**VELEUČILIŠTE SUVREMENIH INFORMACIJSKIH TEHNOLOGIJA**

**STRUČNI PRIJEDIPLOMSKI STUDIJ INFORMACIJSKIH TEHNOLOGIJA**

**ZAGREB**

**ZAVRŠNI RAD**

**NAZIV ZAVRŠNOG RADA**

**Sven Gotal**

**Zagreb, studeni 2023.**

Ova i naslovna stranica dobiju se u Referadi

Sadržaj

[1. UVOD 8](#_Toc182752631)

[1.1. Motivacija Projekta 8](#_Toc182752632)

[1.2. Logička Struktura 8](#_Toc182752633)

[2. OPIS PROBLEMA 10](#_Toc182752634)

[2.1. Struktura Biblije 10](#_Toc182752635)

[2.2. Struktura Aplikacije 10](#_Toc182752636)

[2.3. Strukturno Rješenje 10](#_Toc182752637)

[3. ANALIZA ALATA I MODELA 11](#_Toc182752638)

[3.1. Java Spring Boot Framework 11](#_Toc182752639)

* 1. [3.1.1. Struktura Paketa JSB 11](#_Toc182752640)
  2. [3.1.2. Struktura Mapa Datoteka JSB 11](#_Toc182752641)

[3.2. Model View Controller Pattern 12](#_Toc182752642)

[3.3. H2 Persistence 12](#_Toc182752643)

[3.4. Alati Projekta 12](#_Toc182752644)

* 1. [3.4.1. Maven 12](#_Toc182752645)
  2. [3.4.2. Eclipse IDE 13](#_Toc182752646)
  3. [3.4.3. Spring Initializr 13](#_Toc182752647)
  4. [3.4.4. Git 14](#_Toc182752648)

[3.5. Teorijski Model 14](#_Toc182752649)

* 1. [3.5.1. Modeliranje Biblijskih Modela 14](#_Toc182752650)
  2. [3.5.2. Modeliranje Korisničkih Modela 15](#_Toc182752651)

[4. IZRADA APLIKACIJE 17](#_Toc182752652)

[4.1. Domain/Modeli 17](#_Toc182752653)

* 1. [4.1.1. Bible 17](#_Toc182752654)
  2. [4.1.2. Book 19](#_Toc182752655)
  3. [4.1.3. Chapter 20](#_Toc182752656)
  4. [4.1.4. Verse 20](#_Toc182752657)
  5. [4.1.5. Commentary 21](#_Toc182752658)
  6. [4.1.6. User i UserRoles 22](#_Toc182752659)

[4.2. Repository Interface 24](#_Toc182752660)

* 1. [4.2.1. ChapterRepository 25](#_Toc182752661)
  2. [4.2.2. VerseRepository 26](#_Toc182752662)
  3. [4.2.3. UserRepository 26](#_Toc182752663)
  4. [4.2.4. CommentaryRepository 26](#_Toc182752664)
  5. [4.2.5. Zaključak na Repozitorije 27](#_Toc182752665)

[4.3. Kontroleri 27](#_Toc182752666)

[LITERATURA 30](#_Toc182752667)

[SAŽETAK 31](#_Toc182752668)

[SUMMARY 32](#_Toc182752669)

# **POPIS KODOVA**

[Kod 1: Primjer Bible klase 17](#_Toc182749884)

[Kod 2: Primjer Book klase 18](#_Toc182749885)

[Kod 3: Primjer Chapter klase 19](#_Toc182749886)

[Kod 4: Primjer Verse klase 20](#_Toc182749887)

[Kod 5: Primjer Commentary klase 21](#_Toc182749888)

[Kod 6: Primjer User klase 22](#_Toc182749889)

[Kod 7: Primjer UserRoles enum-a 22](#_Toc182749890)

[Kod 8: Primjer Definicije Repozitorija 24](#_Toc182749891)

[Kod 9: Definicija BibleRepository 24](#_Toc182749892)

[Kod 10: Definicija BookRepository 24](#_Toc182749893)

[Kod 11: Primjer ChapterRepository 24](#_Toc182749894)

[Kod 12: Deklaracija UserRepository 25](#_Toc182749895)

[Kod 13: Delaracija CommentaryRepository 26](#_Toc182749896)

**POPIS SLIKA**

# UVOD

## Motivacija Projekta

Web projekt „Online Biblijski Komentari” (web link: https://www.biblijaija.com) odnosno web aplikacija „BIBLIJA&ja” osmišljena je sa željom kako bi se ljudima približilo Sveto Pismo s pomoću modernog medija - Interneta. Temeljna ideja jest da aplikacija bude alat dostupan svima na svakom mjestu u Hrvatskom govornom području kako bi mogli učiti iz teksta Svetog Pisma (Biblije) te komentara koje ostavljaju drugi korisnici te kako bi konačno i sami mogli dati doprinos i svoje viđenje na određenu cjelinu koja se obrađuje.

Također je želja autora da kroz ovakav projekt potakne korisnike na čitanje Svetog Pisma i njegovo proučavanje i promišljanje o pročitanom te uz čitanje ostalih komentara zadobije dublji uvid u mnoga značenja koja Biblija sadržava. Današnji ubrzani način života nas ne potiče da čitamo s razumijevanjem. Iako puno čitamo, malo ili uopće ne promišljamo o pročitanom i propuštamo mnoga saznanja, uvide i iskustva drugih. Iako cilj ovog rada nije društvene već tehničke prirode motivacija za izradom ovog projekta je ipak društvenog karaktera.

## Logička Struktura

Kako je sama Biblija organizirana u mjerljive jedinice i podjele, u navodu: Zavjet, Knjiga, Poglavlje/Glava te potom Stih tako je i aplikacija podložna takvoj podjeli. Izuzetak je Zavjet koji u samoj aplikaciji nema težinu niti je za korisnika takva podjela bitna. Glavne jedinice mjere u aplikaciji su Knjiga i Poglavlje koje također služe za navigaciju po tekstu Biblije i na koje je komentar korisnika vezan. Korisnik i Komentar su strukture aplikacije koje su neodvojive jedna od druge te imaju ulogu kao što ima struktura jednog internetskog foruma. Tako je ova aplikacija amalgam struktura čitanja knjige online (npr: https://www.readanybook.online) i foruma ali s fokusom na književni dio gdje se komentari i korisnici fokusiraju na poglavlja teksta Biblije umjesto na razne teme i diskusije odnosno imaju jednu temu umjesto više njih. Također korisnici mogu besplatno i čitati Bibliju što je još jedno svojstvo ove aplikacije. Tijekom čitanja poglavlja bit će prikazani i javno dostupni komentari korisnika.

