#### Sadržaj

[1. Uvod 1](#_Toc207839204)

[2. Tehnologije 2](#_Toc207839205)

[2.1. Java 2](#_Toc207839206)

[2.2. Spring Framework 4](#_Toc207839207)

[2.3. MySQL DBMS 4](#_Toc207839208)

[2.4. Thymeleaf pogonitelj predložaka 5](#_Toc207839209)

[2.5. Flyway alat za migraciju baze podataka 6](#_Toc207839210)

[3. Korisničko sučelje 8](#_Toc207839211)

[3.1. Thymeleaf predlošci 8](#_Toc207839212)

[3.2. Izgled korisničkog sučelja 9](#_Toc207839213)

[4. Poslovna logika 10](#_Toc207839214)

[4.1. MVC arhitektura 10](#_Toc207839215)

[4.1.1. Model 10](#_Toc207839216)

[4.1.2. View 11](#_Toc207839217)

[4.1.3. Controller 11](#_Toc207839218)

[4.2. Modularizacija, API baze podataka 12](#_Toc207839219)

[5. Baza podataka 13](#_Toc207839220)

[5.1. MyBatis sustav za komunikaciju s bazom podataka 13](#_Toc207839221)

[6. Povezivanje modula 15](#_Toc207839222)

[6.1. HTTP zahtjevi 15](#_Toc207839223)

[6.2. Autentifikacija pristupa 16](#_Toc207839224)

[6.2.1. JSON Web Token 16](#_Toc207839225)

[6.2.2. API ključevi 17](#_Toc207839226)

[6.3. Prijenos podataka između aplikacija 18](#_Toc207839227)

[6.3.1. Serijalizacija i deserijalizacija 18](#_Toc207839228)

[7. Generacija dokumenata za preuzimanje na korisnikovo računalo 20](#_Toc207839229)

[7.1. Generacija PDF dokumenata 20](#_Toc207839230)

[7.2. Digitalni potpis 22](#_Toc207839231)

[7.3. Preuzimanje datoteka 23](#_Toc207839232)

[8. Zaključak 25](#_Toc207839233)

[9. Popis literature 26](#_Toc207839234)

#### Popis kratica i oznaka

API *Application Programming Interface*

CSS *Cascading Style Sheet*

DBMS *Database Management System*

HTML *Hypertext Markup Language*

HTTP *Hypertext Transfer Protocol*

JSON *Javascript Object Notation*

MVC *Model-View-Controller*

OOP Objektno orijentirano programiranje

PDF *Portable Document Format*

SQL *Structured Query Language*

URL *Uniform Resource Locator*

UX *User experience*

XML *Extensible Markup Language*

#### Popis tablica

[Tablica 1 Najčešće HTTP metode zahtjeva 15](#_Toc207839235)

#### Popis slika

[Slika 1 Slika ekrana Spring Initializr web stranice 4](#_Toc207839236)

[Slika 2 Flyway migracijski direktorij 6](#_Toc207839237)

[Slika 3 Dijagram komunikacije uz JWT ovjeru 17](#_Toc207839238)

[Slika 4 Dijagram procesa digitalnog potpisivanja i verifikacije potpisa 22](#_Toc207839239)

#### Popis programskih isječaka

[Programski isječak 1 Primjer Java klase 3](#_Toc207839240)

[Programski isječak 2 Primjer HTML koda koji koristi Thymeleaf elemente 6](#_Toc207839241)

[Programski isječak 3 Primjer Spring kontrolne jedinice web sučelja 8](#_Toc207839242)

[Programski isječak 4 Primjer klase korištene za instanciranje podataka 11](#_Toc207839243)

[Programski isječak 5 Primjer kontrolne jedinice koja dostavlja podatak u obliku JSON objekta 12](#_Toc207839244)

[Programski isječak 6 Primjer MyBatis repozitorija 13](#_Toc207839245)

[Programski isječak 7 Primjer MyBatis mapper datoteke 14](#_Toc207839246)

[Programski isječak 8 Primjer implementacije različitih funkcionalnosti na istoj pristupnoj točci 16](#_Toc207839247)

[Programski isječak 9 JSON izlaz API-a za dohvat trenutnog vremena 18](#_Toc207839248)

[Programski isječak 10 Primjer implementacije parsiranja JSON podataka uporabom Jackson alata 19](#_Toc207839249)

[Programski isječak 11 Metoda za generaciju PDF dokumenta 21](#_Toc207839250)

[Programski isječak 12 Metoda za pretvorbu HTML elemenata u PDF elemente 22](#_Toc207839251)

[Programski isječak 13 Primjer kontrolne jedinice koja poslužuje tok bajtova 24](#_Toc207839252)

# Uvod

Za razliku od klasičnih samostalnih aplikacija (engl. *desktop application*) koje se izvršavaju na korisnikovom računalu, web aplikacije se obično izvršavaju na udaljenom poslužitelju te se s njima komunicira putem web preglednika kao što su *Google Chrome*, *Mozilla Firefox* ili *Microsoft Edge*. U slučaju programskog okvira *Spring Framework*, aplikacija presreće korisnikov *Hypertext Transfer Protocol* (HTTP) zahtjev te umjesto statičkog resursa korisniku dostavlja dinamički generiran resurs uporabom poslovne logike.

U današnje vrijeme u većini se institucija, bile one službene ili osobne, koriste suvremeni načini komunikacije putem interneta. Zbog ovog razloga sve više se koriste web aplikacije i usluge zbog mogućnosti automatizacije komunikacijskih i poslovnih procesa. Svaka moderna institucija ima svoje web sjedište koje nudi mnoge usluge poput kontaktiranja osoblja, pregled usluga, predaju obrazaca i sl. te su takve aplikacije važan sastavni dio svake organizacije koja ih koristi.

U ovom će radu biti prikazan proces izrade web sjedišta za jednu takvu organizaciju. Biti će opisane metode i tehnike izrade reznih neophodnih dijelova aplikacije, spajanje istih u cjelinu te komunikacija između njih.

Rad je podijeljen na tri dijela. Uvodni dio navest će i opisat sve tehnologije potrebne za izradu web aplikacije. U tehnologije spadaju programski jezici, alati za pomoć u programiranju te sustavi posvećeni određenim ulogama. U drugom dijelu biti će opisan proces izrade korisničkog sučelja te metodologija dizajna. Treći će se dio baviti temom *backend*-a aplikacije (svi procesi koji su nevidljivi korisniku), u što spadaju poslovna logika i funkcionalnosti aplikacije, komunikacija s bazom podataka te moduli aplikacije i njihova međusobna komunikacija.

# Tehnologije

U ovoj sekciji rada navedene su i opisane sve tehnologije, programski jezici i njihove stavke potrebne za izradu web aplikacije.

