Upotreba Spring i Maven tehnologija u razvoju Java aplikacije za kontrolu pristupa podacima

Kudumija, Luka

Sveučilište Sjever

Završni rad br. 04/RINF/2024

Upotreba Spring i Maven tehnologija u razvoju Java aplikacija za kontrolu pristupa podacima

Luka Kudumija, 0068222839

Koprivnica, srpanj 2024. godine

Sveučilište
Sjever

Odjel za računarstvo i informatiku

Završni rad br. 04/RINF/2024

Upotreba Spring i Maven tehnologija u razvoju Java aplikacija za kontrolu pristupa podacima

Student

Luka Kudumija, 0068222839

Mentor

Domagoj Frank, doc.dr.sc.

Koprivnica, srpanj 2024. godine

Sveučilište Sjever
Sveučilišni centar Koprivnica
Trg dr. Žarka Dolinara 1, HR–48000 Koprivnica

Prijava završnog rada

Definiranje teme završnog rada i povjerenstva

ODJEL: Odjel za računarstvo i informatiku

STUDIJ: Prijediplomski stručni studij Računarstvo i informatika

PRISTUPNIK: Luka Kudumija | MATIČNI BROJ: 0068222839

DATUM: 10.06.2024. | KOLEGIJ: Projektiranje računalnih sustava

NASLOV RADA: Upotreba Spring i Maven tehnologija u razvoju Java aplikacija za kontrolu pristupa podacima.

NASLOV RADA NA ENGL. JEZIKU: Use of Spring and Maven technologies in Java application development for data access control.

MENTOR: Domagoj Frank | ZVANJE: Doc. dr. sc.

ČLANOVI POVJERENSTVA:
1. Dražen Crčić, predsjednik
2. Leon Horvat, član
3. Domagoj Frank, mentor
4. Darko Špoljar, zamjenski član
5.

Zadatak završnog rada

BROJ: 4/RINF/2024

OPIS:

Današnja potreba za Javom, a posebno Spring i Maven tehnologijama, proizlazi iz njihove široke primjene u razvoju skalabilnih i održivih poslovnih aplikacija. Njihova primjena može se uočiti kod izrade Java aplikacija u tvrtkama koje se primarno bave financijama kao što su banke ili Financijska agencija.
Cilj završnog rada je istražiti i demonstrirati upotrebu Spring i Maven tehnologija u razvoju Java aplikacija, razumjeti osnovne koncepte Spring Frameworka i Maven alata, te kreirati Java aplikaciju koja ilustrira upotrebu navedenih tehnologija za kontrolu prava pristupa podacima u bazi podataka.

Teorijski dio:
- Kratak pregled Java programskog jezika, detaljan pregled Spring Frameworka, objašnjenje Mavena
- Integracija Springa i Mavena: kako se Maven koristi za upravljanje Spring projektima
Praktični dio:
- Kreiranje novog Spring Maven projekta, uključujući konfiguraciju pom.xml datoteke za upravljanje ovisnostima i pluginovima.
- Razvoj Spring aplikacije: Implementacija jednostavne Spring aplikacije koja koristi osnovne Spring module za kontrolu prava pristupa podacima u bazi podataka.

ZADATAK URUČEN: 13.06.2024. | POTPIS MENTORA:

SVEUČILIŠTE SJEVER

ZAHVALA

Zahvaljujem se svom mentoru, doc.dr.sc. Domagoju Franku, na dragocjenim smjernicama i savjetima oko pisanja završnog rada. Profesor Frank je bio od presudne važnosti prilikom odabira teme, ali i mentoriranja kroz cijeli rad. Također se zahvaljujem svim profesorima Sveučilišta Sjever koji su podijelili svoje znanje kroz sve tri godine studiranja, ali i kolegama s kojima sam dijelio to znanje.

Zahvaljujem se i svojoj obitelji i djevojci na njihovoj bezuvjetnoj podršci i ljubavi, koje nikad nije nedostajalo kroz ove tri godine.