Korisnik može birati hoće li njegov/njezin komentar biti javno vidljiv te stoga i prikazan uz samo poglavlje Biblije na index stranici ili će biti privatan te tako ostati vidljiv samo tom korisniku u odjelu „Moji komentari”. Naime nije moguće ostavljati anonimne komentare te je prvi korak, izrada korisničkog računa ali anonimni korisnici odnosno „guest” mogu čitati tekst i javne komentare koji će biti prikazani uz knjigu te poglavlje knjige koje anonimni korisnik odabere.



Slika 1. Index BIBLIJA&ja

# OPIS PROBLEMA

## Struktura Biblije

Kako je već spomenuto Biblija ima jasno razrađenu podjelu na Zavjet, Knjiga (66 knjiga sveukupno), Poglavlje te potom najmanja mjerna jedinica Stih. Taj predložak je korišten i u aplikativnom rješenju izuzev jedinice Zavjet (Stari ili Novi). Kako je cilj aplikacije da korisnik ima mogućnost dodavanja komentara na tekst Biblije zbog pojednostavljenja problema komentar se veže na cijelo poglavlje radije negoli na neki određeni stih ili na primjer cijelu knjigu što bi bila prevelika generalizacija. Također je cilj da se tekst čita u cjelini poglavlja radije negoli pojedinog stiha ili općenito cijele knjige što bi učinilo sučelje aplikacije nepreglednim. Komentar će također biti prikazan na index stranici uz poglavlje.

## Struktura Aplikacije

Kako bi zadatak bio postignut aplikacija je morala slijediti strukturu subjekta odnosno teksta koji se obrađuje. Tako u programu postoje identične strukture odnosno objekti koji imaju ulogu knjige, poglavlja itd. za kontrolu teksta te objekti koji imaju ulogu komentara, korisnika i korisničkih rola koji imaju zadatak kontrolu nad istoimenim domenama unutar aplikacije.

## Strukturno Rješenje

Pravilno strukturiranje odnosa objekata odnosno domena unutar aplikacije je od najveće važnosti kako bi se postigli svi ciljevi i zahtjevi te su stoga ti odnosi morali biti pravilno postavljeni. Primjer one-to-many odnosa unutar aplikacije su Knjiga sadrži Poglavlja koja sadrže Stihove, Komentar koji sadrži Korisnike te povratnu referencu na Poglavlje. Poglavlje također sadrži one-to-many referencu i na Komentar kako bi tijekom prikaza poglavlja teksta bio prikazan i komentar.

# ANALIZA ALATA I MODELA

## Java Spring Boot Framework

Java Spring Boot (u nastavku JSB) je framework odnosno skup programskih biblioteka, alata, API-ja i auto-konfiguracija s vodiljom "convention over configuration” što znači da se radi o korisnom alatu za izradu samoodrživih web aplikacija opće upotrebe. Početna konfiguracija servera koji je sadržan u frameworku te konfiguracije sigurnosti pristupa i baze podataka je već postavljena te postaje dio fokusa tek u kasnijim fazama kompleksnijih projekata pa je JSB vrlo dobar alat i za učenje. Sami framework je implementiran u Java programskom jeziku te se Java koristi u izradi aplikacije. Server i sve potrebne konfiguracije potrebne kako bi JSB izradio standalone Java aplikaciju su apstrahirane od developera te nije potrebno baviti se tim aspektom u detalje u jednostavnijim aplikacijama. JSB koristi „opinionated“ pristup, što znači da se moraju poštivati određene konvencije tokom izrade projekta. Jedna od tih konvencija je da struktura mapa koju je framework definirao mora biti poštivana.

### Struktura Paketa JSB

JSB ima strogo definiranu strukturu paketa (package u Java, namespace u C# ili C++ npr.) koju je potrebno pratiti zbog osnovne konfiguracije JSB i Java programskog jezika općenito. Uobičajeni sustav imenovanja paketa u Java jest: domena.tvrtka.biblioteka ali se može prilagoditi potrebama pa stoga i JSB slijedi slične standarde ali manje fleksibilno.

JSB sadrži ulaznu točku odnosno početnu klasu koja se mora nalaziti uvijek u vršnom paketu aplikacije te svi drugi paketi s kojima aplikacija upravlja moraju biti u paketima ispod. Primjer u aplikaciji BIBLIJA&ja:

* Vršni paket koji sadrži Aplication.java klasu:

„package com.java.crv.BibleReaderCommentary;”

* npr. Paket koji sadrži domene:

„package com.java.crv.BibleReaderCommentary.domain;”

### Struktura Mapa Datoteka JSB

Također je i struktura mapa strogo definirana. Kako je projekt definiran na filesystemu će utjecati na rad aplikacije. Sve klase moraju biti definirane u „src/main/java” dok svi resursi aplikacije (što uključuje cijelu View domenu) moraju biti sadržani u „src/main/resources” mapi. Uz „src/” mapu postoji i „bin/” mapa u kojoj su sadržani svi .class objekti odnosno kompajlirane klase u byte-code koje JVM čita i interpretira te izvršava. Mapa „target/” je još jedna mapa sadržana u folderu projekta te sadrži auto generirane .class objekte, konfiguracije i instance View resursa te je dom .jar datoteka koje se mogu generirati s pomoću Maven ili Gradle alata, .jar datoteka je aplikacija formatirana u java arhiv koji se može prenijeti u komadu na drugo računalo koje ima instaliran Java Virtual Machine te pokrenuti.

## Model View Controller Pattern

„BIBLIJA&ja” aplikacija je MVC (Model-View-Controller) softverskog obrasca što omogućava pravilnu podjelu odgovornosti projekta. Modeli i kontroleri su domena backend podjele dok View pripada frontend podjeli. Takva podjela odgovornosti omogućava lakše razvijanje projekta bilo da na njemu radi jedan ili više članova razvojnog tima te se time i lakše implementira Single-responsibility princip programiranja gdje je pojedini modul odgovoran za jednu umjesto više funkcionalnosti odnosno domena rada aplikacije. MVC je zbog tih karakteristika jedan od omiljenih obrazaca za izradu web aplikacija.

## H2 Persistence

Već spomenuti RDBMS H2 je gotovo u potpunosti apstrahiran u JSB stoga se developer ne treba fokusirati na izradu, odabir i postavljanje relacijskih baza podataka što je omogućeno kroz JDBC API već se baza modelira programski s pomoću klasa a potom instanciranih objekata u memoriji koji se upisuju u bazu podataka. H2 je vrlo lagana i učinkovita baza podataka te je prikladna za uporabu i u komercijalnim aplikacijama manjih dimenzija. Može se koristiti u dva moda rada, kao volatile da je zadržana u trenutnom memorijskom prostoru ili kao datoteka a developer može birati kakav mod želi koristiti. Memorijski mod rada je preferabilan za testiranje i razvoj ali se može koristiti i u produkcijske svrhe po potrebi. H2 je relacijska baza podataka stoga ima zakonitosti koje se moraju poštivati. Napisana je u Java programskom jeziku te je FOSS (Free Open Source Software) i podržava korištenje PostgreSQL ODBC drivera iako je su glavni API-ji SQL i JDBC.

