



# Univerzitet Singidunum

## Tehnički fakultet

Studijski program:  
*Softversko i informaciono inženjerstvo*

## Aplikacija za evidenciju prisustva studenata

### Diplomski rad

**Mentor:**

*prof. dr Aleksandar Jevremović*

**Kandidat:**

*Marko Dojkić  
2018/201682*

**Beograd, 2022.**

## **Sadržaj**

<b>1. Uvod.....</b>	<b>3</b>
<b>2. Način komunikacije i prosleđivanja poziva između aplikacija.....</b>	<b>4</b>
<b>3. Izrada web aplikacije za nastavno osoblje.....</b>	<b>8</b>
<b>4. Izrada pozadinskog (eng. Backend) Spring servera.....</b>	<b>13</b>
<b>5. Izgled MongoDB baze podataka.....</b>	<b>18</b>
<b>6. Izrada studentske aplikacije za Android operativni sistem.....</b>	<b>19</b>
<b>7. Izrada studentske aplikacije za iOS operativni sistem.....</b>	<b>25</b>
<b>8. Demonstracija rada aplikacije.....</b>	<b>29</b>
9.1. Web aplikacija.....	29
9.2. Studentska Android aplikacija.....	34
9.3. Studentska iOS aplikacija.....	37
<b>9. Zaključna razmatranja.....</b>	<b>39</b>
<b>10. Literatura.....</b>	<b>40</b>

## 1. Uvod

Ovaj rad će prikazati primenu različitih tehnologija prilikom izrade aplikacije koja će omogućiti evidenciju prisustva studenata. Početna ideja je bila da profesori i asistenti kreiraju QR kod, kojeg bi studenti skenirali na početku nastave i time evidentirali svoje prisustvo, ali bi to zahtevalo instaliranje dodatne opreme koja bi isti pokazivala, pa je ipak odlučeno da se kreira mobilna aplikacija. Struktura projekta je sledeća:

1. Web aplikacija koju koriste profesori, asistenti i administrator sistema, koja je izrađena pomoću PHP, HTML, CSS i JavaScript tehnologija.
2. Mobilna studentska aplikacija za Android operativni sistem, koja je izrađena u Android Studiju posredstvom Java programskog jezika.
3. Mobilna studentska aplikacija za iOS operativni sistem, koja je izrađena u xCode alatu posredstvom Swift programskog jezika.
4. Backend server napisan u Javi koji koristi Spring frejmворк i koji je posrednik između klijentskih aplikacija i baze podataka.
5. MongoDB baza podataka koja čuva sve potrebne podatke i parametre za logovanje na klijentske aplikacije.

Izabrana je gore navedena struktura pre svega zbog poznavanja samih tehnologija. Studentske aplikacije su izrađene u okruženjima koji su specijalizovani za razvoj aplikacija za odabrani operativni sistem. Na kraju MongoDB je izabrana baza podataka za skladištenje, zbog boljih performansi i sposobnosti brže obrade velike količine podataka, a takođe i zbog činjenice da ne postoji mnogo veza između samih entiteta, pa nije potrebna upotreba relacionih baza podataka.

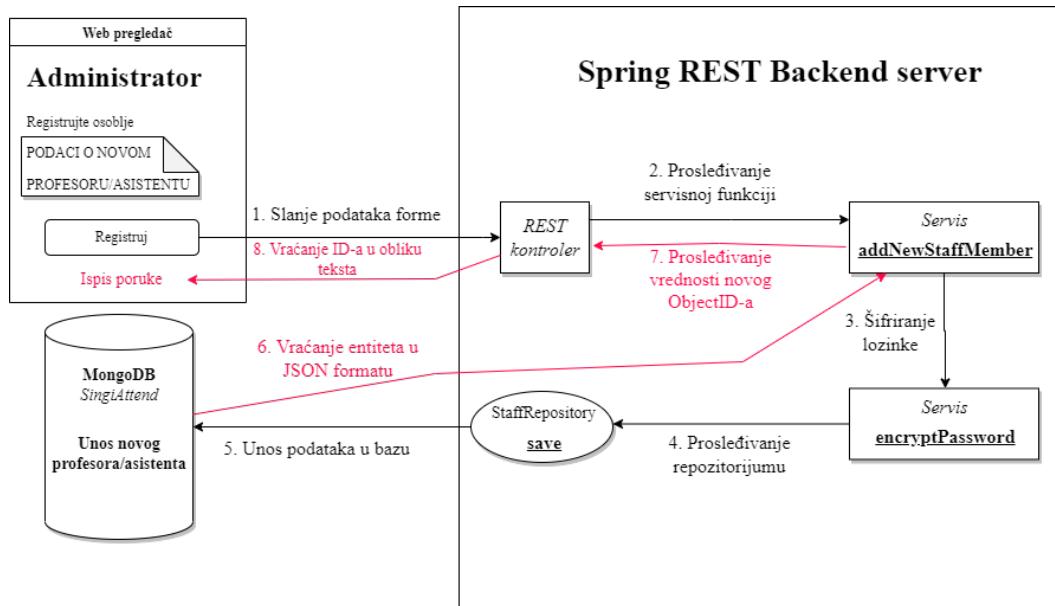
Pored izabranih tehnologija postojala je i mogućnost korišćenja frejmворка za izradu web aplikacije poput Angular-a i React-a, kao i korišćenje višeplatformskih rešenja poput Flutter-a i React Native-a, ali je zbog jednostavnosti projekta, a kasnije i njegovog održavanja, ipak odlučeno korišćenje već navedenih alata.

Na kraju prilikom odabira baze podataka, prvo bitno je korišćena relaciona MySQL baza, ali se ubrzo uvidela ne mogućnost evidentiranja prisutnih studenata unutar same baze. Tada je alternativa bila smeštanje prisutnih studenata unutar zasebnih JSON fajlova, čije ime odgovara primarnom ključu svakog predavanja, odnosno vežbi. Budući da MongoDB već čuva podatke u JSON formatu on je bio logičan izbor.

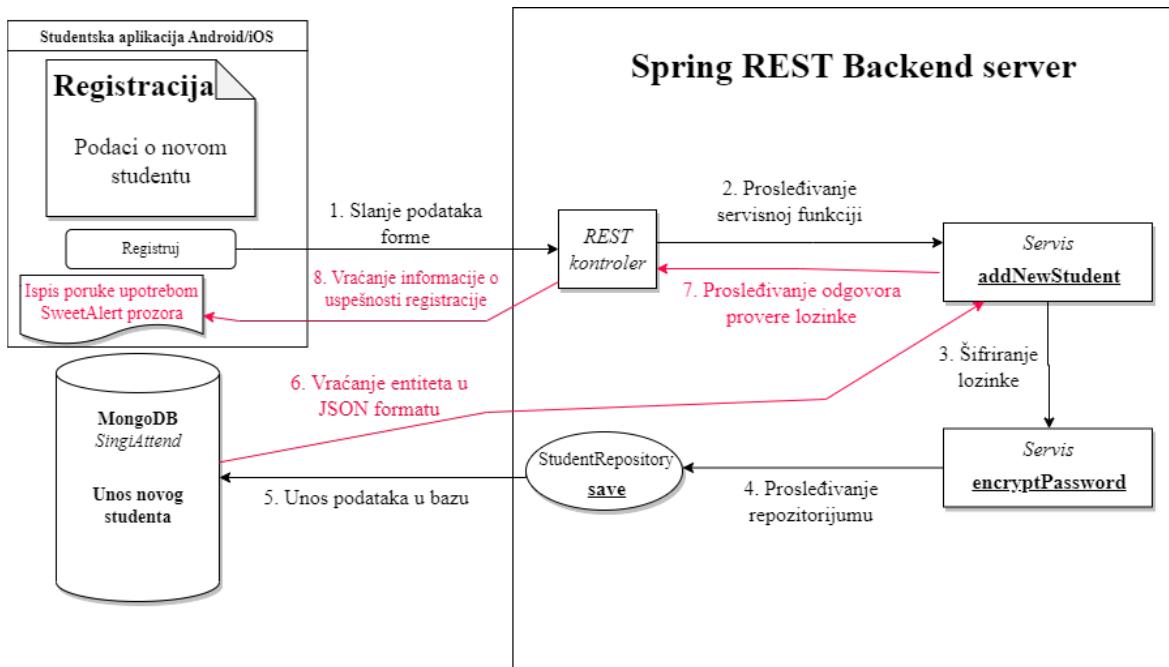
Celokupan projekat je urađen u MVC (eng. *Model-View-Controller*) arhitekturi, a komunikacija se odvija slanjem poziva ka Spring Boot REST kontroleru, koji podatke šalje servisnoj klasi, koja ih obrađuje i pomoću Mongo rezitorijuma šalje, odnosno dobavlja podatke iz baze i na kraju šalje konačan rezultat frontend aplikacijama u JSON formatu.

## 2. Način komunikacije i prosleđivanja poziva između aplikacija

Inicijalno u aplikaciji postoji samo jedan administratorski nalog koji može da registruje nastavno osoblje, tj. profesore i asistente. Studenti se registruju popunjavanjem registracione forme unutar mobilnih aplikacija.

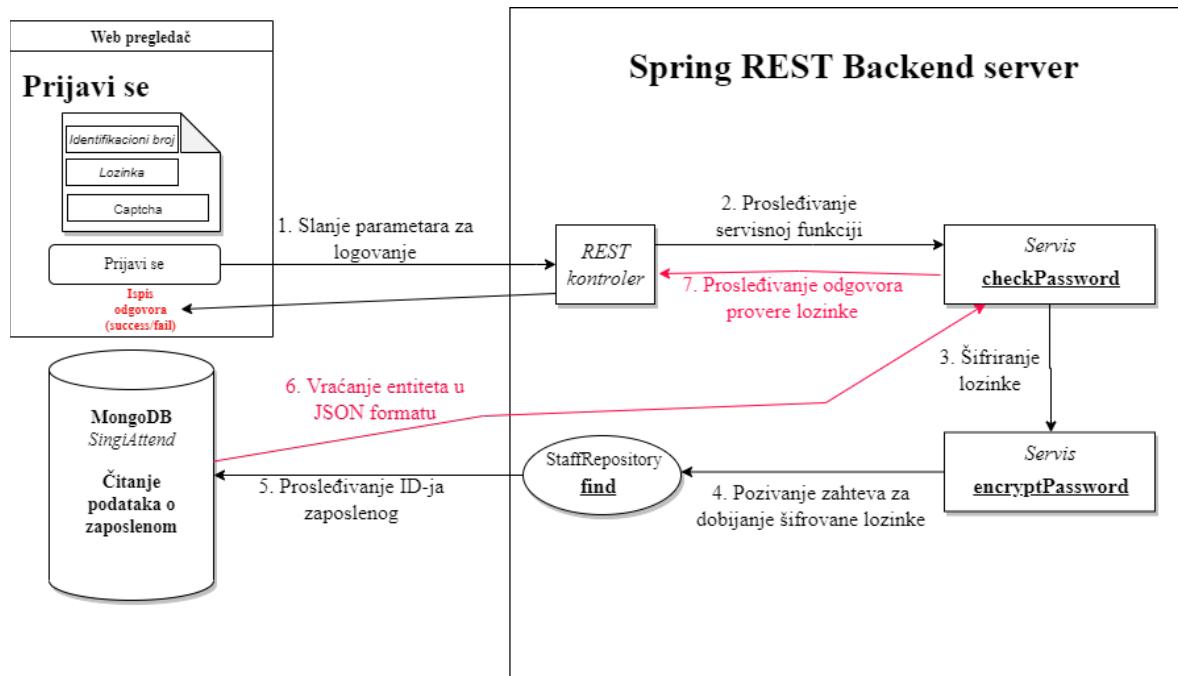


Slika 1: Prikaz toka komunikacije prilikom registracije novog profesora/asistenta

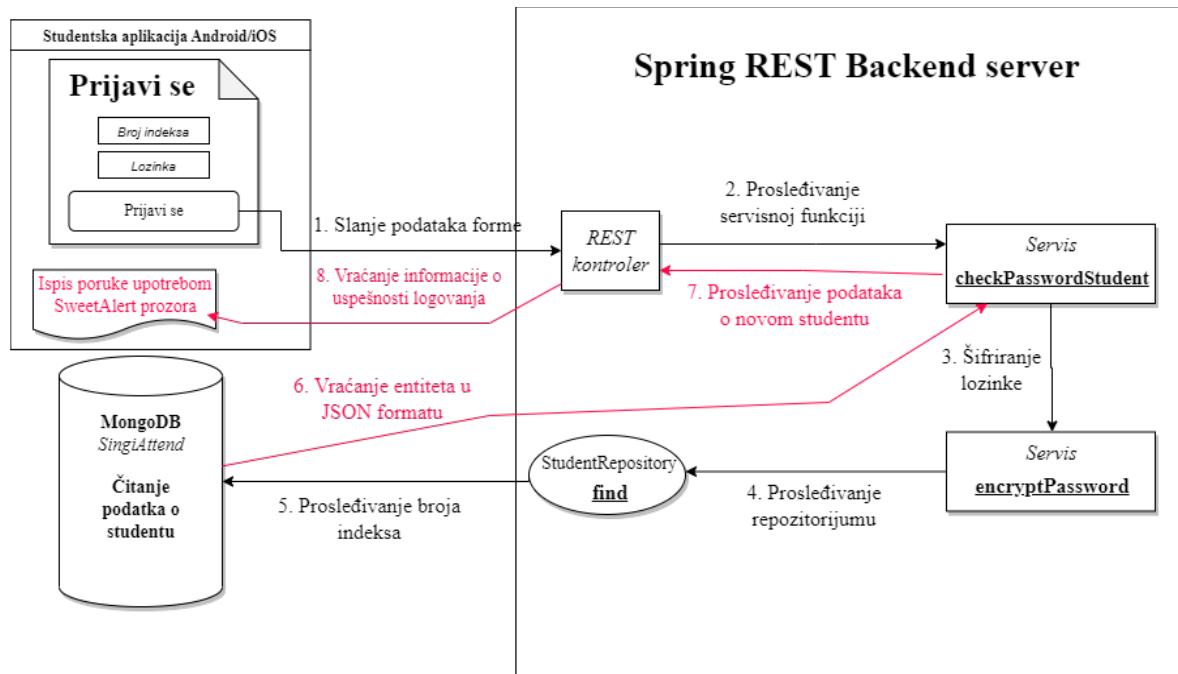


Slika 2: Prikaz toka komunikacije prilikom registracije novog studenta

Nakon uspešne registracije nastavno osoblje se može ulogovati koristeći jedinstveni id, koji je generisan prilikom kreiranja entiteta u bazi, i lozinke, a studenti koriste svoj broj indeksa i lozinku kao parametre.

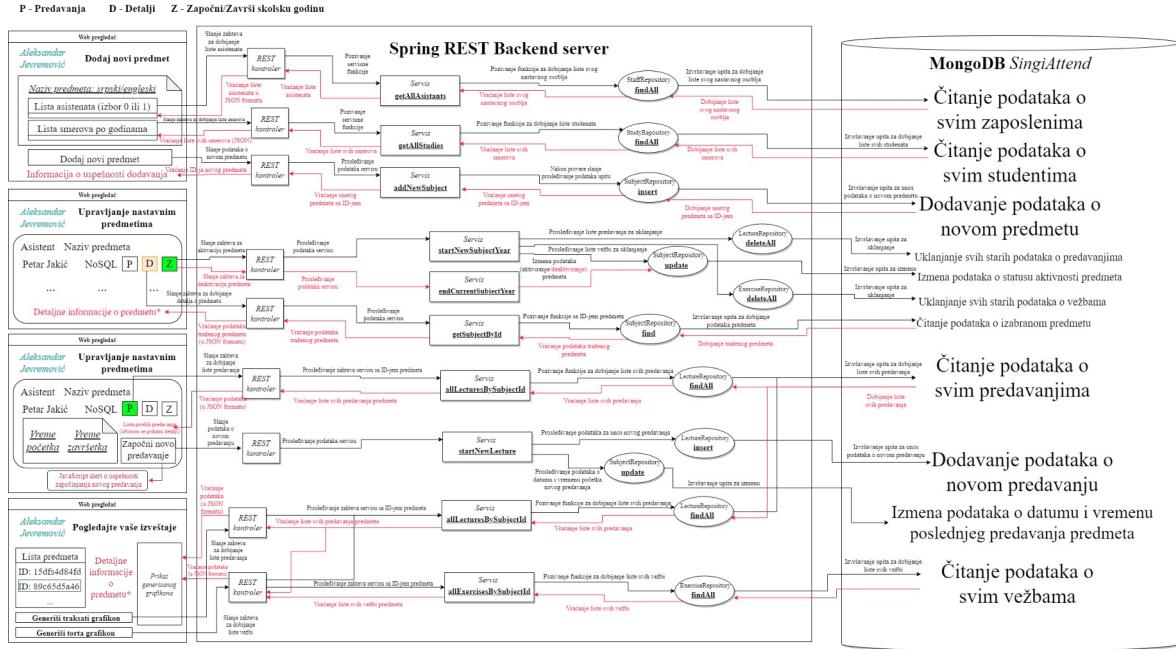


Slika 3: Prikaz toka komunikacije prilikom logovanja profesora/asistenta



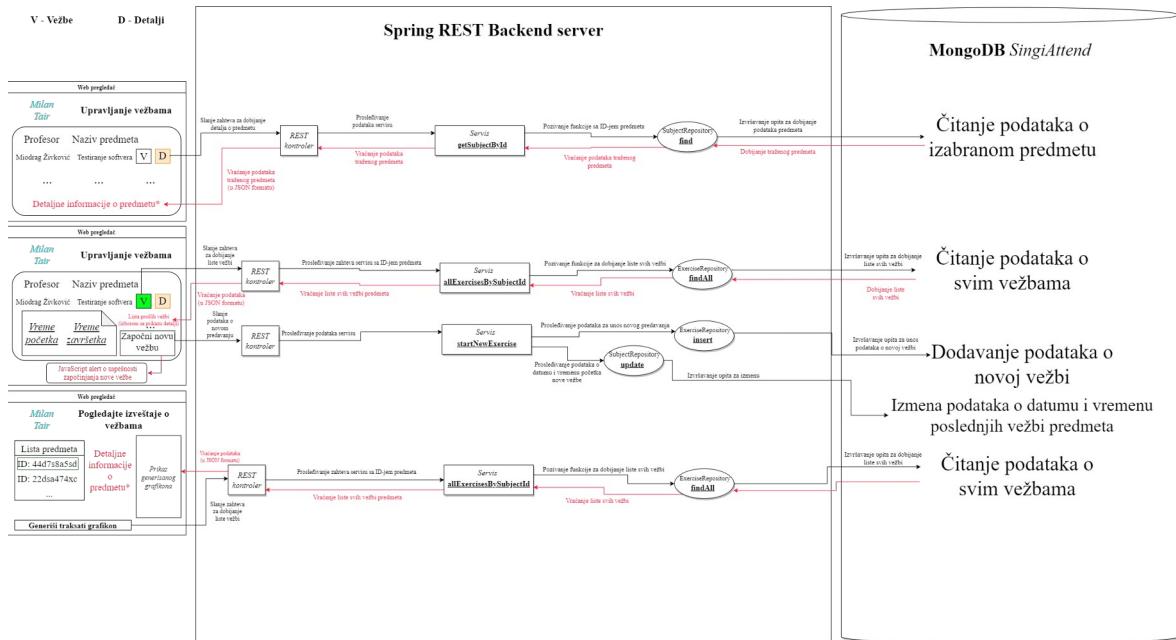
Slika 4: Prikaz toka komunikacije prilikom logovanja studenta

Profesori imaju mogućnost da unose nove predmete, započinju novu školsku godinu, biraju asistente na svojim predmetima, beleže evidenciju prisustva na predavanjima i takođe da vide grafikone prisustva na predavanjima i vežbama.