## Java

Java je visokorazinski objektno-orijentirani programski jezik čija se prva inačica pojavila 1995. godine [1]. Objektno orijentirano programiranje (OOP) temelji se, kao što ime ukazuje, na uporabi objekata koji su pojedinačne instance klasa – programski realiziranih predložaka. Objekt može sadržavati atribute (imenovane podatke) i metode (implementacije poslovne logike za obradu podataka). Ova svojstva mogu biti privatna, što znači da im se može pristupiti samo iz opsega samog objekta, ili javna, što znači da se mogu pozvati u bilo kojem opsegu u kojem je objekt instanciran. Objekt može služiti kao element za spremanje, prijenos i obradu podataka, sučelje za vanjski alat, sučelje za metode i sl.

|  |
| --- |
| @Controller @RequestMapping(*URL\_AUTHORIZATION*) public class AuthorizationController {  private final *ProfessorService* **professorService**;  private final AuthenticationService **authenticationService**;  private final Messages **messages**;   public AuthorizationController(*ProfessorService professorService*,  AuthenticationService *authenticationService*,  Messages *messages*) {  this.**professorService** = *professorService*;  this.**authenticationService** = *authenticationService*;  this.**messages** = *messages*;  }   @GetMapping  public String showAuthorizationView(*Model model*) {  **authenticationService**.refresh();  User userLogin = (User) *model*.getAttribute("userLogin");  initModel(*model*);  return "authorization";  }   @PostMapping  public String processAuthorization(*Model model*,  @ModelAttribute ProfessorBuffer *professorBuffer*) {  **authenticationService**.refresh();  User userLogin = (User) *model*.getAttribute("userLogin");   if (!*professorBuffer*.getProfessors().isEmpty()) {  for (Professor professor : *professorBuffer*.getProfessors())  **professorService**.authorizeProfessor(professor.getId());  *model*.addAttribute(  "success", **messages**.getMessage("success.changes-saved"));  } else  *model*.addAttribute(  "error", **messages**.getMessage("error.missing-professor"));   initModel(*model*);  return "authorization";  }   private void initModel(*Model model*) {  *initialize*(*model*, *URL\_AUTHORIZATION*);  ProfessorBuffer professorBuffer = ProfessorBuffer.*builder*()  .professors(  **professorService**.getUnauthorizedProfessors().stream()  .sorted(new ProfessorComparator())  .toList())  .build();  *model*.addAttribute("professorBuffer", professorBuffer);  } } |

Programski isječak 1 Primjer Java klase

U gornjem prikazu (Programski isječak 1) može se vidjeti definicija klase *AuthorizationController* koja sadrži jedan atribut („*professorService*“) i tri metode („*showAuthorizationView*“, „*processAuthorization*“ i „*initModel*“).

Java se od ostalih programskih jezika izražava time što se aplikacije programirane u Javi ne izvršavaju u korisnikovom izvornom programskom okruženju, već su enkapsulirane u *Java Virtual Machine* – virtualno okruženje koje može izvršavati bilo koji program pisan u Javi. Ovime se postiže neovisnost aplikacije o operacijskom sustavu koji ju pokreće tako da se aplikacije pisane u Javi mogu izvršavati na bilo kojem računalu koje ima instaliranu određenu inačicu Java platforme.

## Spring Framework

A screenshot of a computer

AI-generated content may be incorrect.Spring Framework je programski okvir za Javu razvijen ranih 2000.-ih čija je današnja inačica objavljena 2022. godine [2]. Programski okvir je paradigma za programski razvoj koja olakšava razvoj pružajući korisniku skup alata i modula. Dva najvažnija alata iz okvira Spring nazivaju se *Spring Boot* i *Spring Initializr*.

Slika 1 Slika ekrana Spring Initializr web stranice

(Izvor: <https://start.spring.io/>, pristupljeno: 13.1.2025.)

Spring Initializr koristi se za inicijalizaciju projekta te je vanjski alat. U njemu je moguće odabrati željene funkcionalnosti projekta, inačicu Jave te inačicu Spring Boot alata koja će se koristiti (Slika 1). Izlazna datoteka ovog alata je inicijalizirani projekt koji sadrži sve zavisnosti (engl. *dependency*) potrebne za realizaciju projekta.

Spring Boot je alat koji „pokreće“ Spring aplikaciju, tj. alat koji omogućuje da Spring projekt bude samostalna aplikacija. U njega su uključeni razni ostali alati poput *Apache* *Tomcat servera* (aplikacijski web poslužitelj za Javu) [3].

## MySQL DBMS

Baza podataka je skup podataka organiziran u tablice koje su strukturirane na način povoljan korisniku baze za njegovu upotrebu. Za upravljanje bazom podataka koristi se *Database Management System* (DBMS)– sustav razvijen za svaku od izvedba baza podataka od kojih su danas najpopularniji MySQL i PostgreSQL.

MySQL je relacijski DBMS. Kao takav podržava relacijske veze između tablica, tj. veze koje opisuju način na koji su tablice povezane te njihov međusobni odnos, koje će biti temeljne za izvedbu ovog projekta. Za upravljanje bazom koristi se *Structured Query Language* (SQL), deklarativni programski jezik kojim se dobavljaju i spremaju podaci, izvršavaju transakcije, definiraju strukture tablica i sl.

## Thymeleaf pogonitelj predložaka

Thymeleaf je pogonitelj predložaka (engl. *template engine*) čija je svrha stvaranje dinamičkog *Hypertext Markup Language* (HTML) koda u trenutku korisničkog zahtjeva. Njime se ostvaruje veza između HTML elemenata koji preglednik prikazuje korisniku aplikacije i aplikacijskog koda te njegove poslovne logike. HTML kod je uobičajeno statički resurs, no uporabom pogonitelja predložaka, programer aplikacije može napraviti predložak HTML resursa koji pogonitelj, u trenutku kada korisnik zatraži taj resurs, puni zadanim vrijednostima te preoblikuje po potrebi.

U primjeru (Programski isječak 2) vidljivi su elementi označeni svojstvima koja imaju prefiks „*th:*“. Takvi elementi podložni su Thymeleaf pogonitelju te svoje vrijednosti dobavljaju iz Java aplikacije. Osim dobavljanja dinamičkih vrijednosti, uporabom Thymeleaf pogonitelja moguće je i izvršavati grananje korištenjem „*th:if*“ svojstva te dijeljenje HTML koda u module korištenjem „*th:replace*“ svojstva.

|  |
| --- |
| <!DOCTYPE *html*> <html *lang*="en" *xmlns:***th**="http://www.thymeleaf.org">  <head>  <meta *charset*="UTF-8"/>  <title>Login</title>  <link *rel*="stylesheet" *href*="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/css/bootstrap.min.css">  <link *rel*="stylesheet" **th***:href*="@{/styles.css}">  <script **th***:src*="@{/refreshTime.js}" *defer*></script>  </head>   <body>  <div **th***:replace*="~{fragments::dateTime}"></div>  <div **th***:replace*="~{fragments::languageSwitch}"></div>  <div *class*="container mt-5">  <h1 *class*="mb-4" **th***:text*="#{login}"></h1>   <div **th***:if*="${error != null}" *class*="card mb-4">  <div *class*="card-body">  <div *class*="error-message">  <p **th***:text*="${error}"></p>  </div>  <a **th***:href*="@{/signup}" **th***:text*="#{signup-instead}"></a>  </div>  </div>  <form *method*="POST" **th***:action*="@{/login}" **th***:object*="${userLogin}">  <table *class*="table table-striped">  <tbody>  <tr>  <td><label *for*="username" **th***:text*="#{username} + ': '">  </label></td>  <td>  <input *type*="text"  *id*="username"  **th***:field*="\*{username}"  *required*/>  </td>  </tr>  <tr>  <td><label *for*="password" **th***:text*="#{password} + ': '">  </label></td>  <td>  <input *type*="password"  *id*="password"  **th***:field*="\*{password}"  *required*/>  </td>  </tr>  <tr>  <td></td>  <td>  <button *type*="submit"  *class*="btn btnB"  **th***:text*="#{login}"></button>  </td>  </tr>  </tbody>  </table>  </form>  </div>  <script *src*=https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/js/bootstrap.bundle.min.js >  </script>  </body> </html> |

Programski isječak 2 Primjer HTML koda koji koristi Thymeleaf elemente

## Flyway alat za migraciju baze podataka

A screenshot of a computer

AI-generated content may be incorrect.Flyway je alat za migraciju baze podataka. Njime se postiže automatska validacija i ažuriranje baze podataka na najnoviju verziju određenu SQL skriptama koje programer postavlja u alatom određeni direktorij (Slika 2).