## Alati Projekta

Važan dio svakog projekta je alat koji se koristi za njegovu izgradnju, pogotovo kod složenijih projekata. IDE i Git alati su neizostavni dio te Maven koji je specijaliziran za projekte pisane u Java programskom jeziku.

### Maven

Maven je alat za izgradnju projekta te se koristi .xml formatom točnije pom.xml datotekom u koju su upisani verzija projekta, dependency biblioteke, artefakti odnosno sve datoteke koje su rezultat outputa build procesa i pluginovi. S pomoću Maven alata je generirana i .jar datoteka koju pokreće Java interpreter na serveru kojim se ovaj projekt trenutačno koristi.

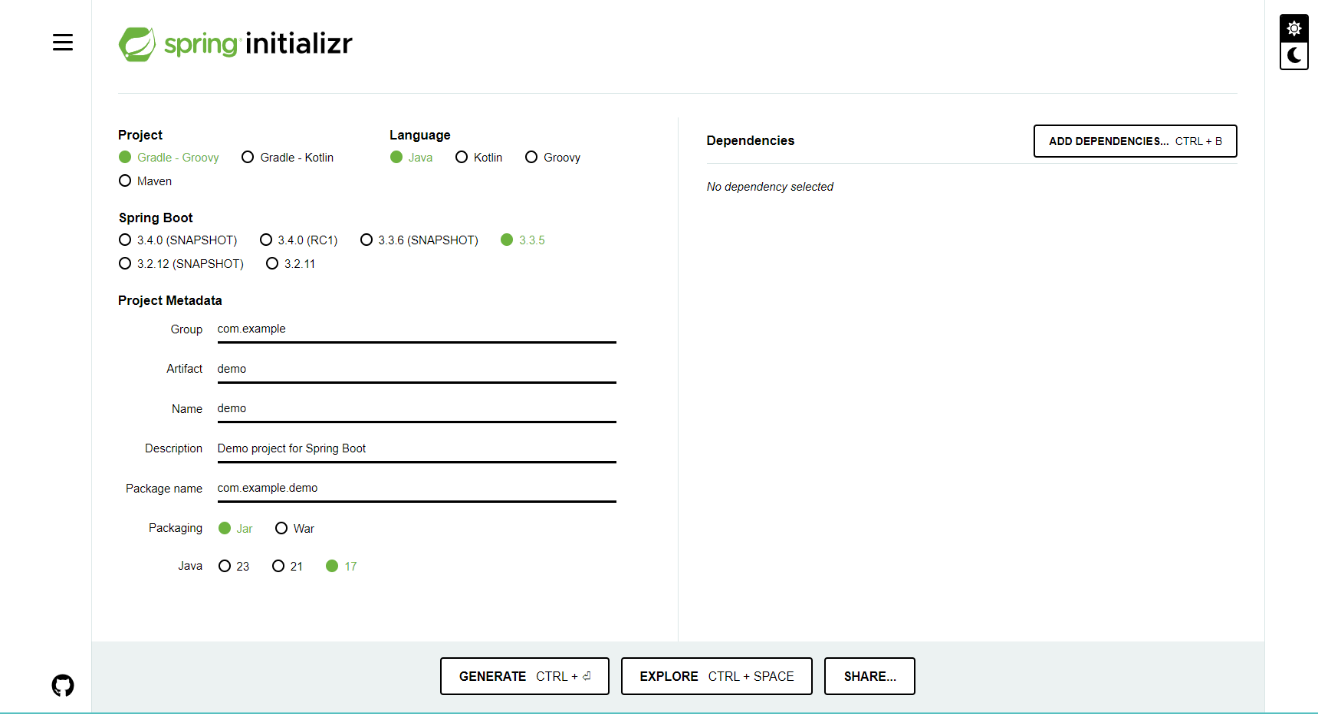
### Eclipse IDE

Eclipse je IDE (Integrated Development Environment) odnosno alat za razvoj programa u nekom od podržanih računalnih jezika. Primarni jezik je Java i sami alat je većinom implementiran u Javi. Eclipse omogućava uporabu tzv. Plugina koji proširuju funkcionalnosti Eclipse IDE-a stoga se radi o moćnom i efikasnom programskom alatu za gotovo sva područja razvoja od skriptnih jezika do jezika opće upotrebe. Verzija Eclipse-a korištena za razvoj ovog projekta je 2024-03 (4.31.0). Inačica odnosno specijalizacija Eclipse-a za Javu već dolazi s Ant, Maven i Gradle alatima tako da developer ima široki spektar odabira.

Kako je Eclipse korišten za razvoj i frontend domene aplikacije bilo je potrebno instalirati plugin s tim funkcionalnostima isto tako i za korištenje Spring Boot Frameworka koji tada prepoznaju da se radi o JSB tipu projekta. Nije potrebno ručno konfigurirati Eclipse da traži prave classpath-ove te posebno dodavati ili instalirati alate potrebne za pokretanje takvog projekta ukoliko je plugin za JSB instaliran u IDE.

### Spring Initializr

Ime „Initializr” (bez slova e na predzadnjem mjestu) nije tipfeler već pravo ime alata za inicijalizaciju Spring projekata. Alat je razvila ista grupa koja je razvila i framework. Taj online alat omogućava developerima da započnu projekt s naprednim startom gdje su početne postavke projekta već postavljene. Stoga umjesto da se kreće iz ničega taj postojeći osnovni projekt postaje bazom za daljnji razvoj. Velika prednost je također što takav početni projekt već ima definiranu strukturu mapa koju je potrebno slijediti a sami Spring inzistira na takvom rasporedu.

Spring Initializr se može pronaći na sljedećoj web stranici: (https://start.spring.io/) i besplatan je za korištenje te je preporučen način započinjanja projekta. 

Slika 2. Spring Initializr

### Git

Git je još jedan alat u nizu mnogih alata potrebnih za razvoj složenijih aplikacija. GIT je sustav verzioniranja odnosno alat koji pohranjuje i pamti trenutačnu i prošle verzije aplikacije tijekom njenog razvoja. Zbog tih funkcionalnosti moguće je vratiti se na prethodne preinake učinjene na projektu te paralelan razvoj s više developera u timu. To omogućava da verzija na kojoj se radi bude što više strukturirana te preinake koje su učinile aplikaciju npr. neupotrebljivom ili bugovitom budu izolirane i uklonjene.