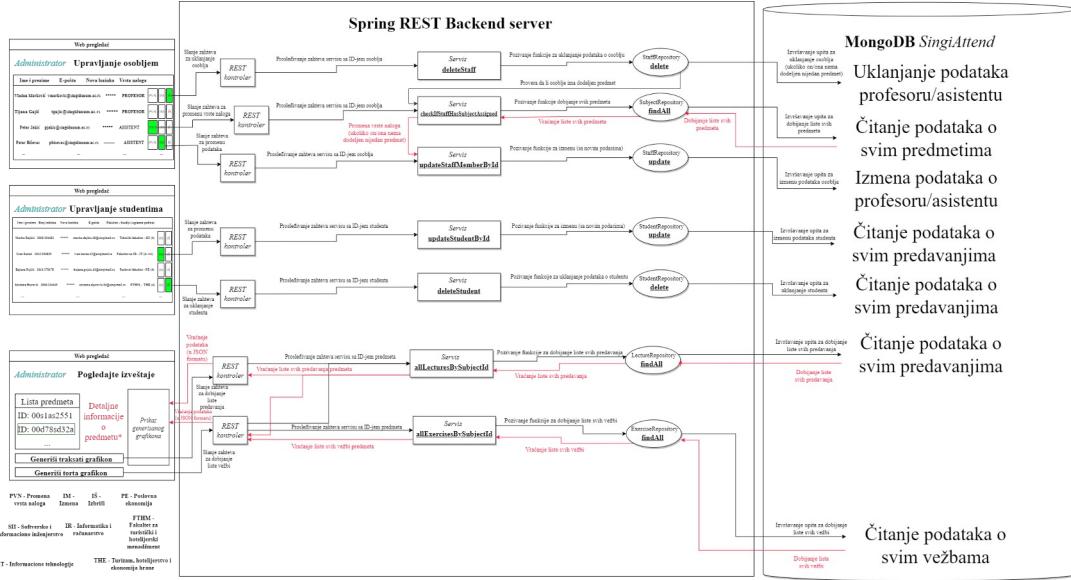
Slika 5: Prikaz tokova komunikacije stranica profesora

Asistenti imaju samo mogućnost da beleže evidenciju prisustva sa vežbi predmeta na kojima su oni angažovani i vide grafikone ukupnog prisustva sa istih.



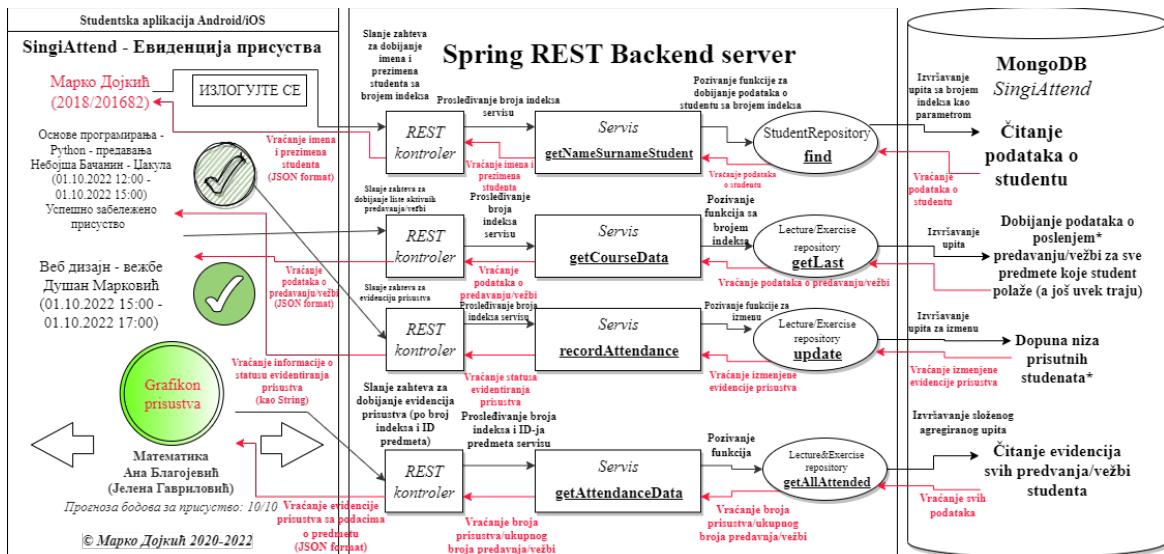
Slika 6: Prikaz tokova komunikacije stranica asistenta

Administrator nakon logovanja sa svojim parametrima, pored mogućnosti registracije osoblja (slika 1), može da menja i uklanja podatke o već registrovanom osoblju, tj. profesorima i asistentima, a takođe i o studentima. Administrator može da vidi grafikone prisustva svih predmeta.



Slika 7: Prikaz tokova komunikacije stranica administratora

Studenti prilikom logovanja mogu videti grafikone evidencije prisustva na predavanjima i vežbama, kao i trenutno planiran broj bodova za prisustvo (maksimalno 10 bodova), a pored toga i da evidentiraju prisustvo na trenutno aktivnim predavanjima, odnosno vežbama.



Slika 8: Prikaz tokova komunikacije studenta

### **3. Izrada web aplikacije za nastavno osoblje**

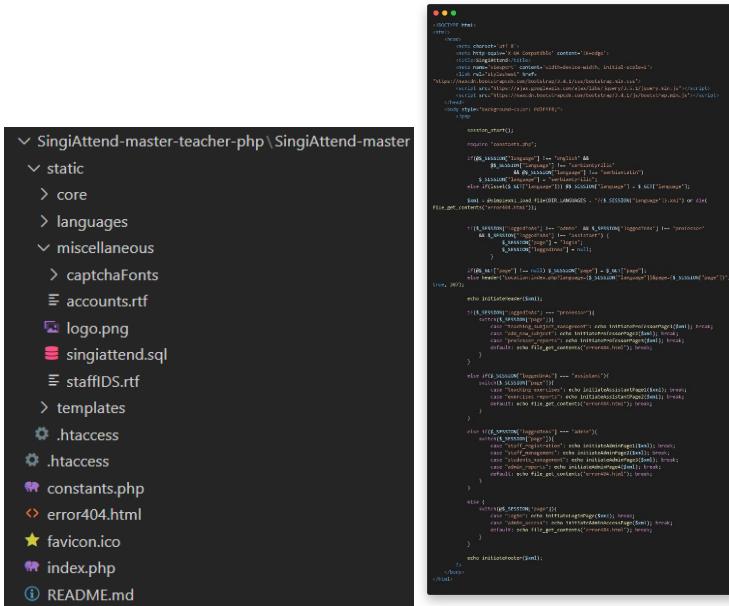
Struktura aplikacije (slika 9) je podeljena po direktorijumima na sledeći način:

- **core** – Sadrži sve PHP pomoćne fajlove koji služe za obradu logike aplikacije
  - **languages** – Sadrži 3 XML fajla koji čuvaju tekst stranica na engleskom i srpskom jeziku (ćirilično i latinično pismo)
  - **miscellaneous** – Sadrži fajlove poput spiska ID-jeva naloga, slika i captcha fontova, u njemu se takođe zbog potrebe arhiviranja nalazi i stara SQL baza
  - **templates** – Sadrži HTML šablonе koji se nakon popunjavanja podacima od PHP koda prikazuju na stranicama

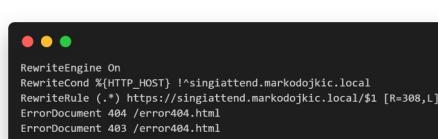
Pored navedene strukture u korenom direktorijumu se nalazi i index stranica, stranica za prikaz ukoliko se nađe na nepostojeću ili stranicu bez prava pristupa, kao i PHP fajl koji sadrži sve konstante, poput putanja i parametara za povezivanje na Spring server. Direktan pristup van je zabranjen htaccess pravilima.

Za inicijalizaciju svake stranice je zadužen indeksni PHP fajl, čiji je glavni HTML deo prikazan na slici 10. Nakon započinjanja sesije učitavaju se konstante, a potom XML fajl izabranog jezika (podrazumevan jezik je srpski – cirilično pismo). Nakon toga se proverava da li je neki korisnik ulogovan, a u suprotnom se učitava stranica za logovanje.

Na kraju se učitavaju redom header, trenutna stranica (ukoliko je korisnik ulogovan) i na kraju footer.



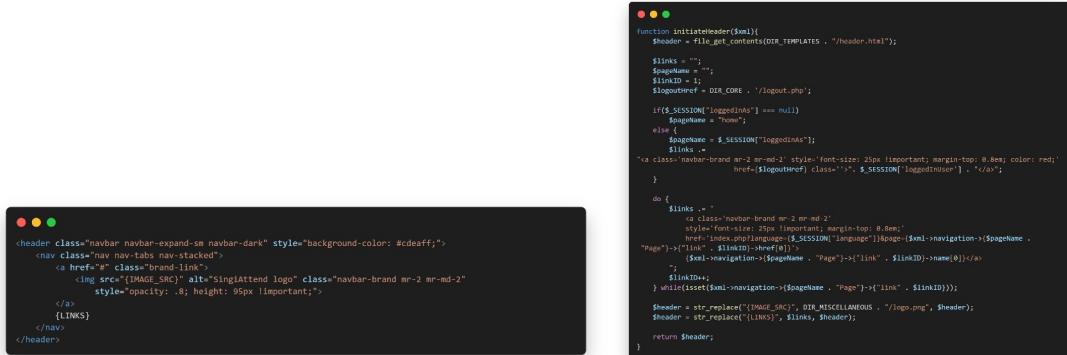
*Slike 9 i 10: Struktura web aplikacije i glavni deo indeks stranice*



*Sljedeći dio: Izgled htaccess fajla i konstanti (constants.php)*

Heder se sastoji od nav taga koji inicijalno sadrži samo logo, a ostale linkove dinamički popunjava PHP funkcija (slika 14), nakon učitavanja templejta.

Ukoliko se u radu nađe na stranicu kojoj korisnik nema pristup ili ne postoji ceo inicijalizuje se HTML kod sa slike 15.



*Slike 13 i 14: Prikaz šablonu i funkcije za inicijalizaciju hedera.*



Slika 15: HTML kod stranice za prikaz greške 404



Slika 16: Prikaz XML-a prevoda teksta na srpskom jeziku (ćirilično pismo)

Administrator se loguje na posebnoj putanji (*/admin\_access*) popunjavanjem iskačućeg prozora (korisničko ime i lozinka). Ukoliko su autentifikacioni parametri ispravni administrator se preusmerava na stranicu za registraciju osoblja, a u suprotnom na „error404“ stranicu.

```

function initiateAdminAccessPage($xml){
    if(@$_SESSION["isAdminLoggedout"]){
        $_SERVER["PHP_AUTH_USER"] = null;
        $_SERVER["PHP_AUTH_PW"] = null;
        $_SESSION["isAdminLoggedOut"] = false;
    }
    authenticateAdmin($xml,$conn);
}

function authenticateAdmin($xml){
    $adminPass = "$2y$10$zeRF8Y01yTitpNMuyHMPuYwBFRcPh96L6Bol0AE1wztZpiUFKU9S";

    header("WWW-Authenticate: Basic realm=\"Administrator panel\"");
    header("HTTP/1.0 401 Unauthorized");
    if (@$_SERVER['PHP_AUTH_USER'] === 'Administrator' && password_verify($_SERVER['PHP_AUTH_PW'], $adminPass)){
        $_SESSION["loggedInAs"] = "admin";
        $_SESSION['loggedInUser'] = "Administrator";
        header("Location:index.php?language=".$_SESSION["language"]&page=staff_registration",true, 301);
    }
}

echo "<script>window.location = 'index.php?language=".$_SESSION["language"]."&page=login';</script>";
}

```

Slika 17: Prikaz koda za autentifikaciju administratora

Prilikom registracije osoblja potrebno je izvršiti „captcha“ proveru. Ona se generiše kodom sa slike 18, crtanjem 6 nasumičnih karaktera koji predstavljaju velika slova engleske abecede. Navedena provera je neophodna svaki put kada se registruje ili jedan profesor, tj. asistent ili pri pokretanju višestrukog unosa putem CSV fajla (kod je prikazan na slici 19).

```

if($_FILES["csv_newuserslist"]["error"] == 0){
    if ($_POST["captcha"] != $_SESSION["captcha_text"]) die("<div style='color:red;font-size:14px;'> " . $xml . "<br>" . $invalid_captcha[0] . "</div><br><br>");
    errors->invalid_captcha();
}

$data_csv = array_map('str_getcsv', file($_FILES["csv_newuserslist"]["tmp_name"]));
$invalid_csv_path = DIR_ROOT . DIR_MISCELLANEOUS . '/invalidCsvRows.csv';

if(file_exists($invalid_csv_path))
    unlink($invalid_csv_path); //izbrisati prethodni fajl ukoliko on postoji

for($i = 0; $i < count($data_csv); $i++){
    if(!preg_match_all($password_pattern, $data_csv[$i][1])
        || !preg_match_all('/^([a-z\d]+)\.([a-z\d]+)\.([a-z\d]+)\.([a-z\d]+)\.([a-z\d]+)$/', $data_csv[$i][1])
        || !preg_match_all('/^([a-z\d]+)\.([a-z\d]+)\.([a-z\d]+)\.([a-z\d]+)\.([a-z\d]+)\.([a-z\d]+)$/', $data_csv[$i][2]))
        file_put_contents($invalid_csv_path, $data_csv[$i][2] . "\n", FILE_APPEND | LOCK_EX);
    else{
        $url = "http://" . SERVER_URL . SERVER_PORT . "/api/insert/staff";
        $user_data = array("name"=>$data_csv[$i][0], "email"=>$conn,$data_csv[$i][2], "password_hash"=>hash($data_csv[$i][1], "sha256"));
        $context = stream_context_create(array(
            'HTTPHEADER' => "Authorization: Basic " . base64_encode(SERVER_USERNAME . ":" . SERVER_PASSWORD),
            'CONTENT_TYPE' => "application/json",
            'CONTENT_LENGTH' => strlen(json_encode($user_data)),
            'HTTPMETHOD' => "POST"
        ));
        $response = json_decode(file_get_contents($url, false, $context));
        if($response->"error" == null){
            file_put_contents($invalid_csv_path, $data_csv[$i][2] . "\n" . $xml->errors->registrationError);
        }
        else{
            $newUserHeader = $response->id . ' ' . strtoupper($data_csv[$i][0]);
            $newUserHeader .= "-----" . $newUserHeader . "-----";
            $data_csv[$i][0] = $newUserHeader;
            $data_csv[$i][1] = $data_csv[$i][1];
            $data_csv[$i][2] = $data_csv[$i][2];
            $data_csv[$i][3] = $newUserHeader;
        }
        file_put_contents(DIR_ROOT . DIR_MISCELLANEOUS . '/accounts.csv', $newUser, FILE_APPEND | LOCK_EX);
    }
}

if(file_exists($invalid_csv_path))
    unlink($invalid_csv_path);

$_SESSION["captcha_text"] = $captcha_string;

for($i = 0; $i < $string_length; $i++){
    $letter_space = 198/$string_length;
    $initial = 15;

    imagerectext($image, 24, rand(15, 30), $initial + $i*$letter_space, rand(2, 4), $letters[rand(0, 3)], $font[rand(0, 3)], $captcha_string[$i]);
}

header("Content-type: image/png");
imagepng($image);
imagedestroy($image);
}

```

Slike 18 i 19: Kodovi za generisanje „captcha“ izazova i čitanje CSV fajla za registraciju osoblja

Ulogovani profesor pored mogućnosti da vidi i menja svoje predmete, kao i da započinje evidentiranje prisustva na predavanjima mora prvo dodati te predmete. Stranica za dodavanje predmeta se sastoji od forme koja sadrži: tekstualno polje (za unos naziva predmeta u formatu *naziv na engleskom/naziv na srpskom jeziku*), dropdown listu za izbor jednog (ili nijednog) asistenta i listu višestrukog izbora (za izbor smerova koji slušaju taj predmet, tj. kurs).

HTML kod stranice je prikazan na slici 20. Placeholderi koji sadrže podatke koji se biraju pomoći listi (asistente i smerove po godinama) se popunjavaju nakon dobavljanja istih pomoću PHP-a (kod sa slike 21) posredstvom Spring servera.

Nakon popunjavanja forme podaci se šalju na server (kod sa slike 22), nakon čega se unosi novi predmet i vraća poruka o uspešnosti izvršavanja unosa ili postojanju greške.

```


Asistant_selection



Studies



Submit



Reset


```

Slike 20 i 21: HTML i PHP kodovi za prikaz stranice za dodavanje novih predmeta

```

session_start();
require '../constants.php';
$url = file_get_contents(ROOT . '_HTTP_LANGUAGES' . (($_SESSION['language']) ? '' : ''));
$data = json_decode(file_get_contents(ROOT . '_HTTP_REQUEST'), true);
$server = array();
$server['port'] = $_POST['server_port'];
$server['url'] = SERVER_URL . $server['port'] . '/api/getAllAssistants';
$server['context'] = stream_context_create(array(
    'http' => array(
        'method' => 'GET',
        'header' => "Authorization: Basic " . base64_encode(SERVER_USERNAME . ":" . SERVER_PASSWORD),
        'content-type' => 'application/json'
    )
));
$server['context']['method'] => 'GET';

$curl = curl_init($server['url']);
curl_setopt($curl, CURLOPT_RETURNTRANSFER, 1);
curl_setopt($curl, CURLOPT_HTTPHEADER, $server['context']['header']);
curl_setopt($curl, CURLOPT_POST, 1);
curl_setopt($curl, CURLOPT_POSTFIELDS, $data);
curl_setopt($curl, CURLOPT_HTTPHEADER, $server['context']['header']);
curl_setopt($curl, CURLOPT_RETURNTRANSFER, 1);
curl_setopt($curl, CURLOPT_POST, 1);
curl_setopt($curl, CURLOPT_POSTFIELDS, $data);

$response = curl_exec($curl);
if ($response === false) {
    echo "Error: " . curl_error($curl);
} else {
    $response = json_decode($response, true);
    if ($response['status'] == 'success') {
        echo "Subject added successfully!";
    } else {
        echo "Error: " . $response['message'];
    }
}

```

Slike 22: PHP kod za slanje podataka forme za dodavanje novog predmeta

Nakon evidentiranja prisustva profesori mogu videti grafikone koji pokazuju prisustva studenata na predavanjima i vežbama. Ukoliko se prikazuju prisustva sa predavanja, nakon izbora predmeta, generiše se stubičasti grafikon koji za svako predavanje pokazuje ukupan broj prisutnih i odsutnih studenata. Kod asistenta ovaj grafikon prikazuje evidenciju sa vežbi. Na kraju moguće je generisati i „pita“ grafikon (samo profesori i administrator) koji prikazuje ukupno prisustvo na svim predavanjima i vežbama (0-100%).

```

if($_POST["graphValue"] === "barGraph"){
    echo "
        <script src='https://cdn.jsdelivr.net/npm/chart.js@2.9.4/dist/Chart.min.js'></script>
        <canvas id='myChart' style='position: relative; height:80vh; width:82vw;'></canvas>
        <script>
            var ctx = document.getElementById('myChart').getContext('2d');
            var myChart = new Chart(ctx, {
                type: 'bar',
                data: {
                    labels: {$datesData},
                    datasets: {$attendancesData}
                },
                options: {
                    scales: {
                        yAxes: [{{
                            ticks: {
                                beginAtZero: true
                            }
                        }}]
                    },
                    title: {
                        display: true,
                        text: '{$xml->$_SESSION["loggedInAs"] . "Page"}->barChartTitle[0]'}
                }
            });
        </script>
    ";
}
else if($_POST["graphValue"] === "pieGraph"){
    $attendedTotalExercises = 0;
    $totalStudentsAllExercises = 0;

    $url = "http://" . SERVER_URL . SERVER_PORT . "/api/getAllExercises/" . $_POST["subjectSelection"];

    $context = stream_context_create(array(
        "http" => array(
            "header" => "Authorization: Basic " . base64_encode(SERVER_USERNAME . ":" . SERVER_PASSWORD) . "\r\n",
            "Content-Type: application/json",
            "protocol_version" => 1.1,
            "method" => 'GET'
        )));
    $exercises = json_decode(file_get_contents($url, false, $context), true);

    if(!is_array($exercises) || $exercises.length == 0) die("<i style='color:red;font-size:28px;'>" . $xml->errors->graphNotGenerated[0] . "</i>");

    foreach($exercises as $exercise){
        $attended .= "\n" . sizeof($exercise["attended_students"]) . ",";
        $notAttended .= "\n" . ($totalStudents - sizeof($exercise["attended_students"])) . ",";
        $datesData = "\n" . explode("T", $exercise["started_at"])[0] . "\n";
        $attendedTotalExercises += sizeof($exercise["attended_students"]);
        $totalStudentsAllExercises += $totalStudents;
    }

    $percentageAL = round($attendedTotalExercises/$totalStudentsAllExercises*100);
    $percentageNAL = 100-$percentageAL;
    $percentageAE = round($attendedTotalExercises/$totalStudentsAllExercises*100);
    $percentageNAE = 100-$percentageAE;

    $label1 = $xml->professorPage->attended[0] . " " . strtr(lower($xml->professorPage->lecturesBtn[0]));
    $label2 = $xml->professorPage->notAttended[0] . " " . strtr(lower($xml->professorPage->lecturesBtn[0]));
    $label3 = $xml->professorPage->attended[0] . " " . strtr(lower($xml->assistantPage->exercisesBtn[0]));
    $label4 = $xml->professorPage->notAttended[0] . " " . strtr(lower($xml->assistantPage->exercisesBtn[0]));

    echo "
        <script src='https://cdn.jsdelivr.net/npm/chart.js@2.9.4/dist/Chart.min.js'></script>
        <canvas id='mychart' style='position: relative; height:80vh; width:164vw;'></canvas>
        <canvas id='mychart2' style='position: relative; height:80vh; width:164vw;'></canvas>
        <script>
            var ctx = document.getElementById('myChart').getContext('2d');
            var myChart = new Chart(ctx, {
                type: 'pie',
                data: {
                    labels: ['$label1','$label2'],
                    datasets: [
                        {backgroundColor: ['#33FF33','#FF3333'], data : [$percentageAL,$percentageNAL] }
                    ]
                },
                options: {
                    title: {
                        display: true,
                        text: '{$xml->professorPage->pieChartTitle[0]}'
                    }
                }
            });
            var ctx = document.getElementById('myChart2').getContext('2d');
            var myChart2 = new Chart(ctx, {
                type: 'pie',
                data: {
                    labels: ['$label3','$label4'],
                    datasets: [
                        {backgroundColor: ['#99ff00','#ff8080'], data : [$percentageAE,$percentageNAE] }
                    ]
                },
                options: {
                    title: {
                        display: true,
                        text: '{$xml->assistantPage->pieChartTitle[0]}'
                    }
                }
            });
        </script>
    ";
}