Slika 2 Flyway migracijski direktorij

Skripte u tom direktoriju moraju imati prefiks koji označava inačicu baze podataka, koji se sastoji od slova „V“ (engl. kratica za *version*), broja inačice te dvije donje crte (engl. *dunder*) kako bi ih Flyway alat mogao prepoznati kao migracijske skripte (nije dovoljno samo staviti ih u direktorij). Također, unutar baze podataka stvara se nova tablica „*flyway\_history\_schema*“ koja služi za validaciju uporabom *checksum* vrijednosti same skripte. *Checksum* je jedinstvena vrijednost izračunata zbrajanjem brojčanih vrijednosti znakova u skripti. Kao takva, kako bi se skripta pravilno verificirala, nakon inicijalne migracije na određenu inačicu, ne smije se mijenjati, čak i ako promjena ne utječe na izvršavanje koda kao npr. dodavanje novog praznog reda ili praznog znaka (razmaka). U slučaju semantičke pogreške u skripti koja se izvršava pravilno no daje krivi rezultat, potrebno je napraviti novu migracijsku skriptu koja ju ispravlja.

# Korisničko sučelje

Ova sekcija rada bavit će se izradom sučelja za komunikaciju s korisnikom aplikacije. Sučelje je realizirano Thymeleaf kodom koji se kompajlira u HTML te poslužuje korisniku kroz njegov web preglednik. U HTML je također nadovezan *Cascading Style Sheet* (CSS) kod koji definira izgled straničnih elemenata dok sami elementi definiraju strukturu web stranice.

## Thymeleaf predlošci

|  |
| --- |
| @Controller public class WebController {  @GetMapping("/playagame")  public String playGame(@RequestParam(name="choice",required=false)  String *theChoice*, *Model model*) {  if (*theChoice* == null) {  return "index";  }   String theOutcome = "error";  if (*theChoice*.equals("rock")) {  theOutcome = "tie";  }  if (*theChoice*.equals("paper")) {  theOutcome = "win";  }  if (*theChoice*.equals("scissors")) {  theOutcome = "loss";  }   *model*.addAttribute("outcome", theOutcome);  return "result";  } } |

Thymeleaf predlošci imaju izgled klasičnog HTML-a, no u elementima stranice moguće je definirati svojstva koja dobavljaju svoje vrijednosti iz određenog objekta „*model*“ koji je definiran kontrolnom jedinicom stranice. Svaka stranica ili skup povezanih stranica ima svoju kontrolnu jedinicu kojom programer prosljeđuje podatke dobivene poslovnom logikom aplikacije u predložak. Osim dohvata vrijednosti iz aplikacije u predložak, moguć je i obratan tok podataka, tj. uporabom raznih tehnika (poput HTML forma, JavaScript skripta, itd.) moguće je slati podatke koje korisnik upisuje ili programer definira u aplikaciju za obradu. Ovom se metodom omogućava eksternalizacija funkcionalnosti aplikacije u programski jezik Java, dok korisničko sučelje postaje lako dostupna web stranica bez potrebe da korisnik preuzme posebnu samostalnu aplikaciju. Također, korisnik nema pristup pozadinskom kodu web aplikacije kojim se osigurava povjerljivost izvornog koda i resursa koji se nalaze na udaljenom poslužitelju.

Programski isječak 3 Primjer Spring kontrolne jedinice web sučelja

(Izvor: <https://www.theserverside.com/tutorial/Spring-MVC-tutorial-How-Spring-Boot-aids-Java-web-development>, pristupljeno: 13.8.2025.)

U primjeru (Programski isječak 3) prikazana je struktura tipičnog Java kontrolera koji sadrži metodu za posluživanje predloška „*result.html*“ ili predloška „*indeks.html*“ (u slučaju nepostojećeg ulaznog podatka) koja se vrši kada korisnik pristupi određenoj pristupnoj točci (engl. *endpoint*) „*/playagame*“. Kontroler prima ulazni podatak „*theChoice*“ koji je u predlošku referenciran s „*choice*“. Unutar metode izvršava se logika kojom se određuje ishod „*theOutcome*“ te se sprema u objekt za prosljeđivanje predlošku kao izlazni podatak „*outcome*“.

## Izgled korisničkog sučelja

Osim funkcionalnosti, identitet web stranice definira i njen izgled. Izgled nije samo pitanje estetike već i korisničkog doživljaja (engl. *user experience*, UX). UX filozofija je dizajna koja se temelji na mnoštvu ljudskih doživljaja te postavlja pitanja o tome kako ih ostvariti ili potaknuti interakcijom s određenim proizvodom [4].

Korisničko sučelje, osim svoje praktičke namjene, ima zadaću biti pristupačno i lako za upotrebu. Iz ovog razloga potrebno je obratiti pažnju tijekom razvoja kako bi se osigurao najbolji mogući korisnički dojam. Neke od najvažnijih stavaka tijekom razvoja korisničkog sučelja su intuitivnost uporabe te korisnička očekivanja. Intuitivnost je najlakše inkorporirati uporabom boja, npr. zelena se boja asocira s dobitkom i potvrdom te je ona izvrstan izbor za elemente kojima korisnik potvrđuje izbor ili upis. Korisnička su očekivanja, s druge strane, teža za predvidjeti iz razloga što se razlikuju od osobe do osobe s obzirom na kulturu [5], npr. većina ljudi u globalnom zapadu pišu latiničkim pismom koje se piše s lijeva na desno, no članovi drugih kultura poput većine arapskog svijeta pišu arapskim pismom s desna na lijevo. Ovo dovodi u pitanje raspored elemenata u korisničkom sučelju te zahtijeva analizu ciljane publike.

# Poslovna logika

Poslovna logika obuhvaća cijelu funkcionalnost aplikacije. To uključuje primanje i obrađivanje korisničkih podataka, posluživanje korisničkih sučelja, čitanje, spremanje i modifikaciju podataka u bazi podataka, autorizaciju i autentifikaciju korisnika te sve povezane operacije nevidljive korisniku. Ova sekcija rada bavi se poslovnom logikom, povezivanjem dijelova aplikacije i njihovom komunikacijom.

## MVC arhitektura

Struktura aplikacije definirana je obrascem dizajna koji se koristi tijekom razvoja. U ovom radu koristi se *Model-View-Controller* (MVC) arhitekturni obrazac. Prema obrascu, funkcionalnosti projekta strukturirane su u zasebne datoteke. Kao što ime sugerira MVC arhitektura sastoji se od tri dijela [6].

### Model

Pojam *Model* se u MVC arhitekturi odnosi na prikaz podataka unutar aplikacije. U Java aplikaciji model obuhvaća sve podatkovne klase i pripadajuće repozitorije.

|  |
| --- |
| public class Grade extends BaseEntity {  private Integer **grade**;  private String **student**;  private String **course**;   public Grade() {  super(null);  }   public Grade(final String *id*,  final Integer *grade*,  final String *student*,  final String *course*) {  super(*id*);  this.**grade** = *grade*;  this.**student** = *student*;  this.**course** = *course*;  }   public String getId() {  return super.getId();  }   public Integer getGrade() {  return this.**grade**;  }   public String getStudent() {  return this.**student**;  }  public String getCourse() {  return this.**course**;  }   public void setId(final String *id*) {  super.setId(*id*);  }   public void setGrade(final Integer *grade*) {  this.**grade** = *grade*;  }   public void setStudent(final String *student*) {  this.**student** = *student*;  }   public void setCourse(final String *course*) {  this.**course** = *course*;  } } |
|  |

Programski isječak 4 Primjer klase korištene za instanciranje podataka

Podaci se unutar aplikacije instanciraju definiranim klasama koje sadrže atribute koji opisuju podatak te metode kojima se taj podatak obrađuje (Programski isječak 4). Klasa označava predložak konkretnog podatka, a objekt označava instancu klase popunjenu atributima (nižim objektima ili atomskim podacima).