SAŽETAK

Ovaj se rad fokusira na primjenu Java programskog jezika, te Spring i Maven tehnologija u izgradnji aplikacija za kontrolu pristupa podacima. Spring, kao radni okvir, donosi nove značajke u programiranju u odnosu na tradicionalni pristup, što su aspektno orijentirano programiranje, injekcija ovisnosti i inverzija kontrole. Sve ove značajke služe kao pomoć programerima u izgradnji sigurnih i skalabilnih aplikacija, koje su potrebne za svaku tvrtku u poslovanju, no posebno za one koje se bave financijskim uslugama, poput banaka ili financijskih agencija. Maven nam tad služi za automatizaciju izgradnje aplikacija te se brine za sve ovisnosti i biblioteke koje su nam potrebne u aplikaciji. Na kraju rada je prikazana aplikacija koja koristi značajke Spring i Maven tehnologija kako bi se ograničio pristup podacima za koje smatramo da su osjetljive prirode. Kroz autentifikaciju korisnika, brinemo oko toga da korisnik zapravo je onaj kojim se predstavlja, dok nam autorizacija služi za kontrolu pristupa podacima s obzirom na rolu korisnika koju posjeduje u aplikaciji.

Ključne riječi: Java, Spring, inverzija kontrole, injektiranje ovisnosti, Spring Security, Maven, kontrola pristupa podacima

SUMMARY

This paper focuses on the application of the Java programming language and the use of Spring and Maven technologies in building applications for data access control. Spring, as a framework, introduces new features compared to traditional programming approaches, such as aspect-oriented programming, dependency injection, and inversion of control. These features assist developers in creating secure and scalable applications, which are essential for any business but particularly crucial for those in the financial services industry, such as banks or financial agencies. Maven is used to automate the build process of applications, managing all dependencies and libraries required by the application. At the end of the paper, an application is demonstrated that leverages Spring and Maven technologies to restrict access to data considered sensitive. Through user authentication, we ensure that the user is indeed who they claim to be, while authorization controls access to data based on the user's role within the application.

Keywords: Java, Spring, inversion of control, dependency injection, Spring Security, Maven, data access control

1. Sadržaj

1. UVOD

Prema grčkom filozofu Heraklitu „Jedina konstanta je promjena“, a ta izjava također obuhvaća temeljnu istinu o razvoju softvera. Način na koji danas razvijamo aplikacije drugačiji je nego prije godinu dana ili 5 godina, a svakako drugačiji nego prije 10 ili 15 godina, kada je nastao početni oblik Spring radnog okvira predstavljen u knjizi Roda Johnsona.

Konstantni razvoj tehnologije doveo nas je do razvitka kompleksnih informacijskih sustava koji imaju potrebu zaštite osjetljivih podataka, pogotovo kad se radi o financijskim institucijama poput banaka ili financijskih agencija. Uz razvitak kompleksnih sustava razvijali su se i alati koji bi pomogli u razvitku takvih sustava. Jedan takav alat je Spring radni okvir, odnosno njegov modul – Spring Security (Spring sigurnost). Kod zaštite podataka dolazi do izražaja važnost Spring Security modula koji omogućava sofisticiranu kontrolu pristupa podacima i bolje upravljanje sigurnosnim aspektima aplikacije koju koristimo. Kroz pristup temeljen na rolama unutar aplikacije, postiže se pravilna autentifikacija i autorizacija korisnika aplikacije.

Drugi takav alat, Maven, dodatno olakšava proces razvijanja kompleksnih računalnih sustava tako da automatizira strukturu aplikacije, ali i njezine ovisnosti, kako bi se programeri mogli fokusirati na svoj pravi posao – programiranje.

U ovom radu istražit će se Spring i Maven tehnologije, kako one pomažu u razvoju aplikacija te pokazati na pravom primjeru prednosti koje takve tehnologije pružaju.

