Универзитет у Београду
Факултет организационих наука
Лабораторија за електронско пословање

**Интернет технологије**

# Семинарски рад

Студентски сервис са
друштвеним елементима

Студент:
Иван Рашић 197/09 И

Ментор:
доцент др Душан Бараћ

# Садржај


1 Опис корисничких захтева ..... 4

2 Случајеви коришћења ..... 4

2.1 СК 1: Додавање студента ..... 5

2.2 СК 2: Пријава корисника на систем ..... 5

2.3 СК 3: Пријава преко Facebook-а ..... 5

2.4 СК 4: Пријава испита за испитни рок ..... 5

2.5 СК 5: Измена пријаве (оцењивање) ..... 6

2.6 СК 6: Објава догађаја на Facebook-у ..... 6

3 Анализа ..... 7

3.1 Понашање софтверског система – системски дијаграми секвенци ..... 7

3.1.1 ДС 5: Измена пријаве (оцењивање) ..... 7

3.1.2 ДС 6: Објава догађаја на Facebook-у ..... 7

4 Архитектура система ..... 9

4.1 Опис ..... 9

4.2 Дијаграм доменског модела ..... 10

5 Пројектовање базе података ..... 11

6 Имплементација ..... 15

6.1 Приказ кода – сервер ..... 15

6.1.1 JPA/Hibernate ..... 15

6.1.2 Spring – „ресурс“ класе ..... 16

6.1.3 Spring – класе пословне логике ..... 18

6.1.4 Spring – „репозиторијум“ интерфејси ..... 20

6.1.5 Liquibase – скрипте за ажурирање базе ..... 20

6.2 Приказ кода – клијент ..... 22

6.2.1 Компоненте ..... 22

6.2.2 Сервиси ..... 24

6.2.3 Модули ..... 25

6.2.4 Руте ..... 26

7 Корисничко упутство ..... 27

7.1 Пријава корисника на систем ..... 27

7.2 Пријава преко Facebook-а ..... 28

7.3 Пријава испита за испитни рок ..... 30

7.4 Измена пријаве (оцењивање) ..... 31

7.5 Објава догађаја на Facebook-у ..... 33

# 1 Опис корисничких захтева

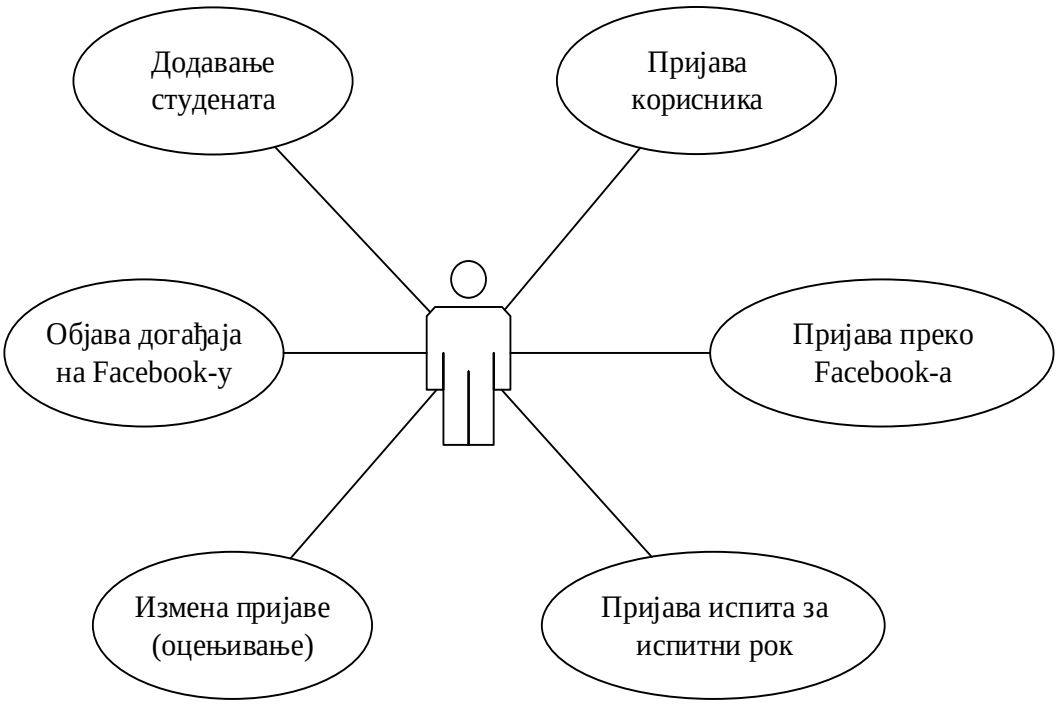
Потребно је направити софтверски систем за потребе факултета, који обезбеђује базу података са подацима о свим елементима наставног процеса, као и подацима о студентима. Приступ систему треба да имају наставници и административно особље ради вођења евиденције о студентима, предметима, испитима, испитним роковима, оценама и другим елементима наставног процеса. Приступ такође треба да имају и студенти ради пријава испита, увида у своја постигнућа.

Специјалан захтев за овај софтверски систем је и повезивање са друштвеним мрежама (Facebook), ради могућности да студенти директно из система могу да креирају објаве везане за одређене промене у систему везане за њих (положен испит, пао испит...)

# 2 Случајеви коришћења

Идентификовани су следећи случајеви коришћења:

1. Додавање студената
2. Пријава корисника
3. Пријава преко Facebook-а
4. Пријава испита за испитни рок
5. Измена пријаве (оцењивање)
6. Објава догађаја на Facebook-у



## *2.1 СК 1: Додавање студента*

**Актери СК:** **Запослени**

**Учесници СК:** **Систем** и **запослени**

**Предуслови:** **Запослени** је пријављен на систем.
Учитана је форма за унос података о студенту.

**Сценарио СК:**

1. **Запослени** уноси податке о студенту.
2. **Запослени** позива **систем** да сачува студента.
3. **Систем** снима новог студента у базу података.
4. **Систем** враћа снимљеног студента.

## *2.2 СК 2: Пријава корисника на систем*

**Актери СК:** **Корисник**

**Учесници СК:** **Систем** и **корисник**

**Предуслов:** **Корисник** је уписан у систему.
Учитана је форма за пријаву на систем.

**Сценарио СК:**

1. **Корниск** уноси своје корисничко име и шифру.
2. **Корисник** позива **систем** да га улогује и започне сесију.
3. **Систем** проверава унесене податке.
4. **Систем** приказује почетну форму за избор области апликације.

## *2.3 СК 3: Пријава преко Facebook-а*

**Актери СК:** **Корисник**

**Учесници СК:** **Корисник**, **систем**, **Facebook сервис**

**Предуслов:** **Корисник** је уписан у **систему**.
**Корисник** има Facebook налог повезан са налогом у систему.
Учитана је форма за пријаву на систем.

**Сценарио СК:**

1. **Корисник** позива **систем** да започне пријаву путем Facebook-а.
2. **Систем** позива **Facebook сервис** да аутентификује корисника и врати податке.
3. **Facebook сервис** аутентификује **корисника** и враћа **систему** податке.
4. **Систем** проверава добијене податке.
5. **Систем** приказује почетну форму за избор области апликације.

## *2.4 СК 4: Пријава испита за испитни рок*

**Актери СК:** **Студент**

**Учесници СК:** **Студент** и **систем**

**Предуслов:** **Студент** је пријављен на **систем**.
Учитана је форма за пријаву испита.

**Сценарио СК:**

1. **Студент** бира предмет који жели да пријави за следећи испитни рок.
2. **Студент** позива **систем** да забележи пријаву.
3. **Систем** бележи пријаву испита за изабрани предмет.
4. **Систем** враћа листу пријављених испита и предмета могућих за пријаву.

## *2.5 СК 5: Измена пријаве (оцењивање)*

**Актери СК:** **Наставник**

**Учесници СК:** **Наставник** и **систем**

**Предуслов:** **Наставник** је пријављен на **систем**.
Учитана је форма за управљање испитних пријава.

**Сценарио СК:**

1. **Наставник** бира пријаву испита коју жели да измени.
2. **Наставник** позива **систем** да врати податке за изабрану пријаву.
3. **Систем** добавља изабрану пријаву.
4. **Систем** враћа изабрану пријаву.
5. **Наставник** уноси нове податке.
6. **Наставник** позива **систем** да сачува унесене податке и креира предлоге објава.
7. **Систем** снима нове податке и по потреби креира предлоге објава.
8. **Систем** враћа листу пријава са унесеним изменама.

## *2.6 СК 6: Објава догађаја на Facebook-у*

**Актери СК:** **Студент**

**Учесници СК:** **Студент**, **систем, Facebook сервис**

**Предуслов:** **Студент** је пријављен на **систем**.
Учитана је форма са предлозима објава.

**Сценарио СК:**

1. **Студент** бира предлог објаве који жели да реализује.
2. **Студент** позива **систем** да састави предлог текста објаве.
3. **Систем** саставља предлог текста објаве.
4. **Систем** враћа предлог текста објаве.
5. **Студент** уноси измене текста по потреби.
6. **Студент** позива **систем** да започне процес објаве.
7. **Систем** позива **Facebook сервис** да започене процес објаве.
8. **Facebook сервис** аутентификује корисника и приказује форму за објаву са унесеним тексстом.
9. **Студент** позива **Facebook сервис** да изврши објаву.
10. **Facebook сервис** извршава објаву.

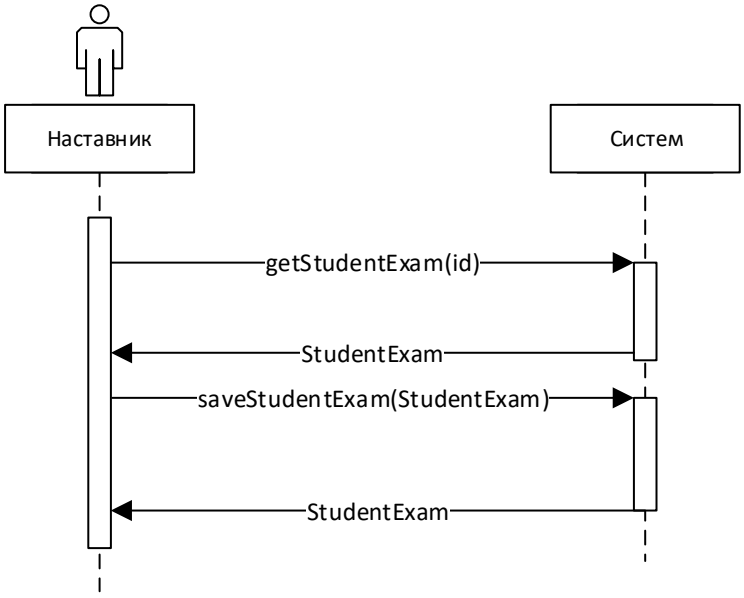
# 3 Анализа

## *3.1 Понашање софтверског система – системски дијаграми секвенци*

### 3.1.1 ДС 5: Измена пријаве (оцењивање)

**Сценарио СК:**

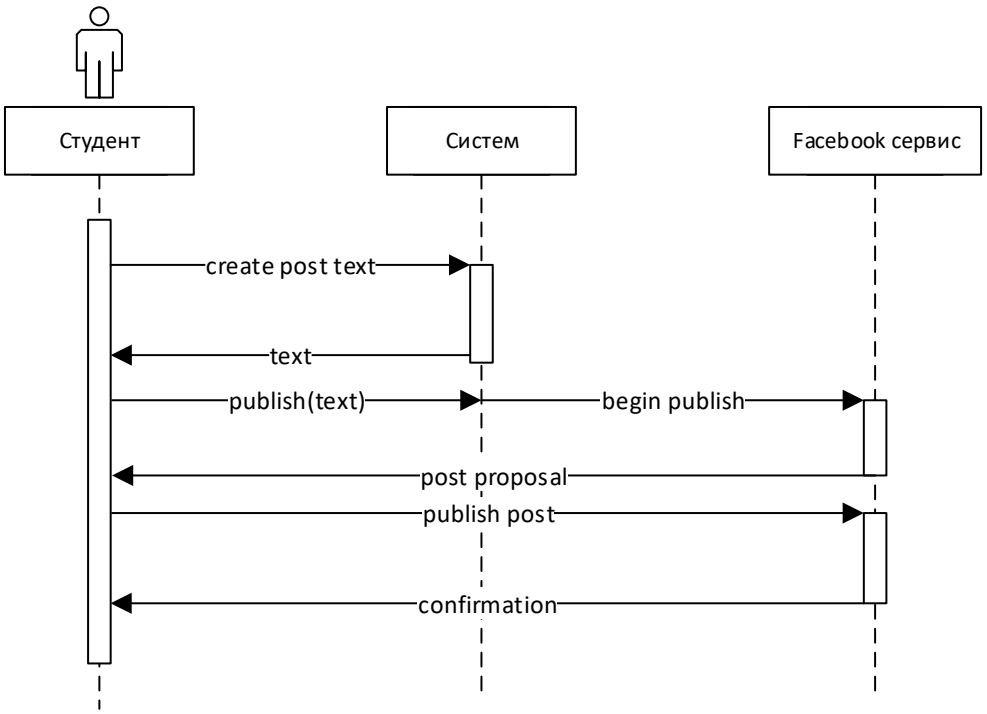
1. **Наставник** позива **систем** да врати податке за изабрану пријаву.
2. **Систем** враћа изабрану пријаву.
3. **Наставник** позива **систем** да сачува унесене податке и креира предлоге објава.
4. **Систем** враћа листу пријава са унесеним изменама.