### View

*View* predstavlja prezentacijski sloj. Ovaj sloj varira u implementaciji ovisno o uporabi aplikacije. Odgovoran je za pripremu podataka te njihov izlazni oblik. U slučaju web aplikacije podaci se pripremaju za prikaz u obliku web stranica. Osim prikaza podaci se mogu pripremiti i za prijenos kao *Javascript Object Notation* (JSON) ili *Extensible Markup Language* (XML) objekti [7]. Osim prezentacije izlaznih podataka prezentacijski je sloj također odgovoran za primanje i temeljnu obradu ulaznih podataka. Obrada podataka uobičajeno je zadatak sloja modela ili kontrolnog sloja, no u slučaju obaveza poput ograničavanja ulaznih podataka i validacije obrazaca ova zadaća pripada prezentacijskom sloju.

### Controller

*Controller* je kontrolna jedinica MVC arhitekture. Korisnik komunicira s kontrolnom jedinicom putem prezentacijskog sloja te manipulira podacima koji ustraju kroz isti. Kontrolni se sloj sastoji od mnogih kontrolnih jedinica od kojih je svaka odgovorna za svoj dio web sjedišta (svaka web stranica ima svoju kontrolnu jedinicu). Ovaj sloj komunicira sa slojem modela kako bi dohvatio potrebne podatke iz baze podataka, a korisniku dostavlja podatke pripremljene prezentacijskim slojem.

## Modularizacija, API baze podataka

*Application Programming Interface* (API) je programsko rješenje koje služi komunikaciji između dva aplikacijska procesa. U primjeru (Programski isječak 5) vidljiva je implementacija kontrolne jedinice koja prima zahtjeve na pristupnoj točci „*/api/v1/students/{id}*“ (označenoj s „URL\_STUDENT“ i „URL\_ID“) te dostavlja JSON objekt koji sadrži podatak iz tablice u bazi podataka.

|  |
| --- |
| @RestController @RequestMapping(*URL\_STUDENT*) public class StudentController {  private final StudentService **studentService**;    public StudentController(StudentService *studentService*) {  this.**studentService** = *studentService*;   }   @GetMapping(*URL\_ID*)  public Student getById(@PathVariable String *id*) {  return **studentService**.getById(*id*);  } } |

Programski isječak 5 Primjer kontrolne jedinice koja dostavlja podatak u obliku JSON objekta

Aplikacija koja šalje zahtjev mora biti spremna interpretirati odgovor na zahtjev, tj. pretvoriti dobiveni objekt u njoj razumljiv oblik. Pretvorba tokom koje se objekt dijeli u manje dijelove za cilj interpretacije naziva se parsiranje. Glavna namjena JSON objekta nije biti čitljiv čovjeku, već imati strukturu podataka koju je lako parsirati.

Modularizacija je tehnika kojom programer izdvaja funkcionalnosti projekta u zasebne module. Moduli mogu biti samostalne aplikacije ili vanjske zavisnosti poput biblioteka. Ova je tehnika uporabljena kako bi se komunikacija s bazom podataka izdvojila u zaseban modul, što pridonosi skalabilnosti projekta. Na ovaj način sučelje baze podataka implementirano je odvojeno od aplikacije koja upravlja web sjedištem te u slučaju promjene temeljnog okvira aplikacije moguća je upotreba sučelja baze podataka bez modifikacija.

# Baza podataka

Baza podataka je strukturirano skladište podataka kojem se pristupa u svrhu trajne pohrane podataka te dobavljanja istih. Baze su podataka sveprisutne u suvremenim programskim rješenjima. U ovoj sekciji opisana je implementacija baze podataka u MySQL relacijskom DBMS-u što olakšava pristup podacima i povezuje podatke u koherentan skup.

## MyBatis sustav za komunikaciju s bazom podataka

Izravan pristup bazi podataka iz aplikacije pisane u programskom jeziku Java nije praktičan te postoje mnogi alati koji tu ulogu obavljaju učinkovitije. Jedan takav alat je MyBatis koji preslikava Java metode na SQL izjave. Ovime se olakšava i pojednostavljuje pristup bazi podataka nasuprot *Object-Role Modeling* alatima poput *Java Persistence API* koji izravno preslikavaju podatke iz baze podataka na Java objekte te zahtijevaju definiranje odnosa podataka u samim Java klasama [8]. MyBatis alatom također je moguće detaljno podešavanje SQL izjava po potrebama aplikacije.

Implementacija MyBatis sustava svodi se na dva dijela: deklaracija Java metoda kojom se definiraju ulazni i izlazni podaci te definiranje XML datoteka za preslikavanje (engl. *mapper*) kojima se podaci dobavljaju iz baze podataka te preslikavaju u Java objekte. Također postoji i neobavezan treći korak kojim programer može definirati način na koji se podaci iz baze pretvaraju u Java objekte ili atomske podatke uporabom posebnih pretvorbenih klasa (engl. *converter*). U slučaju da pretvorbene klase nisu definirane MyBatis sadrži rudimentarne pretvorbe koje su dovoljne za većinu primjena.

U primjeru (Programski isječak 6) vidljiva je implementacija MyBatis repozitorija gdje je klasom „*RefreshTokenRepository*“ moguće dobaviti objekt tipa *JwToken* za danu značku ili izbrisati podatak iz baze za dano korisničko ime.

|  |
| --- |
| @Mapper public interface *RefreshTokenRepository* extends *BaseRepository*<JwToken>{  JwToken findByToken(String *token*);  int deleteByUsername(String *username*); } |

Programski isječak 6 Primjer MyBatis repozitorija

|  |
| --- |
| <!DOCTYPE *mapper* PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"  "http://mybatis.org/dtd/mybatis-3-mapper.dtd"> <mapper *namespace*="tvz.ntpr.api.repo.RefreshTokenRepository">  <resultMap *id*="base" *type*="tvz.ntpr.api.auth.JwToken">  <id *column*="id" *jdbcType*="VARCHAR" *property*="id"/>  <result *column*="token" *jdbcType*="VARCHAR" *property*="token"/>  <result *column*="username" *jdbcType*="VARCHAR" *property*="username"/>  <result *column*="valid\_until" *jdbcType*="DATE" *property*="validUntil"/>  </resultMap>   <select *id*="findByToken" *parameterType*="String" *resultMap*="base">  SELECT \*  FROM `tokens`  <where>  `token` = #{token,jdbcType=VARCHAR}  </where>  </select>   <delete *id*="deleteByUsername" *parameterType*="String">  DELETE  FROM `tokens`  <where>  `username` = #{username,jdbcType=VARCHAR}  </where>  </delete> </mapper> |

Programski isječak 7 Primjer MyBatis mapper datoteke

U slijedećem primjeru (Programski isječak 7) vidljiva je izvedba pripadajuće datoteke za preslikavanje. Unutar „*mapper*“ objekta definirana su tri niža objekta. Prvi objekt „*resultMap*“ služi agregaciji rezultata pretrage baze podataka u Java objekt. Druga dva objekta definiraju SQL izjave koje se izvršavaju pozivom metoda deklariranih u repozitoriju unutar aplikacije.

# Povezivanje modula

Moduli projekta realizirani su u obliku dvije zasebne aplikacije. Komunikacija između njih vrši se pomoću HTTP zahtjeva. Kao takva moguća je preko lokalne mreže (ako se obje aplikacije izvršavaju na istom računalu ili mreži) ili preko interneta (u slučaju da se aplikacije nalaze na dva različita udaljena poslužitelja). Prvi modul je aplikacija koja korisniku poslužuje web stranice („*core*“ modul), a drugi modul je aplikacija s kojom se vrši komunikacija putem programskih sučelja u svrhu dobavljanja podataka iz baze podataka („*api*“ modul).