```

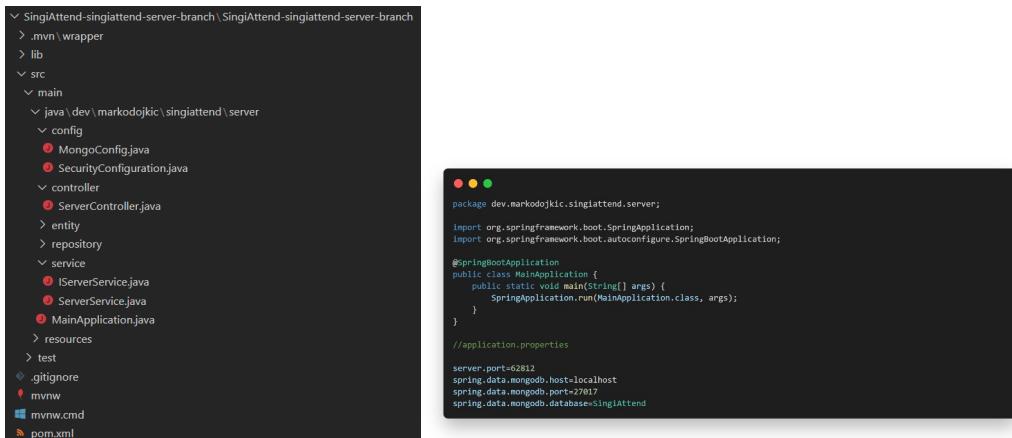
Slika 23: PHP kod za generisanje JavaScript koda koji prikazuje grafikone upotrebom Chart.js biblioteke

## 4. Izrada pozadinskog (eng. Backend) Spring servera

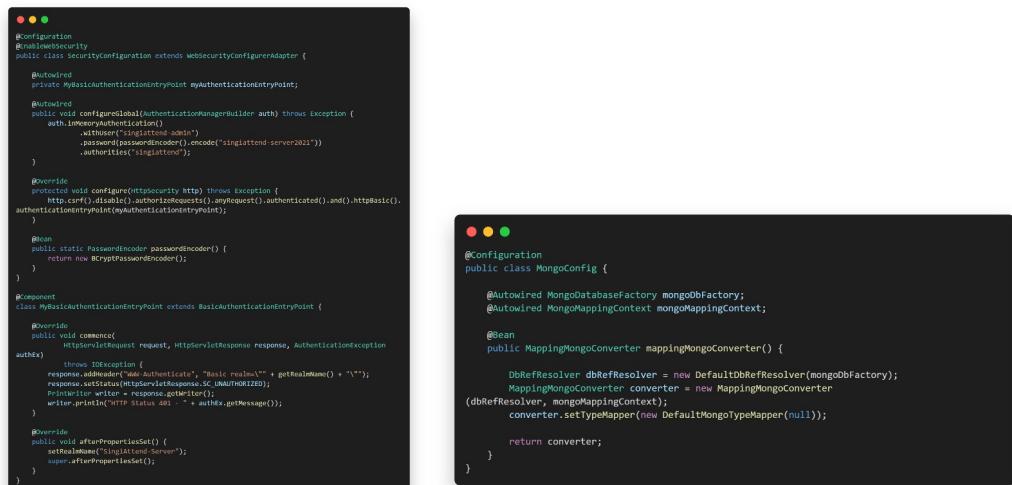
Server koji komunicira sa MongoDB bazom se sastoji od standardne strukture Spring frejmverka. Klase unutar glavnog Java paketa su grupisane na sledeći način (slika 24):

- **config** – Konfiguracione klase Mongo baze i bezbednosna podešavanja
- **controller** – Sadrži REST kontroler klasu za obradu HTTP zahteva
- **entity** – Sadrži klase koje predstavljaju modele tabela u Mongo bazi podataka
- **repository** – Sadrži interfejse koji nasleđuju **MongoRepository** interfejse za svaki entitet. Ti interfejsi služe da bi izvršavali sve upite i CRUD operacije.
- **service** – Sadrži servisnu klasu koja izvršava celokupnu logiku i služi kao posrednik između kontrolera i baze podataka, tj. repozitorijuma.

Pored navedene strukture postoji i MainApplication klasa koja sadrži glavni metod i prva se poziva prilikom pokretanja servera. Unutar *resources* foldera se nalazi fajl (*application.properties*) koji sadrži parametre za pristup, tj. ime host-a, port na kome se server izvršava, ime baze podataka i port na kome se komunicira sa MongoDB.



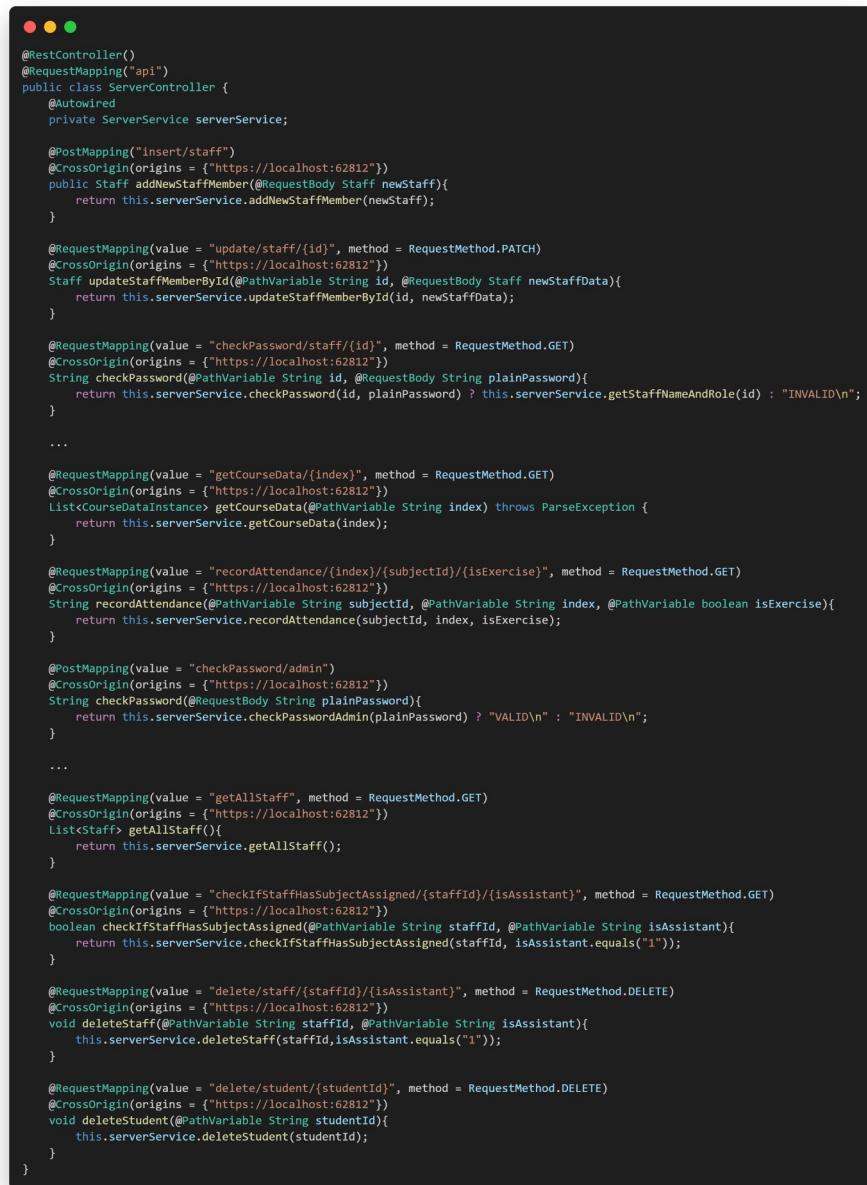
Slike 24 i 25: Prikaz strukture Spring servera, glavne klase i pristupnih parametara aplikacije i baze



Slike 26 i 27: Izgled konfiguracionih klasa, sigurnosna podešavanja i specijalnog MongoDB konvertera

Da bi se pristupilo serveru potrebno je poslati autentifikacione parametre, tj. korisničko ime i lozinku, kao vrednost *Authorization* parametra zaglavlja HTTP zahteva. Ovaj podatak se šalje enkodiran i koristi se bazična autorizacija. Kako je ovaj server planiran da se koristi samo u lokalnoj mreži, trenutno nema potrebe za naprednjim nivoom zaštite.

Na slici 28 vidimo deo koda REST kontrolera. Putanja kojom se pristupa metodama kontrolera je formatirana na sledeći način: **[https://localhost:62812/api/<putanja\\_metode>](https://localhost:62812/api/<putanja_metode>)**. Metode koje dodaju ili ažuriraju podatke kao dolazni parametar očekuju JSON objekat novih podataka (koji odgovara modelu entiteta u bazi), vraćaju celokupni objekat nakon izmene ili prazan u slučaju greške. Ukoliko metoda vraća podatke na osnovu ID-ja ili nekog drugog identifikacionog parametra, pr. kod provere lozinke studenta, taj parametar se unosi unutar putanje kao „*PathVariable*“. Svi metodi izvršavaju prosleđuju dobijene podatke servisu i od njega dobijaju odgovor, a potom isti vraćaju u JSON formatu ili PHP ili mobilnim aplikacijama. Dostupni HTTP metodi su: **GET, POST, PATCH i DELETE**.



```

@RestController()
@RequestMapping("api")
public class ServerController {
    @Autowired
    private ServerService serverService;

    @PostMapping("insert/staff")
    @CrossOrigin(origins = {"https://localhost:62812"})
    public Staff addNewStaffMember(@RequestBody Staff newStaff){
        return this.serverService.addNewStaffMember(newStaff);
    }

    @RequestMapping(value = "update/staff/{id}", method = RequestMethod.PATCH)
    @CrossOrigin(origins = {"https://localhost:62812"})
    Staff updateStaffMemberById(@PathVariable String id, @RequestBody Staff newStaffData){
        return this.serverService.updateStaffMemberById(id, newStaffData);
    }

    @RequestMapping(value = "checkPassword/staff/{id}", method = RequestMethod.GET)
    @CrossOrigin(origins = {"https://localhost:62812"})
    String checkPassword(@PathVariable String id, @RequestBody String plainPassword){
        return this.serverService.checkPassword(id, plainPassword) ? this.serverService.getStaffNameAndRole(id) : "INVALID\n";
    }

    ...

    @RequestMapping(value = "getCourseData/{index}", method = RequestMethod.GET)
    @CrossOrigin(origins = {"https://localhost:62812"})
    List<CourseDataInstance> getCourseData(@PathVariable String index) throws ParseException {
        return this.serverService.getCourseData(index);
    }

    @RequestMapping(value = "recordAttendance/{index}/{subjectId}/{isExercise}", method = RequestMethod.GET)
    @CrossOrigin(origins = {"https://localhost:62812"})
    String recordAttendance(@PathVariable String subjectId, @PathVariable String index, @PathVariable boolean isExercise){
        return this.serverService.recordAttendance(subjectId, index, isExercise);
    }

    @PostMapping(value = "checkPassword/admin")
    @CrossOrigin(origins = {"https://localhost:62812"})
    String checkPassword(@RequestBody String plainPassword){
        return this.serverService.checkPasswordAdmin(plainPassword) ? "VALID\n" : "INVALID\n";
    }

    ...

    @RequestMapping(value = "getAllStaff", method = RequestMethod.GET)
    @CrossOrigin(origins = {"https://localhost:62812"})
    List<Staff> getAllStaff(){
        return this.serverService.getAllStaff();
    }

    @RequestMapping(value = "checkIfStaffHasSubjectAssigned/{staffId}/{isAssistant}", method = RequestMethod.GET)
    @CrossOrigin(origins = {"https://localhost:62812"})
    boolean checkIfStaffHasSubjectAssigned(@PathVariable String staffId, @PathVariable String isAssistant){
        return this.serverService.checkIfStaffHasSubjectAssigned(staffId, isAssistant.equals("1"));
    }

    @RequestMapping(value = "delete/staff/{staffId}/{isAssistant}", method = RequestMethod.DELETE)
    @CrossOrigin(origins = {"https://localhost:62812"})
    void deleteStaff(@PathVariable String staffId, @PathVariable String isAssistant){
        this.serverService.deleteStaff(staffId, isAssistant.equals("1"));
    }

    @RequestMapping(value = "delete/student/{studentId}", method = RequestMethod.DELETE)
    @CrossOrigin(origins = {"https://localhost:62812"})
    void deleteStudent(@PathVariable String studentId){
        this.serverService.deleteStudent(studentId);
    }
}

```

Slika 28: Prikaz dela koda REST kontrolera

Da bi izvršavali CRUD (eng. Create, Read, Update, Delete) operacije nad tabelama u našoj bazi podataka, potrebno je da napišemo klase koje će oslikavati strukturu podataka, tj. atribute tih tabela. Svi atributi se mapiraju u odgovarajuće tipove podataka u Javi, a upotrebom `@Document` anotacije označavamo naziv kolekcije na koju se entitet odnosi. `@Field` anotacija označava tačni naziv kolone kojoj promenljiva pripada, tako da je njen naziv u klasi proizvoljan.

Ukoliko atribut predstavlja strani ključ ka drugom entitetu potrebno je označiti da je parametar tipa `objectId`.

Tabele takođe sadrže i podatke koji označavaju datum i vreme (pr. datum i vreme početka i završetka predavanja). Ti podaci zahtevaju upotrebu `@Temporal` anotacije, kao i specificiranje formata (u našem slučaju „`yyyy-MM-dd HH:mm:ss.SSS`“).

Na kraju predmeti, vežbe i predavanja zahtevaju upotrebu nizova koji će čuvati ID-jeve studenta koji ih prate, a u te svrhe korišćen je `ArrayList` tip, koji je mapiran kao niz karaktera (`@Type(type = "string-array")`).

Izgled ovih tabela, kao i primera podataka u bazi će biti prikazan u nastavku rada.

```
● ● ●
@data
@Document(collection = "Students")
public class Student {
    @Id
    private String id;
    @Field("name_surname")
    private String name_surname;
    @Field("index")
    private String index;
    @Field("password_hash")
    private String password_hash;
    @Field("email")
    private String email;
    @Field(targetType = FieldType.OBJECT_ID, value = "study_id")
    private String studyId;
    @Field("year")
    private String year;
    @Field("study")
    private ArrayList<Study> study;
}
```

```
● ● ●
@data
@Document(collection = "Exercises")
public class Exercise {
    @Id
    private String id;
    @Field("subject_id")
    private String subject_id;
    @Field("started_at")
    @Temporal(TemporalType.TIMESTAMP)
    @DateTimeFormat(style = "yyyy-MM-dd HH:mm:ss.SSS")
    private Date started_at;
    @Field("ended_at")
    @Temporal(TemporalType.TIMESTAMP)
    @DateTimeFormat(style = "yyyy-MM-dd HH:mm:ss.SSS")
    private Date ended_at;
    @Type(type = "string-array")
    @Field("attended_students")
    private ArrayList<String> attended_students;
}
```

```
● ● ●
@data
@Document(collection = "Subjects")
public class Subject {
    @Id
    private String id;
    @Field("title")
    private String title;
    @Field("title_english")
    private String title_english;
    @Field(targetType = FieldType.OBJECT_ID, value = "professor_id")
    private String professorId;
    @Field(targetType = FieldType.OBJECT_ID, value = "assistant_id")
    private String assistantId;
    @Field("last_lecture_at")
    @Temporal(TemporalType.TIMESTAMP)
    @DateTimeFormat(style = "yyyy-MM-dd HH:mm:ss.SSS")
    private Date lastLectureAt;
    @Field("last_exercise_at")
    @Temporal(TemporalType.TIMESTAMP)
    @DateTimeFormat(style = "yyyy-MM-dd HH:mm:ss.SSS")
    private Date lastExerciseAt;
    @Type(type = "string-array")
    @Field("enroled_students")
    private ArrayList<String> enroled_students;
    @Field("isInactive")
    private String isInactive;
    @Field("professor")
    private ArrayList<Staff> professor;
    @Field("assistant")
    private ArrayList<Staff> assistant;
}
```

Slike 29-31, odozgo na dole: Prikaz koda entiteta studenta, vežbe i predmeta

Svi potrebne metode koje koristimo za izvršavanje CRUD operacija, nad našim entitetima, nudi nam već pomenuuti Mongo repozitorijum. Njegov interfejs nasleđuju naši repozitorijumi za svaki entitet, a kao parametar prosleđujemo klasu entiteta i tip primarnog ključa (kako koristimo Mongo biće u pitanju tekstualni tip, tj. *String*).

Pored osnovnih metoda unutar našeg repozitorijuma možemo pisati i naše, a potom upotrebom `@Query` anotacije dodeliti im upite koji će se izvršiti nad našim tabelama.

Posebna mogućnost koju nam Mongo baza podataka nudi je pisanje agregacija (anotacija `@Aggregation`). Agregacije nam omogućavaju upotrebu naprednijih operacija za dobijanje željenih rezultata, poput korišćenja `$lookup` operacije da bi dobili podatke iz drugih tabela na osnovu ID-ja ili `$project` za izbor željenih kolona.