### 3.1.2 ДС 6: Објава догађаја на Facebook-у

**Сценарио СК:**

5. **Студент** позива **систем** да састави предлог текста објаве.
6. **Систем** враћа предлог текста објаве.
7. **Студент** позива **систем** да започне процес објаве.
8. **Систем** позива **Facebook сервис** да започене процес објаве.
9. **Facebook сервис** аутентификује корисника и приказује форму за објаву са унесеним тексстом.
10. **Студент** позива **Facebook сервис** да изврши објаву.
11. **Facebook сервис** извршава објаву и потврђује.


Студент
Систем
Facebook сервис
create post text
text
publish(text)
begin publish
post proposal
publish post
confirmation

# 4 Архитектура система

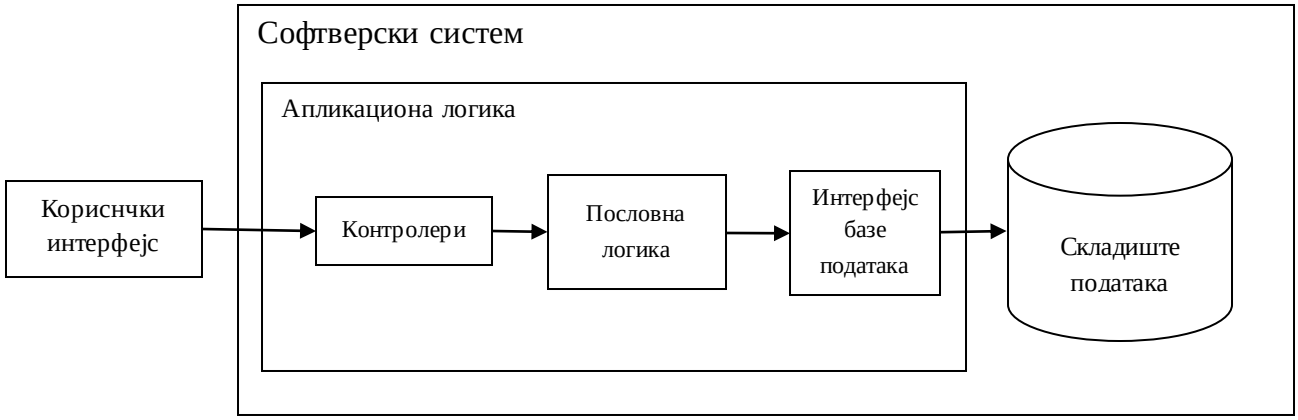
## *4.1 Опис*

Коришћена је класична тронивојска архитектура.

Тронивојска архитектура се састоји из:

1. Корисничког интерфејса који предствађа реализацију улаза и излаза софтверског система.
2. Апликационе логике која описује структуру и понашање софтверског система.
3. Складишта података које чува стање атрибута софтверског система.

Ниво корисничког интерфејса је на страни клијента, а апликациона логика и складиште подата на страни сервера.



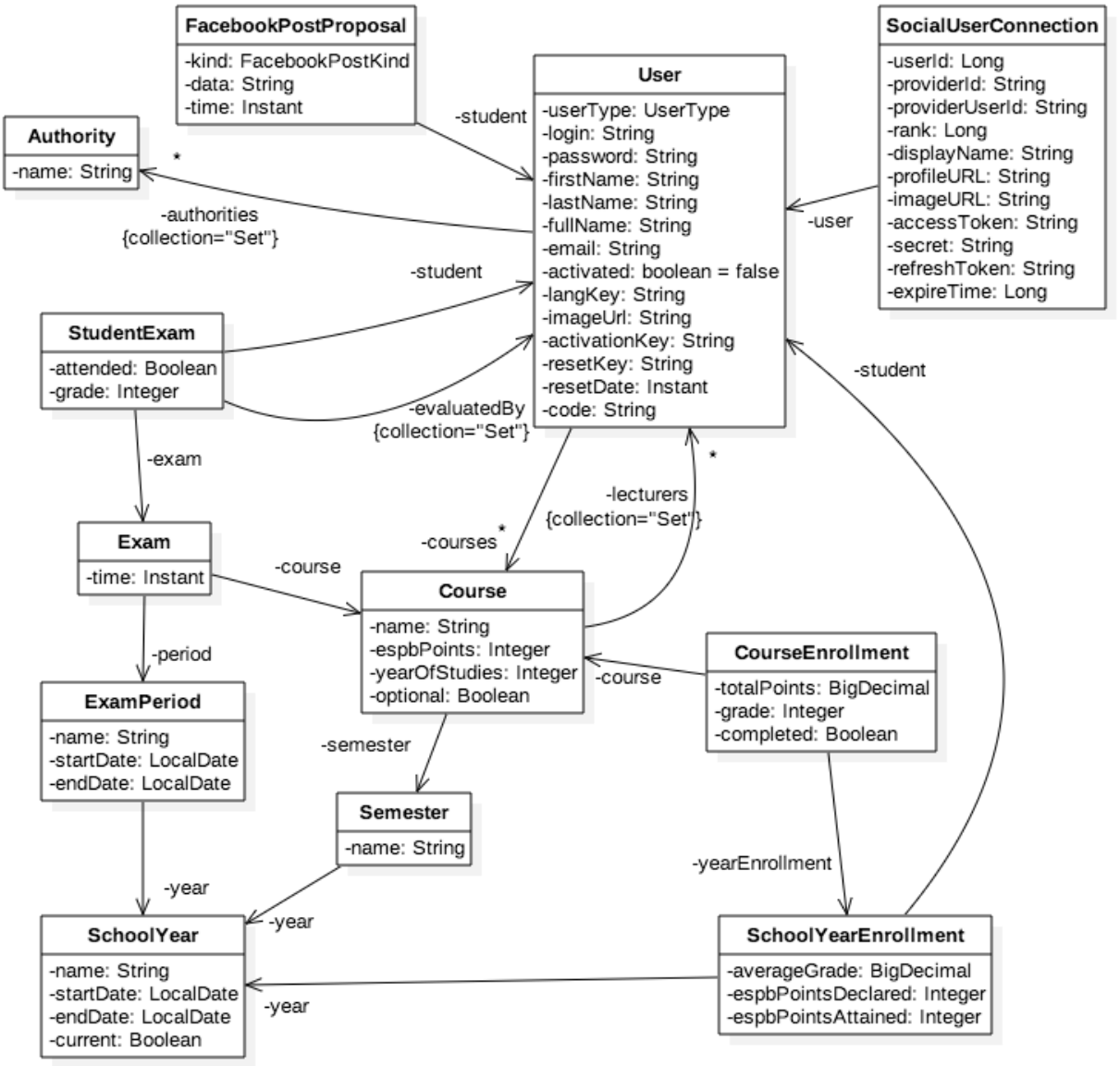

Кориснички интерфејс се извршава у web browser-у на клијентској страни у виду веб апликације која са сервером комуницира искључиво путем HTTP захтева који шаљу и примају JSON податке.

На серверској страни се налази апликациона логика која је подељена у три нивоа.

1. Први (контролерски) ниво комуницира са корисничким интерфејсом и претвара поруке из текстуалног облика у облик објеката и обратно. Контролерски ниво позива ниво пословне логике који извршава функције пословне логике.
2. Ниво пословне логике комуницира извршава функције пословне логике, са базом података преко слоја интерфејса базе.
3. Слој интерфејса базе извршава упите и команде које су потребне слоју пословне логике.

Сваки од ових слојева се састоји од карактеристичних класа и интерфејса.

## *4.2 Дијаграм доменског модела*



# 5 Пројектовање базе података

На основу софтверских класа структуре пројектоване су табеле релационог система за управљање базом података. У овом раду коришћен је MySQL релациони систем за управљање базом података.

| Tabela: user | |
|---|---|
| Naziv kolone | Tip |
| id | bigint(20) |
| user_type | varchar(5) |
| login | varchar(50) |
| password | varchar(50) |
| first_name | varchar(50) |
| last_name | varchar(50) |
| full_name | varchar(110) |
| email | varchar(50) |
| activated | boolean |
| lang_key | varchar(10) |
| image_url | varchar(255) |
| activation_key | varchar(100) |
| reset_key | varchar(100) |
| reset_date | datetime |
| code | varchar(20) |
| Indeksi | |
| Naziv indeksa | Kolone |
| PRIMARY | id |

| Tabela: social_user_connection | |
|---|---|
| Naziv kolone | Tip |
| id | bigint(20) |
| user_id | bigint(20) |
| provider_id | varchar(20) |
| provider_user_id | varchar(100) |
| rank | int |
| display_name | varchar(255) |
| profile_url | varchar(255) |
| image_url | varchar(255) |
| access_token | varchar(100) |
| secret | varchar(255) |
| refresh_token | varchar(100) |
| expire_time | datetime |
| Indeksi | |
| Naziv indeksa | Kolone |
| PRIMARY | id |
| connection_user | user_id |

| Tabela: facebook_post_proposal | |
|---|---|
| Naziv kolone | Tip |
| id | bigint(20) |
| kind | varchar(5) |
| data | varchar(255) |
| time | datetime |
| student_id | bigint(20) |
| Indeksi | |
| Naziv indeksa | Kolone |
| PRIMARY | id |
| post_proposal_student | student_id |

| Tabela: student_exam | |
|---|---|
| Naziv kolone | Tip |
| id | bigint(20) |
| attended | boolean |
| grade | int |
| student_id | bigint(20) |
| exam_id | bigint(20) |
| evaluated_by_id | bigint(20) |
| Indeksi | |
| Naziv indeksa | Kolone |
| PRIMARY | id |
| student_exam_student | student_id |
| student_exam_exam | exam_id |
| student_ex_avaluated | evaluated_by_id |

| Tabela: exam | |
|---|---|
| Naziv kolone | Tip |
| id | bigint(20) |
| time | datetime |
| course_id | bigint(20) |
| period_id | bigint(20) |
| Indeksi | |
| Naziv indeksa | Kolone |
| PRIMARY | id |
| exam_course | course_id |
| exam_period | period_id |

| Tabela: course | |
|---|---|
| Naziv kolone | Tip |
| id | bigint(20) |
| name | varchar(100) |
| espb_points | int |
| year_of_studies | int |

| | |
|---|---|
| semester_id | bigint(20) |
| Indeksi | |
| Naziv indeksa | Kolone |
| PRIMARY | id |
| course_semester | semester_id |

| Tabela: exam_period | |
|---|---|
| Naziv kolone | Tip |
| id | bigint(20) |
| start_date | datetime |
| end_date | datetime |
| year_id | bigint(20) |
| Indeksi | |
| Naziv indeksa | Kolone |
| PRIMARY | id |
| exam_period_year | year_id |

| Tabela: school_year | |
|---|---|
| Naziv kolone | Tip |
| id | bigint(20) |
| name | varchar(20) |
| start_date | datetime |
| end_date | datetime |
| current | boolean |
| Indeksi | |
| Naziv indeksa | Kolone |
| PRIMARY | id |

| Tabela: semester | |
|---|---|
| Naziv kolone | Tip |
| id | bigint(20) |
| name | varchar(50) |
| year_id | bigint(20) |
| Indeksi | |
| Naziv indeksa | Kolone |
| PRIMARY | id |
| semester_year | year_id |

| Tabela: course_enrollment | |
|---|---|
| Naziv kolone | Tip |
| id | bigint(20) |
| total_points | decimal |
| grade | int |
| completed | boolean |

| | |
|---|---|
| year_enrollment_id | bigint(20) |
| Indeksi | |
| Naziv indeksa | Kolone |
| PRIMARY | id |
| course_enrollment_year | year_enrollment_id |

| Tabela: school_year_enrollment | |
|---|---|
| Naziv kolone | Tip |
| id | bigint(20) |
| average_grade | decimal |
| espb_points_declared | int |
| espb_points_attained | int |
| student_id | bigint(20) |
| year_id | bigint(20) |
| Indeksi | |
| Naziv indeksa | Kolone |
| PRIMARY | id |
| year_enrollment_student | student_id |
| year_enrollment_year | year_id |

# 6 Имплементација

Софтверски систем је развијен коришћењем бројних технологија.