## Teorijski Model

Kako bi se cilj i ideja projekta pravilno prenijela na software potrebno je identificirati potrebne funkcionalnosti i modele koji će vršiti određene funkcije. Počevši od gore prema dolje odnosno top-down strategijom, prvo se identificiraju više odnosno veće organizacijske strukture informacijskog sustava pa se potom razlažu na manje dok se ne razlože sve do najmanjeg elementa u sustavu. U sadržaju ovog projekta prepoznate su dvije velike strukture koje moraju biti međusobno povezane. To su:

* Biblija i njezini modeli te
* Korisnik i njegovi modeli.

### Modeliranje Biblijskih Modela

Kako je već prethodno opisana struktura Biblije odnosno podjela na ovdje relevantne - Knjiga, Poglavlje i Stih tako je potrebno modelirati i Java klase koje će predstavljati te istoimene podjele te imati identičnu programsku logiku. Stoga se Biblija kao model može razložiti na sljedeći način:

* Biblija mora sadržavati knjige,
* knjiga mora sadržavati poglavlja,
* poglavlje mora sadržavati stihove,
* stih mora sadržavati tekst.

Stih kao najmanji element Biblije je zadnji u ovom nizu. Iako bi tekst bio teoretski najmanji element nije ga potrebno definirati u ovom projektu jer je već po defaultu definiran. U Java je tekst već definiran kao String klasa.

### Modeliranje Korisničkih Modela

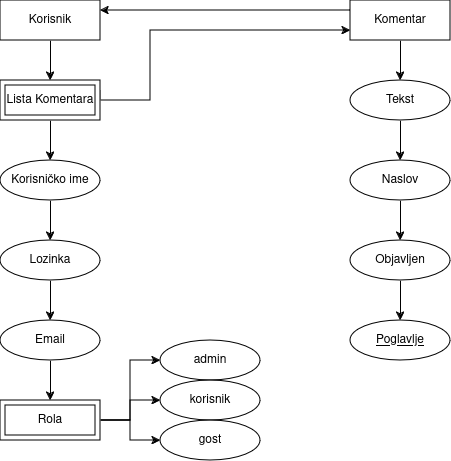
S obzirom na to da aplikaciju trebaju koristiti osobe koji doprinose projektu svojim komentarima, potrebno je također izraditi i potrebne modele. Naime model korisnika ima na sebe vezane i druge manje modele kao lozinka, korisničko ime, email, korisničke role itd. Korisničke role su konstante koje se ne mijenjaju a pridodaju se korisnicima kako bi im se omogućilo ili ograničilo njihovo korištenje web aplikacije iz sigurnosnih razloga. S ove strane je važno razmišljati o sigurnosti jer model korisnika je pod izravnom kontrolom osobe koja koristi aplikaciju a namjere mogu biti i maligne.

Također imamo i komentar koji korisnik ostavlja. Komentar mora biti vezan za poglavlje na koje je napisan stoga model poglavlja iz Biblije mora sadržavati referencu odnosno listu referenci na model komentara. Sami komentar također mora sadržavati i referencu na model korisnika. Korisnik sadrži listu referenci na model komentara jer jedan korisnik može napisati više komentara i na više od jednog poglavlja.

Stoga model korisnik se može razložiti na:

* listu referenci na model komentar,
* korisničko ime,
* lozinku i
* korisničku rolu.

Model komentara se može razložiti na:

* referencu na poglavlje u Bibliji,
* povratnu referencu na jednog korisnika,
* naslov komentara,
* boolean je li komentar javan te
* tekstualni sadržaj.

Slika 3. Korisnički modeli

# IZRADA APLIKACIJE

Pisanje programskog koda nije težak zadatak a pogotovo kada je analiza problema i modeliranje odrađeno unaprijed. Ono što predstavlja najveći problem jest rješavanje bugova kako dolaze te je potrebno redovito testiranje aplikacije bilo kroz automatizirane testove ili ručno. Za početak bitno je što vjernije prenijeti teoretski model u pisani kod. S vremenom a pogotovo tijekom testiranja se u praktičnom radu otkrivaju detalji koje je često potrebno doraditi ili prilagoditi potrebama aplikacije koji nisu bili jasno ili na prvu vidljivi u teoretskom modelu.

## Domain/Modeli

Domain ili modeli u MVC softverskom obrascu su „M” u kratici MVC. Radi se o Model dijelu te predstavlja unutarnju logiku aplikacije, tzv. „Business logic”. Modeli mogu biti korisnici, entiteti koji predstavljaju programsku stranu baze podataka, klase koje pružaju neku funkcionalnost aplikaciji i slično. S obzirom na to da se u ovom projektu koristi OOP (Object-Oriented-Programming) korisnici su klase, poglavlja u Bibliji i drugi modeli su također klase odnosno objekti u memoriji kad je proces aplikacije pokrenut na operativnom sistemu. Unutar JSB i Java programskog jezika općenito te gotovo svake aplikacije koja koristi MVC, modeli moraju biti odvojeni dio programa u zasebni namespace/package radi organizacije i kako ne bi dolazilo do sukoba s drugim domenama aplikacije.

Za funkcionalnost ovog projekta iz teoretskih modela izvedene su sljedeće klase:

* Bible,
* Book,
* Chapter,
* Verse,
* Commentary,
* User,
* UserRoles.

### Bible

Bible je najviša klasa u hijerarhiji koja sadrži unutar sebe referencu na listu Poglavlja, kako je opisano u teoretskom modelu. Biblija je javna (public) klasa s anotacijom @Entity. Entity anotacija je specifična za JPA (Java Persistence API) koja omogućava kompajleru da tu klasu prepozna kao entitet prilagođen za bazu podataka i pravilno ju kompajlira. Anotacije nisu tema ovog rada te se često koriste u Java programskom jeziku pa će biti viđena ovdje i u daljnjem prikazu koda. Kako se anotacije budu pojavljivale tako će biti i ukratko objašnjene. Bible klasa sadrži nekoliko polja (fields) koja ju definiraju:

* private Long id; koji predstavlja njezin id u bazi podataka,
* private String translation; je polje koje služi za prikaz prijevoda Biblije,
* private String emptyField; je polje ostavljeno za dodatne komentare kao ime prevoditelja teksta, koje je godine prevedeno itd. Dakle manji detalji koji su definirani pod jednim komadom teksta te zadnje polje jest
* private List<Book> books; polje koje sadrži referencu na klasu List u memoriji koja u sebi sadrži klase/objekte tipa Book. Teoretski model Biblije sadrži više od jedne knjige stoga je i programska logika identična.

package com.java.crv.BibleReaderCommentary.domain;

import java.util.List;

import jakarta.persistence.CascadeType;

import jakarta.persistence.Entity;

import jakarta.persistence.GeneratedValue;

import jakarta.persistence.GenerationType;

import jakarta.persistence.Id;

import jakarta.persistence.OneToMany;

@Entity

public class Bible {

@Id

@GeneratedValue(strategy = GenerationType.AUTO)

private Long id;

private String translation;

private String emptyField;

@OneToMany(mappedBy = "bible", cascade = CascadeType.ALL)

private List<Book> books;

public Long getId() {

return id;

}

/\*Getters and setters here\*/

}

Kod 1: Primjer Bible klase

Oznake većine anotacija su samoopisne kao što je @Id koja određeno polje označava kompajleru da se radi o jedinstvenom identifikatoru. Postoje i anotacije koje nemaju jednostavan opis ali vrše jednostavnu funkciju kao npr. @GeneratedValue(strategy = GenerationType.AUTO) što govori kompajleru ili točnije upućuje JDBMS da pridoda vrijednost tom unosu u bazu podataka po vlastitoj unutarnjoj logici. Developer može birati od nekoliko strategija koje taj API nudi.