Ukoliko vraćamo podatke koji imaju drugaćiju strukturu od našeg entiteta, pr. vraćanje podataka o tekućem predavanju (metoda `getCourseDataByLectures`), potrebno je navesti kao povratnu vrednost posebnu klasu čija struktura odgovara rezultatu (`AggregationResults<naziv_klase>`). Takvi podaci će biti vraćeni kao JSON objekat čiji ključevi će odgovarati nazivima promenljive te klase.

```
public interface IStudentRepository extends MongoRepository<Student, String> {
    @Query(value="{'index': {$regex : ?0, $options: 'i'}}")
    Student getByIndex(String index);
    @Aggregation(pipeline={
        "{$lookup: {from:'Studies',localField:'study_id',foreignField:'_id',as:'study'}}",
        "{$match: { '_id': ObjectId(?0 ) }}"}
    )
    Student find(String id);
    @Aggregation(pipeline={
        "{$lookup: {from:'Studies',localField:'study_id',foreignField:'_id',as:'study'}}"})
    List<Student> findAll();
}
```

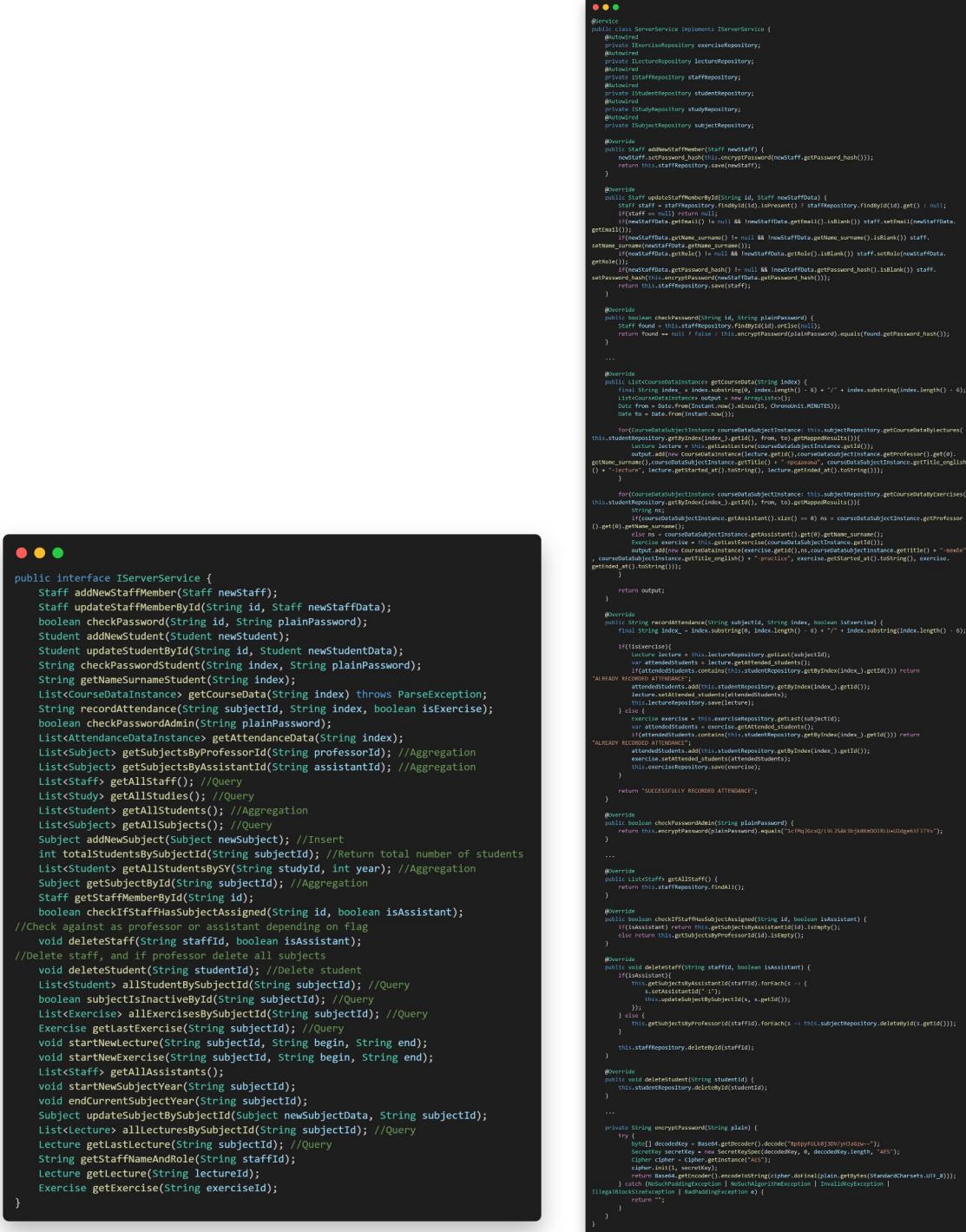
```
public interface ISubjectRepository extends MongoRepository<Subject, String> {
    @Aggregation(pipeline={
        "{$lookup: {from:'Staff',localField:'professor_id',foreignField:'_id',as:'professor'}}",
        "{$lookup: {from:'Staff',localField:'assistant_id',foreignField:'_id',as:'assistant'}}",
        "{$match: {enroled_students: {$eq: ?0}} }"}
    )
    AggregationResults<AttendanceSubobjectInstance> getSubobjectdata(String id);
    @Aggregation(pipeline={
        "{$match : { 'enroled_students' : { $eq: ?0 } } } \n",
        "{$match : { 'last_lecture_at' : { $gte: ?1, $lte: ?2 } } } \n",
        "{$lookup: {from:'Staff',localField:'professor_id',foreignField:'_id',as:'professor'}} \n",
        "{$lookup: {from:'Staff',localField:'assistant_id',foreignField:'_id',as:'assistant'}} \n",
        "{$project: {'professor_id':0,'assistant_id':0,'date':0,'enroled_students':0, '_class':0}}"}}
        AggregationResults<CourseDataSubjectInstance> getCourseDataByLectures(String studentId, Date from, Date to);
        @Aggregation(pipeline={
            "{$match : { 'enroled_students' : { $eq: ?0 } } } \n",
            "{$match : { 'last_exercise_at' : { $gte: ?1, $lte: ?2 } } } \n",
            "{$lookup: {from:'Staff',localField:'professor_id',foreignField:'_id',as:'professor'}} \n",
            "{$lookup: {from:'Staff',localField:'assistant_id',foreignField:'_id',as:'assistant'}} \n",
            "{$project: {'professor_id':0,'assistant_id':0,'date':0,'enroled_students':0, '_class':0}}"}}
            AggregationResults<CourseDataSubjectInstance> getCourseDataByExercises(String studentId, Date from, Date to);
            @Aggregation(pipeline={
                "{$lookup: {from:'Staff',localField:'professor_id',foreignField:'_id',as:'professor'}}",
                "{$lookup: {from:'Staff',localField:'assistant_id',foreignField:'_id',as:'assistant'}}"}})
                AggregationResults<Subject> findAllAggregation();
}
```

Slake 32 i 33, odozgo na dole: Prikaz koda repozitorijuma studenta i predmeta

Kao što će već ranije rečeno celokupna logika aplikacije je sadržana unutar servisne klase. Skup potpisa metoda, tj. interfejs servisa, je prikazan na slici 34, dok je deo implementacije nekolicine metoda prikazan na slici 35.

U našem slučaju zbog činjenice da nije prevelik broj metoda postoji samo jedna servisna klasa, dok u realnom i obimnijem projektu bi poželjno bilo korišćenje više servisa (u našem slučaju postoji mogućnost podele servisa po repozitorijumima).

Pored navedenih metoda postoji i privatna metoda koja šifrira lozinku upotrebom AES kriptografskog standarda.



```

public interface IServerService {
    Staff addNewStaffMember(Staff newStaff);
    Staff updateStaffMemberById(String id, Staff newStaffData);
    boolean checkPassword(String id, String plainPassword);
    Student addNewStudent(Student newStudent);
    Student updateStudentById(String id, Student newStudentData);
    String checkPasswordStudent(String index, String plainPassword);
    String getNamesurnameStudent(String index);
    List<CourseDataInstance> getCourseData(String index) throws ParseException;
    String recordAttendance(String subjectId, String index, boolean isExercise);
    boolean checkPasswordAdmin(String plainPassword);
    List<AttendanceDataInstance> getAttendanceData(String index);
    List<Subject> getSubjectsByProfessorId(String professorId); //Aggregation
    List<Subject> getSubjectsByAssistantId(String assistantId); //Aggregation
    List<Staff> getAllStaff(); //Query
    List<Study> getAllStudies(); //Query
    List<Student> getAllStudents(); //Aggregation
    List<Subject> getAllSubjects(); //Query
    Subject addNewSubject(Subject newSubject); //Insert
    int totalStudentsBySubjectId(String subjectId); //Return total number of students
    List<Student> getAllStudentsByStudyId(String studyId, int year); //Aggregation
    Subject getSubjectById(String subjectId); //Aggregation
    Staff getStaffMemberById(String id);
    boolean checkIfStaffHasSubjectAssigned(String id, boolean isAssistant);
    //Check against as professor or assistant depending on flag
    void deleteStaff(String staffId, boolean isAssistant);
    //Delete staff, and if professor delete all subjects
    void deleteStudent(String studentId); //Delete student
    List<Student> getAllStudentsBySubjectId(String subjectId); //Query
    boolean subjectIsInactiveById(String subjectId); //Query
    List<Exercise> allExercisesBySubjectId(String subjectId); //Query
    Exercise getLastExercise(String subjectId); //Query
    void startNewLecture(String subjectId, String begin, String end);
    void startNewExercise(String subjectId, String begin, String end);
    List<Staff> getAllAssistants();
    void startNewSubjectYear(String subjectId);
    void endCurrentSubjectYear(String subjectId);
    Subject updateSubjectBySubjectId(Subject newSubjectData, String subjectId);
    List<Lecture> allLecturesBySubjectId(String subjectId); //Query
    Lecture getLastLecture(String subjectId); //Query
    String getStaffNameAndRole(String staffId);
    Lecture getLecture(String lectureId);
    Exercise getExercise(String exerciseId);
}

public class ServerService implements IServerService {
    @Autowired
    private IExerciseRepository exerciseRepository;
    @Autowired
    private ILectureRepository lectureRepository;
    @Autowired
    private IStaffRepository staffRepository;
    @Autowired
    private IStudentRepository studentRepository;
    @Autowired
    private ISubjectRepository studyRepository;
    @Autowired
    private ISubjectRepository subjectRepository;

    @Override
    public Staff addNewStaffMember(Staff newStaff) {
        newStaff.setPassword(newStaff.getPassword().hashCode());
        return this.staffRepository.save(newStaff);
    }

    @Override
    public Staff updateStaffMemberById(String id, Staff newStaffData) {
        Staff staff = staffRepository.findById(id);
        if(staff == null) return null;
        if(newStaffData.getMail() != null && !newStaffData.getMail().isEmpty()) staff.setEmail(newStaffData.getEmail());
        if(newStaffData.getNamesurname() != null && !newStaffData.getNamesurname().isBlank())
            staff.setNamesurname(newStaffData.getNamesurname());
        if(newStaffData.getPassword() != null && !newStaffData.getPassword().isBlank())
            staff.setPassword(newStaffData.getPassword().hashCode());
        if(newStaffData.getPlainPassword() != null && !newStaffData.getPlainPassword().isBlank())
            staff.setPassword(this.encryptPassword(newStaffData.getPlainPassword().hashCode()));
        return this.staffRepository.save(staff);
    }

    @Override
    public Staff checkPassword(String id, String plainPassword) {
        Staff found = this.staffRepository.findById(id);
        if(found == null) return null;
        if(this.encryptPassword(found.getPassword().hashCode()) == plainPassword)
            return found;
        else return null;
    }

    ...
}

@Override
public List<CourseDataInstance> getCourseData(String index) {
    final String index_ = index.substring(0, index.length() - 6) + "/" + index.substring(index.length() - 6);
    List<CourseDataInstance> output = new ArrayList<>();
    Date from = Date.from(Instant.now().minus(15, ChronoUnit.MINUTES));
    Date to = Date.from(Instant.now());
    for(CourseDataInstance courseData : courseDataRepository.findAll()) {
        if(courseData.getFrom() != null && courseData.getTo() != null) {
            if(courseData.getFrom().isBefore(from) && courseData.getTo().isAfter(to)) continue;
            Lecture lecture = this.lectureRepository.getLecture(courseData.getLectureId());
            if(lecture != null && courseData.getFrom().isBefore(lecture.getCreatedAt()) && courseData.getTo().isAfter(lecture.getCreatedAt())) {
                output.add(courseData);
            }
        }
    }
    for(CourseDataSubjectInstance courseDataSubject : courseDataSubjectRepository.findAll()) {
        if(courseDataSubject.getFromIndex() != null && courseDataSubject.getToIndex() != null) {
            if(courseDataSubject.getFromIndex().isBefore(index_) && courseDataSubject.getToIndex().isAfter(index_)) {
                if(courseDataSubject.getSubject().getProfessor().getSurname() != null && courseDataSubject.getSubject().getProfessor().getSurname().equals(this.encryptPassword(courseDataSubject.getPlainSurname())))
                    output.add(courseDataSubject);
            }
        }
    }
    return output;
}

@Override
public void recordAttendance(String subjectId, String index, boolean isExercise) {
    final String index_ = index.substring(0, index.length() - 6) + "/" + index.substring(index.length() - 6);

    if(isExercise) {
        Lecture lecture = this.lectureRepository.getLecture(subjectId);
        var attendedStudents = lecture.getAttendedStudents();
        if(attendedStudents.size() <= 0) continue;
        if(subjectId.equals(lecture.getSubjectId())) {
            this.attendedStudents.add(this.studentRepository.getByIndex(index_, getId()));
            lecture.setAttendedStudents(attendedStudents);
            this.lectureRepository.save(lecture);
        } else {
            Exercise exercise = this.exerciseRepository.getExercise(subjectId);
            var attendedStudents = exercise.getAttendedStudents();
            if(attendedStudents.size() <= 0) continue;
            if(subjectId.equals(exercise.getSubjectId())) {
                this.attendedStudents.add(this.studentRepository.getByIndex(index_, getId()));
                exercise.setAttendedStudents(attendedStudents);
                this.exerciseRepository.save(exercise);
            }
        }
        return "SUCCESSFULLY RECORDED ATTENDANCE!";
    }
}

@Override
public boolean checkPasswordAdmin(String plainPassword) {
    return this.encryptPassword(plainPassword).equals("1f9e0c401d153a511a1d2e03ff171a");
}

...
}

@Override
public List<Staff> getAllStaff() {
    return this.staffRepository.findAll();
}

@Override
public boolean checkIfStaffHasSubjectAssigned(String id, boolean isAssistant) {
    if(isAssistant) {
        List<Subject> subjects = this.subjectRepository.findAll();
        for(Subject s : subjects) {
            if(s.getSubjectId().equals(id))
                return true;
        }
    } else {
        List<Subject> subjects = this.subjectRepository.findAll();
        for(Subject s : subjects) {
            if(s.getProfessorId().equals(id))
                return true;
        }
    }
    return false;
}

@Override
public void deleteStaff(String staffId, boolean isAssistant) {
    if(isAssistant) {
        List<Subject> subjects = this.subjectRepository.findAll();
        for(Subject s : subjects) {
            if(s.getSubjectId().equals(staffId))
                this.subjectRepository.deleteBySubjectId(s.getId());
        }
    } else {
        List<Subject> subjects = this.subjectRepository.findAll();
        for(Subject s : subjects) {
            if(s.getProfessorId().equals(staffId))
                this.subjectRepository.deleteBySubjectId(s.getId());
        }
    }
}

@Override
public void deleteStudent(String studentId) {
    this.studentRepository.deleteById(studentId);
}

...
}

private String encryptPassword(String plain) {
    try {
        byte[] decodedKey = Base64.getDecoder().decode("8pgyFUJX8jDv/vDjGw==");
        SecretKeySpec secretKey = new SecretKeySpec(decodedKey, 0, decodedKey.length, "AES");
        Cipher cipher = Cipher.getInstance("AES");
        cipher.init(Cipher.ENCRYPT_MODE, secretKey);
        String result = cipher.doFinal(plain.getBytes(StandardCharsets.UTF_8));
        return Base64.getEncoder().encodeToString(cipher.doFinal(plain.getBytes(StandardCharsets.UTF_8)));
    } catch (NoSuchAlgorithmException | NoSuchPaddingException | InvalidKeyException | IllegalBlockSizeException | BadPaddingException e) {
        return "";
    }
}

```

Slike 34 i 35: Kod interfejsa servisa i dela koda njegove implementacije

## **5. Izgled MongoDB baze podataka**

Kao što je već rečeno, svi podaci su smešteni unutar MongoDB baze podataka. Ona se sastoji od 6 kolekcija: *osoblje (Staff)*, *studenti (Students)*, *smerovi (Studies)*, *predmeti (Subjects)*, *predavanja (Lectures)* i *vežbe (Exercises)*.

Tipovi podataka unutar kolekcija su:

- **ObjectId** – polja primarnog i stranog ključa (`_id`)
  - **ISODate** – polja koja predstavljaju datum i vreme
  - **String** – sva polja koja sadrže karaktere, uključujući i boolean vrednosti
  - **Niz karaktera** – polja evidencije studenta koji pohađaju ili su prisustvovali predavanjima i vežbama (niz tekstualnih vrednosti id-jeva studenata)

Na slikama u nastavku su prikazani primeri podataka nekih kolekcija.

*Slike 36-38, s leva na desno: Prikaz dela podataka kolekcija predmeta, smerova i osoblja*

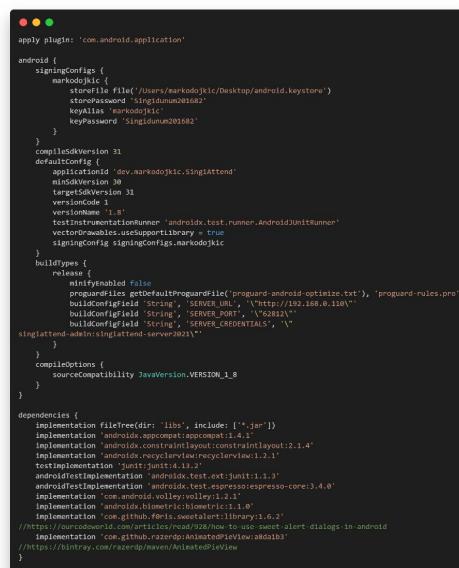
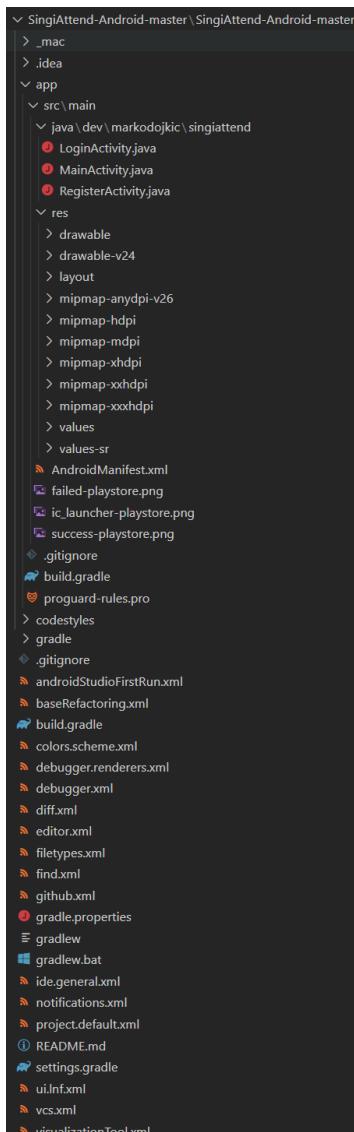
*Slika 39: Prikaz dela kolekcije evidencije predavanja (identično izgleda i evidencija vežbi)*

## 6. Izrada studentske aplikacije za Android operativni sistem

Studentska Android aplikacija je napravljena upotrebom **Android Studio** integrisanog okruženja. Standardna struktura koja je automatski generisana je prikazana na slici 40. Tu vidimo da postoje 3 klase koje odgovaraju aktivnostima, odnosno delovima aplikacije. Te aktivnosti su registracija, logovanje i glavna aktivnost, koja predstavlja prikaz i evidentiranje prisustva.

Kako bi se kreirala aplikacija korišćena je *build.gradle* skripta (slika 41), koja sadrži konfiguracione parametre, poput ID-ja, minimalne potrebne verzije android-a, tj. verzije SDK platforme, verzije aplikacije i parametara za pristup serveru. U ovoj skripti se takođe nalaze parametri i putanja za pristup **keystore** fajlu, koji služi za potpisivanje izvršnog APK fajla, kao i lista zavisnih biblioteka (eng. *dependecies*), koji se implementiraju u svrhe crtanja pita grafikona evidencije prisustva i prikaz *SweetAlert* iskačućeg prozora.