## HTTP zahtjevi

HTTP temeljni je protokol kojim se vrši komunikacija putem računalne mreže. Definira nekoliko metoda zahtjeva prikazanih u tablici (Tablica 1):

Tablica 1 Najčešće HTTP metode zahtjeva

|  |  |
| --- | --- |
| Metoda | Namjena |
| GET | dohvat podataka |
| POST | slanje podataka |
| PUT | ažuriranje postojećih podataka |
| DELETE | brisanje podataka |

HTTP zahtjev sastoji se od tri dijela: metoda zahtjeva, zaglavlje i sadržaj. U zaglavlje zahtjeva spremaju se *Uniform Resource Locator* (URL) traženog sadržaja te informacije o zahtjevu i sadržaju zahtjeva. U sadržaj zahtjeva spremaju se podaci koji se šalju poslužitelju [9].

Moguće je implementirati više funkcionalnosti na jednoj pristupnoj točci uporabom ovih HTTP metoda. U primjeru (Programski isječak 8) vidljiva je implementacija kontrolne jedinice koja koristi različite HTTP metode za izvršavanje različitih funkcija u bazi podataka (čitanje, pisanje, ažuriranje i brisanje) uporabom iste pristupne točke (osim DELETE metode koja na pristupnu točku dodaje „*/{id}*“)

|  |
| --- |
| @RestController @RequestMapping(*URL\_PROFESSOR*) public class ProfessorController {  private final ProfessorService **professorService**;  private final CourseService **courseService**;   public ProfessorController(ProfessorService *professorService*,  CourseService *courseService*) {  this.**professorService** = *professorService*;  this.**courseService** = *courseService*;  }   @GetMapping  public *List*<Professor> getAll() {  return **professorService**.getAll();  }   @PostMapping  public void saveProfessor(@RequestBody Professor *professor*) {  **professorService**.create(*professor*);  }   @PutMapping  public void updateProfessor(@RequestBody Professor *professor*) {  **professorService**.update(*professor*);  }   @DeleteMapping(*URL\_ID*)  public void deleteGrade(@PathVariable String *id*) {  for (Course course : **courseService**.getByProfessorId(*id*)) {  course.setProfessor(null);  **courseService**.update(course);  }  **professorService**.deleteById(*id*);  } } |

Programski isječak 8 Primjer implementacije različitih funkcionalnosti na istoj pristupnoj točci

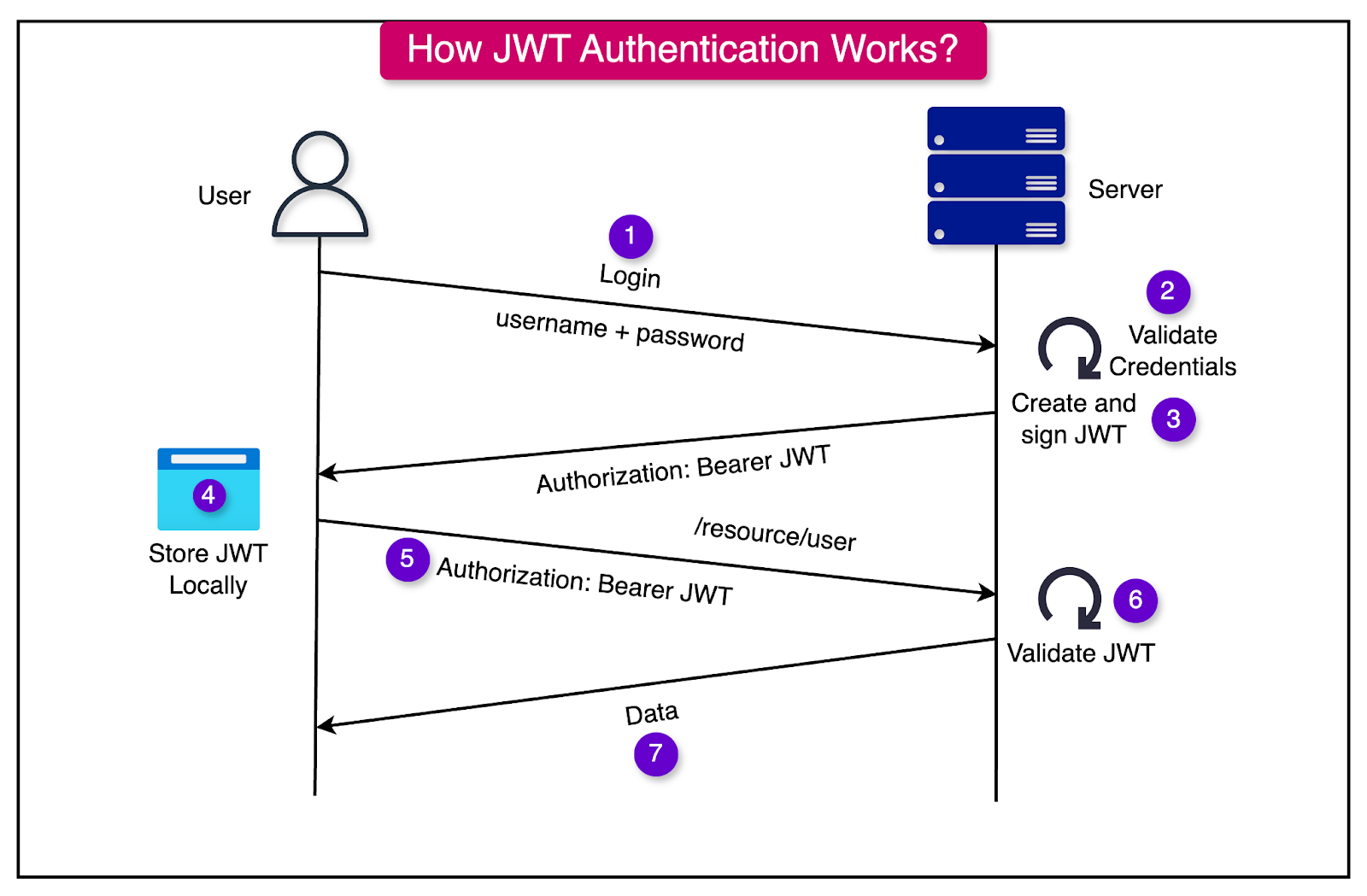
## Autentifikacija pristupa

Budući da je sučelje baze podataka izloženo internetu javlja se potreba da to sučelje bude osigurano od neautoriziranog pristupa. Za ovaj zadatak postoje dvije glave metodologije rješenja: korisničke značke i ključevi.

### JSON Web Token

Najčešća implementacija korisničke značke je JSON Web Token (JWT) koja se temelji na dvije vrste znački: značka za pristup (engl. *access token*) i značka za osvježivanje (engl. *refresh token*). Značka ima oblik prividno nasumičnih znakova, no unutar njih šifrirane su informacije o vlasniku, roku valjanosti te vlasnikovim privilegijama. Svaka JWT značka sadrži zaglavlje, sadržaj i potpis [10].

Princip rada ovog rješenja je registracija značke za osvježivanje za svakog korisnika u trenutku prijave. Značka za osvježivanje najčešće se sprema u bazu podataka te je jedinstvena za svakog korisnika. Svrha ovog podatka je „osvježiti“ drugu vrstu značke: značku za pristup. Značka za pristup koristi se za autentificirani pristup nekom resursu te ima iznimno kratak rok valjanosti.