Anotacija @OneToMany(mappedBy = "bible", cascade = CascadeType.ALL) koja se nalazi odmah iznad liste knjiga govori o relaciji modela Bible sa modelom Book. OneToMany jednostavno označuje da jedna Biblija ima jednu ili više relacija te se i u samoj anotaciji definira naziv te relacije odnosno entitet Biblija ima naziv „bible” u tom odnosu.

Metode koje su vidljive u prikazu koda su getteri i setteri odnosno metode koje pružaju funkcionalnost klasi Bible da se njena polja koja su privatnog mogu na siguran način

### Book

Book je idući model u hijerarhiji te je isto označen s anotacijom @Entity. Vrlo sličan modelu Bible ali u ovom slučaju ima povratnu referencu na model Bible (ManyToOne) te referencu na listu modela Chapter (OneToMany). Logika ovoga je uočljiva već iz prethodnih poglavlja i očigledan je niz kojim ide. U nastavku je skraćeni primjer koda klase Book:

@Entity

public class Book {

@Id

@GeneratedValue(strategy = GenerationType.AUTO)

private Long id;

private String name;

@ManyToOne

@JsonIgnore

@JoinColumn(name = "bible\_id")

private Bible bible;

@OneToMany(mappedBy = "book", cascade = CascadeType.ALL)

private List<Chapter> chapters;

}

Kod 2: Primjer Book klase

Ovdje je uočljiva i nova anotacija @JoinColumn koja opisuje da je Id u referenci na objekt tipa Bible strani ključ ove veze i naziv tog stupca jest „bible\_id”. Također je ovdje i @JsonIgnore anotacija koja onemogućava neprekidnu petlju referenci. Bez te anotacije tijekom korištenja reference na Bible bismo dobili ne samo referencu na Bible već ponovno natrag na sve objekte koji se nalaze u listi. Tako bismo završili ponovno u objektu iz kojeg smo referirali Bibliju i tako dok program ne izbaci exception i sruši se. Za razliku od Bible, ovdje imamo i dodatno polje „name” koje predstavlja ime knjige, npr. „Izaija” ili „Postanak” itd.

### Chapter

Chapter model je reprezentacija poglavlja unutar knjige. Poput Book ima listu referenci na niži model – u ovom slučaju Verse. Te ima povratnu referencu na Book unutar kojeg se nalazi. Chapter za razliku od svojeg „roditelja” Book nema ime već samo broj (tipa int). Još jedna od razlika je što Chapter ima listu referenci i na korisnički model Commentary (više o tom modelu u idućim poglavljima).

@Entity

public class Chapter {

@Id

@GeneratedValue(strategy = GenerationType.AUTO)

private Long id;

private int number;

@ManyToOne

@JsonIgnore

@JoinColumn( name = "book\_id")

private Book book;

@OneToMany(mappedBy = "chapter", cascade = CascadeType.ALL)

private List<Verse> verses;

@OneToMany(mappedBy = "chapter", cascade = CascadeType.ALL)

@JsonIgnore

private List<Commentary> comments;

}

Kod 3: Primjer Chapter klase

### Verse

Verse odnosno stih je zadnji u nizu modela koji su vezani za Bible model. Ta klasa sadržava sami tekst iz Biblije putem String klase predefinirane u Java programskom jeziku. Najduži stih u Bibliji je iz knjige po imenu Estera te sadržava oko 487 znakova ovisno o prijevodu. U tom slučaju se može staviti anotacija @Column preko koje je moguće konfigurirati stupac u bazi podataka. Ovdje je taj stupac definiran da sadrži i do 500 znakova (VARCHAR(500)). Radi učinkovitosti memorije i prostora koju baza koristi i zauzima na računalu bolje je da to bude varijabilna ali ograničena vrijednost. Defaultna vrijednost je uglavnom 255 znakova stoga je potrebno koristiti tu anotaciju. Verse klasa također sadrži povratnu referencu na Chapter kojem pripada.

@Entity

public class Verse {

@Id

@GeneratedValue(strategy = GenerationType.AUTO)

private Long id;

private int number;

@Column(length = 500)

private String text;

@ManyToOne

@JsonIgnore

@JoinColumn(name = "chapter\_id")

private Chapter chapter;

}

Kod 4: Primjer Verse klase

### Commentary

Commentary je jedna od korisničkih klasa te opisuje i sadrži podatke vezane za komentar koji korisnik ostavlja. Commentary zajedno uz Chapter čini most između dvije velike domene opisanih u uvodu. Sadrži referencu na Chapter u kojoj je odnos ManyToOne što znači da Commentary sadrži referencu na jedno i isključivo jedno poglavlje tj. Chapter. Također sadrži referencu na jednog User-a te uz to i sadrži ime autora (User), subjekt odnosno naslov komentara te sami tekst koji je korisnik napisao i datum pisanja teksta. Polje u klasi: private Boolean published služi kako bi komentator mogao definirati svoj komentar kao javni ili privatni komentar. Dakle komentar će se prikazivati drugim korisnicima ovisno o vrijednosti u tom boolean polju. Također ova klasa nema defaultni konstruktor nego koristeći predefinirane Java klase LocalDateTime i DateTimeFormatter se u konstruktoru polje timestamp postavlja na datum pisanja komentara.

Kako bi autor komentara imao dovoljno prostora za pisanje, polje „text” unutar Commentary klase je definirano na raspon do deset tisuća znakova (VARCHAR(10000)). Takva strategija omogućava da korisnik ima dovoljno prostora, da ne opterećuje bazu podataka jer nije očekivano da će komentari biti veći od te granice te će u bazu biti alocirano onoliko memorije koliko je zaista i potrebno. Dakle ako je korisnik unio komentar od tisuću znakova tada iako je gornja granica deset tisuća bit će alocirano memorije samo za tih tisuću znakova.