2. JAVA

Java je jedan od najpopularnijih programskih jezika današnjice i temelj milijunima aplikacija na mnogim platformama, kao što su Windows, MacOS, UNIX, ručni uređaji temeljeni na Androidu, ugrađeni sustavi i korporativna rješenja. Jedan od najvećih razloga zašto je Java tako popularna je neovisnost o platformi. Programi se mogu izvoditi na nekoliko različitih vrsta računala, odnosno, sve dok računalo ima instalirano Java Runtime Environment (JRE), na njemu se može izvoditi Java program. Većina vrsta računala bit će kompatibilna s JRE-om, uključujući računala koja rade na Windowsima, Macintosh računala, Unix ili Linux računala i velika središnja računala, kao i mobilni telefoni. Uz to, budući da postoji tako dugo, neke od najvećih svjetskih organizacija izgrađene su pomoću jezika Java. Mnoge banke, trgovci na malo, osiguravajuća društva, komunalna poduzeća i proizvođači koriste Javu.[1]

2.1 Povijest Jave

James Gosling, poznat kao "otac Jave", predstavio je Java jezik u lipnju 1991. pod nazivom 'Oak' projektom. Gosling je imao za cilj razviti virtualni stroj i jezik s poznatom notacijom sličnom programskom jeziku C, ali s većom preciznošću i jednostavnošću od C i C++ programskih jezika. Prva javna verzija, Java 1.0, objavljena je 1995. godine. Ta verzija Jave obećavala je jednostavnu stvar: 'Write Once, Run Anywhere' (hrv. „Napiši jednom, pokreni svugdje“). Jezik je bio vrlo siguran i konfigurabilan, s ograničenjima pristupa mreži i datotekama, zbog čega su ga brzo implementirali u svoje standardne postavke značajni web preglednici.[2]

Godine 1997., Sun Microsystems je zatražio pomoć od ISO/IEC JTC1 (International Organization for Standardization and the International Electrotechnical Commission) i kasnije Ecma International kako bi formalizirali Javu, ali oni su se brzo povukli od te ideje. Java je i dalje "de facto" vlasnički standard koji regulira Java Community Process (JCP). Uz napredak kao što je Java Enterprise Framework (JEF), Sun je razvio nekoliko Java implementacija potpuno besplatno. Kritična razlika je u tome što kompajler nije uključen u JRE-u, što razlikuje njegov Software Development Kit (SDK) i JRE.[2]

Sun je 13. studenog 2006. godine objavio velik dio Jave pod GNU General Public License kao besplatni softver otvorenog koda. Dana 8. svibnja 2007., Sun je završio proces objavom potpuno dostupne, potpuno besplatne i otvorene Java jezgre, izuzev malog dijela koda za koji Sun nije zadržao autorska prava.[2][3]

2.2 Filozofija Jave

Razvoj Java jezika imao je pet primarnih ciljeva:

- Treba koristiti objektno orijentirani pristup.
- Trebao bi omogućiti da više operativnih sustava pokreće isti program.
- Trebao bi imati ugrađenu podršku računalne mreže.
- Trebao bi biti konfiguriran za sigurno izvršavanje koda iz udaljenih izvora.
- Odabirom koji se dijelovi drugih objektno orijentiranih jezika smatraju prihvatljivima. Trebao bi biti jednostavan za korištenje.[2]

2.2.1 Objektno orijentirani pristup

Objektno orijentiran pristup programiranju i dizajnu temelji se na organiziranju procesa oko entiteta koji kombiniraju podatke i operacije. Entiteti, poznati kao objekti, sadrže kod i stanje koje se može samostalno mijenjati. Ključno načelo je dizajniranje programa za manipulaciju različitim tipovima podataka putem odgovarajućih operacija. Podaci se integriraju s kodom kako bi se stvorili entiteti koji pružaju pouzdanu osnovu za dizajn softverskog sustava. Cilj je poboljšati upravljanje projektima, povećati kvalitetu i smanjiti broj neuspjelih projekata.[2]