Komunikacija između klijenta i poslužitelja vrši se u nekoliko koraka. U trenutku prijave klijentu se izdaje značka za osvježivanje. Klijent prije svakog zahtjeva prvo šalje svoju značku za osvježivanje koja se na poslužitelju provjerava te poslužitelj klijentu izdaje novu značku za pristup. Klijent tada šalje zahtjev za resursom te uz njega izdanu značku za pristup. Nakon provjere značke za pristup poslužitelj klijentu šalje traženi resurs. Dijagram komunikacije ovjerene JWT značkama vidljiv je na slici (Slika 4):

Slika 3 Dijagram komunikacije uz JWT ovjeru

(Izvor: <https://blog.bytebytego.com/p/mastering-modern-authentication-cookies>, pristupljeno: 15.8.2025.)

### API ključevi

Korisnički ključ, češće zvan API ključ, je generirana statička vrijednost vezana uz određeni korisnički račun. Javne API usluge često ga koriste za potrebe regulacije prometa i pružanja plaćenih usluga poput dodatnih funkcionalnost API-a. U usporedbi s korisničkim značkama, ovaj je sustav jednostavnije izvedbe. Budući da je API ključ vrijednost koja se ne mijenja dovoljno je implementirati sustav na strani klijenta koji šalje osobni API ključ u sklopu svakog zahtjeva prema API-u. Sustav ovjere API ključeva se uglavnom koristi u automatiziranim, tzv. *pipeline,* sustavima.

## Prijenos podataka između aplikacija

Aplikacije implementirane na različitim poslužiteljima komuniciraju mrežnim protokolima, naročito HTTP-om. HTTP je standardni protokol mrežne komunikacije zbog svoje svestranosti i prilagodljivosti. Označava ga mrežni prefiks „*http://*“. Također postoji i inačica usredotočena na sigurnost komunikacije označena mrežnim prefiksom „*https://*“. Ovim je protokolom moguć pouzdan prijenos datoteka različitih formata od kojih je najčešći HTML [11].

### Serijalizacija i deserijalizacija

|  |
| --- |
| {  **"year"**: 2025,  **"month"**: 8,  **"day"**: 15,  **"hour"**: 12,  **"minute"**: 6,  **"seconds"**: 19,  **"milliSeconds"**: 652,  **"dateTime"**: "2025-08-13T12:06:19.6523518",  **"date"**: "08/13/2025",  **"time"**: "12:06",  **"timeZone"**: "Europe/Zagreb",  **"dayOfWeek"**: "Friday",  **"dstActive"**: true } |

Kako bi prijenos Java objekata preko HTTP protokola bio moguć objekte je potrebno pretvoriti u oblik povoljan za prijenos. Pretvorba memorijskih objekata u oblik povoljan za pohranu ili prijenos naziva se „serijalizacija“. Pretvorba iz dobivenog oblika natrag u memorijske objekte naziva se „deserijalizacija“.

Programski isječak 9 JSON izlaz API-a za dohvat trenutnog vremena

(Izvor: <https://timeapi.io/api/time/current/zone?timeZone=Europe%2FZagreb>, pristupljeno: 15.8.2025.)

Jedna od najčešćih implementacija serijalizacije za prijenos HTTP-om je pretvorba u JSON oblik zbog jednostavnosti i učinkovitosti. Java objekti, koji se sastoje od atributskih vrijednosti se serijalizacijom u JSON format pretvaraju u niz znakova koji sadrži parove ključeva i vrijednosti (engl. *key-value pair*). Ključevi označavaju imena članova klase objekta, a vrijednosti su stvarne vrijednosti tih članova. JSON format također podržava nizove vrijednosti s jednim ključem. Serijalizaciju, izlaganjem Java objekata na pristupne točke, automatski vrši *Spring* *Framework*. Primjer izlazne vrijednosti u JSON formatu vidljiv je u prikazu (Programski isječak 9).

Zadnji korak prijenosa je deserijalizacija. Deserijalizacija se vrši u aplikaciji koja prima vrijednosti iz API-a. Izlazne vrijednosti API-a u aplikaciju dostižu u obliku *HTTP response* objekta. Iz tog objekta vadi se sadržaj te se sprema u *String* objekt kao niz znakova. Kako bi se taj niz parsirao u Java objekt potrebna je implementacija pretvorbene metode. U svrhu ove pretvorbe postoje alati za interpretiranje JSON podataka poput sučelja *Jackson*.

|  |
| --- |
| public class JsonParser {  private static final ObjectMapper *OBJECT\_MAPPER* =  new ObjectMapper().registerModule(new JavaTimeModule());   public static <*T*> *T* parseIntoObject(String *json*, Class<*T*> *clazz*)  throws Exception {  if (*json* == null || *json*.isEmpty()) return null;  return *OBJECT\_MAPPER*.readValue(*json*, *clazz*);  }   public static <*T*> *List*<*T*> parseIntoList(String *json*, Class<*T*> *clazz*)  throws Exception {  if (*json* == null || *json*.isEmpty()) return new ArrayList<>();  return *OBJECT\_MAPPER*.readValue(*json*, *OBJECT\_MAPPER* .getTypeFactory()  .constructCollectionType(*List*.class, *clazz*));  } } |

Programski isječak 10 Primjer implementacije parsiranja JSON podataka uporabom Jackson alata

U primjeru (Programski isječak 10) vidljiva je implementacija deserijalizacije iz JSON formata u Java objekte uporabom Jackson API-a. Definirane su dvije metode. Prva metoda služi parsiranju u pojedinačan objekt, a druga parsiranju u zbirku objekata. Ulazni podaci za obje metode su niz znakova koji tvore JSON objekt i ciljana klasa, a izlazni podaci su objekt ili zbirka objekata ciljane klase.

# Generacija dokumenata za preuzimanje na korisnikovo računalo

Gotovo svaka suvremena web aplikacija korisniku nudi mogućnost preuzimanja datoteka. Ove datoteke mogu biti slike, dokumenti, komprimirane arhive ili neka druga važna stavka koju korisnik ima razloga htjeti trajno spremiti na svoje računalo. U ovoj će se sekciji rada razraditi tema generacije i izlaganja dokumenata na pristupne točke za preuzimanje.

## Generacija PDF dokumenata

*Portable Document Format* (PDF) je format datoteka koji se koristi za prikaz tekstualnih dokumenata. Ovo je najčešći oblik dokumenata na internetu iz razloga što je lagan za uporabu, sveobuhvatan i neovisan o platformi, što znači da oblikovanje dokumenta (korišteni fontovi, raspored elemenata, korištene boje, itd.) ostaje nepromjenjivo neovisno o programskoj podršci koja ga prikazuje [12].

Jednostavna implementacija generacije ovakvog dokumenta je tzv. *web site scraper*, tj. alat za izvlačenje podataka s web stranica. Uporabom ovakvog alata moguće je sadržaj web stranice pretvoriti u PDF elemente te ih svrstati u PDF datoteku. Ovim pristupom nije potrebna ponovna obrada podataka iz baze podataka zbog toga što su podaci već pripremljeni za prikaz uporabom prezentacijskog sloja web aplikacije.