@Entity

@JsonIdentityInfo(generator = ObjectIdGenerators.PropertyGenerator.class, property = "id")

public class Commentary {

@Id

@GeneratedValue(strategy = GenerationType.AUTO)

private Long id;

private String subject;

private Boolean published;

@Column(length = 10000)

private String text;

private String timestamp;

private String author;

@ManyToOne

@JsonProperty("user")

@JsonManagedReference

@JoinColumn(name = "user\_id")

private User user;

@ManyToOne

@JsonIgnore

@JoinColumn(name = "chapter\_id")

private Chapter chapter;

public Commentary() {

LocalDateTime ldt = LocalDateTime.now();

DateTimeFormatter format = DateTimeFormatter.ofPattern("dd/MM/yyyy");

this.timestamp = ldt.format(format);

}

}

Kod 5: Primjer Commentary klase

### User i UserRoles

User klasa predstavlja samog korisnika te ima zadatak pohraniti sve korisničke podatke kojima aplikacija raspolaže. Neke od tih podataka korisnik ispunjava i pod korisničkom su kontrolom dok neke dodjeljuje korisnik sa statusom admina ili sama aplikacija tijekom izrade računa. Klasa User ima najveću ulogu u sigurnosti ove web aplikacije jer ovisno o korisničkim rolama koje određeni korisnik posjeduje ima i pristup određenim domenama rada aplikacije. UserRoles je tipa enum a ne klasa te u sebi sadrži sveukupno četiri definirane razine odnosno četiri role koje korisnik može imati. Podjela tih rola od najviših ovlasti do najmanjih su: owner, admin, user te konačno guest. Na primjer User koji ima rolu „admin” ima administratorske ovlasti nad aplikacijom te može pristupiti direktno: bazi podataka ili sučelju za upload baze podataka, korisnika ili Biblije te download istih podataka. Isto vrijedi i za rolu „owner”, koji ima pristup potpunoj aplikaciji. Svakom novom korisniku se dodjeljuje rola „user” koji ima ovlasti isključivo nad svojim komentarima i pisanju/brisanju/revidiranju istih te pri odabiru hoće li komentar biti javnog ili privatnog tipa. Najniža rola guest se zapravo ne primjenjuje nad niti jednim korisnikom i nije korištena u praksi već služi za primjer hijerarhije korisnika unutar aplikacije. Svaki posjetitelj web aplikacije koji još nije izradio profil ima virtualnu „guest” rolu. To se može primijeniti u projektu kroz cookies pa se kroz automatiku može izraditi privremeni user sa „guest” rolom. To bi bilo slično pripremi izrade profila ali u trenutačnoj verziji ovog projekta to nije implementirano.

Korisnik web aplikaciji odnosno svom profilu pristupa preko korisničkog imena (username) i lozinke (password) iako je u trendu da je email glavni način pristupanja. Ovo će se najvjerojatnije promijeniti u idućim verzijama aplikacije kako bi se omogućilo lakše pristupanje s npr. Google accounta ili sličnih.

@Entity

@JsonIdentityInfo(generator = ObjectIdGenerators.PropertyGenerator.class, property = "id")

@Table(name= "`USER`")

public class User {

@Id

@GeneratedValue(strategy = GenerationType.AUTO

private Long id;

private String password;

private String username;

private String email;

private UserRoles role;

@OneToMany(mappedBy = "user")

@JsonProperty("comments")

@JsonBackReference

private List<Commentary> comments; }

Kod 6: Primjer User klase

public enum UserRoles {

GUEST,

USER,

ADMIN,

OWNER }

Kod 7: Primjer UserRoles enum-a

## Repository Interface

U sklopu JSB projekata potrebno je definirati tzv. Repository interface-ove, oni su usko povezani s modelima unutar poslovne logike. Te interface-ovi se označavaju s anotacijom @Repository koja ih definira kao programatski model baze podataka odnosno radi se o kolekciji objekata određenog tipa. Oni pripadaju Data Access Layer-u aplikacije koji služi kao mehanizam pohrane/povrata/izmjene/brisanja podataka (CRUD operacije – Create, Read, Update, Delete). Ukratko imaju funkciju mosta između programatskog dijela aplikacije i njene baze podataka.

Svaki model koji želimo pohraniti u bazu podataka mora imati i svoj Repository. Tako klase Bible, Book, Chapter....User itd. Svaki ima svoj repozitorij koji omogućava developeru da koristi taj apstrahirani način rada a da se ne bavi pisanjem SQL naredbi. To je svakako moguće i dio funkcionalnosti repozitorija te developer može pisati SQL naredbe ali za potrebe CRUD operacije to je svakako suvišno te je bolje koristiti već isprobani, testirani i pouzdani mehanizam koji će osigurati integritet podataka.

U opsegu projekta tako imamo sljedeće repozitorije:

* BibleRepository,
* BookRepositorym
* ChapterRepository,
* VerseRepository,
* UserRepository te
* CommentaryRepository.

Uobičajena je praksa da ime repozitorija bude po principu <model>Repository. Repozitorijski interface definira developer te je on nasljednik CrudRepository interfacea preko kojeg će unutarnji mehanizam omogućiti CRUD funkcionalnosti. Definirani repozitorij moguće je proširiti vlastitim funkcionalnostima. To je jedna od velikih prednosti JSB-a gdje se samo koristeći konvencije imenovanja metoda unutar interface-a mogu omogućiti nove funkcionalnosti. Java Persistence API proširen unutar JSB implementira tu funkcionalnost koristeći tzv. „Query Method Derivation”. U sklopu ovog projekta BibleRepository i BookRepository su čiste izvedenice CrudRepository interfacea te nisu proširivane. Svaki repozitorij je definiran u kodu:

@Repository

public interface MyModelRepository extends CrudRepository<T, Id>{}

Kod 8: Primjer Definicije Repozitorija

Tako su BibleRepository i BookRepository definirani kao:

@Repository

public interface BibleRepository extends CrudRepository<Bible, Long>{ }

Kod 9: Definicija BibleRepository

@Repository

public interface BookRepository extends CrudRepository<Book, Long>{.......}

Kod 10: Definicija BookRepository

Ostali repozitoriji su morali biti proširivani kako bi se omogućilo bolje dohvaćanje podataka i bit će opisani u daljnjem tekstu.

### ChapterRepository

Na primjer ChapterRepository ima proširenje preko deklaracije metode List<Chapter> findByBookId(Long bookId koji nema definiranu pretragu objekata po povratnoj referenci u baznom interfaceu CrudRepository. S obzirom na to da developer definira vlastiti repozitorij, JSB ne može predvidjeti sve mogućnosti koje bi CrudRepository trebao omogućiti ili kakva će sve polja određena klasa sadržavati ili kakve će međusobni odnosi između objekata postojati. U ovome leži prednost „Query Method Derivation” načina, gdje developer definira metodu po određenom načinu imenovanja metode te davanjem informacije o kojem se objektu radi a API ju implementira.

@Repository

public interface ChapterRepository extends CrudRepository<Chapter, Long>{

List<Chapter> findByBookId(Long bookId);

}

Kod 11: Primjer ChapterRepository

Kako je Chapter jedan od objekata u listi Book modela, nije jednostavno doći do njegovog Id polja. Ova metoda rješava tu kompleksnost, olakšava developeru da programatski riješi problem te se ne mora baviti SQL programiranjem.