Drugi bitan fajl je manifest (*AndroidManifest.xml*, slika 42) koji sadrži potrebne dozvole za rad aplikacije (pristup internetu i čitanje stanja veze ka istom), nazine paketa koji sadrže klase aktivnosti i izabranu temu.



```
apply plugin: 'com.android.application'

android {
    signingConfigs {
        markodojcic {
            storeFile('/Users/markodojcic/Desktop/android.keystore')
            storePassword('singiangtend')
            keyAlias('markodojcic')
            keyPassword('Singiangtend201682')
        }
    }
    compileSdkVersion 31
    defaultConfig {
        applicationId 'dev.markodojcic.singiangtend'
        minSdkVersion 30
        targetSdkVersion 31
        versionCode 1
        versionName "1.0"
        testInstrumentationRunner 'androdkx.test.runner.AndroidJUnitRunner'
        vectorDrawables.useSupportLibrary = true
    }
    signingConfigs {
        markodojcic
    }
    buildTypes {
        release {
            minifyEnabled false
            proguardFiles getDefaultProguardFile("proguard-android-optimize.txt"), "proguard-rules.pro"
            buildConfigField "String", "SERVER_URL", "\"http://192.168.0.118\""
            buildConfigField "String", "SERVER_PORT", "\"62812\""
            buildConfigField "String", "SERVER_CREDENTIALS", "\"singiangtend-admin@singiangtend-server001\""
        }
    }
    compileOptions {
        sourceCompatibility JavaVersion.VERSION_1_8
    }
}

dependencies {
    implementation fileTree(dir: 'libs', include: ['*.jar'])
    implementation 'com.google.code.gson:gson:2.8.5'
    implementation 'androidx.constraintlayout:constraintlayout:2.1.4'
    implementation 'androidx.recyclerview:recyclerview:1.2.1'
    testImplementation 'junit:junit:4.13.2'
    androidTestImplementation 'androidx.test.ext:junit:1.1.3'
    androidTestImplementation 'androdkx.test.espresso:espresso-core:3.4.0'
    implementation 'com.android.volley:volley:1.2.5'
    implementation 'com.google.android.gms:play-services-base:18.0.0'
    implementation 'com.github.F0ris:sweetalert:library:3.6.2'
    //https://ourcodeworld.com/articles/read/924/how-to-use-sweet-alert-dialogs-in-android
    implementation 'com.github.Razored:SweetAlertView:1.0.0'
    //https://bitly.com/caser0qjneven/AnimatedSweetAlert
}
```

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="dev.markodojcic.singiangtend">
    <uses-permission android:name="android.permission.INTERNET" />
    <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
    <application
        android:allowBackup="false"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportRtl="true"
        android:theme="@style/Theme.AppCompat.DayNight.NoActionBar"
        android:usesClearTextTraffic="true">
        <activity
            android:name="dev.markodojcic.singiangtend.LoginActivity"
            android:theme="@style/AppTheme.TransparentPopUp" />
        <activity
            android:name="dev.markodojcic.singiangtend.RegisterActivity"
            android:screenOrientation="fullSensor"
            android:theme="@style/Theme.AppCompat.DayNight.NoActionBar" />
        <activity
            android:name="dev.markodojcic.singiangtend.MainActivity"
            android:exported="true"
            android:screenOrientation="fullSensor"
            android:theme="@style/Theme.AppCompat.DayNight.NoActionBar" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>

```

Slike 40-42, s leva na desno, odozgo na dole: Prikaz strukture studentske Android aplikacije, *build.gradle* skripte i manifesta (*AndroidManifest.xml*)

Kako je već rečeno ova aplikacija ima 3 aktivnosti, a za svaku je potrebno kreirati njihove korisničke interfejse. Informacije o strukturi korisničkih interfejsa se nalaze u .xml fajlovima koji su smešteni unutar foldera `/app/src/main/res/layout`.

Korisnički interfejs čine komponente koje su grupisane u željeni „*layout*“ (u našem slučaju **ConstraintLayout**), organizuje njihov raspored na ekranu telefona. Upotreba ovakvog načina grupisanja nam omogućava da tačno preciziramo udaljenost svake komponente međusobno i u odnosu na margine samog ekrana.

Pored informacija o poziciji svakog elementa ovi fajlovi sadrže i posebna svojstva svakog atributa (pr. njihova veličina, izvor teksta, boja, veličina i stil fonta, naziv funkcije koja se okida pritiskom tastera, itd.).

Na slici 46 je prikazana struktura komponente koja prikazuje evidenciju svih prisustva po predmetima (grafikon, prognozu ostvarenih broja bodova za prisustvo, podatke predmetu), kao i implementacija „više linijskog spinera“, koji se koristi prilikom izbora smera tokom registracije novog studenta.



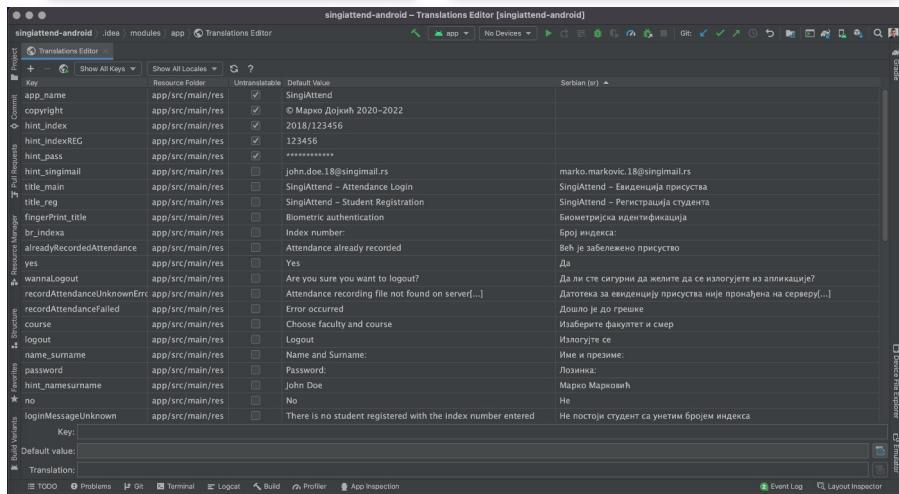
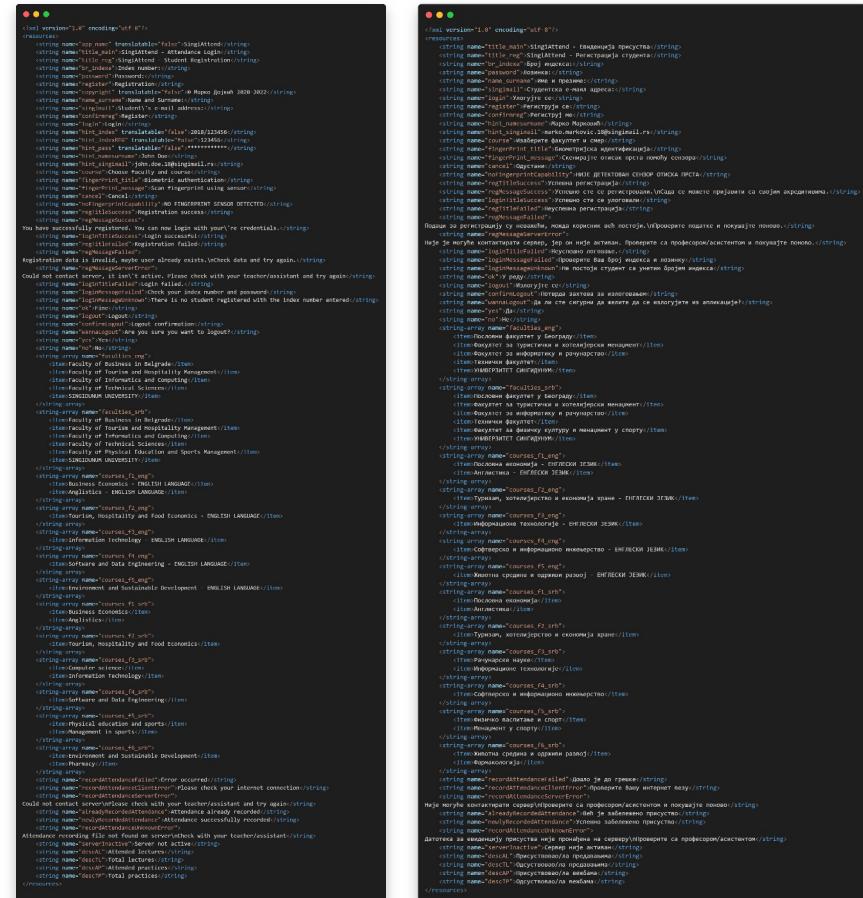
Slike 43-46, s leva na desno: Prikaz strukture korisničkih interfejsa registracije, logovanja, glavne aktivnosti, komponente koja prikazuje evidenciju svih prisustva i modifikovane, tj. „višelinjske“ verzije spinner-a

Kako bi se ostvarila više jezičnost aplikacije, potrebno je da kreiramo fajlove koji sadrže prevod svakog tekstualnog podatka.

Ti fajlovi se nalaze u posebnim folderima na putanji `/app/src/main/res/values` (podrazumevani tekst, ukoliko prevod nije pronađen) ili `/app/src/main/res/values-<prefiks_jezika_na_koji_prevodimo>`. U našem slučaju podrazumevani tekst je na engleskom jeziku, a postoji i prevod na srpski, koji je napisan na ciriličnom pismu. Nazivi ovih fajlova su striktno `strings.xml`.

Ukoliko se neki tekstovi ne prevode (poput naziva aplikacije ili autorskog prava), njih označimo kao **translatable="false"**.

Ove vrednosti možemo unositi direktno u .xml fajlove ili grafički, upotrebom editora prevoda (eng. **TranslationsEditor**), koji je prikazan na slici 49.



Slike 47-49, s leva na desno, odgozo na dole: Prikaz xml fajlova prevoda i TranslationsEditor-a

Kada korisnik prvobitno pristupi aplikaciji pokreće se glavna aktivnost, koja proverava da li je student već ulogovan u memoriji aplikacije. Ukoliko jeste izvršava se kod za dobijanje potrebnih informacija o evidenciji prisustva, kojeg ćemo detaljnije objasniti u nastavku rada, a ukoliko nije pokreće se aktivnost logovanja, čiji korisnički interfejs prikazujemo na sredini ekrana.

Kod koji pokreće ovu aktivnost je prikazan na slici 50. Tu vidimo da veličina prozora te aktivnosti zauzima 95% širine i 35% visine ekrana.

Pored tog koda vidimo i upotrebu filtera za ograničenje dužine unosa broja indeksa (koji odgovara standardu univerziteta). Kako smo polje prethodno označili da može da koristi samo brojčane karaktere, potrebno je automatski dodati znak /, koji razdvaja godinu upisa od ostatka broja indeksa. Primer ispravno unetog broja indeksa je **2018/201682**.

Na slici 51 vidimo kod koji proverava ispravnost unete lozinke. On funkcioniše unutar niti koja šalje HTTP zahtev serveru i prosleđuje unetu lozinku, a kao odgovor očekuje ili tekst **VALID** (ukoliko je lozinka ispravna), **INVALID** (ukoliko lozinka nije ispravna) ili **UNKNOWN** (ukoliko je unet nepostojeći broj indeksa). U bilo kom slučaju se pokazuje odgovarajući „SweetAlert“ iskačući prozor sa porukom.

Ukoliko je uspešno logovanje, tj. ispravno su uneti broj indeksa i lozinka, ova aktivnost se završava korišćenjem metode *finish()*.

```

public void editText(EditText brIndekskaTxt) {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_login);
        WindowMetrics windowMetrics = getWindowManager().getCurrentWindowMetrics();
        getWindow().setLayout((int)(windowMetrics.getBounds().width()*0.95), (int)(windowMetrics.getBounds().height()*0.3));
        WindowManager.LayoutParams params = getWindow().getAttributes();
        params.gravity = Gravity.CENTER;
        params.x = 0;
        params.y = 0;
        getWindow().setAttributes(params);
    }
}

brIndekskaTxt = findViewById(R.id.index_txt);
brIndekskaTxt.addTextChangedListener(new InputFilter() {
    @Override
    public CharSequence filter(CharSequence source, int start, int end, int count, int after) {
        if(brIndekskaTxt.getText().length() >= 5){
            if(source.toString().matches("[0-9]{2}[.][0-9]{3}")) {
                if(source.toString().length() == 5) {
                    if(source.toString().charAt(4) == '/') {
                        String[] arr = source.toString().split("/");
                        if(arr[0].length() == 4) {
                            brIndekskaTxt.setSelection(4);
                            brIndekskaTxt.setText(String.format("%s/", brIndekskaTxt.getText()));
                            brIndekskaTxt.setSelection(5);
                        }
                    }
                }
            }
        }
        return false;
    }
});

@Override
public void onTextChanged(CharSequence s, int start, int before, int count) {
}

@Override
public void afterTextChanged(Editable s) {
}
}

```

Slike 50 i 51, s leva na desno: Kod za inicijalizaciju aktivnosti za logovanje studenta i kod za proveru ispravnosti unete lozinke

Prilikom registracije studenta potrebno je uneti adresu elektronske pošte studenta, koja na Univerzitetu Singidunum ima sledeći format: **<ime>.< prezime>.<poslednje\_dve\_cifre\_godine\_upisa><dodata\_cifra\_ukoliko\_se\_poklop\_e\_dva\_studenta\_sa\_istim\_imenom\_i\_prezimenom\_u\_jednoj\_godini>@singimail.rs**.

Kako bi olakšali i osigurali ispravan unos koristimo ugrađenu funkciju **onTextChanged** (slika 52), koja prilikom svake promene polja za unos adrese elektronske pošte proverava da li je unet ispravan format prefiksa (dela pre @ znaka) upotrebom regularnog izraza. Ukoliko ova provera prođe unos se dopunjuje tekstom „**@singimail.rs**“.

```

@Override
public void onTextChanged(CharSequence s, int start, int before, int count) {
    if(studentskaEmail.getText().toString().matches("^[a-z]+\\.[a-z]+\\.[0-9]{2,3}@$")){
        studentskaEmail.setText(String.format("%s@", studentskaEmail.getText()));
        subSequence(0, studentskaEmail.getText().toString().indexOf("@") + 1, "singimail.rs"));
        studentskaEmail.setSelection(studentskaEmail.getText().toString().indexOf("@"));
    }
}

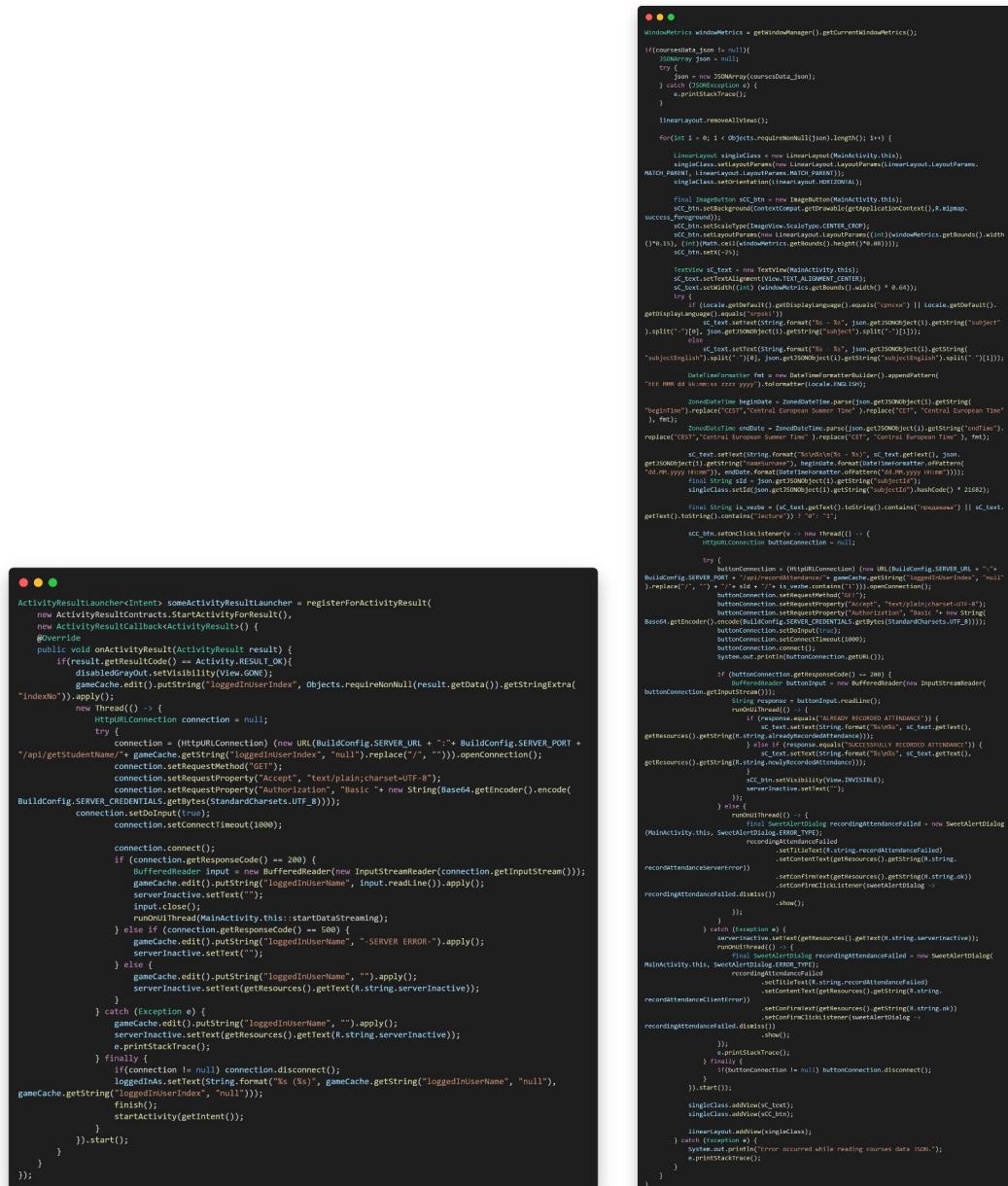
```

Slika 52: Kod za proveru ispravnosti unete adrese elektronske pošte (format Univerziteta Singidunum)

Nakon što se završi aktivnost logovanja studenta, broj indeksa se unosi u keš memoriju aplikacije i tu ostaje dokle god se student ne izloguje.

Potom u novoj niti započinjemo komunikaciju sa serverom u cilju dobijanja imena i prezimena studenta (kod sa slike 53). Svaka komunikacija sa serverom se odvija unutar **nove niti** kako ne bi blokirali **glavnu**, tj. nit koja obrađuje prikaz celokupnog grafičkog korisničkog interfejsa. Kako je ova komunikacija prva koja će se uvek izvršiti prilikom pokretanja aplikacije (ukoliko je student ulogovan), možemo da proverimo da li je server aktivan, tj. komunikacija uspešna, a ukoliko utvrdimo da nije popunjavamo polje koje pokazuje da server nije aktivan.

Nakon dobijanja imena i prezimena studenta, sledeći korak je dobijanje liste trenutno aktivnih predavanja, odnosno vežbi (kod sa slike 54). Za svaki dobijeni JSON objekat dodajemo prikaz u listu, tj. vertikalno orijentisanu komponentu linearног rasporeda, koja se nalazi unutar grupe „**ScrollView**“ (koja nam omogućava vertikalno skrolovanje liste).



Slike 53 i 54, s leva na desno: Prikaz kodova za dobijanje imena i prezimena studenta, kao i njegovih trenutno aktivnih predavanja, odnosno vežbi

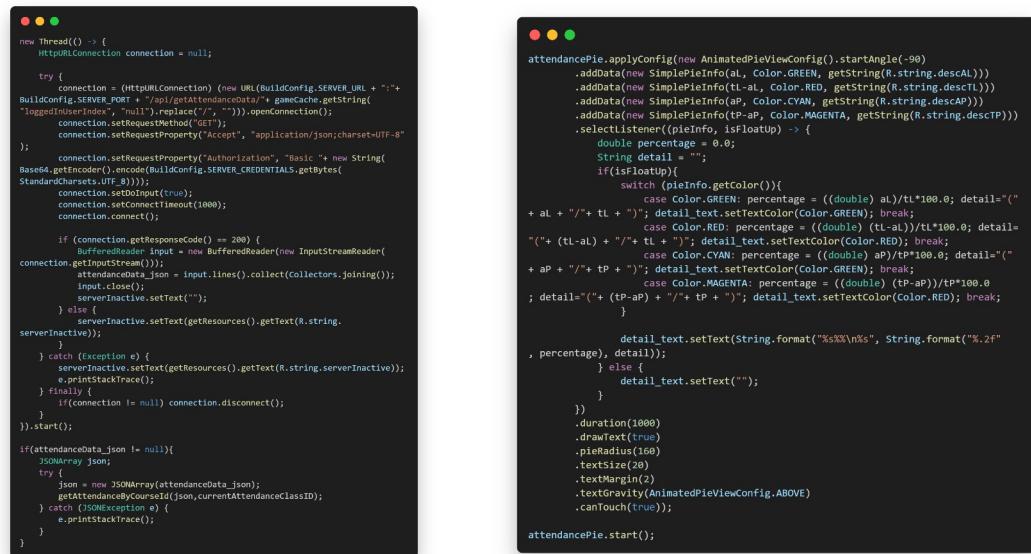
Na kraju nakon dobijanja trenutno aktivnih predavanja i vežbi, potrebno je preuzeti listu prisustva na svim predmetima kojih student prati. Podaci za svaki predmet su smešteni u poseban JSON objekat, a potom se ti objekti smeštaju u JSON niz. Inicijalno se prikazuje prvi objekat (čiji je indeks u nizu **0**), a student posredstvom tastera na ekranu se može kretati kroz niz, odnosno videti evidenciju prisustva na različitim predmetima.

Za prikaz pita grafikona evidencije prisustva koristimo „**AnimatedPieView**“ API autora „**razerdp**“. Podatke dodeljujemo grafikonu instanciranjem *SimplePieInfo* objekata upotrebom konstruktora. Konstruktor kao parametre uzima brojčanu vrednost, boju i naziv podatka. U našem slučaju postoje 4 vrednosti:

1. Broj predavanja na kojima je student bio prisutan – zelena boja
2. Broj predavanja na kojima je student nije bio prisutan – crvena boja
3. Broj vežbi na kojima je student bio prisutan – cijan boja
4. Broj vežbi na kojima je student nije bio prisutan – magenta boja

Takođe za svaku od ovih vrednosti, pritiskom na deo grafikona u centru se pojavljuje tekst koji ove vrednosti pokazuje u brojčanom i procentualnom obliku (pr. **50.00% (5/10)**).

Izvršavanje funkcija koji preuzimaju podatke o trenutno aktivnim predavanjima, odnosno vežbama, kao i evidenciju prisustva se izvršavaju automatski u intervalu od 10 sekundi.