Princip generacije PDF dokumenta svodi se na izvlačenje HTML koda web stranice te pretvorbu HTML elemenata u PDF elemente očuvanjem njihovog okvira i rasporeda.

|  |
| --- |
| public static File scrapeHtmlToPdf(String *url*) throws IOException {  *Path* jarDirectory =  Paths.*get*(System.*getProperty*("java.class.path"))  .toAbsolutePath()  .getParent();  *Path* studentReportsDirectory = jarDirectory.resolve(*OUTPUT\_DIRECTORY*);  String timestamp =  String.*valueOf*(  *TIME\_API*.getCurrentTime().toEpochSecond(ZoneOffset.*UTC*));  File output =  Paths.*get*(  studentReportsDirectory.toString(),  "student\_report(" + timestamp + ").pdf")  .toFile();  Files.*createDirectories*(studentReportsDirectory);  org.jsoup.nodes.Document htmlDoc = Jsoup.*connect*(*url*).get();   PdfWriter writer = new PdfWriter(output);  PdfDocument pdfDoc = new PdfDocument(writer);  Document pdfDocument = new Document(pdfDoc);   String title = htmlDoc.title();  pdfDocument.add(  new Paragraph(title)  .setFont(FontSupplier.*BOLD*)  .setFontSize(18)  .setTextAlignment(*TextAlignment*.*CENTER*));   Elements bodyChildren = htmlDoc.body().children();  for (Element element : bodyChildren)  *convertHtmlElementToPdf*(element, pdfDocument);   pdfDocument.close();   return output; } |

Programski isječak 11 Metoda za generaciju PDF dokumenta

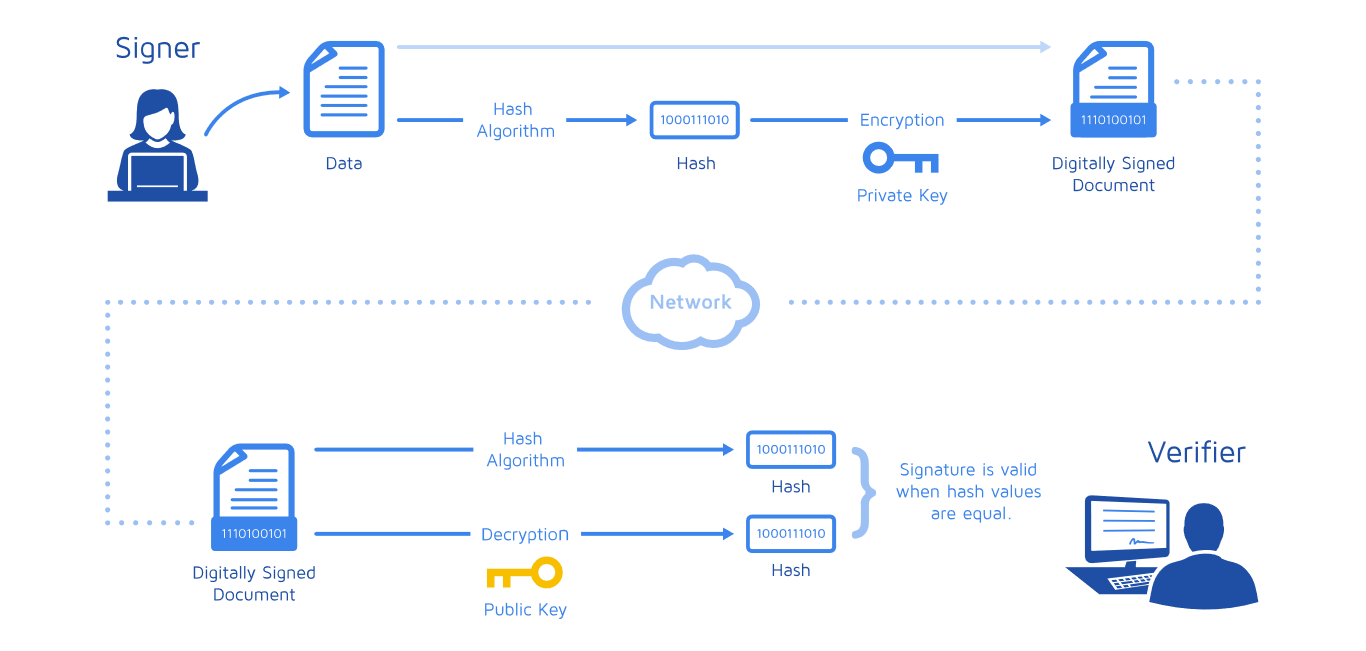
U primjeru (Programski isječak 11) prikazana je Java metoda koja dohvaća HTML kod web stranice s ulaznog URL-a te ga uporabom *iText* alata pretvara u PDF dokument. *iText* je alat za manipuliranje PDF datotekama putem Java koda. U primjeru se vrše tri koraka generacije datoteke: definiranje izlazne lokacije, podjela HTML koda na pojedinačne elemente te pretvorba svakog HTML elementa u PDF element. Zadnji korak vrši se posebnom metodom vidljivom u slijedećem primjeru (Programski isječak 12).

|  |
| --- |
| private static void convertHtmlElementToPdf(Element *element*,  Document *pdfDocument*) {  switch (*element*.tagName()) {  case "p":  *pdfDocument*.add(  new Paragraph(*element*.text())  .setFont(FontSupplier.*REGULAR*)  .setFontSize(12)  .setMarginTop(5));  break;  case "h1":  *pdfDocument*.add(  new Paragraph(*element*.text())  .setFont(FontSupplier.*BOLD*)  .setFontSize(18)  .setMarginTop(15));  break;  case "h2":  *pdfDocument*.add(  new Paragraph(*element*.text())  .setFont(FontSupplier.*BOLD*)  .setFontSize(16)  .setMarginTop(12));  break;  case "table":  Table table = new Table(*element*.select("thead th").size());  Elements rows = *element*.select("tr");  for (Element row : rows) {  Elements cols = row.select("td, th");  for (Element col : cols)  table.addCell(  new Cell().add(  new Paragraph(col.text()).setFontSize(10)));  }  *pdfDocument*.add(table.setMarginTop(10));  break;  case "div":  Elements divChildren = *element*.children();  for (Element child : divChildren)  *convertHtmlElementToPdf*(child, *pdfDocument*);  break;  default:  break;  } } |

Programski isječak 12 Metoda za pretvorbu HTML elemenata u PDF elemente

U ovom programskom isječku vidljiv je način pretvorbe elemenata. Ako se radi o tekstualnom elementu („*p*“, „*h1*“, „*h2*“), on se u PDF dokument dodaje u obliku stiliziranog odlomka kojemu svojstva ovise o tipu elementa. U slučaju tablice izrađuje se objekt tipa „*Table*“ u koji se umeću podaci iz elemenata tipa „*th*“ i „*td*“ koji označuju ćelije tablice. Ako je proslijeđeni element tipa „*div*“, koji se u HTML-u koristi za ugnježđivanje, njegovi se pod‑elementi prosljeđuju u rekurzivni poziv metode kako bi se došlo do osnovnih elemenata. U protivnom, element koji nije označen niti jednim od navedenih tipova se zanemaruje.

## Digitalni potpis

Digitalni je potpis kriptografska tehnika verifikacije autentičnosti digitalnih dokumenata. Temelji se na kriptografiji javnog ključa (engl. *public-key cryptography*) čime se osigurava nepromjenjivost potpisanih podataka te njihova autentičnost [13].

Slika 4 Dijagram procesa digitalnog potpisivanja i verifikacije potpisa

(Izvor: <https://meruja.medium.com/digital-signature-generation-75cc63b7e1b4>, pristupljeno: 3.9.2025.)

U dijagramu (Slika 4) vidljiv je proces digitalnog potpisivanja. Dokument prolazi kroz algoritam sažimanja (engl. *hash algorithm*) te se dobivena vrijednost šifrira potpisnikovim privatnim ključem čime se dobiva jedinstvena vrijednost koja se može priložiti na dokument ili spremiti kao posebna datoteka. Osoba koja želi verificirati potpis to vrši potpisnikovim javnim ključem. Javni se ključ koristi za dešifriranje kriptografskog sažetka. Verifikator tada dokument provodi kroz isti algoritam sažimanja te uspoređuje rezultirajuće vrijednosti. Ako se podudaraju, to znači da dokument nije mijenjan od trenutka potpisivanja te da je potpisnikov privatan ključ uistinu korišten za šifriranje sažetka. Ovim činjenicama uspostavlja se punovažnost i autentičnost dokumenta.