Серверска страна:

- Програмски језик Јава
- JPA/Hibernate – ORM библиотека
- Spring библиотеке – обезбеђују лаку комуникацију са базом, трансакције, лако примање и одговарање на HTTP захтеве, имплементирају протоколе за комуникацију са друштвеним мрежама
- Swagger – библиотека за подршку документовању REST API-ја (са plugin-ом који документује API-је имплементиране путем Spring-а и алатом за генерисање документације свих REST API-ја апликације у виду једног HTML фајла)
- Liquibase – библиотека за аутоматско ажурирање структуре базе података приликом стартовања апликације.

Клијентска страна:

- Програмски језик TypeScript – надградња над JavaScript-ом која доноси објектно оријентисано програмирање засновано на класама и декларисање типова података у коду.
- Angular 5 – свеобухватан framework за писање клијентске стране модерних веб апликација које се заснивају на комуникацији са сервером искључиво преко AJAX-а.
- Bootstrap – CSS framework за добар изглед сајта.
- Webpack – обезбеђује сажимање свих TypeScript фајлова, као и свих HTML фајлова у пројекту у један JS фајл који се само једном скида са сервера кад се апликација први пут отвори у једном табу браузера.

## *6.1 Приказ кода – сервер*

### 6.1.1 JPA/Hibernate

Класе доменског модела се везују за релационе табеле преко JPA анотација. Пример доменске класе за предлог објаве за Facebook (гетер и сетер методе изостављене).

```
@Entity
@Table(name = "facebook_post_proposal")
public class FacebookPostProposal extends AbstractEntity {

    @NotNull
    @Enumerated(EnumType.STRING)
    @Column(name = "kind", nullable = false)
    private FacebookPostKind kind;

    @NotNull
    @Column(name = "data", nullable = false)
    private String data;

    @NotNull
    @Column(name = "time", nullable = false)
    private Instant time;

    @ManyToOne(optional = false)
    @NotNull
    private User student;
```

```
}


@MappedSuperclass
public abstract class AbstractEntity implements Serializable {

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

    public final Long getId() {
        return id;
    }

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

@Entity анотација означава да је ово JPA класа. Свако поље се мапира у једну колону у табели. Поља чији тип је нека друга доменска класа или колекција истих, представљају колоне са спољним кључевима у бази и она се означавају анотацијама @ManyToOne, @OneToMany или @ManyToMany.

Све доменске класе у пројекту наслеђују класу AbstractEntity која дефинише примарни кључ.

### 6.1.2 Spring – „ресурс“ класе

Ово су класе које престављају улаз REST API-ја апликације који је једини интерфејс ка спољном свету. Пример, ресурс класа за URL-ове који се тичу пријаве испита (изостављени су коментари).

```
@RestController
@RequestMapping("/api")
public class StudentExamResource {

    private final Logger log = LoggerFactory.getLogger(StudentExamResource.class);

    private static final String ENTITY_NAME = "studentExam";

    private final StudentExamService studentExamService;
    private final StudentExamQueryService studentExamQueryService;

    public StudentExamResource(StudentExamService studentExamService, StudentExamQueryService
studentExamQueryService) {
        this.studentExamService = studentExamService;
        this.studentExamQueryService = studentExamQueryService;
    }

    @PutMapping("/student-exams")
    @Timed
    @ApiOperation(
        value = "Update exam application (grade)",
        notes = "Updates the exam application. If the grade has changed it " +
            "will initiate Facebook post proposal creation."
    )
    public ResponseEntity<StudentExamDTO> updateStudentExam(@Valid @RequestBody StudentExamDTO
studentExamDTO) throws URISyntaxException {
        log.debug("REST request to update StudentExam : {}", studentExamDTO);
        if (studentExamDTO.getId() == null) {
            return createStudentExam(studentExamDTO);
        }
        StudentExamDTO result = studentExamService.save(studentExamDTO);
        return ResponseEntity.ok()
            .headers(HeaderUtil.createEntityUpdateAlert(ENTITY_NAME,
studentExamDTO.getId().toString()))
```

```
            .body(result);
    }

    @GetMapping("/student-exams")
    @Timed
    public ResponseEntity<List<StudentExamDTO>> getAllStudentExams(StudentExamCriteria criteria,
Pageable pageable) {
        log.debug("REST request to get StudentExams by criteria: {}", criteria);
        Page<StudentExamDTO> page = studentExamQueryService.findByCriteria(criteria, pageable);
        HttpHeaders headers = PaginationUtil.generatePaginationHttpHeaders(page, "/api/student-
exams");
        return new ResponseEntity<>(page.getContent(), headers, HttpStatus.OK);
    }

    @GetMapping("/student-exams/{id}")
    @Timed
    public ResponseEntity<StudentExamDTO> getStudentExam(@PathVariable Long id) {
        log.debug("REST request to get StudentExam : {}", id);
        StudentExamDTO studentExamDTO = studentExamService.findOne(id);
        return ResponseUtil.wrapOrNotFound(Optional.ofNullable(studentExamDTO));
    }

    @DeleteMapping("/student-exams/{id}")
    @Timed
    public ResponseEntity<Void> deleteStudentExam(@PathVariable Long id) {
        log.debug("REST request to delete StudentExam : {}", id);
        studentExamService.delete(id);
        return ResponseEntity.ok().headers(HeaderUtil.createEntityDeletionAlert(ENTITY_NAME,
id.toString())).build();
    }

    @GetMapping("/exam-application-data")
    @ApiOperation(
        value = "Return data for exam applications page",
        notes = "Returns all data for the exam application page for the logged in student, " +
        "which is all exams enrolled, not passed, not applied for and exams applied for the next
period."
    )
    public ResponseEntity<ExamApplicationPageData> examApplicationPageData() {
        ExamApplicationPageData data = studentExamService.getApplicationPageData();
        return new ResponseEntity<>(data, HttpStatus.OK);
    }

    @PutMapping("/apply-for-exam")
    @ApiOperation(
        value = "Apply for exam",
        notes = "Creates an exam application for the next period identified by examId. Returns the
page data again"
    )
    public ResponseEntity<ExamApplicationPageData> applyForExam(@RequestParam(name = "exam") Long
examId) {
        studentExamService.applyForExam(examId);
        return examApplicationPageData();
    }

    @PutMapping("/cancel-exam-application")
    @ApiOperation(
        value = "Cancel exam application",
        notes = "Cancels the exam application identified by examId. Returns the page data again."
    )
    public ResponseEntity<ExamApplicationPageData> cancelExamApplication(@RequestParam(name =
"exam") Long examId) {
        studentExamService.cancelExamApplication(examId);
        return examApplicationPageData();
    }
}
```

Свака класа REST API-ја је анотирана са @RestController и @RequestMapping ради декларације почетка путање. Свака метода је анотирана неком од анотација @PostMapping, @GetMapping, @PutMapping, @DeleteMapping у зависности од HTTP методе коју прихвата.

Методе су такође анотиране и анотацијама @ApiOperation. Оне служе за документвање. Када се документација REST API-ја генерише коришћењем Swagger-а текст из тих анотација ће бити тамо исписан.

### 6.1.3 Spring – класе пословне логике

Ове класе се у Spring терминологији називају сервисне класе. Анотиране су са @Service. Анотација @Transactional говори Spring-у да на почетку извршења сваке методе треба да отвори трансакцију са базом података и да је затвори на крају. Пример, сервисна класа са логиком око пријава испита.

```
@Service
@Transactional
public class StudentExamService {

    private final Logger log = LoggerFactory.getLogger(StudentExamService.class);

    private final StudentExamRepository studentExamRepository;
    private final StudentExamMapper studentExamMapper;
    private final StudentExamSearchRepository studentExamSearchRepository;
    private final ExamPeriodRepository examPeriodRepository;
    private final ExamPeriodMapper examPeriodMapper;
    private final ExamMapper examMapper;
    private final ExamRepository examRepository;
    private final UserRepository userRepository;
    private final FacebookPostProposalService facebookPostProposalService;

    public StudentExamService(StudentExamRepository studentExamRepository,
                              StudentExamMapper studentExamMapper,
                              StudentExamSearchRepository studentExamSearchRepository,
                              ExamPeriodRepository examPeriodRepository,
                              ExamPeriodMapper examPeriodMapper,
                              ExamMapper examMapper,
                              ExamRepository examRepository,
                              UserRepository userRepository,
                              FacebookPostProposalService facebookPostProposalService) {
        this.studentExamRepository = studentExamRepository;
        this.studentExamMapper = studentExamMapper;
        this.studentExamSearchRepository = studentExamSearchRepository;
        this.examPeriodRepository = examPeriodRepository;
        this.examPeriodMapper = examPeriodMapper;
        this.examMapper = examMapper;
        this.examRepository = examRepository;
        this.userRepository = userRepository;
        this.facebookPostProposalService = facebookPostProposalService;
    }

    public StudentExamDTO save(StudentExamDTO studentExamDTO) {
        log.debug("Request to save StudentExam : {}", studentExamDTO);
        return studentExamDTO.getId() == null
            ? create(studentExamDTO)
            : update(studentExamDTO);
    }

    private StudentExamDTO create(StudentExamDTO studentExamDTO) {
        StudentExam studentExam = studentExamMapper.toStudentExam(studentExamDTO);
        studentExam = studentExamRepository.save(studentExam);
        StudentExamDTO result = studentExamMapper.toDto(studentExam);
        studentExamSearchRepository.save(studentExam);
        return result;
    }

    private StudentExamDTO update(StudentExamDTO studentExamDTO) {
```

```
        StudentExam studentExam = studentExamRepository.findOne(studentExamDTO.getId());

        boolean shouldPropose = studentExam.getEvaluatedBy() == null &&
studentExamDTO.getEvaluatedById() != null;
        studentExamMapper.toStudentExam(studentExamDTO, studentExam);
        if (shouldPropose) {
            FacebookPostKind postKind = studentExamDTO.getGrade() > 5
                ? FacebookPostKind.EXAM_PASSED
                : FacebookPostKind.EXAM_FAILED;
            facebookPostProposalService.saveExamProposal(studentExam, postKind);
        }

        studentExam = studentExamRepository.save(studentExam);
        StudentExamDTO result = studentExamMapper.toDto(studentExam);
        studentExamSearchRepository.save(studentExam);
        return result;
    }

    @Transactional(readOnly = true)
    public Page<StudentExamDTO> findAll(Pageable pageable) {
        log.debug("Request to get all StudentExams");
        return studentExamRepository.findAll(pageable)
            .map(studentExamMapper::toDto);
    }

    @Transactional(readOnly = true)
    public StudentExamDTO findOne(Long id) {
        log.debug("Request to get StudentExam : {}", id);
        StudentExam studentExam = studentExamRepository.findOne(id);
        return studentExamMapper.toDto(studentExam);
    }

    public void delete(Long id) {
        log.debug("Request to delete StudentExam : {}", id);
        studentExamRepository.delete(id);
        studentExamSearchRepository.delete(id);
    }