```

new Thread() -> {
    HttpURLConnection connection = null;
    try {
        connection = (HttpURLConnection) (new URL(BuildConfig.SERVER_URL + ":" +
BuildConfig.SERVER_PORT + "/api/getAttendanceData/" + gameCache.getString(
"loggedInUserIndex", "null").replace("/", ""))).openConnection();
        connection.setRequestMethod("GET");
        connection.setRequestProperty("Accept", "application/json;charset=UTF-8");
    } catch (Exception e) {
        serverInactive.setText(getResources().getText(R.string.
serverInactive));
        e.printStackTrace();
    } finally {
        if(connection != null) connection.disconnect();
    }
}.start();

if(attendanceData_json != null){
    JSONArray json;
    try {
        json = new JSONArray(attendanceData_json);
        getAttendanceByCourseId(json,currentAttendanceClassID);
    } catch (JSONException e) {
        e.printStackTrace();
    }
}
}

attendancePie.applyConfig(new AnimatedPieViewConfig().startAngle(-90)
    .addData(new SimplePieInfo(aL, Color.GREEN, getString(R.string.descAL)))
    .addData(new SimplePieInfo(tL-aL, Color.RED, getString(R.string.descTL)))
    .addData(new SimplePieInfo(aP, Color.CYAN, getString(R.string.descAP)))
    .addData(new SimplePieInfo(tP-aP, Color.MAGENTA, getString(R.string.descTP)))
    .selectListener((pieInfo, isFloatUp) -> {
        double percentage = 0.0;
        String detail = "";
        if(isFloatUp){
            switch (pieInfo.getColor()){
                case Color.GREEN: percentage = ((double) aL)/tL*100.0; detail=(
                    +(aL + "/" + tL + ")); detail_text.setTextColor(Color.GREEN); break;
                case Color.RED: percentage = ((double) (tL-aL))/tL*100.0; detail=(
                    +(tL-aL) + "/" + tL + ")); detail_text.setTextColor(Color.RED); break;
                case Color.CYAN: percentage = ((double) aP)/tP*100.0; detail=(
                    +aP + "/" + tP + ")); detail_text.setTextColor(Color.GREEN); break;
                case Color.MAGENTA: percentage = ((double) (tP-aP))/tP*100.0;
                    detail=(+(tP-aP) + "/" + tP + ")); detail_text.setTextColor(Color.RED); break;
            }
            detail_text.setText(String.format("%s%%\n%s", String.format("%.2f",
                percentage), detail));
        } else {
            detail_text.setText("");
        }
    })
    .duration(1000)
    .drawText(true)
    .pieRadius(100)
    .textSize(20)
    .textMargin(2)
    .textGravity(AnimatedPieViewConfig.ABOVE)
    .canTouch(true);
}

attendancePie.start();

```

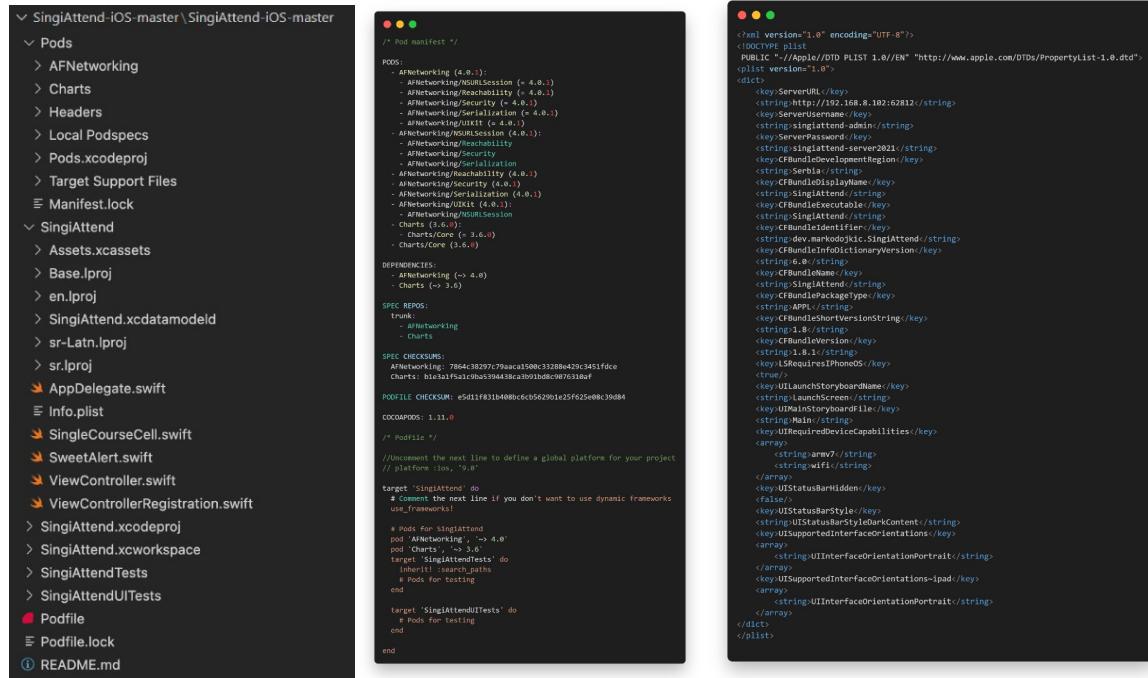
Slike 55 i 56, s leva na desno: Prikaz kodova za dobijanje evidencija prisustva po predmetima i generisanje pita grafikona na osnovu tih podataka

## 7. Izrada studentske aplikacije za iOS operativni sistem

Studentska iOS aplikacija je napravljena upotrebom xCode integrisanog okruženja. Standardnu strukturu vidimo na slici 57. Vidimo da se ona sadrži folder u kome su smeštene biblioteke od kojih aplikacija zavisi, poput **AFNetworking** (biblioteka koja omogućava umrežavanje iOS, macOS, watchOS, i tvOS aplikacija) i **Charts** (crtanje grafikona).

„**CocoaPods**“ menadžer zavisnih biblioteka koji funkcioniše na aplikacionom nivou, a njega koriste programski jezici koji koriste Objective-C kao runtime biblioteku (pr. *Objective-C*, *Swift*, *RubyMotion*, itd.). Listu biblioteka kojih želimo da koristimo navodimo unutar „PodFile“ fajla (slika 58) u formatu „**pod <ime\_biblioteke>**‘-> **<željena\_verzija>**“. Postoji mogućnost izbora biblioteka koje se koriste samo prilikom testiranja. Nakon što unesemo sve željene biblioteke potrebno je u direktorijumu gde je smešten „PodFile“, upotrebom terminala, izvršimo komandu „**pod install**“, kako bi preuzele biblioteke u Pods direktorijum.

Kao što Android sistem sadrži konfiguracione parametre unutar *build.gradle* fajla, iOS koristi *info.plist* (slika 59). Unutar njega smo takođe specificirali pristupne parametre i URL putanju ka Spring serveru, a takođe smo naveli naziv aplikacije, orijentaciju uređaja, kao i zahtevane mogućnosti koje uređaj treba da ima (armv7+ arhitekturu i WiFi modul).



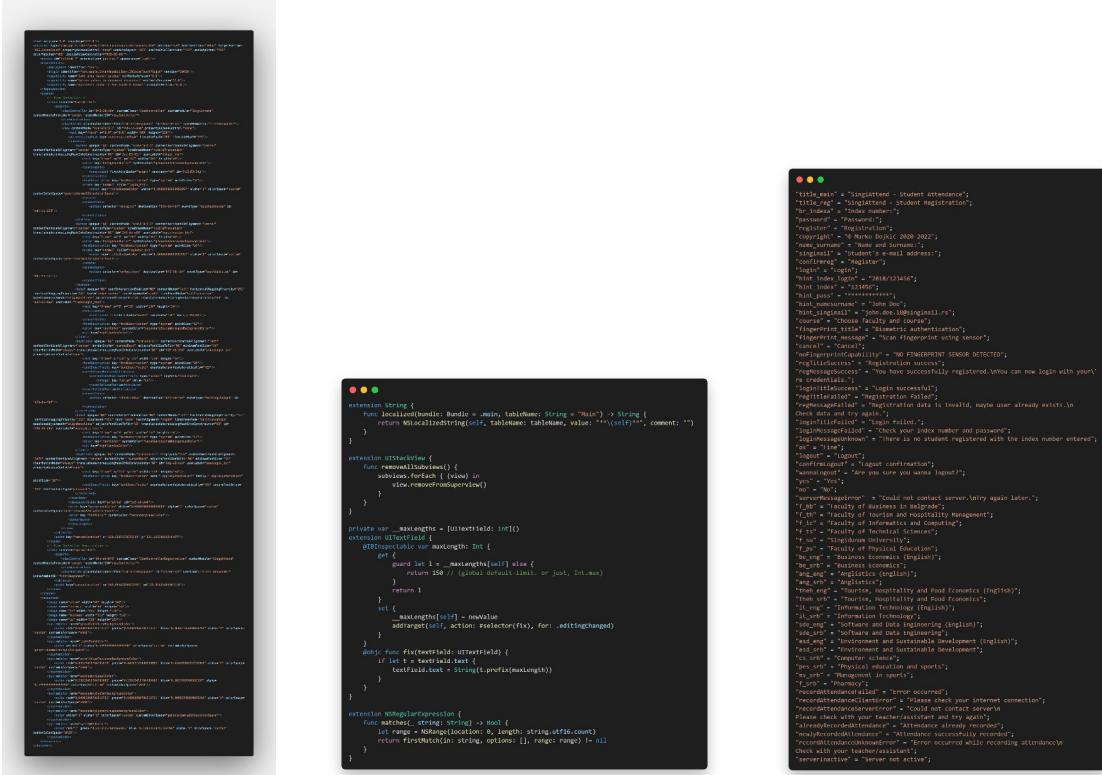
Slike 57-59, s leva na desno: Prikaz strukture iOS aplikacije, PodManifest-a i PodFile-a i info.plist fajla

Za prikaz i navigaciju unutar iOS aplikacije koristi se posebno svojstvo xCode okruženja koje se zove „storyboarding“, u našem slučaju izgledi logovanja, registracije, evidentiranja i prikaza prisustva se nalaze u okviru **main.storyboard** fajla (deo fajla je prikazan na slici 60). Unutar njega su ovi izgledi podeljeni unutar više „**<scene>...</scene>**“ tagova.

Naziv .swift fajlova koji sadrže kod koji omogućava funkcionisanje korisničkog interfejsa je naveden u okviru *scene* taga (jedan fajl može da deli više scena). Upotreboom anotacija „@IBOutlet“ (oznaka da je promenljiva vezana za komponentu korisničkog interfejsa) i „@IBAction“ (oznaka da se funkcija poziva pritiskom na određeno dugme) ostvarujemo komunikaciju između korisničkog interfejsa i koda.

Swift programski jezik nam omogućava kreiranje ekstenzija (slika 61), odnosno pisanje novih funkcionalnosti nad postojećim klasama, strukturama, enumeracijama ili protokolnim tipovima podataka. U našem slučaju mi smo String tipu podataka dodali funkciju **localized** koja nam olakšava lokalizaciju teksta (koji će vratiti rezultat NSLocalizedString metode, a naziv ključa vrednosti odgovaraće nazivu našeg string-a), primer upotrebe "*loginTitleSuccess*".**localized()** će vratiti lokalizovanu poruku da je uspešno logovanje ili smo dodali **UIStackView** klasi metodu koja uklanja sve pod poglede iz grupe.

Na slici 62 vidimo prikaz prevoda, čiji je format “**<ključ>**” = “**<vrednost>**” na engleski jezik, na identičan način su napisani prevodi na srpski jezik (latinično i cirilično pismo). Tekst ovih prevoda se nalazi unutar *Main.strings* fajlova koji se nalaze na putanji */SingiAttend/<prefiks\_jezika\_na\_koji\_prevodimo>.lproj*.



*Slike 60-62, s leva na desno: Deo koda glavnog storyboard-a, koda dodatih ekstenzija i teksta prevoda na engleski jezik*

Nakon pokretanja aplikacije korisnik dobija prozor na kome ima mogućnost da unese broj indeksa i lozinku, da bi se ulogovao ili da otvori prozor za registraciju (upotrebom metode **performSegue**, koja kao parametar uzima identifikator *ViewController-a* koji sadrži komponente registracione forme).

Ukoliko korisnik izabere da se uloguje pozvaće se funkcija sa slike 64 i koja šalje HTTP zahtev serveru i prosleđuje unetu lozinku (ovaj deo funkcioniše na identičan način kao u slučaju Android aplikacije), a nakon dobijanja odgovora pokazuje odgovarajući „SweetAlert“ iskačući prozor sa porukom (kod za ovaj prikaz se mora izvršiti kao asinhron unutar glavne niti, a to nam omogućava upotreba sintakse **DispatchQueue.main.async** {}).

```
#login func login(_ sender: UIButton) {
    let loginTxt = loginTxt.replaceOccurrences(of: ":", with: "")!, (passloginTxt.text),
    dataString: utf8!)!, completionHandler: {
        response, error in
        if let message = response {
            DispatchQueue.main.async {
                //This method must be called on the main thread because it may not be performed from a background thread after
                //it has been removed from the main thread
                if(message == "INVALIDID") {
                    AlertService.showAlert("Login failed", localizedTitle: "Login Failed", subtitle: "Please enter a valid ID", style: AlertStyle.error, buttonTitle: "OK").localized(),
                    buttonColor: UIColor.blue)
                } else if(message == "INVALIDP") {
                    AlertService.showAlert("Login failed", localizedTitle: "Login Failed", subtitle: "Please enter a valid Password", style: AlertStyle.error, buttonTitle: "OK").localized(),
                    buttonColor: UIColor.blue)
                } else if(message == "NOINPUT") {
                    AlertService.showAlert("Login failed", localizedTitle: "Login Failed", subtitle: "Please enter both ID and Password", style: AlertStyle.error, buttonTitle: "OK").localized(),
                    buttonColor: UIColor.blue)
                } else if(message == "NOLOGIN") {
                    AlertService.showAlert("Login failed", localizedTitle: "Login Failed", subtitle: "Please enter a valid ID", style: AlertStyle.error, buttonTitle: "OK").localized(),
                    buttonColor: UIColor.blue)
                } else if(message == "NOPASSWORD") {
                    AlertService.showAlert("Login failed", localizedTitle: "Login Failed", subtitle: "Please enter a valid Password", style: AlertStyle.error, buttonTitle: "OK").localized(),
                    buttonColor: UIColor.blue)
                } else if(message == "NOTLOGGEDIN") {
                    AlertService.showAlert("Login failed", localizedTitle: "Login Failed", subtitle: "Please enter a valid ID", style: AlertStyle.error, buttonTitle: "OK").localized(),
                    buttonColor: UIColor.blue)
                } else if(message == "NOTLOGGEDOUT") {
                    AlertService.showAlert("Login failed", localizedTitle: "Login Failed", subtitle: "Please enter a valid Password", style: AlertStyle.error, buttonTitle: "OK").localized(),
                    buttonColor: UIColor.blue)
                } else if(message == "OK") {
                    AlertService.showAlert("Success", localizedTitle: "Success", subtitle: "Login successful", style: AlertStyle.success, buttonTitle: "OK").localized(),
                    buttonColor: UIColor.green)
                }
            }
        }
    }
}

@IBAction func onRegister(_ sender: UIButton) {
    performSegue(withIdentifier: "toRegister", sender: sender)
}
```

Slike 63 i 64, s leva na desno: Prikaz koda za izbor opcije logovanja ili registracije i proveru unete lozinke

Na slici 65 vidimo inicijalizaciju promenljivih koje koristimo prilikom registracije studenata. Komponente korisničkog interfejsa su označene kao **weak var** promenljive odgovarajućeg tipa (labele, tekstualna polja, tasteri, liste za izbor vrednosti – *UIPickerView*).

Da bi omogućili izbor fakulteta i smera koristimo *UIPickerView* koji nam omogućava izbor jedne vrednosti nalik dropdown listi u web aplikacijama, a podatke smeštamo unutar nizova **faculties**, **courses** i **f\_courses**, koji se dinamički popunjavaju pomoću funkcija prikazanih na slici 66.

Na kraju da bi poslali JSON objekat kreirali smo strukturu **Student** i označili je tipom **Codable**, koji omogućava da se objekat konvertuje iz Swift u JSON objekat i obrnuto.

```
    #!Boutlet weak var titleReg_text: UITextField!
    #!Boutlet weak var nameSurname_text: UILabel!
    #!Boutlet weak var nameSurname_Text: UITextField!
    #!Boutlet weak var weakIndexReg_text: UITextField!
    #!Boutlet weak var weakIndexYear_pv: UIPickerView!
    #!Boutlet weak var singEmail_text: UITextField!
    #!Boutlet weak var singEmail_Text: UITextField!
    #!Boutlet weak var passEmail_text: UITextField!
    #!Boutlet weak var reg_email_RegText: UITextField!
    #!Boutlet weak var facultyCs_Text: UILabel!
    #!Boutlet weak var copyright_text: UILabel!
    #!Boutlet weak var register_BTN: UIButton!
    #!Boutlet weak var facultys_pv: UIPickerView!
    #!Boutlet weak var courses_pv: UIPickerView!
    #!Boutlet weak var eng_BTN: UIButton!
    #!Boutlet weak var srn_BTN: UIButton!