Ako se web aplikacija koristi za generaciju službenih dokumenata poput izvještaja ili potvrda važno je te dokumente osigurati protiv krivotvorenja ili neslužbenih izmjena uporabom sustava za digitalno potpisivanje.

## Preuzimanje datoteka

Kako bi korisnik aplikacije datoteke mogao spremiti na svoje računalo potrebno je datoteke izložiti za preuzimanje na pristupnu točku. Ovaj je rad do sada pokrio dva načina oblikovanja izlaznih podataka – HTML kod i JSON objekti, no ovi oblici nisu povoljni za prijenos datoteka. Kako bi omogućili prijenos datoteka putem interneta potreban je način slanja neobrađenih bajtova datoteke preko HTTP protokola. Ovo se postiže pisanjem podataka izravno u izlazni tok (engl. *output stream*) objekta HTTP odgovora.

|  |
| --- |
| @GetMapping(*URL\_ID*) public void download(@PathVariable String *id*, *HttpServletResponse response*)  throws IOException {  Report report = **reportService**.findByStudent(*id*);  if (report == null  || report.getData() == null  || report.getSignature() == null) {  *response*.setStatus(*HttpServletResponse*.*SC\_NOT\_FOUND*);  }   String filename = report.getFileName();  byte[] data = report.getData();  byte[] signature = report.getSignature();   *response*.setHeader(HttpHeaders.*CONTENT\_TYPE*, "application/zip");  *response*.setHeader(HttpHeaders.*CONTENT\_DISPOSITION*, "attachment");  *response*.setHeader("filename", filename);   try (ZipOutputStream os = new ZipOutputStream(*response*.getOutputStream())) {  ZipEntry dataEntry = new ZipEntry(filename);  os.putNextEntry(dataEntry);  os.write(data);  os.closeEntry();   ZipEntry signatureEntry = new ZipEntry(filename + ".p7s");  os.putNextEntry(signatureEntry);  os.write(signature);  os.closeEntry();  } } |

Programski isječak 13 Primjer kontrolne jedinice koja poslužuje tok bajtova

U gornjem primjeru (Programski isječak 13) vidljiva je kontrolna jedinica koja ne poslužuje obrađeni podatak, već dinamički puni izlazni tok bajtova podacima. U metodu je proslijeđen objekt tipa „*HttpServletResponse*“ kojim se ostvaruje izravan pristup HTTP odgovoru. U primjeru je također ostvarena funkcionalnost generacije *ZIP* arhive kao komprimiranog formata podataka za lakši prijenos preko mreže. Ovime se podaci ujedinjuju u jednu datoteku čija je veličina manja od ukupne veličine njenih dijelova.

Slanjem zahtjeva na pristupnu točku ovakve metode korisnik prima podatke u trenutku kada se oni dinamički upisuju na izlazni tok HTTP odgovora, pri čemu se sadržaj prenosi postupno, čime je omogućen primitak podataka prije nego što je cijeli sadržaj spreman, tj. bez potrebe da se cijeli odgovor prethodno generira u memoriji.

# Zaključak

Web aplikacije u današnje su vrijeme neizostavan dio svake organizacije. Pomažu nam u obradi podataka, izvršavanju poslova, vođenju evidencije i mnogim drugim svrhama. Suvremeni programski proizvodi gotovo uvijek imaju inačicu za korištenje putem web preglednika. Primjeri ovakvih aplikacija su *Microsoft 365 Online*, *Google Workspace*, *Netflix* i mnogi drugi. Trend premještanja usluga klasičnih *desktop* aplikacija u web aplikacije javlja se zbog pristupačnosti i lakoće održavanja.

U ovom radu opisane su tehnologije i metode izrade jedne ovakve web aplikacije. Rad je bio podijeljen na tri dijela: opis tehnologija potrebnih za izradu, izrada korisničkog sučelja te izrada poslovne logike i povezivanje modula aplikacije. U prvom dijelu navedene su i opisane tehnologije korištene za izradu aplikacije, odgovorena su pitanja vezana uz odabir tehnologija te su dani primjeri uporabe. U drugom dijelu opisana je izvedba korisničkog sučelja te je utvrđena važnost svrsishodnog dizajna istog. Treći se dio bavio najopširnijom temom ovoga rada, a to je poslovna logika i svi procesi aplikacije koji se izvršavaju u pozadini. U ovom su dijelu bile opisane specifične implementacije tehnologija navedenih u prvom dijelu rada te njihove svrhe i ciljevi. Također, bile su opisane metode komunikacije između različitih modula aplikacije, sustav komunikacije s bazom podataka i autentifikacija pristupa osjetljivim podacima.

Osim tehničkog aspekta, rad upućuje i na važnost razumijevanja potreba krajnjih korisnika kao i na brigu o budućoj prilagodljivosti i skalabilnosti proizvoda. U ovom pogledu web aplikacije nisu samo programski proizvod, već i strateški alati za unaprjeđenje poslovnih procesa i učinkovitosti radnika.

Analizom trenutnih trendova može se zaključiti da će web aplikacije u budućnosti imati još veći značaj, a poznavanje metoda i tehnologija njihove izrade već i danas predstavlja ključnu prednost u kompetenciji suvremenih programskih inženjera.

# Popis literature

|  |  |
| --- | --- |
| [1] | M. R. Hoffman, »Pre Java 1.0,« The Java Version Almanac, [Mrežno]. Dostupno: https://javaalmanac.io/jdk/pre1.0/. [Pokušaj pristupa 13. 1. 2025.]. |
| [2] | J. Hoeller, »Spring Framework 6.0 goes GA,« Spring blog, [Mrežno]. Dostupno: https://spring.io/blog/2022/11/16/spring-framework-6-0-goes-ga. [Pokušaj pristupa 14. 1. 2025.]. |
| [3] | S. Nicoll, »spring-boot,« Github, [Mrežno]. Dostupno: https://github.com/spring-projects/spring-boot. [Pokušaj pristupa 14. 1. 2025.]. |
| [4] | D. Benyon, Designing user experience, London: Pearson, 2019. |
| [5] | M. Springett i T. French, »User experience and it's relationship to usability,« u *Towards a UX Manifesto*, Lancaster, 2007. |
| [6] | S. Wanigasooriya, »A Deep Dive into Spring MVC: Architecture, Request Flow, and Modern Best Practices,« Medium, [Mrežno]. Dostupno: https://sithara-wanigasooriya.medium.com/a-deep-dive-into-spring-mvc-architecture-request-flow-and-modern-best-practices-7fc5a9d510ed. [Pokušaj pristupa 3. 8. 2025.]. |
| [7] | F. Rosenberg, F. Curbera, M. J. Duftler i R. Khalaf, »Composing RESTful Services and Collaborative Workflows: A Lightweight Approach,« *IEEE Internet Computing,* svez. 12, br. 5, pp. 24-31, 2008. |
| [8] | T. Halpin, Object-role modeling fundamentals: a practical guide to data modeling with ORM, Sedona: Technics Publications, 2015. |
| [9] | D. Nolan i D. T. Lang, XML and web technologies for data sciences with R, New York: Springer, 2014. |
| [10] | P. Mahindrakar, »Insights of JSON Web Token,« *International Journal of Recent Technology and Engineering,* svez. 8, br. 6, pp. 1707-1710, 2020. |
| [11] | D. Gourley i B. Totty, HTTP: the definitive guide, Sebastopol: O'Reilly Media, Inc., 2002. |
| [12] | S. L. Abrams i S. P. Levenson, »PDF/A: An Electronic Document File Format for Long-Term Preservation,« u *Archiving Conference*, San Antonio, 2004. |
| [13] | S. Goldwasser i M. Bellare, *Lecture notes on cryptography,* Cambridge: Massachusetts Institute of Technology, 1999. |