    @Transactional(readOnly = true)
    public Page<StudentExamDTO> search(String query, Pageable pageable) {
        log.debug("Request to search for a page of StudentExams for query {}", query);
        Page<StudentExam> result = studentExamSearchRepository.search(queryStringQuery(query),
pageable);
        return result.map(studentExamMapper::toDto);
    }

    @Transactional(readOnly = true)
    public ExamApplicationPageData getApplicationPageData() {
        Long studentId = SecurityUtils.getCurrentUserId();

        Page<ExamPeriod> page = examPeriodRepository.findNextPeriod(LocalDate.now(), new
PageRequest(0, 1));
        ExamPeriodDTO nextPeriodDTO = examPeriodMapper.toDto(page.getContent().get(0));

        List<Exam> exams = studentExamRepository.findNextExamsForStudent(studentId,
nextPeriodDTO.getId());
        List<ExamDTO> examDTOS = examMapper.toDto(exams);

        List<Exam> applied = studentExamRepository.findAppliedExams(studentId,
nextPeriodDTO.getId());
        List<ExamDTO> appliedDTOs = examMapper.toDto(applied);

        return new ExamApplicationPageData(nextPeriodDTO, examDTOS, appliedDTOs);
    }

    public void applyForExam(Long examId) {
        Long studentId = SecurityUtils.getCurrentUserId();

        Exam exam = examRepository.findOne(examId);
```

```
        StudentExam studentExam = new StudentExam()
            .exam(exam)
            .student(userRepository.findOne(studentId));

        studentExamRepository.save(studentExam);
    }

    public void cancelExamApplication(Long examId) {
        Long studentId = SecurityUtils.getCurrentUserId();
        studentExamRepository.deleteByExam_IdAndStudent_Id(examId, studentId);
    }
}
```

### 6.1.4 Spring – „репозиторијум“ интерфејси

Сви упити који су апликацији потребани у комуникацији са базом су дефинисани као методе у овим интерфејсима. Они немају имплементацију у изворном коду, већ Spring за сваки интерфејс генерише имплементациону класу (када се сервер покрене) која имплементира упите на основу имена метода или упита писаним JPA језиком у анотацијама. Пример, репозиторијум са упитима везаним за пријаве испита.

```
@Repository
public interface StudentExamRepository extends JpaRepository<StudentExam, Long>,
JpaSpecificationExecutor<StudentExam> {

    @Query("select e from Exam e \n" +
        "join CourseEnrollment ce on (ce.course = e.course)" +
        "join ce.yearEnrollment ye \n" +
        "where ye.student.id = :studentId and e.period.id = :periodId \n" +
        "and e.id not in (select e.id \n" +
        "                 from StudentExam se join se.exam e \n" +
        "                 where se.student.id = :studentId and e.period.id = :periodId)")
    List<Exam> findNextExamsForStudent(@Param("studentId") Long studentId, @Param("periodId") Long
periodId);

    @Query("select e \n" +
        "from StudentExam se join se.exam e \n" +
        "where se.student.id = ?1 and e.period.id = ?2")
    List<Exam> findAppliedExams(Long studentId, Long periodId);

    void deleteByExam_IdAndStudent_Id(long examId, Long studentId);
}
```

За прве две методе упит је уписан у анотацију, док за трећу методу Spring сам закључује какв упит желимо из имена методе које задовољава одређену синтаксу.

### 6.1.5 Liquibase – скрипте за ажурирање базе

Ове скрипте су XML фајлови у којима се промене на бази дефинишу синтаксом независном од произвођача базе. Пример је креирање табеле за предлоге објава.

```
<?xml version="1.0" encoding="utf-8"?>
<databaseChangeLog
    xmlns="http://www.liquibase.org/xml/ns/dbchangelog"
    xmlns:ext="http://www.liquibase.org/xml/ns/dbchangelog-ext"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.liquibase.org/xml/ns/dbchangelog
http://www.liquibase.org/xml/ns/dbchangelog/dbchangelog-3.5.xsd
                        http://www.liquibase.org/xml/ns/dbchangelog-ext
http://www.liquibase.org/xml/ns/dbchangelog/dbchangelog-ext.xsd">

    <property name="now" value="now()" dbms="h2"/>

    <property name="now" value="now()" dbms="mysql"/>
    <property name="autoIncrement" value="true"/>

    <property name="floatType" value="float4" dbms="postgresql, h2"/>
```

```
    <property name="floatType" value="float" dbms="mysql, oracle, mssql"/>

    <!--
        Added the entity FacebookPostProposal.
    -->
    <changeSet id="20180202195353-1" author="ivan">
        <createTable tableName="facebook_post_proposal">
            <column name="id" type="bigint" autoIncrement="${autoIncrement}">
                <constraints primaryKey="true" nullable="false"/>
            </column>
            <column name="kind" type="varchar(255)">
                <constraints nullable="false" />
            </column>

            <column name="data" type="varchar(255)">
                <constraints nullable="false" />
            </column>

            <column name="time" type="timestamp">
                <constraints nullable="false" />
            </column>

            <column name="student_id" type="bigint">
                <constraints nullable="false" />
            </column>

        </createTable>
        <dropDefaultValue tableName="facebook_post_proposal" columnName="time"
columnDataType="datetime"/>

    </changeSet>
</databaseChangeLog>
```

## 6.2 *Приказ кода – клијент*

Angular апликације се састоје од више елемената:

- Компоненте – класе које представљају елементе корисничког интерфејса које у себи садрже одређену логику као и HTML код.
- Сервиси – класе које садрже било какав код који се користи из разних делова апликације. Ове класе се позивају из компоненти и других сервиса.
- Модули – класе које служе за дефинисање одређених целина пликације. Један модул садржи своје компоненте, сервисе и руте.
- Руте – спецификације стаза које се приказују у адресној траци браузера. Ту се за сваку руту дефинише улазна компонента и параметери.

### 6.2.1 Компоненте

Пример логике компоненте у TypeScript-у (дијалог за измену предложеног текста објаве):

```
@Component({
    selector: 'app-publish-facebook-post',
    templateUrl: './publish-facebook-post.component.html'
})
export class PublishFacebookPostComponent implements OnInit, OnDestroy {

    @ViewChild('template')
    templateRef: TemplateRef<any>;

    private modalRef: NgbModalRef;
    private routeSub: Subscription;

    private id: number;

    text: string;

    constructor(private modal: NgbModal,
                private router: Router,
                private route: ActivatedRoute,
                private http: Http,
                private eventManager: JhiEventManager,
                private translate: TranslateService,
                private facebookPostProposalService: FacebookPostProposalService,
                private facebook: FacebookService) {
    }

    ngOnInit() {
        this.facebook.init({
            appId: '146176679417761',
            xfbml: true,
            version: 'v2.12'
        });

        this.routeSub = this.route.params.subscribe((params) => {
            this.id = params['id'];
            this.activate();
        });
    }

    ngOnDestroy() {
        this.routeSub.unsubscribe();
    }

    private async activate() {
        try {
            const prop = await this.facebookPostProposalService.find(this.id).toPromise();
            const key = 'app.facebookPostProposal.dataToText.' + prop.kind;
            this.text = await this.translate.get(key, prop.data).toPromise();
```

```
            await this.runModal();
            await this.runFacebookShareDialog();
        } catch (err) {
            console.log(err);
        }
        this.resetRoute();
    }

    continue() {
        this.modalRef.close();
    }

    cancel() {
        this.modalRef.dismiss('cancel');
    }

    private resetRoute() {
        this.router.navigate([{ outlets: { popup: null } }], { replaceUrl: true,
queryParamsHandling: 'merge' });
    }

    private async runModal() {
        try {
            this.modalRef = this.modal.open(this.templateRef, { size: 'lg', backdrop: 'static' });
            await this.modalRef.result;
        } finally {
            this.modalRef = null;
        }
    }

    private runFacebookShareDialog() {
        const params: UIParams = {
            method: 'share_open_graph',
            action_type: 'og.shares',
            action_properties: JSON.stringify({
                scrape: false,
                object: {
                    'og:url': 'http://www.fon.bg.ac.rs',
                    'og:title': this.text,
                }
            })
        };
        return this.facebook.ui(params);
    }
}
```

Свака компонент је декорисана са @Component где је уписано и име фајла са HTML кодом који следи.

```
<ng-template #template>
    <form name="postForm" role="form" novalidate (ngSubmit)="continue()" #postForm="ngForm">

        <div class="modal-header">
            <h4 class="modal-title" id="myFacebookPostProposalLabel"
                jhiTranslate="app.facebookPostProposal.createPostTitle"></h4>
            <button type="button" class="close" data-dismiss="modal" aria-hidden="true"
(click)="cancel()">&times;
            </button>
        </div>
        <div class="modal-body">
            <jhi-alert-error></jhi-alert-error>
            <div class="form-group">
                <label class="form-control-label" jhiTranslate="app.facebookPostProposal.postText"
for="text"></label>
                <textarea class="form-control" name="text" id="text" [(ngModel)]="text"
                          required minlength="10"></textarea>
                <div [hidden]="!(postForm.controls.text?.dirty && postForm.controls.text?.invalid)">
                    <small class="form-text text-danger"
[hidden]="!postForm.controls.text?.errors?.required"
                           jhiTranslate="entity.validation.required">
                    </small>
```

```
                    <small class="form-text text-danger"
[hidden]="!postForm.controls.text?.errors?.minlength"
                           jhiTranslate="entity.validation.minlength" translateValues="{min:10}">
                    </small>
                </div>
            </div>
        </div>
        <div class="modal-footer">
            <button type="button" class="btn btn-secondary" data-dismiss="modal" (click)="cancel()">
                <span class="fa fa-ban"></span> <span
jhiTranslate="entity.action.cancel">Cancel</span>
            </button>
            <button type="submit" [disabled]="postForm.form.invalid" class="btn btn-primary">
                <span class="fa fa-save"></span> Objavi
            </button>
        </div>
    </form>
</ng-template>
```

Ове се може видети извођење валидирања форми. Angular обезбеђује валидирање на основу уписаних ограничења (на textarea елементу имамо required minlength=”10”). Али ми морамо да имплементирамо интефјес који ће приказивати грешке на основу резултата валидације које нам Angular даје, као и да спречимо наставак радње ако форма није валидна, али је то шаблонски процес.

### 6.2.2 Сервиси

Пример сервиса који се користи на више места. Бави се дозволама које има тренутни корисник. Сервисне класе су декорисане са @Injectable().

```
@Injectable()
export class Principal {
    private userIdentity: User;
    private authenticated = false;
    private authenticationState = new Subject<User>();

    constructor(
        private account: AccountService,
        private trackerService: JhiTrackerService
    ) {}

    authenticate(identity) {
        this.userIdentity = identity;
        this.authenticated = identity !== null;
        this.authenticationState.next(this.userIdentity);
    }

    hasAnyAuthority(authorities: string[]): Promise<boolean> {
        return Promise.resolve(this.hasAnyAuthorityDirect(authorities));
    }

    hasAnyAuthorityDirect(authorities: string[]): boolean {
        if (!this.authenticated || !this.userIdentity || !this.userIdentity.authorities) {
            return false;
        }

        for (let i = 0; i < authorities.length; i++) {
            if (this.userIdentity.authorities.includes(authorities[i])) {
                return true;
            }
        }

        return false;
    }

    hasAuthority(authority: string): Promise<boolean> {
        if (!this.authenticated) {
           return Promise.resolve(false);
```

```
        }

        return this.identity().then((id) => {
            return Promise.resolve(id.authorities && id.authorities.includes(authority));
        }, () => {
            return Promise.resolve(false);
        });
    }

    identity(force?: boolean): Promise<User> {
        if (force === true) {
            this.userIdentity = undefined;
        }

        // check and see if we have retrieved the userIdentity data from the server.
        // if we have, reuse it by immediately resolving
        if (this.userIdentity) {
            return Promise.resolve(this.userIdentity);
        }

        // retrieve the userIdentity data from the server, update the identity object, and then
resolve.
        return this.account.get().toPromise().then((account) => {
            if (account) {
                this.userIdentity = account;
                this.authenticated = true;
                this.trackerService.connect();
            } else {
                this.userIdentity = null;
                this.authenticated = false;
            }
            this.authenticationState.next(this.userIdentity);
            return this.userIdentity;
        }).catch((err) => {
            if (this.trackerService.stompClient && this.trackerService.stompClient.connected) {
                this.trackerService.disconnect();
            }
            this.userIdentity = null;
            this.authenticated = false;
            this.authenticationState.next(this.userIdentity);
            return null;
        });
    }

    isAuthenticated(): boolean {
        return this.authenticated;
    }

    isIdentityResolved(): boolean {
        return this.userIdentity !== undefined;
    }

    getAuthenticationState(): Observable<User> {
        return this.authenticationState.asObservable();
    }

    getImageUrl(): String {
        return this.isIdentityResolved() ? this.userIdentity.imageUrl : null;
    }
}
```

### 6.2.3 Модули

Пример дефиниције модула. Модул класа је празна класа која је декорисана са @NgModule({…}). Ту је специфицирано шта све модул садржи. Декларисао је које друге модуле овај модул импортује, које компоненте садржи и које су улазне компоненте.