    let localStorage = UserDefaults.standard
    var faculties = [""]
    var courses: Array<String:Int> = []
    var f_courses: [String:Int] = []
    let i_name = Array<String>(["2006...2099"])
    var selectedCourseString = ""
    var reg_email = ""
    var reg_email_RegText = ""
    struct Student: Codable {
        var nameSurname: String
        var index: String
        var passwordHash: String
        var email: String
        var studyId: String
        var year: String
    }
}
```

*Slike 65 i 66, s leva na desno: Prikaz koda inicializacije promenljivih registracije i UIPickerView-a*

Nakon uspešno logovanja, i u slučaju pokretanja aplikacije kada je student već ulogovan, preuzimamo podatke o trenutno aktivnim predavanjima i vežbama studenta (kod sa slike 67). Svaki pojedinačni podatak se prikazuje unutar jedne ćelije tabele, koja je kreirana kodom sa slike 68, a čija struktura se nalazi na slici 69.

Struktura jedne ćelije se sastoji od tekstualnog polja koje sadrži podatke o predmetu (naziv predmeta, ime i prezime profesora ili asistenta), referencu ka tasteru za evidentiranje prisustva, kao i kod funkcije koja će serveru poslati HTTP zahtev za evidentiranje prisustva i prikazati odgovarajuću poruku nakon dobijanja odgovora o uspešnosti evidentiranja.

Kao što je slučaj i kod Android aplikacije, vreme početka i završetka predavanja, odnosno vežbi je prikazano u formatu „**dd.MM.yyyy HH:mm**“ (pr. *11.05.2022 12:15*).

*Slike 67-69, s leva na desno: Prikaz koda za dobijanje trenutno aktivnih predavanja, odnosno vežbi studenta i smeštanje tih podataka u ćeliju table, kao i koda za prikaz te ćelije i funkcije za evidentiranje prisustva*

Nakon dobijanja gore navedenih podataka, preostaje samo dobijanje liste prisustva na svim predmetima kojih student prati, a upotrebom **Chart** biblioteke instanciramo pita grafikon koji prikazuje evidenciju prisustva na identičan način kao **AnimatedPieView**.

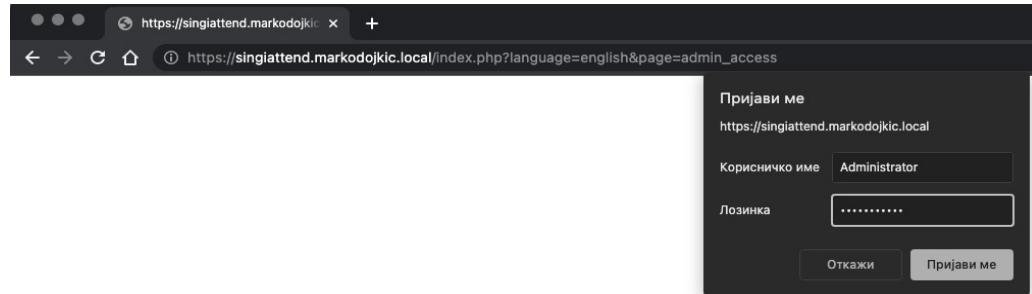
*Slike 70 i 71, s leva na desno: Prikaz koda za dobijanje evidencije svih prisustva studenta i crtanje grafikona*

## 8. Demonstracija rada aplikacije

### 9.1. Web aplikacija

Inicijalni prikaz web aplikacije predstavlja stranicu za logovanje profesora, odnosno asistenta. Na slikama 73 i 74 je prikazan izgled nakon ne validnog popunjavanja forme, tj. neispravnog unosa „captcha“ teksta i lozinke respektivno.

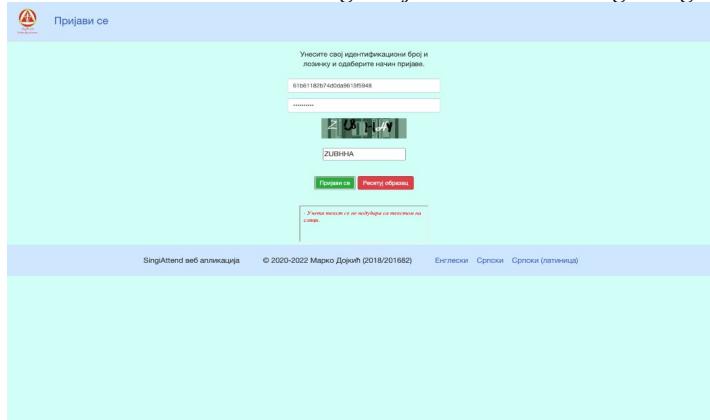
Logovanje administratora (slika 72) se odvija na posebnoj stranici *admin\_access*, unosom korisničkog imena (**Administrator**) i lozinke. U slučaju da je unos neuspešan pokazuje se stranica greške 404 (prikazuje samo poruku da stranica nije pronađena).



Prijavite se  
https://singiattend.markodokic.local

Korisnicko ime: Administrator  
Lozinka: .....  
Prijava me

Slika 72: Prikaz stranice za logovanje administratorskog naloga



Slike 73 i 74, odozgo na dole: Prikaz ispisa grešaka prilikom logovanja profesora, pogrešan „captcha“ tekst i lozinka

Pre nego što bi profesori, a i asistenti, mogli da vode evidenciju potrebno je dodati predmete u sistem. Forma za dodavanje predmeta je dostupna profesorima na stranici *add\_new\_subject*.

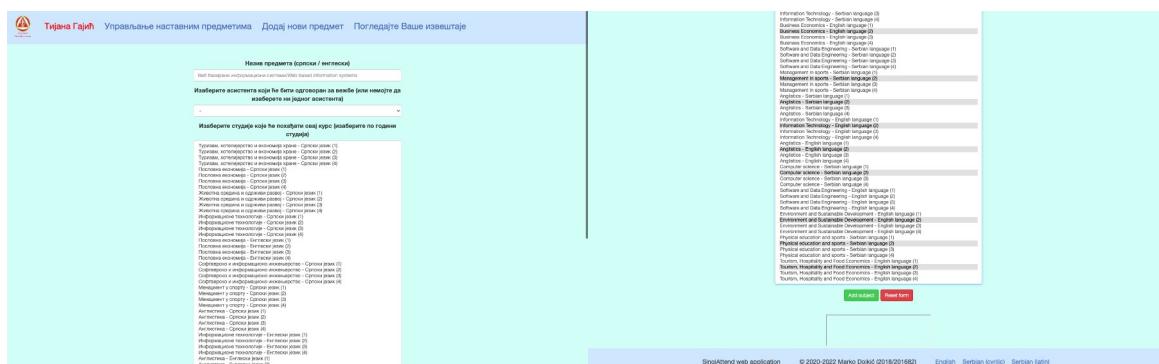
Forma se sastoji od:

- Tekstualnog polja – za unos naziva predmeta na srpskom i engleskom jeziku (u slučaju ne validnog unosa prikazuje se greška - slika 75)
- Padajuće liste (eng. *dropdown*) – koja služi za izbor asistenta ili – ukoliko on trenutno nije dodeljen (prikazana na slici 76)
- Liste višestrukog izbora (eng. *multiline select*) – koja služi za izbor smerova koji slušaju izabrani predmet (prikazana na slikama 77 i 78)

Nakon uspešnog popunjavanja forme predmet se kreira u bazi i onda postaje dostupan profesoru i izabranom asistentu. Da bi se omogućila evidencija prisustva, potrebno je da profesor započne novu školsku godinu, tj. omogući evidentiranje i izbriše sve prethodne evidencije prisustva na predavanjima i vežbama tog predmeta.

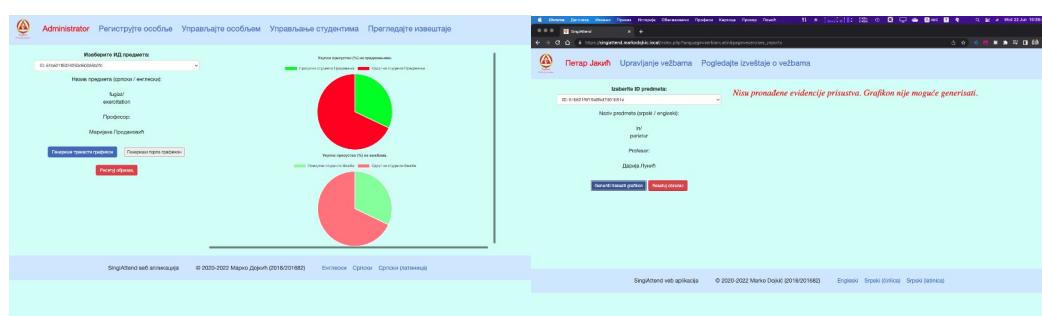


Slike 75 i 76, s leva na desno: Prikaz greške zbog ne validnog unosa naziva predmeta i liste asistenata



Slike 77 i 78, s leva na desno: Prikaz izgleda forme dodavanja predmeta, odnosno liste smerova i godina

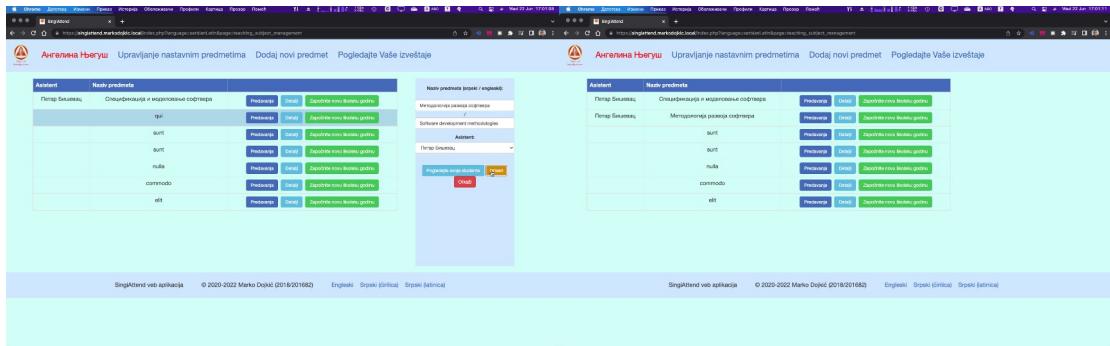
Profesori, asistenti i administrator mogu videti grafikone koji prikazuju ukupno procentualno prisustvo studenata po predmetima (profesori – **predavanja i vežbi svojih predmeta**, asistenti – **vežbi na kojima su oni angažovani** i administrator – **svi predmeti**).



Slike 79 i 80, s leva na desno: Prikaz grafikona iz ugla administratora/profesora i kada predmet nije aktivan

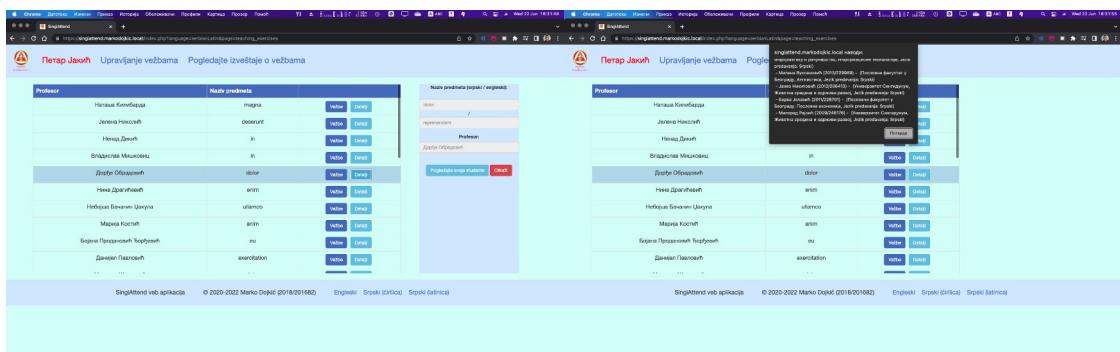
Na glavnoj stranici profesora i asistenta, moguć je pregled i izmena detalja, kao i započinjanje evidentiranja novih predavanja, odnosno vežbi.

Na slikama 81 i 82 vidimo postupak izmene naziva predmeta i izbor novog asistenta. Nakon ove izmene stranica se ponovo učitava i vidimo nove podatke, a takođe se podaci o ovom predmetu pojavljuju i kod izabranog asistenta (**Petar Biševac**).



Slike 81 i 82, s leva na desno: Prikaz izmene podataka predmeta (promjena naziva i izbor novog asistenta)

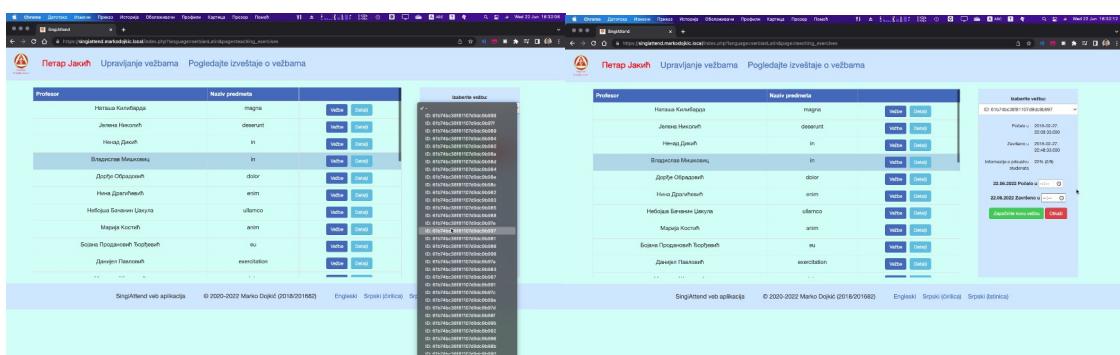
Prikaz ovo dela aplikacije kod asistenta se razlikuje samo u nemogućnosti izmjene podataka, tj. omogućeno je samo čitanje. Na slici 84 vidimo „alert“ prozor koji pokazuje par studenata koji pohađaju izabrani predmet, ova opcija je dostupna i za profesore.



Slike 83 i 84, s leva na desno: Prikaz detalja o predmetu (iz ugla asistenta) i informacije o studentima

Na slikama ispod je prikazan izbor detalja o vežbama i forme za započinjanje nove. U našem slučaju vidimo podatke o vežbi održanoj 27. Februara 2018 godine u terminu 22:03-22:48, a takođe da je njoj prisustvovalo 22% studenata (njih dvoje od devetoro).

Započinjanje nove vežbe je moguće nakon unosa vremena u početka i završetak u predviđena polja (datum je automatski podešen na dan započinjanja). Nakon unosa sistem proverava validnost termina, odnosno da li je vremenski duži od 45 minuta, kraći od 6 sati i da je definisan u budućnosti.

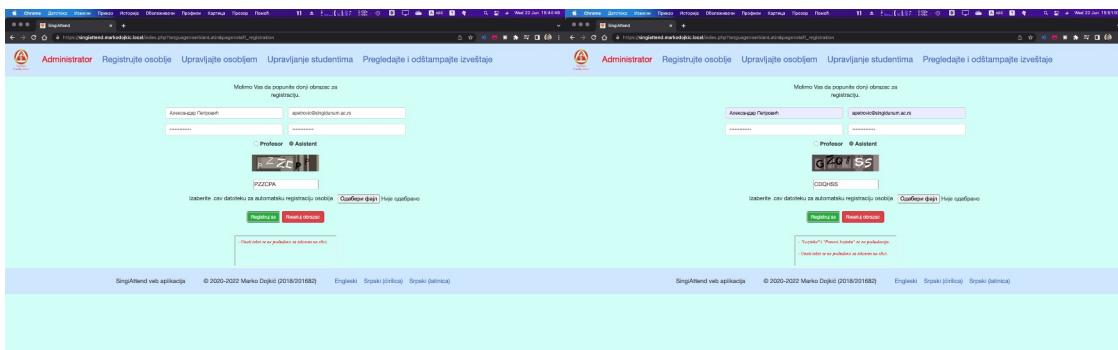


Slike 85 i 86, s leva na desno: Prikaz detalja o izabranoj vežbi i forme za započinjanje nove

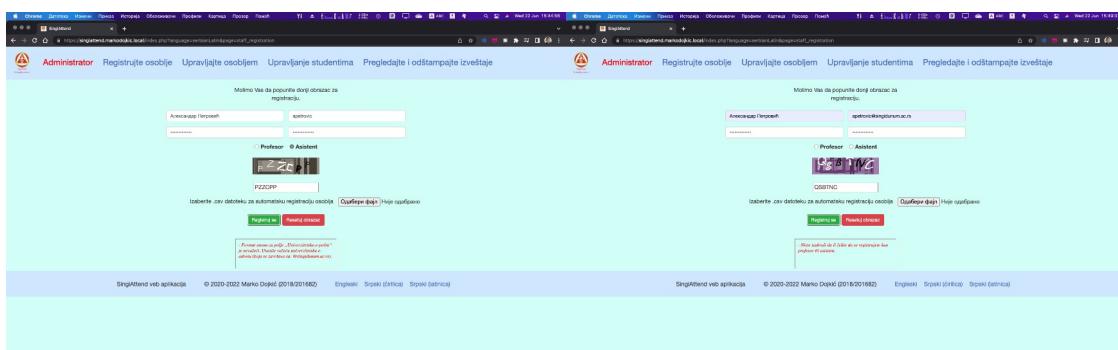
Kao što već rečeno registraciju nastavnog osoblja obavlja administrator sistema. Forma za unos se sastoji od polja koji sadrže ime i prezime zaposlenog, adresu elektronske pošte (format Univerziteta Singidunum), dva polja za unos i potvrdu lozinke, „radio“ tastera za izbor vrste naloga (da li je zaposleni profesor ili asistent). Prilikom registracije je potrebno popuniti i „captcha“ tekst.

Pored ručnog unosa podataka, moguće je višestruki unos podataka iz .csv fajla.

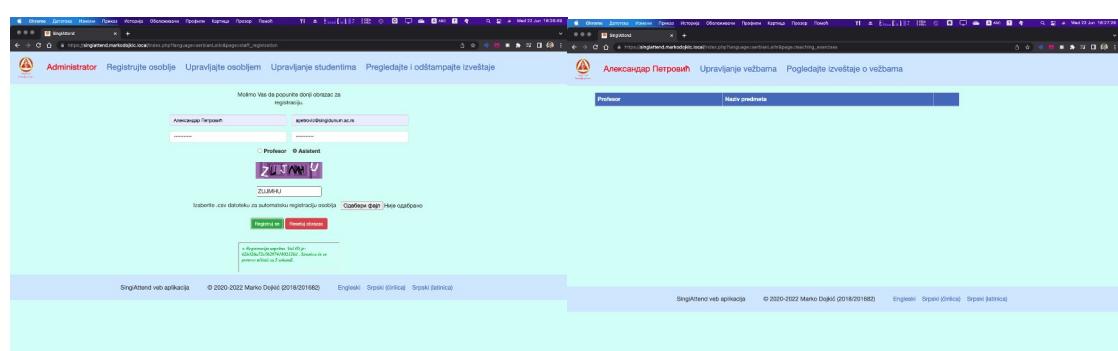
Na slikama 87-90 je prikazan izgled mogućih grešaka, nakon provere prilikom registracije, a na slici 91 poruka o uspešnosti registracije i inicijalni prikaz nakon logovanja novog asistenta (**Aleksandar Petrović**) na slici 92.



Slike 87 i 88, s leva na desno: Prikaz grešaka ne validnog unosa „captcha“ teksta i lozinke



Slike 89 i 90, s leva na desno: Prikaz greške ne validnog unosa adrese elektronske pošte i izbora vrste naloga



Slike 91 i 92, s leva na desno: Prikaz poruke o uspešnosti registracije novog asistenta (sa njegovim dodeljenim ID-jem) i izgled stranice nakon logovanja tog asistenta

Administrator takođe ima mogućnost izmena i brisanja osoblja i studenata. Na slikama 93 i 94 vidimo prikaz tabele, čiji redovi predstavljaju podatke o osoblju, i primećujemo da asistentkinja **Maja Obradović** nema unetu adresu elektronske pošte. Nakon popunjavanja i klika na taster *izmeni* (slika 95), a ta izmena se odmah oslikava na našoj stranici.

Pored mogućnosti brisanja podataka, u slučaju osoblja moguća je izmena vrste naloga (prekvalifikacija asistenta u profesora i suprotno). Na slici 96 vidimo **pokušaj** ovakve izmene, ali dobijamo poruku da ona nije moguća jer asistent ima već dodeljene predmete. Da bi se ova izmena izvršila, svi profesori koji su dodelili predmete tom asistentu moraju taj podatak da izmene (dodele predmet drugom ili nijednom asistentu). U slučaju da smo izmenu pokušali da uradimo na profesorskog naloga bilo bi potrebno uklanjanje svih njegovih predmeta.

Slike 93 i 94, s leva na desno: Prikaz tabele zaposlenog osoblja (pre izmena)

Slike 95 i 96, s leva na desno: Prikaz izmene adrese elektronske pošte zaposlenog i pokušaja izmene vrste naloga asistenta, tj. njegova prekvalifikacija u profesora