### VerseRepository

Ovaj repozitorij poput ChapterRepository-a sadrži samo jedno proširenje interfacea kroz deklaraciju metode List<Verse> findByChapterId(Long chapterId. Ta metoda će omogućiti pronalazak svih stihova koji sadrže određeno poglavlje po njegovom Id-u. Razlog ovakvoj pretrazi podataka jest to što VerseRepository sadrži sve Verse objekte unutar sebe. Ako bismo htjeli dohvatiti sve stihove iz određenog poglavlja mogli bismo to također učiniti i preko ChapterRepository ali u sklopu ovog projekta se stihovi izvlače iz VerseRepository kako bi se zadržala logika fokusirana na taj repozitorij te pretraga baze podataka se zadržava na stihovima radije negoli na poglavljima.

### UserRepository

Korisnički repozitorij kao i po principu ostalih sadrži User objekte. Ovaj interface je proširivan samo s jednom vrlo jednostavnom i izravnom metodom User findByUsername(String username. Iako se ovakva pretraga vrši po samom objektu bilo je potrebno proširiti interface. Razlog tome jest što defaultni odnosno bazni CrudRepository interface sadrži deklaraciju metode koja vrši pretragu po Id-jevima a to prepoznaje prema anotaciji @Id unutar klase. Ova findByUsername metoda će izvršiti pretragu po cijelom UserRepository ali filtrirati će pretragu samo po korisničkom imenu te vratiti listu sa svim User objektima koji sadrže to korisničko ime. Korisničko ime (username) primjetiti ćemo nema nikakvu anotaciju u klasi User nego je to polje deklarirano kao private String username a JPA ga ipak prepozna. Ovdje na djelo ponovno dolazi „Query Method Derivation” koji prepoznaje polje username kao predmet pretrage.

@Repository

public interface UserRepository extends CrudRepository<User, Long>{

User findByUsername(String username);

}

Kod 12: Deklaracija UserRepository

### CommentaryRepository

Najkompleksniji repozitorij unutar ovog projekta s tri proširenja. S obzirom na to da komentari korisnika imaju nekoliko načina pretrage kao uvjet je li komentar javno objavljen ili nije te je li komentar napisao određeni korisnik i slično. Ove pretrage su bitne u kontekstu vidljivosti komentara te da se omogući korisniku da vidi sve svoje komentare na jednom mjestu na primjer ili da se prikažu svi komentari određenog korisnika.

@Repository

public interface CommentaryRepository extends CrudRepository<Commentary, Long> {

List<Commentary> findAllByPublished(Boolean published);

List<Commentary> findAllCommentaryById(Long commentaryId);

List<Commentary> findAllByUserId(Long userId);

}

Kod 13: Delaracija CommentaryRepository

### Zaključak na Repozitorije

Repozitoriji su usko povezani s modelima koje definira developer te predstavljaju bazu podataka u programatskom obliku. Repozitoriji su dio JSB te surađuju s JPA kako bi odradili CRUD operacije na siguran i učinkovit način. „Query Method Derivation” omogućava developeru da proširuje interface repozitorija te ih prilagodi svojim potrebama u odnosu na bazu podataka te tako oslobađa developera da se bavi programskim dijelom aplikacije umjesto samim SQL upitima u bazu podataka. Svi interface-ovi moraju biti izvedenica CrudRepository interface kako bi se omogućile CRUD operacije. Postoji još izvedenica definiranih u samom JSB-u ali one nisu korištene u ovom projektu.

## Kontroleri

Kontroleri su drugi dio backend logike web aplikacije, do sada su objašnjeni modeli koje aplikacija koristi a kontroleri su aktivni dio aplikacije. Figurativno rečeno modeli su „što” a kontroleri su „kako” unutar aplikacije. Kontroleri upravljaju funkcionalnostima web aplikcije od prikaza resursa i podataka na web do unosa, kontrole i provjere unesenih podataka. Kontroleri odgovaraju na zahtjeve upućene serveru preko GET i POST te drugih metoda ali ove dvije su najčešće. Kako bi kontroler znao kada treba odraditi svoj zadatak potrebno je definirati tzv. „endpoint”. Endpoint je određeni URL koji korisnik traži, na primjer „/login” koji je zadužen za prijavu korisnika u sustav, dakle u kontekstu ovog projekta kada korisnik upiše u URL svojeg browsera: <https://www.biblijaija.com/login> kontroler koji je zadužen za „/login” će biti pozvan i odraditi ono što je developer programirao unutar tog kontrolera. U ovom slučaju će biti dohvaćena html web stranica koja ima funkciju prijave korisnika u aplikaciju. Unutar ovog projekta korištena je strategija da jedan kontroler vrši sve funkcije unutar jednog URL-a kako bi se smanjila količina potrebnih kontrolera te radi preglednosti napisanog koda. Mana ove strategije jest to što se kod većih i kompleksnijih aplikacija povećava broj funkcionalnosti stoga i raste broj funkcija koje određeni kontroler mora izvršavati te se time krši SRP (Single Responsibility Principle). Zbog umjerene kompleksnosti ovog projekta odabrana je ta strategija te većina kontrolera nema više od šest metoda.

Druga strategija koja bi bila prilagođenija većim aplikacijama bi bila da se napišu kontroleri po funkciji umjesto po URL putanji. Na taj način bi svaka funkcionalnost na određenom URL-u imala svoj zasebni kontroler.

Package com.java.myProject.controllers;

import org.springframework.stereotype.Controller;

@Controller

Public class MyController{

@GetMapping(„/myURL“)

public T doSomething()

{

Return T;

}

}

Slika 4. Defaultna definicija kontrolera

### API Controller

API kontroller ima funkciju dinamičkog dohvaćanja podataka iz baze prilikom odabira knjige a potom poglavlja. Kako bi korisnik mogao dohvatiti tekst potrebno je prvo izabrati „Book“ odnosno knjigu a potom „Chapter“ odnosno poglavlje unutar kojeg se nalazi lista svih stihova sadržanih unutar baze za to poglavlje.



Slika 5. Odabir teksta korak prvi



Slika 6. Odabir teksta korak drugi

Redoslijed kojim korisnik bira knjigu a potom poglavlje odgovara redoslijedu kojim će i kontroler odrađivati dohvaćanje podataka. Proces se odvija dinamički tako da se ne mijenja cijela HTML stranica već samo odabrani elementi pomoću AJAX tehnologije. Odabir poglavlja neće biti moguć dok korisnik ne odabere prvo knjigu, razlog tome jest što svaka knjiga ima različit broj poglavlja i stoga je tokom odabira knjige potrebno prvo dohvatiti broj poglavlja.

package com.java.crv.BibleReaderCommentary.controllers;

@Controller

public class ApiController {

private BookRepository bookRepository;}

Kod 14. Deklaracija APIController-a

APIController će izvršiti tri metode u redoslijedu a zadnja će biti dohvaćanje svih Commentary objekata koji se nalaze u bazi podataka vezanih na odabrano poglavlje i knjigu.

