogu koristiti različiti klijenti
- Temelji se na jednostavnom HTTP protokolu
- Može se temeljiti na JSON ili XML formatu podataka
- Omogućava manipulaciju resursima temeljenu na HTTP metodama
  - POST: kreiranje resursa
  - GET: dohvaćanje resursa
  - PUT: ažuriranje resursa
  - DELETE: brisanje resursa

REST



https://image.slidesharecdn.com/springone2gx2014widedatameetshypermedia-141104091750-conversion-gate02/95/spring-data-rest-data-meets-hypermedia-14-638.jpg?cb=1415092791

Anotacija @RestController

- Spring *controller* koji se razvija u klasi s REST principima označava se s @RestController anotacijom:

```
@RestController
@RequestMapping("student")
public class StudentController {

    private final StudentService studentService;

    public StudentController(StudentService studentService) {
        this.studentService = studentService;
    }

    @GetMapping
    public List<StudentDTO> getAllStudents(){
        return studentService.findAll();
    }
```

Anotacija @RestController

```
    @GetMapping(params = "JMBAG")
    public StudentDTO getStudentByJMBAG(@RequestParam final String JMBAG){
        return studentService.findStudentByJMBAG(JMBAG);
    }

}
```

Servisni sloj

- Servisni sloj služi kao sučelje prema poslovnoj logici te repozitorijima koje aplikacija koristi
- Oblikuje se na način da se najprije definira sučelje sa svim operacijama/metodama koje je potrebno podržati:

```
public interface StudentService {

    List<StudentDTO> findAll();

    StudentDTO findStudentByJMBAG(String JMBAG);

}
```

Servisni sloj

- Nakon toga se implementiraju metode iz sučelja uz korištenje repozitorija:

```
@Service
class StudentServiceImpl implements StudentService {

    private static final int YEARS_AFTER_WHICH_TUITION_SHOULD_BE_PAYED = 26;

    private final StudentRepository studentRepository;

    public StudentServiceImpl(StudentRepository studentRepository) {
        this.studentRepository = studentRepository;
    }

    @Override
    public List<StudentDTO> findAll() {
        return studentRepository.findAll().stream().map(this::mapStudentToDTO).collect(Collectors.toList());
    }
```

Servisni sloj

```
    @Override
    public StudentDTO findStudentByJMBAG(final String JMBAG) {
        return studentRepository.findStudentByJMBAG(JMBAG).map(this::mapStudentToDTO).orElse(null);
    }

    private StudentDTO mapStudentToDTO(final Student student){
        return new StudentDTO(student.getJMBAG(), student.getNumberOfECTS(), shouldTuitionBePayed(student.getDateOfBirth()));
    }

    private boolean shouldTuitionBePayed(LocalDate dateOfBirth){
        return dateOfBirth.plusYears(YEARS_AFTER_WHICH_TUITION_SHOULD_BE_PAYED).isBefore(LocalDate.now());
    }
}
```

Sloj repozitorija

- Servisni sloj poziva sloj repozitorija predstavlja sloj aplikacije koji komunicira s bazom podataka
- Sloj repozitorija se također definira pomoću sučelja i implementacije:

```
interface StudentRepository {

    List<Student> findAll();

    Optional<Student> findStudentByJMBAG(String JMBAG);

}
```

Sloj repozitorija

- Umjesto komuniciranja s bazom podataka, radi jednostavnosti je moguće u implementaciji definirati „fiksne” podatke koje sloj repozitorija vraća:

```
@Repository
class MockStudentRepository implements StudentRepository {

    private final List<Student> MOCKED_STUDENTS = Arrays.asList(
            new Student("Ivo", "Ivić", "0256423323", LocalDate.now().minusYears(27), 120),
            new Student("Lucija", "Lucić", "0256423324", LocalDate.now().minusYears(25), 98)
    );
```

Sloj repozitorija

```
    @Override
    public List<Student> findAll() {
        return MOCKED_STUDENTS;
    }

    @Override
    public Optional<Student> findStudentByJMBAG(final String JMBAG) {
        return MOCKED_STUDENTS.stream().filter(it -> Objects.equals(it.getJMBAG(), JMBAG)).findAny();
    }
}
```

DTO objekti

- DTO (engl. *Data Transfer Object*) objekti predstavljaju strukture koji se prenose od aplikacije do preglednika i jednostavniji su od klasičnih domenskih objekata (ne sadrže ugnježđivanje podataka) te sadrže one podatke koji su potrebni za prikaz na ekranu
- Primjer DTO objekta izgleda ovako:

```
public class StudentDTO {

    private final String JMBAG;
    private final Integer numberOfECTS;
    private final boolean tuitionShouldBePaid;

    public StudentDTO(String JMBAG, Integer numberOfECTS, boolean tuitionShouldBePaid) {
        this.JMBAG = JMBAG;
        this.numberOfECTS = numberOfECTS;
        this.tuitionShouldBePaid = tuitionShouldBePaid;
    }
```