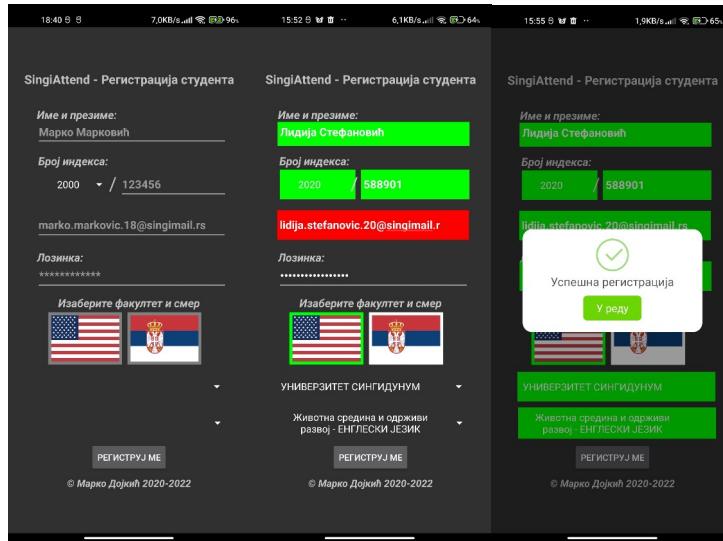
Mogućnost uklanjanja naloga ćemo prikazati uklanjanjem studenta sa brojem indeksa **2019/253015** (slika 97), jer se recimo on ispisao sa fakulteta. Pritiskom na taster *izbriši* podaci o tom studentu se uklanjuju iz tabele i na slici 98 vidimo novi izgled tabele.

Slike 97 i 98, s leva na desno: Prikaz uklanjanja podataka studenta sa brojem indeksa 2019/253015

## 9.2. Studentska Android aplikacija

Studenti se samostalno registruju popunjavanjem forme unutar dela aplikacije za registraciju. Forma (slika 99) se sastoji od: imena i prezimena studenta, godine upisa i broja indeksa, adrese elektronske pošte (format Univerziteta Singidunum), lozinke i izbora fakulteta i smera (izbor u zavisnosti o upisu studija na engleskom ili srpskom jeziku).

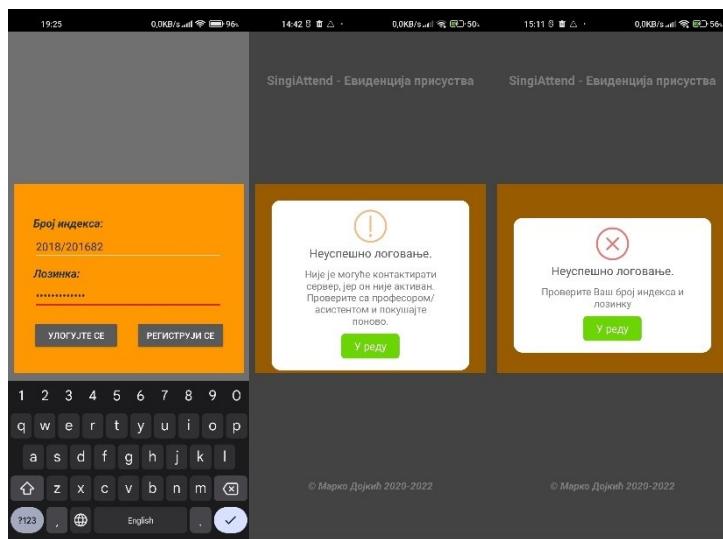
U slučaju ne validnog unosa nekog podatka to polje će postati crveno (slika 100), a ukoliko sve provere prođu uspešno dobijamo prikaz „SweetAlert“ iskačućeg prozora (slika 101).



Slike 99-101, s leva na desno: Prikaz registracione forme (prazne, ne validno i ispravno popunjene)

Nakon registracije, potrebno je da se student uloguje u aplikaciju. Inicijalni prozor za logovanje studenta je prikazan na slici 102, na kojoj student popunjava broj indeksa i lozinku.

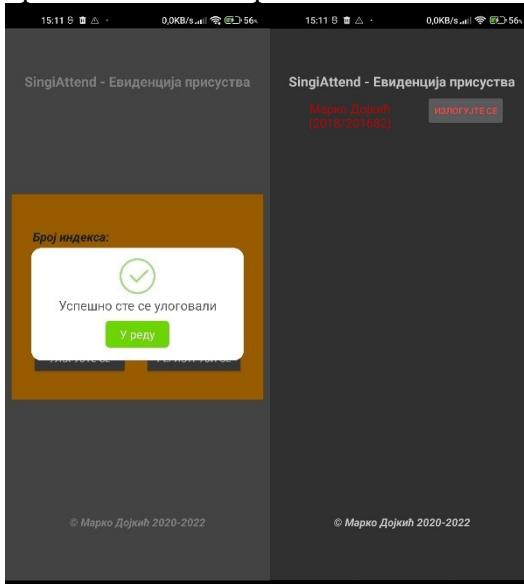
U slučaju da je server nedostupan prilikom logovanja ili su uneti parametri netačni, tj. pogrešno je uneta lozinka ili broj indeksa, prikazuju se odgovarajuće poruke o greškama (slike 103 i 104 respektivno).



Slike 102-104, s leva na desno: Prikaz forme logovanja i gresaka koji mogu da nastanu prilikom logovanja

Nakon uspešnog logovanja studenta (slika 105), a i u slučaju prvog pokretanja aplikacije dok je student ulogovan, dobijamo glavni prikaz koji pokazuje ime, prezime i broj indeksa ulogovanog studenta. U ovom trenutku je potrebno napomenuti da se ime i prezime dobijaju od strane servera, a u slučaju nemogućnosti dobijanja istog ovaj podatak će ostati prazan i ispod taster za izlogovanje će se pojaviti poruka da server trenutno nije dostupan.

U slučaju da je server dostupan pokreće se rutinski kod koji dobavlja spisak trenutno aktivnih predavanja, odnosno vežbi ulogovanog studenta, a potom i ukupne evidencije prisustva na svim predmetima.



Slike 105 i 106, s leva na desno: Prikaz uspešnosti logovanja i inicijalnog prozora ulogovanog studenta

Na slici 107 vidimo izgled evidencije prisustva, gde je izabran prikaz detalja prisustva na predavanjima predmeta „**consectetur**“, na slici 108 vidimo i spisak aktivnih predavanja i vežbi (predmeti „**sint**“, „**Testiranje softvera**“ i „**Metodologija razvoja softvera**“ respektivno).

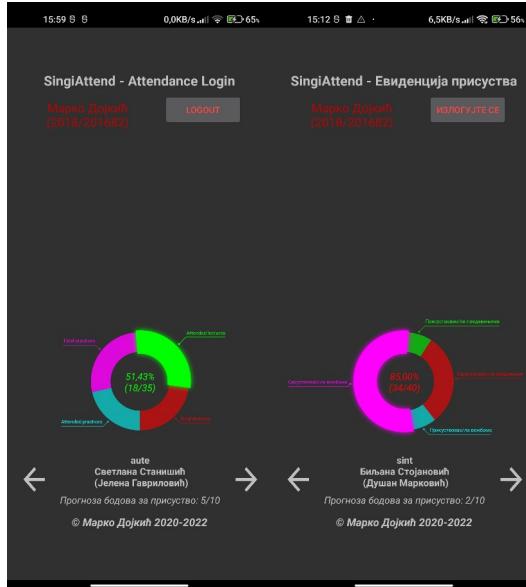


Slike 107 i 108, s leva na desno: Prikaz evidencije prisustva na predavanjima i liste aktivnih predavanja/vežbi studenta

Podaci o ukupnoj evidenciji prisustva se učitavaju po predmetima (kretanje kroz JSON niz predmeta je moguće pritiskom strelica). Na slikama 109 i 110 vidimo izgled evidencije prisustvu predavanjima predmeta „**aute**“ i „**sint**“ respektivno.

Pored podataka o evidenciji prisustva, vidimo i ime i prezime profesora i asistenta izabranog predmeta, kao i informaciju o prognozi bodova za prisustvo (koji se računa po formuli:

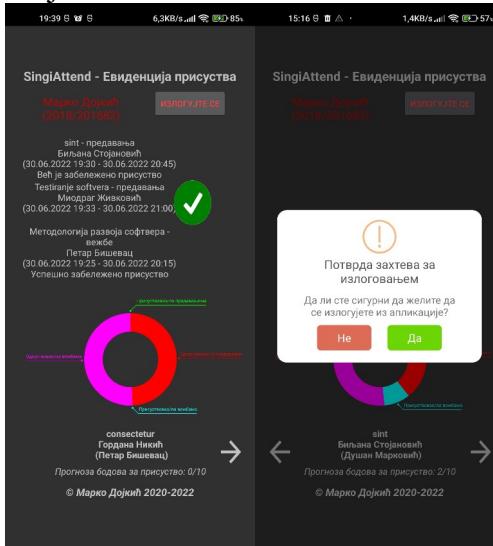
***10.0 \*zbir\_ukupnog\_prisustva\_na\_predavanjima\_i\_vežbama/ zbir\_ukupnog\_broja\_predavanja\_i\_vežbi*** (da bi se nakon zaogruživanja na ceo broj dobio broj između 0 i 10 koliko je moguće i dobiti bodova na Univerzitetu Singidunum).



Slike 109 i 110, s leva na desno: Prikaz grafikona evidencije prisustva na predmetima „**aute**“ i „**sint**“

Izgled evidentiranja prisustva je prikazan na slici 111, na kojoj vidimo da smo uspešno zabeležili prisustvo na vežbama „**Metodologije razvoja softvera**“, a već smo evidentirali ranije na predavanjima predmeta „**sint**“.

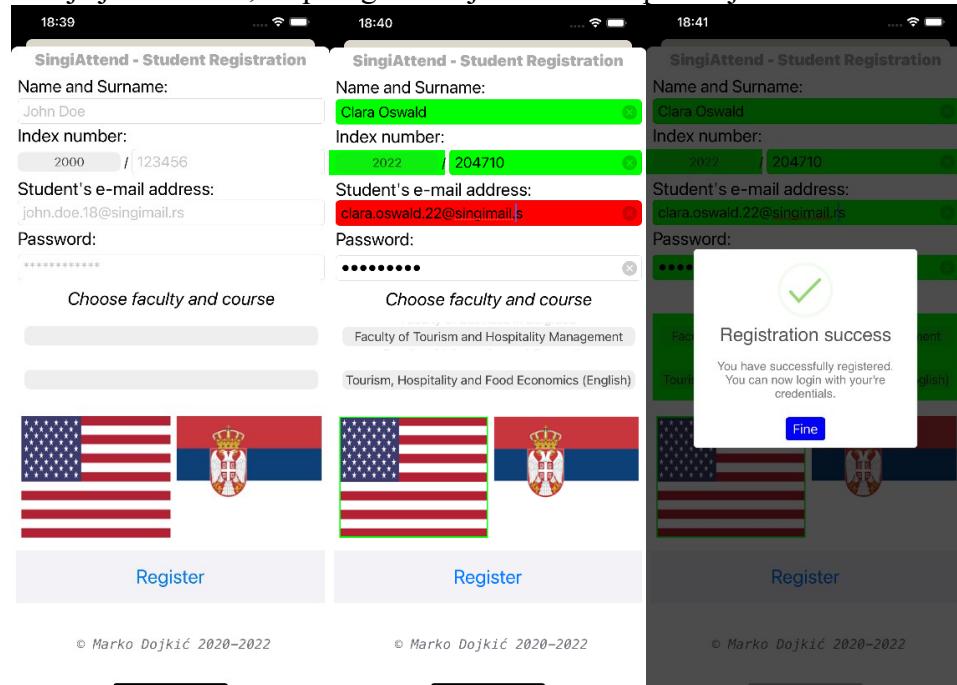
Na kraju na slici 112 vidimo prikaz dijaloga koji nas pita da li želimo da se izlogujemo iz aplikacije.



Slike 111 i 112, s leva na desno: Prikaz rezultata evidentiranja prisustva i prozora za potvrdu izlogovanja

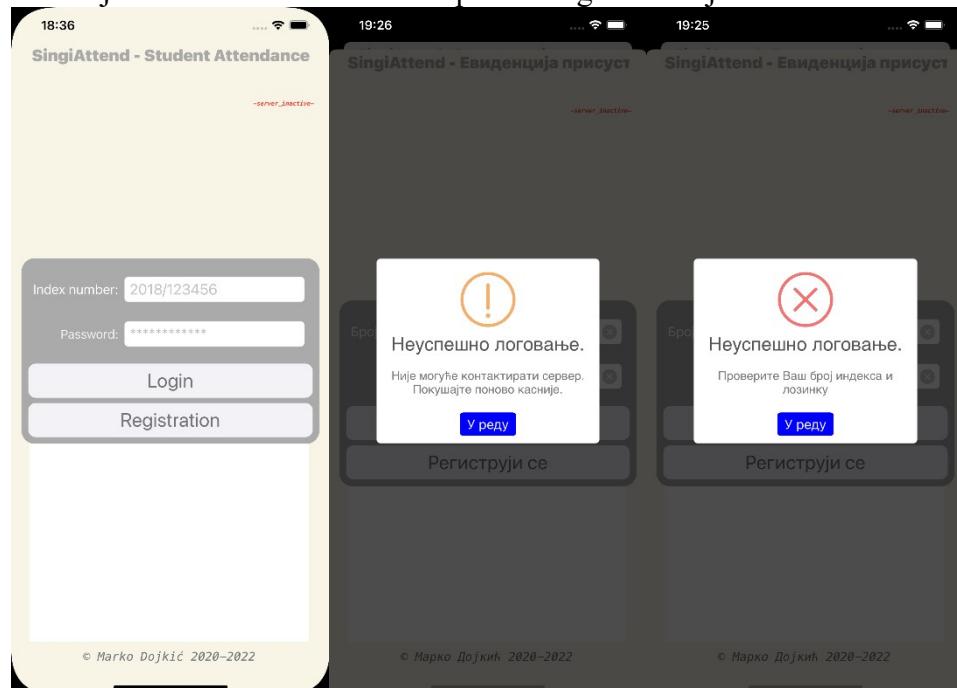
### 9.3. Studentska iOS aplikacija

Kao što je slučaj i kod Android aplikacije, studenti se mogu registrovati i korišćenjem iOS aplikacije, upotrebom identične forme. Način provere i registracije je identičan, uz prilagođavanje koda iOS aplikaciji.



Slike 113-115, s leva na desno: Prikaz registracione forme (prazne, ne validno i ispravno popunjene)

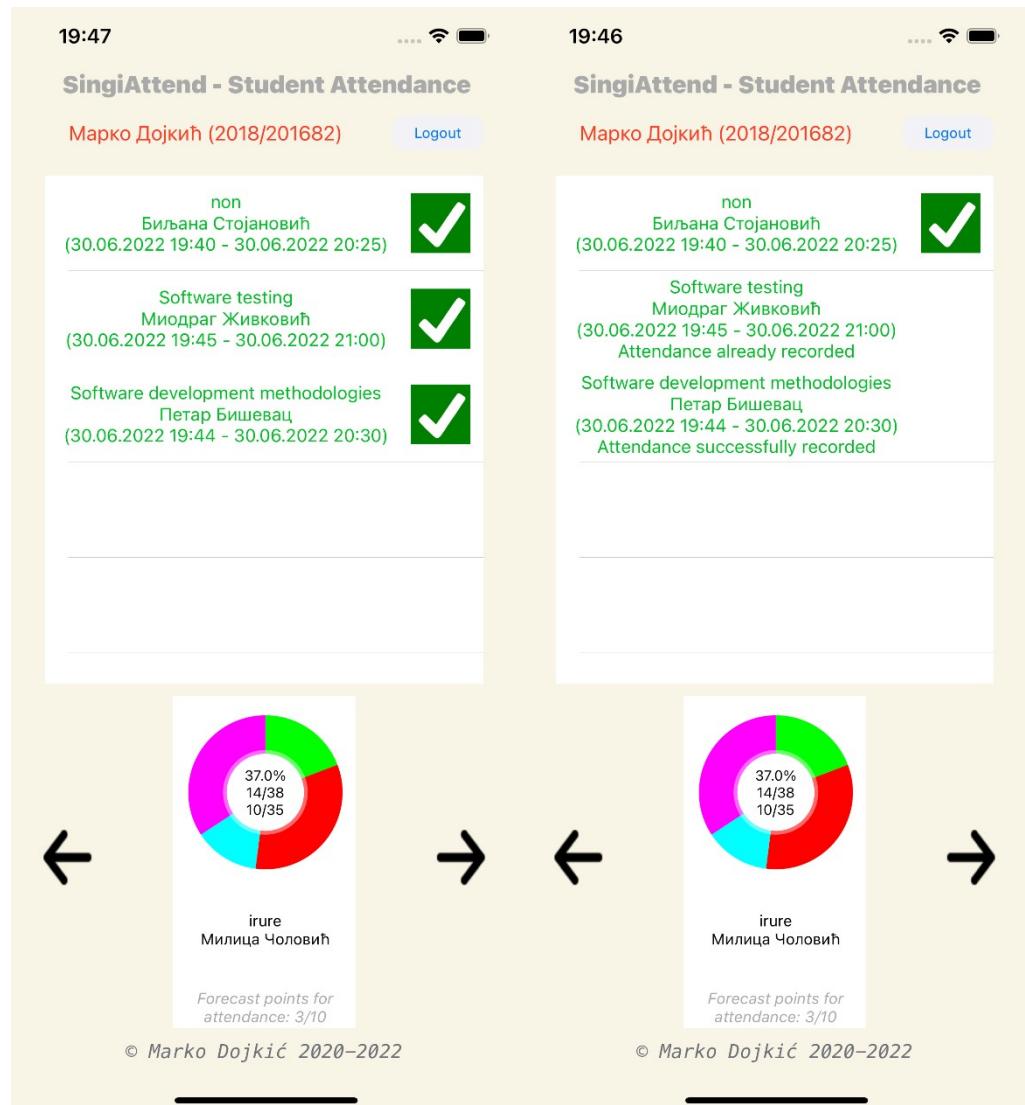
Nakon uspešne registracije student se može ulogovati, na identičan način, unosom broja indeksa i lozinke. Prikaz poruka o greškama je takođe identičan.



Slike 116-118, s leva na desno: Prikaz forme logovanja i gresaka koji mogu da nastanu prilikom logovanja

Na slici 120 vidimo prikaz glavnog prozora nakon učitanih informacija o trenutno aktivnim predavanjima, odnosno vežbama i evidenciji prisustva trenutno ulogovanog studenta. Ovi podaci se dobijaju od strane servera, na identičan način kao u slučaju Android aplikacije. Izgled evidentiranja prisustva je prikazan na slici 121, na kojoj vidimo da smo uspešno zabeležili prisustvo na **predmetu „Software development methodologies“**, a već smo evidentirali ranije na **predmetu „Software testing“**.

Student se takođe može izlogovati upotrebom „Logout“ tastera, nakon čijeg pritiska će dobiti identičan „SweetAlert“ dijalog za potvrdu.



*Slike 119-121: Prikaz glavnog prozora iOS aplikacije i poruka o uspešnosti evidentiranja prisustva na predmetima*

## **9. Zaključna razmatranja**

Ovaj rad predstavlja jednu celinu mog četvorogodišnjeg školovanja i dosadašnjeg učenja programiranja, kao i samih informacionih tehnologija. Takođe u ovom radu se može videti spoj različitih tehnologija, koje za cilj imaju rešavanje jednog, na izgled jednostavnog, problema evidentiranja prisustva studenata na Univerzitetu Singidunum.

Koncept koji je primjenjen prilikom izrade ove aplikacije se može primeniti i prilikom izrade aplikacije evidencije prisustva na bilo kojoj drugoj visokoškolskoj ustanovi, uz minimalnu izmenu grafičkog korisničkog interfejsa i strukture baze podataka.

Izbor tehnologija korisničkih aplikacija je bio vođen prepostavkom da studenti uvek sa sobom nose pametne telefone, koji koriste Android ili iOS operativni sistem, a da profesori i asistenti imaju dostupne fakultetske ili svoje računare. Shodno toj prepostavci korisničke aplikacije su kreirane za web i mobilna okruženja.

Jedna od preporuka za dalji razvoj aplikacije bi bila unapređenje sigurnosti glavnog servera, koji izvršava celu logiku i zadužen je za rad celokupnog sistema. Naime trenutna sigurnost je zasnovana na upotrebi jednostavne autentifikacije korisničkim imenom i lozinkom (koji su hardkodovani unutar svih aplikacija), čijom upotrebotom se može direktno pristupiti servisima i doći do kompromitovanja podataka unutar same baze od strane zlonamernih korisnika. Bolji pristup bi bila upotreba, recimo JWT (eng. *Java Web Token*) autentifikacije, koja bi bila zasnovana na već postojećim autentifikacionim parametrima koji koriste web i studentske aplikacije, a koja bi omogućila i vremensko ograničenje pristupa (zahtevalo bi se ponovno logovanje studenata i nastavnog osoblja u toku jednog dana). Pored te mere unapređenja sigurnosti moguća je i primena sigurnijeg metoda šifrovanja lozinki korisničkih naloga.

Na kraju potrebno je testirati rad same aplikacije u realnom okruženju i praćenje performansi iste, a ukoliko se utvrdi da ih je moguće poboljšati, potrebno je to i učiniti.

## **10. Literatura**

1. Kojić, N. (2020) *Web dizajn: HTML, CSS i JavaScript*, treće izdanje, Univerzitet Singidunum, Beograd
2. Dobrojević M., Bačanin Džakula N. (2021) *Veb programiranje*, prvo izdanje, Univerzitet Singidunum, Beograd
3. Živković M., Bačanin Džakula N., Tuba E. (2022), *Programski jezici*, drugo izdanje, Univerzitet Singidunum, Beograd
4. Jevremović A., Jakić P. (2022), prezentacije sa predmeta *NoSQL baze*
5. Živković M. (2020), *Razvoj mobilnih aplikacija*, prvo izdanje, Univerzitet Singidunum, Beograd
6. <https://developer.android.com/guide/> (dostupno dana: 11.07.2022.)
7. <https://github.com/razerdp/AnimatedPieView> (dostupno dana: 11.07.2022.)
8. <https://developer.apple.com/documentation> (dostupno dana: 11.07.2022.)
9. <https://sweetalert2.github.io/> (dostupno dana: 11.07.2022.)