```
import {
    PassedCoursesComponent,
    DueCoursesComponent,
    ExamApplicationComponent,
    pageRoutes
} from './';

@NgModule({
    imports: [
        ItehProjectSharedModule,
        RouterModule.forChild(pageRoutes)
    ],
    declarations: [
        PassedCoursesComponent,
        DueCoursesComponent,
        ExamApplicationComponent
    ],
    entryComponents: [
        PassedCoursesComponent,
        DueCoursesComponent,
        ExamApplicationComponent
    ]
})
export class PagesModule {
}
```

### 6.2.4 Руте

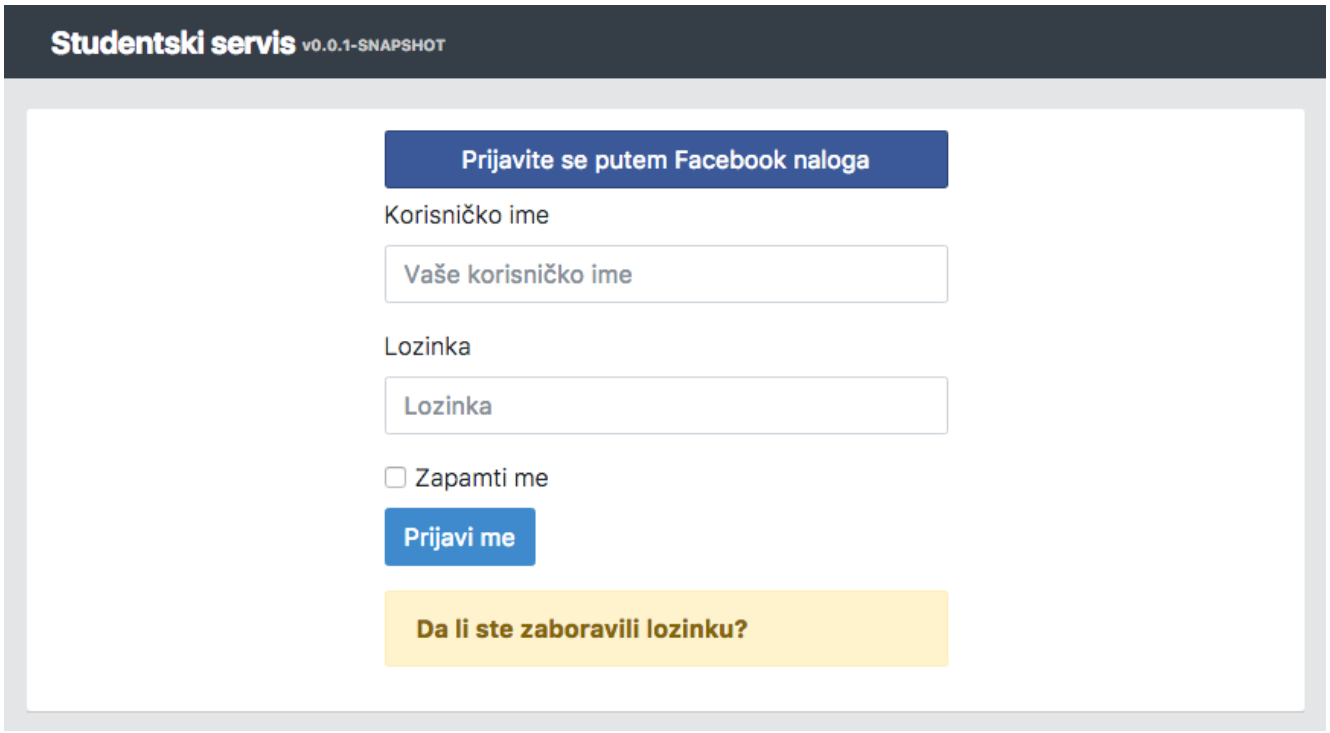
Пример су руте за модул представљен изнад. За сваку руту је дефинисана путања, улазна компонента, одређени подаци (углавном наслов странице у виду кода за локализацију) и конвертери параметара.