Važno je naglasiti da ovaj i ostali kontroleri koriste „Dependency Injection“ tehniku kako bi mogli koristiti bazu podataka. Repository objekti su ti koji posreduju između baze i programskog koda koji se izvršava.

@GetMapping({"/submitComment/fetchChapters", "/public/fetchChapters"})

@ResponseBody

public List<Chapter> fetchChapters(@RequestParam Long bookId)

{

Optional<Book> bk = bookRepository.findById(bookId);

ArrayList<Chapter> chapters;

if(bk.isPresent()) {

List<Chapter> list = bk.get().getChapters();

chapters = new ArrayList<Chapter>();

for(Chapter ch : list) {

Chapter chapt = new Chapter();

chapt.setId(ch.getId());

chapt.setNumber(ch.getNumber());

chapters.add(chapt);

}

return chapters;

}

else

return Collections.emptyList();

}

Kod 15. APICont. fetchChapters metoda

Metoda „fetchChapters“ je prva metoda u nizu izvršavanja koja će se aktivirati tokom promjene vrijednosti u index.html formi. Zahvaljujući anotaciji @GetMapping ova metoda će se aktivirati kada se pokuša dohvatiti URL „"/submitComment/fetchChapters" i "/public/fetchChapters". U trenutku kada korisnik odabere knjigu na index stranici, preko template enginea pod imenom „Thymeleaf“ koji je ugrađen u kod html forme pozvati URL "/public/fetchChapters" na koji će se „fetchChapters“ metoda odazvati i poslati na browser sva poglavlja koju odabrana knjiga sadržava i popuniti <select> element sa <option> child elementima, za svako poglavlje po jedan i sa brojem poglavlja.

<form method="post">

<select id="bookSelection" name="selectedBook"

onchange="updateIndexChapters()">

<option selected disabled th:placeholder="'Odaberite knjigu'">Odaberite knjigu</option>

<option th:each="book : ${books}"th:value="${book.id}"

th:text="${book.name}"></option>

</select>

<select id="chapterSelection" name="selectedChapter" onchange="fetchVersesAndComments()" disabled>

<option selected disabled th:placeholder="'Glava'"> Glava </option>

</select>

</form>

Kod 16. Index forma za odabir prikaza

Oznake koje unutar HTML-a počinju s „th:“ su dio Thymeleaf funkcionalnosti koji obrađuje html formu prije nego bude poslana klijentu. Tijekom promjene vrijednosti na elementu <select> će se svaki puta pozvati funkcija „updateIndexChapters()“ definirana u ajax\_script.js datoteci. Ta funkcija će poslati zahtjev na URL „public/fetchChapters“ zajedno sa podatkom o Id-u knjige.

function updateIndexChapters() {

var selectedBookId = document.getElementById('bookSelection').value;

console.log("Performing fetch...")

// Perform an Ajax request to fetch chapters for the selected book

fetch('public/fetchChapters?bookId=' + selectedBookId)

.then(response => response.json())

.then(data => {

// Populate the 'chapterSelection' dropdown based on the response

var chapterSelection = document.getElementById('chapterSelection');

chapterSelection.innerHTML = ''; // Clear existing options

// Add a placeholder option

var placeholderOption = document.createElement('option');

placeholderOption.value = '';

placeholderOption.text = 'Glava';

placeholderOption.disabled = true;

placeholderOption.selected = true;

chapterSelection.appendChild(placeholderOption);

// Add options for each chapter

data.forEach(chapter => {

var option = document.createElement('option');

option.value = chapter.number;

option.text = chapter.number;

chapterSelection.appendChild(option);

});

// Enable the 'chapterSelection' dropdown

chapterSelection.disabled = false;

})

.catch(error => {

console.error('Error fetching chapters:', error);

});

}

ZAKLJUČAK

U zaključku treba ukratko prikazati rezultate i spoznaje do kojih se u radu došlo. Zaključak mora započeti na zasebnoj stranici. Mora imati više od jedne a manje od dvije stranica teksta. U Zaključku nema podnaslova, slika, tablica….

# LITERATURA

1. Boots Group Plc., (2003), Corporate social responsibility, http://www.Boots‐Plc.Com/Information/Info.Asp?Level1id=447&Level 2id=0 (pristupljeno 23. 7. 2020.)
2. Boynlon, A.C., Zmud, R.W., (1984), An Assessment of Critical Success Factors, *Sloan Management Review*, 25 (4), pp. 17–27
3. Čulina, B., Čulina, D., Čančarević, M., (2020), [Zaboravljena matematička povijest  
   računarstva](https://www.bib.irb.hr/1088451), *Poučak: časopis za metodiku i nastavu matematike*, 21 (83), pp. 56–73
4. Srića, V., Spremić, M., (2000), *Informacijskom tehnologijom do uspjeha*, Zagreb: Sinergija
5. Tran J. M., D. Yean C., Beaumont M., (2002), Exploring the rationales for ERP and SCM Integration, *Industrial management & Data Systems*, 102 (1), pp. 26–34
6. Wikipedia, (2021), SWOT\_analysis, http://www.wikipedia.org (pristupljeno 2. 1. 2021.)

# SAŽETAK

Ovaj završni rad temelji se na analizi postojećeg sustava korisničke podrške te njegovog unapređenja prema zahtjevima korisnika, predstavnika IT tvrtke i Odjela korisničke pomoći. Detekcijom potreba za poboljšanje usluga pružanja podrške korisnicima/klijentima tvrtke identificirane su i predložene mogućnosti poboljšanja u vidu aplikacije za prijavu problema putem interneta i nadogradnju baze podataka te ih izraditi i implementirati s ciljem pojednostavljenja poslova korisničke podrške, omogućavanje uključenosti korisnika u proces omogućavanjem prijave problema putem internet sučelja te povećanje efikasnosti IT tvrtke i brzine rada s klijentima.

**Ključne riječi:** najmanje 5 i najviše 10 ključnih riječi

# SUMMARY

This paper is based on the analysis of existing customer support system and its improvement according to customer requirements: representatives of IT companies and customer support department. After needs to improve services was detected with main goal to create better support for customers/clients, identification and proposal of improvements for the support platform where suggested in the form of crating online application for problem registration whit regular database update. Next step was development and implementation of proposed solution with the aim to facilitate the work of customer support, enabling user involvement in the process of facilitating the application problems through Internet site and increase efficiency of an IT company and improve response time to clients needs.

**Keywords:** at least 5 and at most 10 keywords