DTO objekti

```
public String getJMBAG() {
        return JMBAG;
    }

    public Integer getNumberOfECTS() {
        return numberOfECTS;
    }

    public boolean isTuitionShouldBePaid() {
        return tuitionShouldBePaid;
    }

    @Override
    public String toString() {
        return "StudentDTO{" +
                "JMBAG='" + JMBAG + '\'' +
                ", numberOfECTS=" + numberOfECTS +
                ", tuitionShouldBePaid=" + tuitionShouldBePaid +
                '}';
    }
}
```

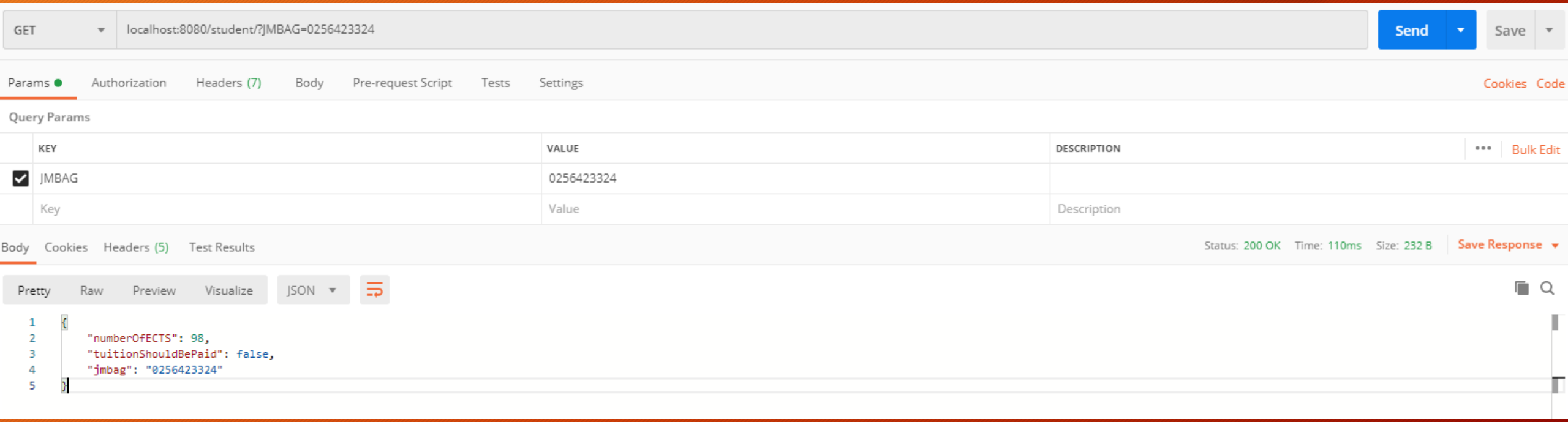
Anotacije @RequestBody i @ResponseBody

- Anotacija @RestController eliminira potrebu da se svakoj metodi unutar *controllera* postavlja anotacija @ResponseBody
- Anotacijom @ResponseBody Spring mapira povratnu vrijednost *controllera* u tijelo HTTP odgovora
- Anotacijom @RequestBody omogućava mapiranje dolazećeg zahtjeva na zadani parametar temeljenog na ACCEPT ili Content-Type zaglavlju
- Spring „u pozadini” koristi razne HTTP konvertere za pretvaranje poruke u domenske objekte metodama serijalizacije i deserijalizacije

Testiranje REST metoda

- Za testiranje REST metoda je moguće koristiti mnoge alate, a jedan od najpopularnijih je Postman
- Koristi se kao zasebna aplikacija
- Omogućava biranje URL-a, metode i definiranja parametara koji će se slati u URL zahtjevu, a vraća odgovor u definiranom formatu (npr. JSON)

Primjer korištenja Postmana - GET



Konfiguracija pom.xml

- Za konfiguriranje osnovne Spring web aplikacije potrebne su sljedeće ovisnosti u pom.xml datoteci:

```
<properties>
    <java.version>11</java.version>
</properties>

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

    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-test</artifactId>
        <scope>test</scope>
        <exclusions>
            <exclusion>
                <groupId>org.junit.vintage</groupId>
                <artifactId>junit-vintage-engine</artifactId>
            </exclusion>
        </exclusions>
    </dependency>
</dependencies>
```

Pitanja?