```
export const pageRoutes: Routes = [
    {
        path: 'passed-courses',
        component: PassedCoursesComponent,
        data: {
            pageTitle: 'app.page.passedCourses.title'
        },
        resolve: {
            pagingParams: ResolvePagingParams
        }
    },
    {
        path: 'due-courses',
        component: DueCoursesComponent,
        data: {
            pageTitle: 'app.page.dueCourses.title'
        },
        resolve: {
            pagingParams: ResolvePagingParams
        }
    },
    {
        path: 'exam-applications',
        component: ExamApplicationComponent,
        data: {
            pageTitle: 'app.page.dueCourses.title'
        },
        resolve: {
            pagingParams: ResolvePagingParams
        }
    }
];
```

# 7 Корисничко упутство

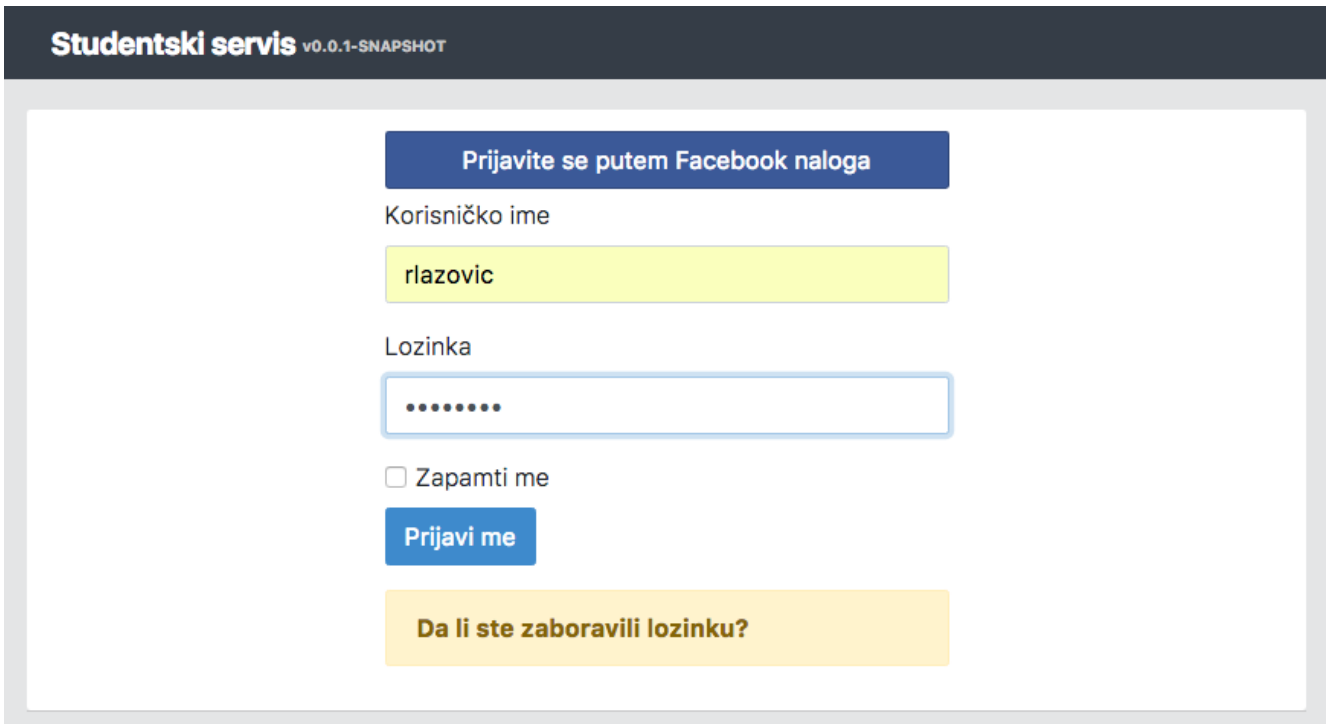
## *7.1 Пријава корисника на систем*

**Предуслов:** **Корисник** је уписан у систему.
Учитана је форма за пријаву на систем.



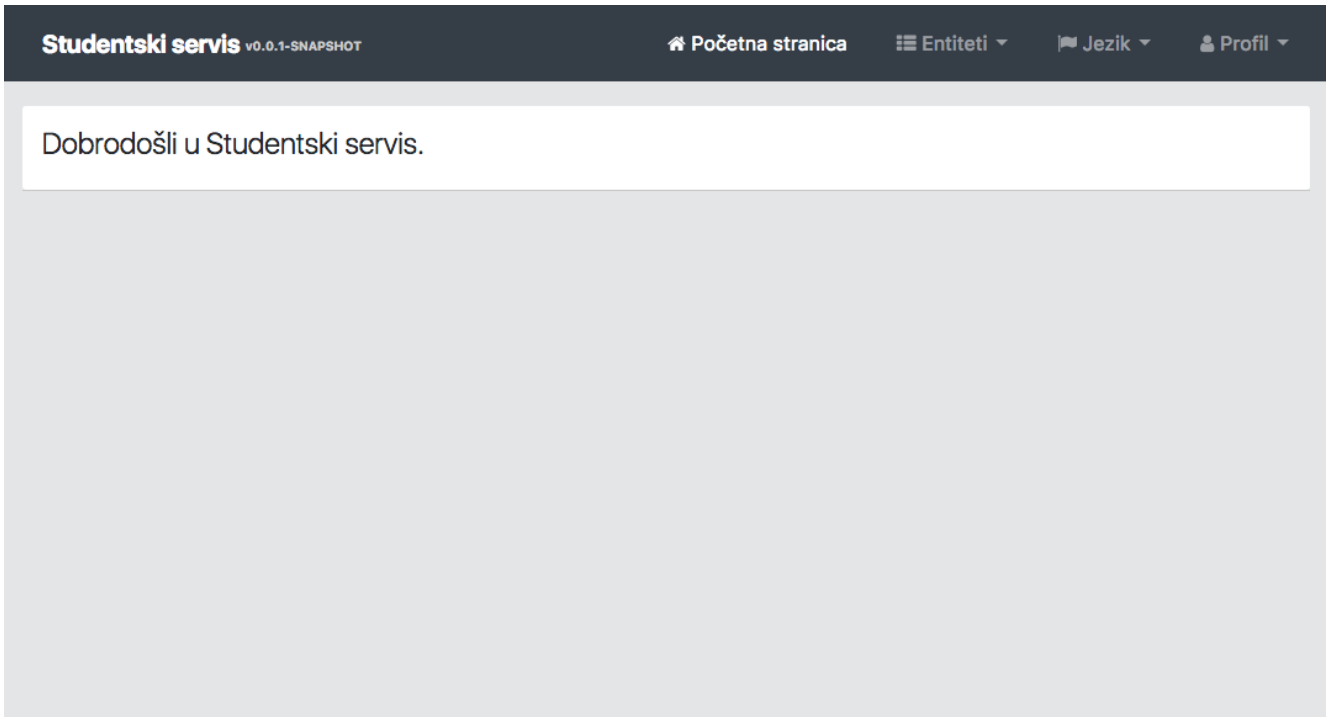

**Сценарио СК:**

1. **Корниск** уноси своје корисничко име и шифру.



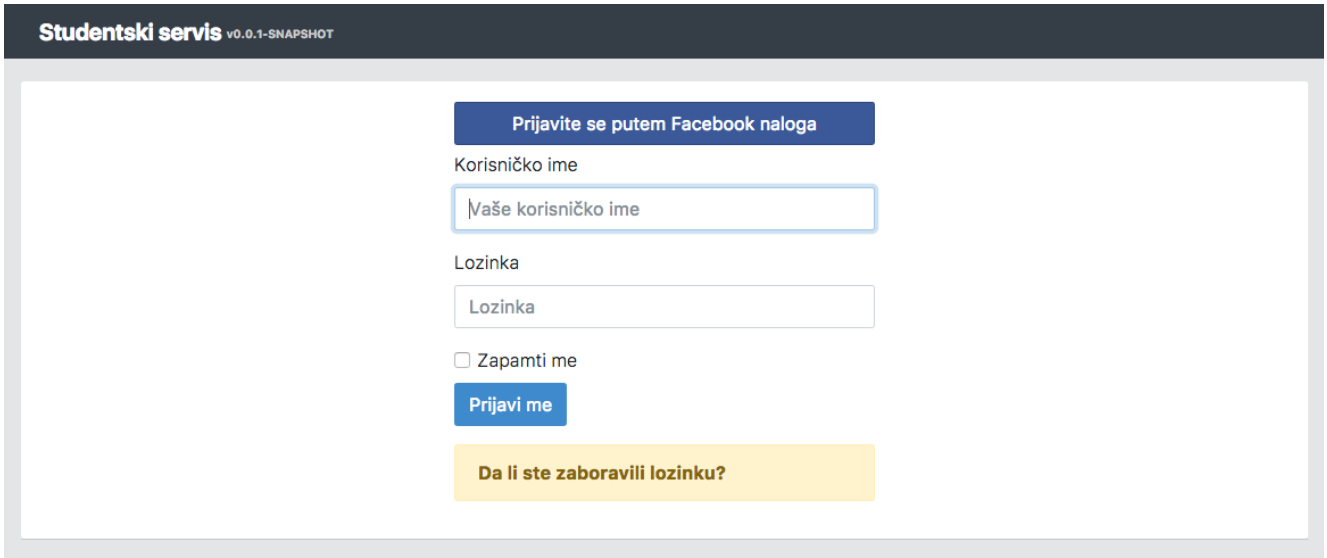

2. **Корисник** позива **систем** да га улогује и започне сесију.
3. **Систем** проверава унесене податке.

4. **Систем** приказује почетну форму за избор области апликације.



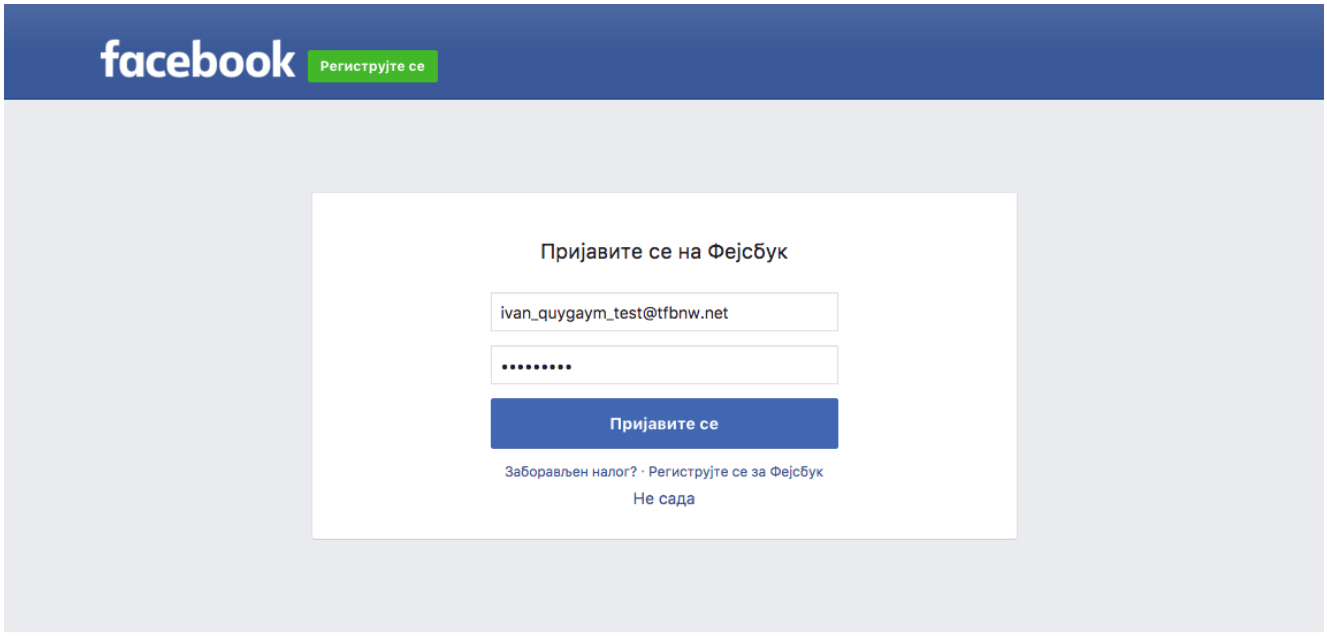

## *7.2 Пријава преко Facebook-а*

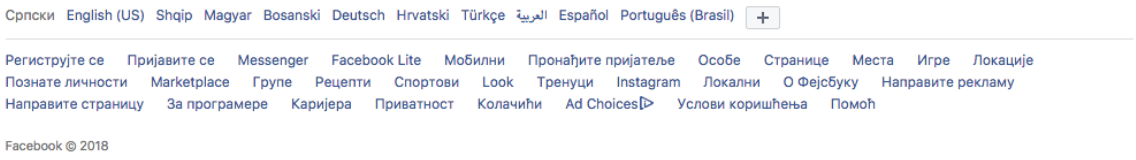
**Предуслов:** **Корисник** је уписан у **систему**.
**Корисник** има Facebook налог повезан са налогом у систему.
Учитана је форма за пријаву на систем.



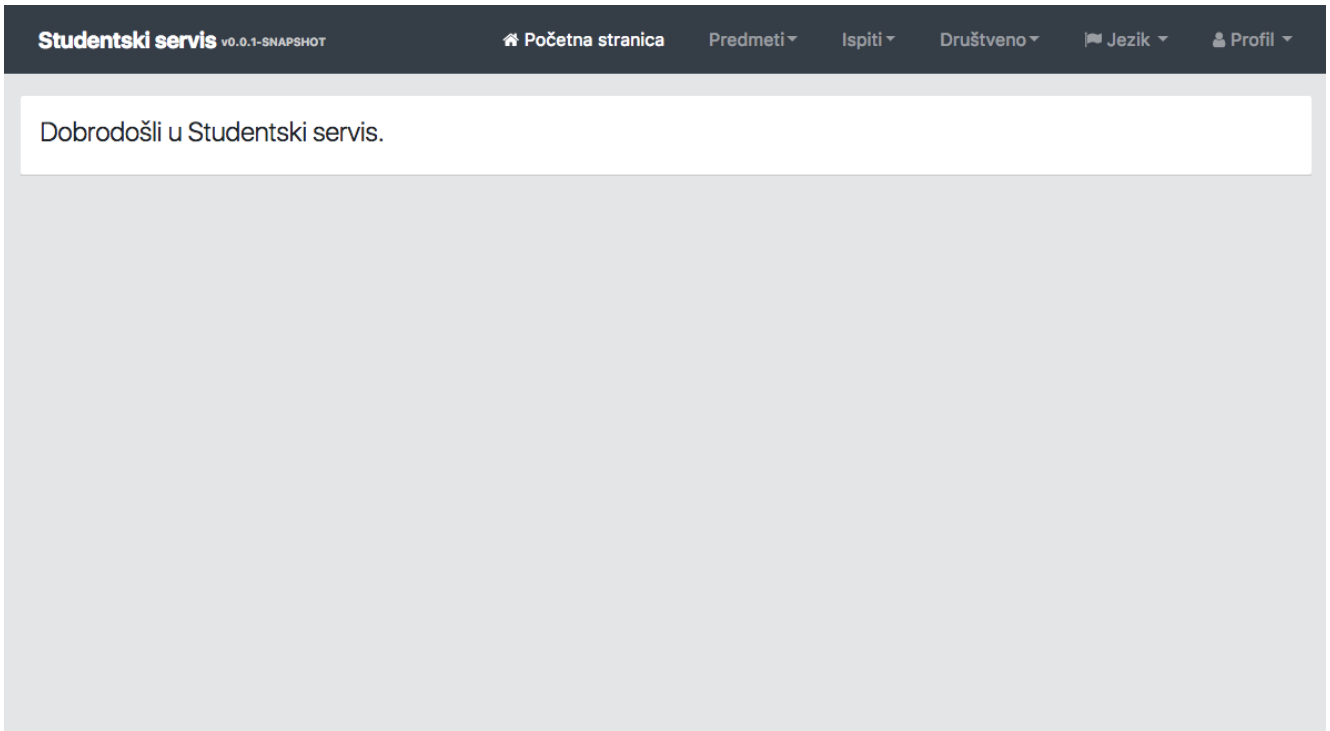

**Сценарио СК:**

1. **Корисник** позива **систем** да започне пријаву путем Facebook-а.
2. **Систем** позива **Facebook сервис** да аутентификује корисника и врати податке.



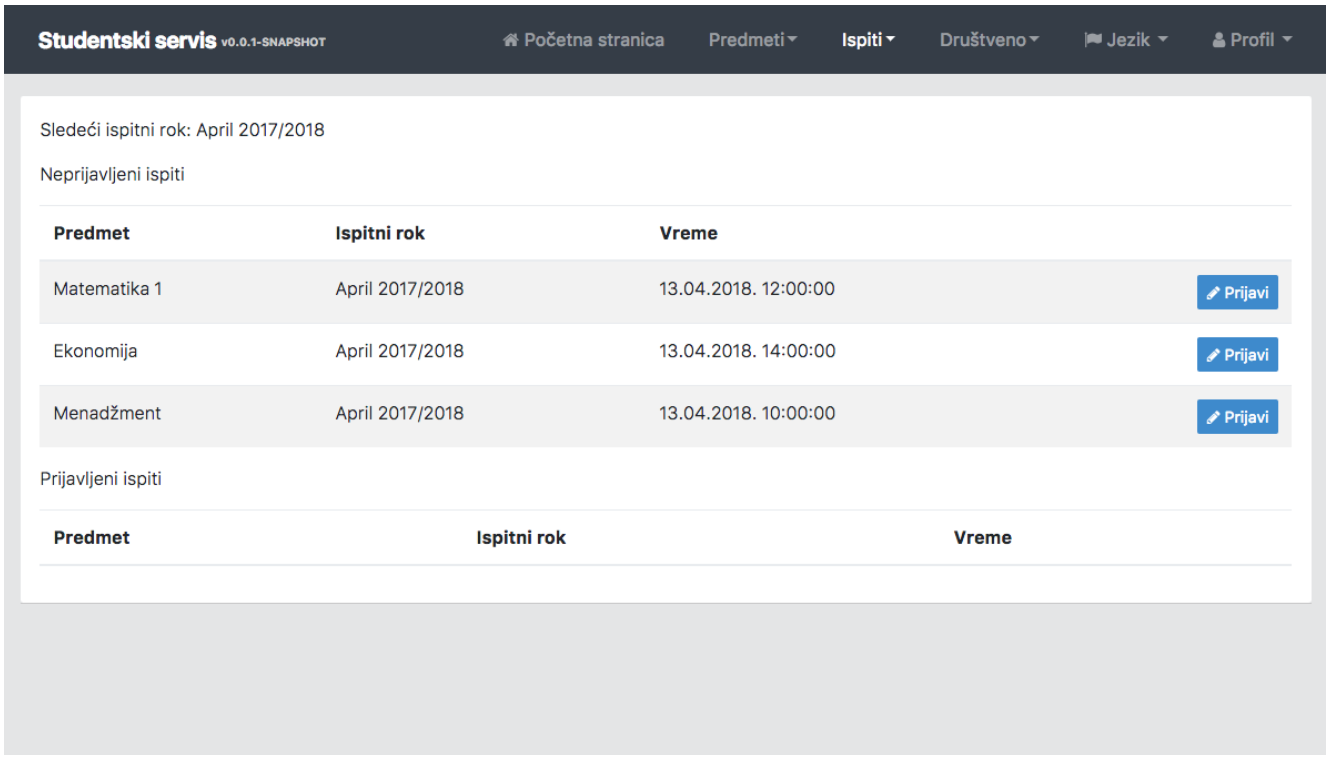




3. **Facebook сервис** аутентификује **корисника** и враћа **систему** податке.
4. **Систем** проверава добијене податке.
5. **Систем** приказује почетну форму за избор области апликације.



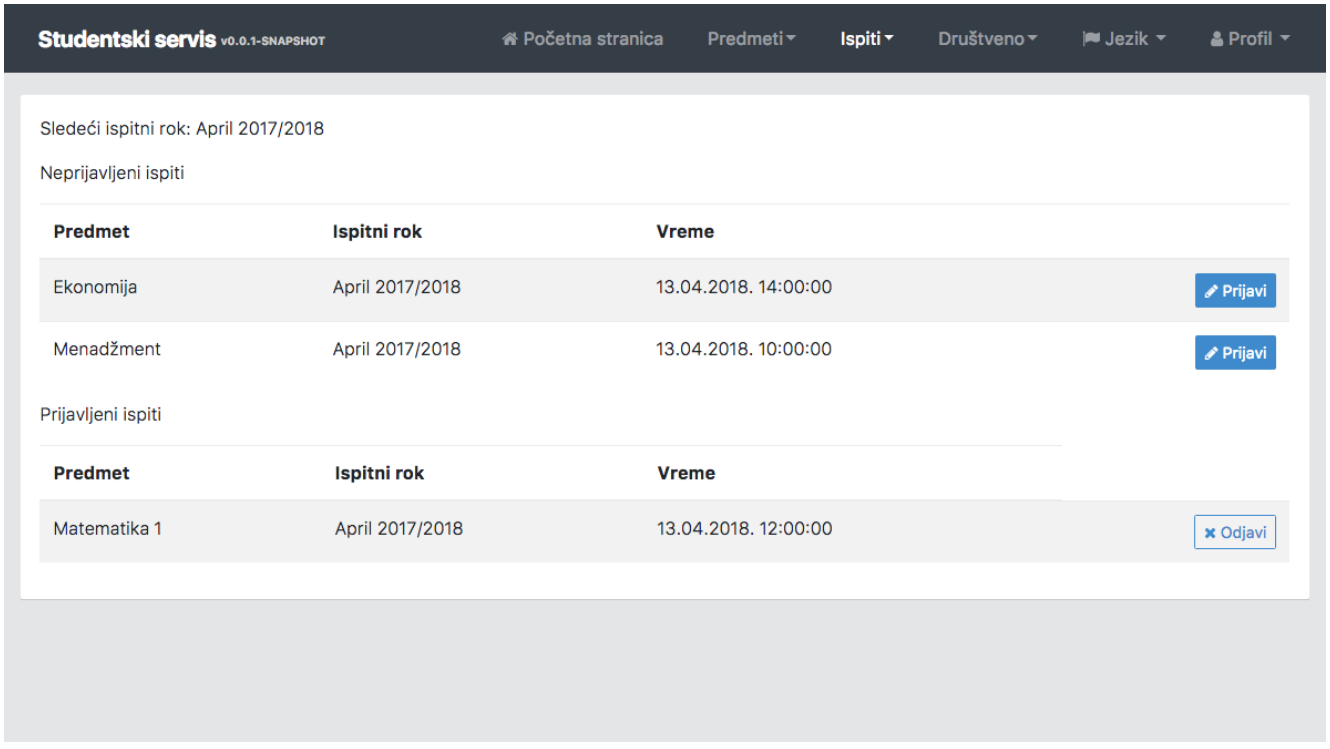
## *7.3 Пријава испита за испитни рок*

**Предуслов:** **Студент** је пријављен на **систем**.
Учитана је форма за пријаву испита.



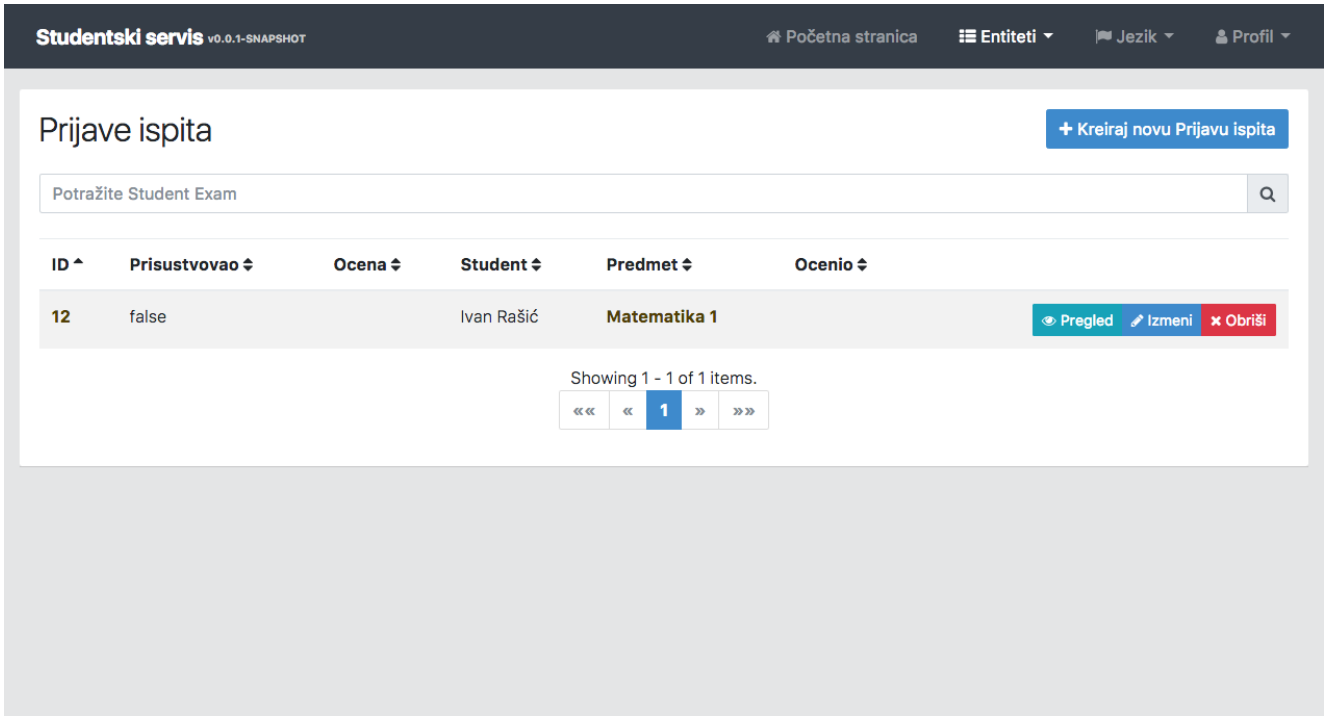

**Сценарио СК:**

1. **Студент** бира предмет који жели да пријави за следећи испитни рок.
2. **Студент** позива **систем** да забележи пријаву.
3. **Систем** бележи пријаву испита за изабрани предмет.
4. **Систем** враћа листу пријављених испита и предмета могућих за пријаву.



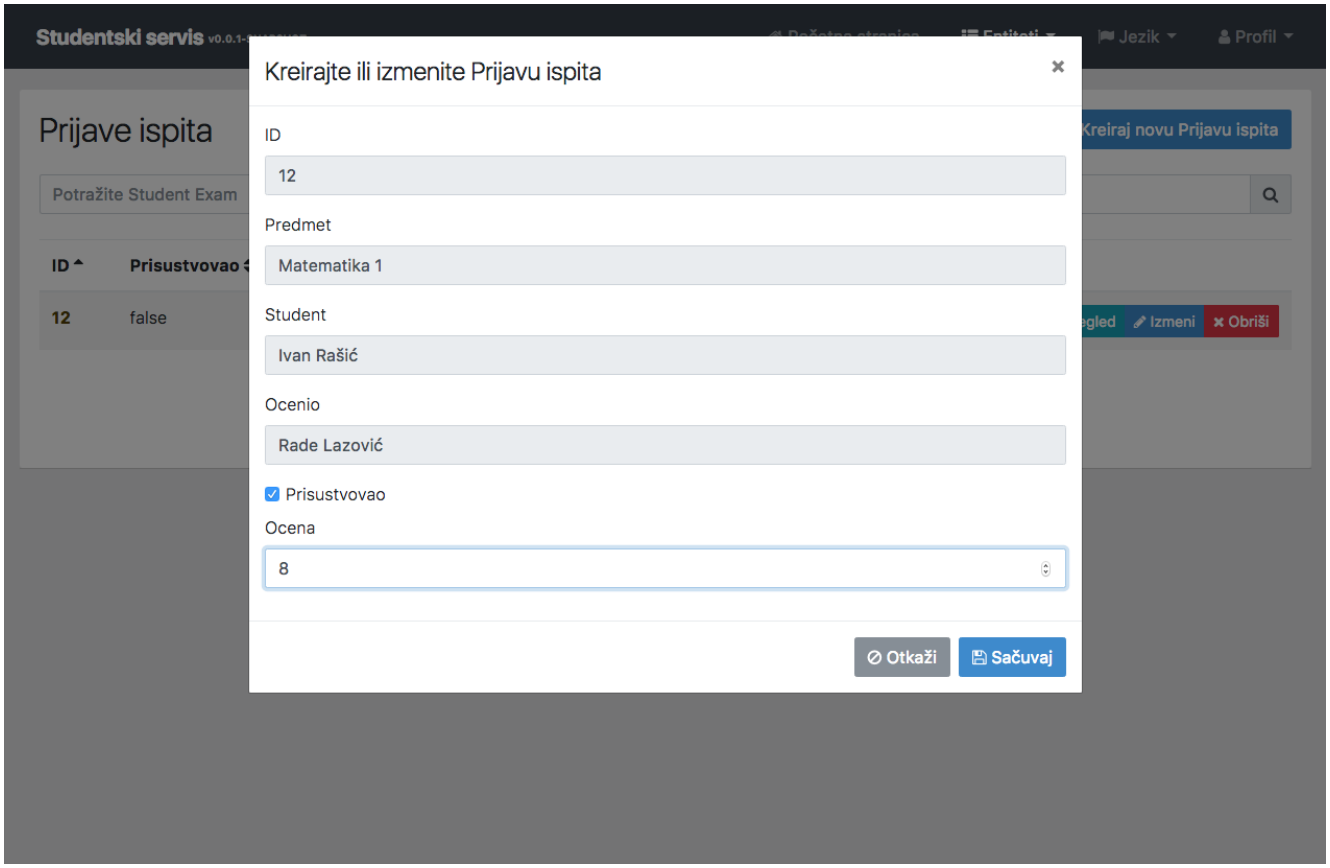
## *7.4 Измена пријаве (оцењивање)*

**Предуслов:** **Наставник** је пријављен на **систем**.
Учитана је форма за управљање испитних пријава.

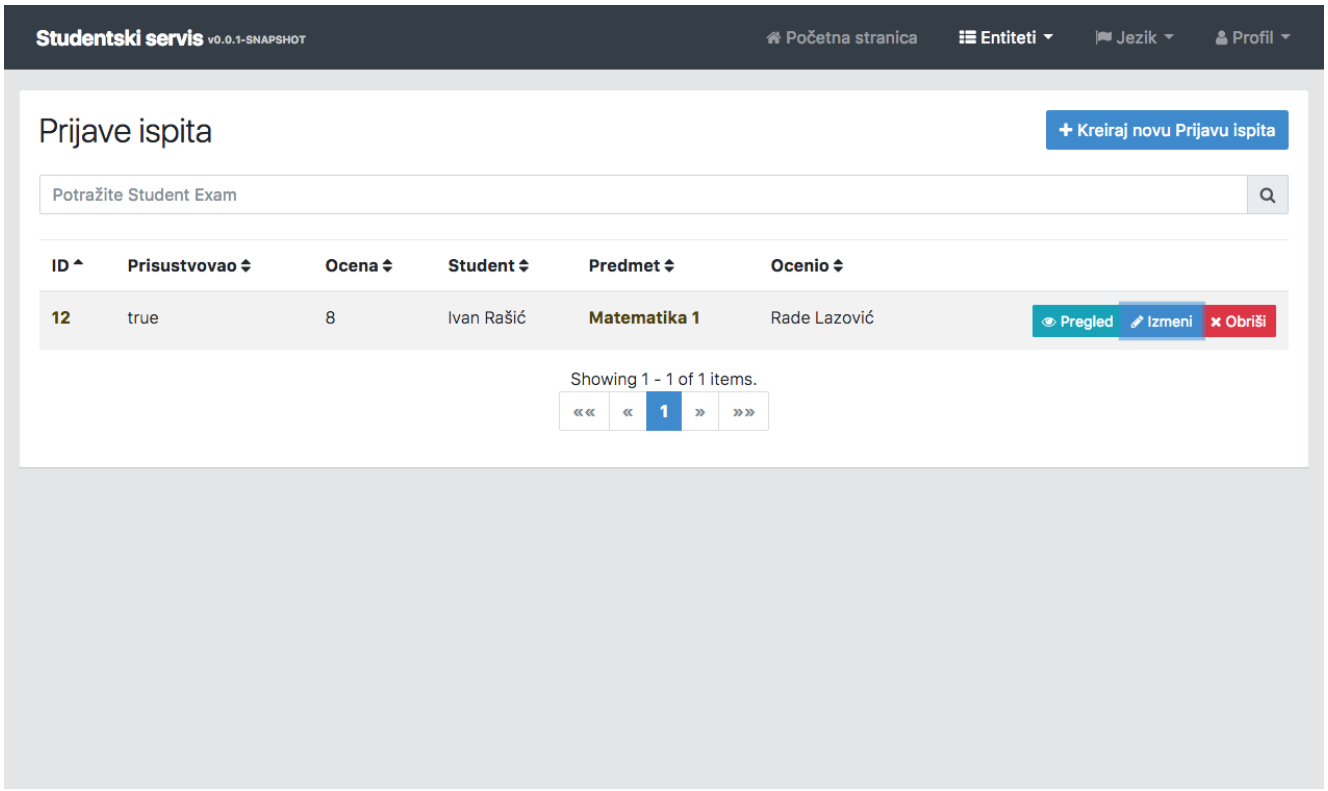



**Сценарио СК:**

1. **Наставник** бира пријаву испита коју жели да измени.
2. **Наставник** позива **систем** да врати податке за изабрану пријаву.
3. **Систем** добавља изабрану пријаву.
4. **Систем** враћа изабрану пријаву.
5. **Наставник** уноси нове податке.

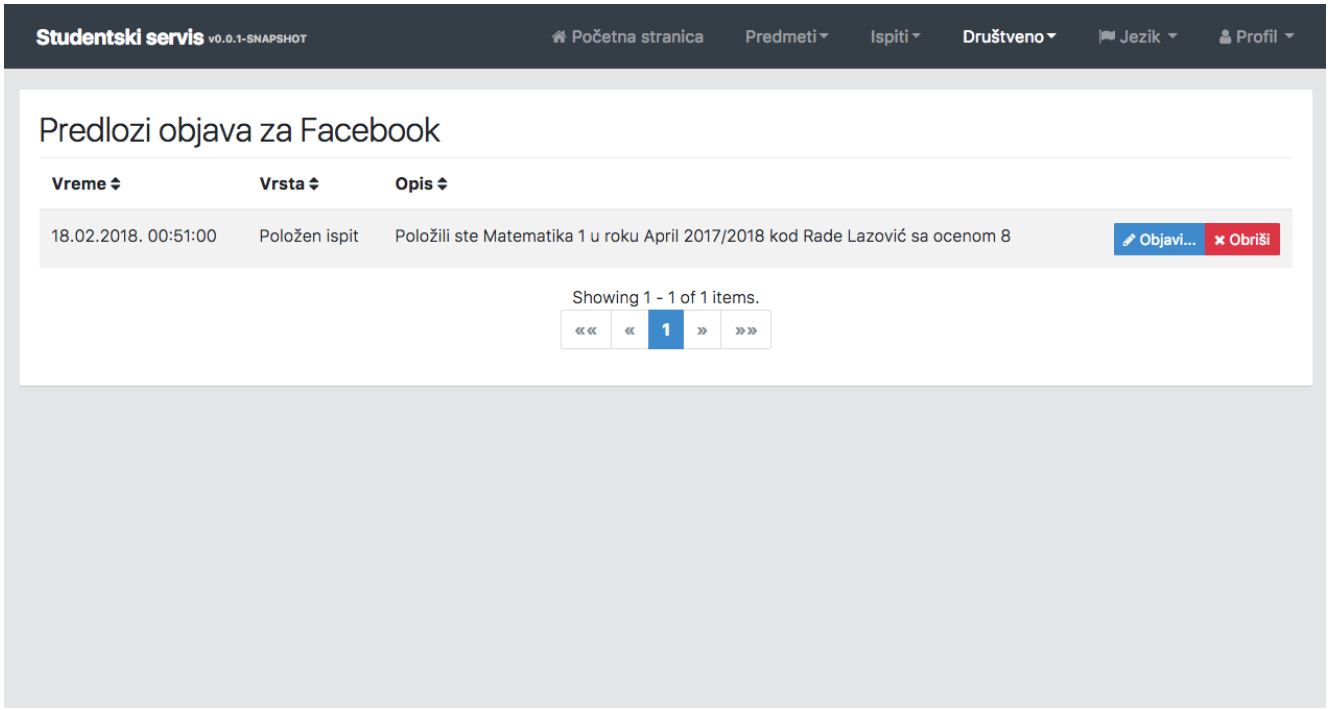



6. **Наставник** позива **систем** да сачува унесене податке и креира предлоге објава.
7. **Систем** снима нове податке и по потреби креира предлоге објава.
8. **Систем** враћа листу пријава са унесеним изменама.



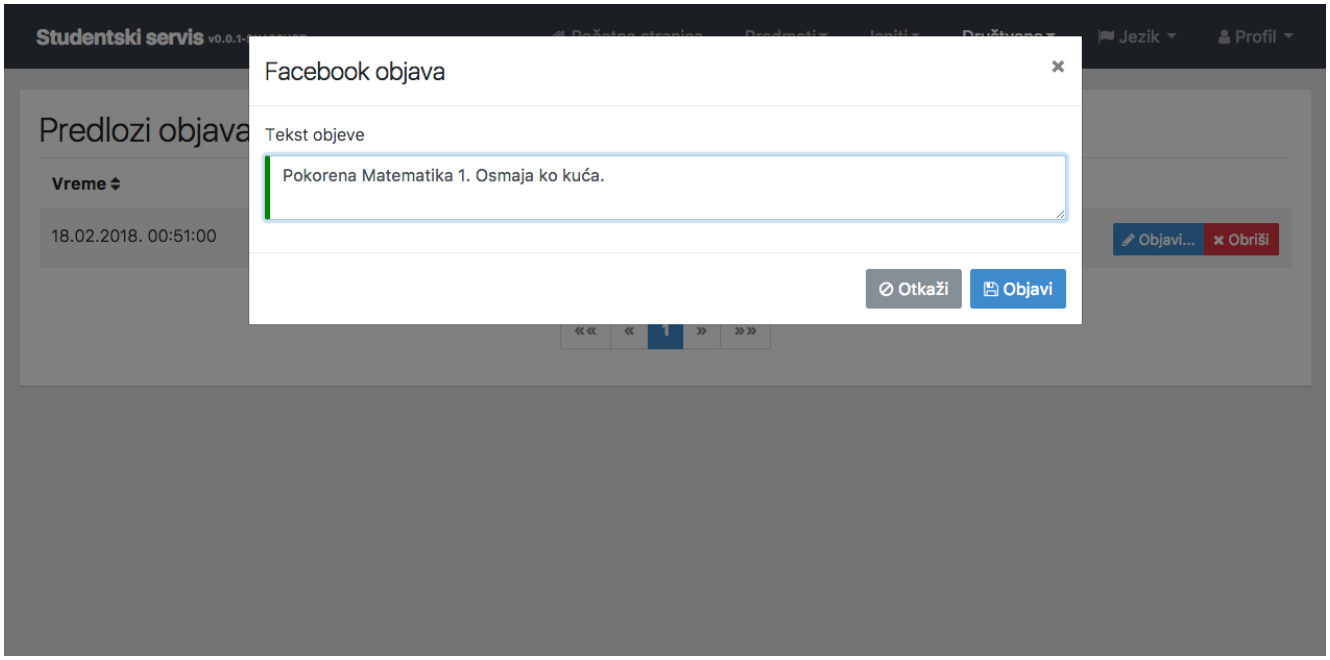
## *7.5 Објава догађаја на Facebook-у*

**Предуслов:** **Студент** је пријављен на **систем**.
Учитана је форма са предлозима објава.



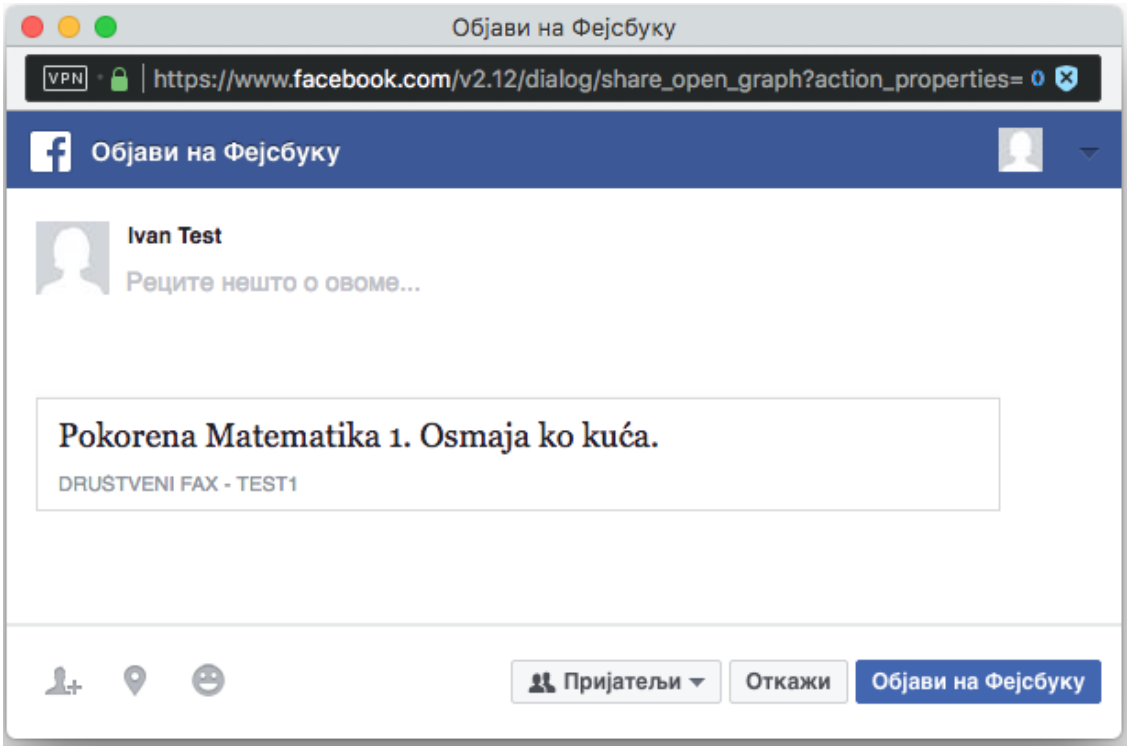

**Сценарио СК:**

1. **Студент** бира предлог објаве који жели да реализује.
2. **Студент** позива **систем** да састави предлог текста објаве.
3. **Систем** саставља предлог текста објаве.
4. **Систем** враћа предлог текста објаве.




5. **Студент** уноси измене текста по потреби.
6. **Студент** позива **систем** да започне процес објаве.
7. **Систем** позива **Facebook сервис** да започене процес објаве.

8. **Facebook сервис** аутентификује корисника и приказује форму за објаву са унесеним тексстом.




9. **Студент** позива **Facebook сервис** да изврши објаву.
10. **Facebook сервис** извршава објаву.