Objektna orijentiranost također teži stvaranju uobičajenih artefakata koji omogućuju ponovno korištenje softvera među projektima. Na primjer, generički objekt "kupac" trebao bi imati sličan skup ponašanja koji se koristi u različitim projektima, što olakšava preklapanje funkcionalnosti unutar organizacija. Softverski objekti se mogu tretirati kao plug-in (umetljive) komponente, što omogućuje brže razvojno vrijeme i smanjuje ovisnost o novim implementacijama. Ipak, izazovi poput konceptualnih razumijevanja i komunikacijskog pristupa generičkim objektima često predstavljaju prepreku.[2]

Uz to, objektno orijentirani pristup uključuje 4 osnovna principa programiranja: apstrakciju, enkapsulaciju, nasljeđivanje i polimorfizam.[2]

Apstrakcija podataka je svojstvo koje prikazuje samo bitne detalje korisniku, dok se trivijalne ili nebitne jedinice zanemaruju. Na primjeru automobila, korisnik vidi automobil kao cjelinu, a ne kao skup pojedinačnih komponenti. Ova vrsta apstrakcije podataka fokusira se na identifikaciju

važnih karakteristika objekta, dok se nevažni detalji zanemaruju. Svojstva i ponašanja objekta razlikuju ga od drugih sličnih objekata i pomažu u njegovom klasificiranju ili grupiranju.[4]

U stvarnom životnom primjeru, ne moramo znati što se zapravo događa kad koristimo lift u zgradi, već samo da nas on može dovesti na željeni kat pritiskom gumba. To je suština apstrakcije - skrivanje složenosti detalja iza jednostavnog sučelja koje korisnik može koristiti.[4]

U programskom jeziku poput Jave, apstrakcija se postiže korištenjem sučelja i apstraktnih klasa. Sučelja omogućuju potpunu apstrakciju definiranjem samo potpisa metoda, bez njihove implementacije. S druge strane, apstraktne klase sadrže apstraktne metode koje su definirane bez implementacije, omogućujući podklasama da ih implementiraju prema potrebi.[4]

Enkapsulacija je mehanizam koji omogućuje omotavanje podataka u jednu jedinicu. Ovaj proces povezuje kod i podatke s kojima manipulira, osiguravajući da pristup podacima izvan te jedinice bude ograničen. Drugi aspekt enkapsulacije je njezina uloga kao zaštitnog štita koji sprječava izravan pristup podacima iz drugih dijelova koda. Tehnički, enkapsulacija omogućuje da se varijable ili podaci unutar klase skrivaju od vanjskog pristupa te im se može pristupiti samo putem metoda koje su definirane unutar iste klase. Koncepti "enkapsulacija" i "skrivanje podataka" često se koriste kao sinonimi, budući da oba principa podrazumijevaju zaštitu podataka od izravnog vanjskog pristupa.[4]

Postizanje enkapsulacije uključuje deklariranje svih varijabli u klasi kao privatnih te implementaciju javnih metoda unutar iste klase za postavljanje i dohvaćanje vrijednosti tih varijabli.[4]

Nasljeđivanje je ključni element objektno orijentiranog programiranja, omogućujući jednoj klasi da naslijedi značajke (polja i metode) druge klase. U Javi se nasljeđivanje postiže korištenjem ključne riječi extends (proširuje). Također se naziva odnosom "je" (primjerice, auto je vozilo). Evo pregleda nekih važnih termina:

- Superklasa: Klasa čije se karakteristike nasljeđuju naziva se superklasa (također poznata kao osnovna ili roditeljska klasa).
- Podklasa: Klasa koja nasljeđuje drugu klasu zove se podklasa (također poznata kao izvedena ili proširena klasa). Podklasa može dodati vlastita polja i metode uz polja i metode nadklase.

Ponovna upotreba: Nasljeđivanje podržava koncept ponovne upotrebe, omogućujući stvaranje nove klase koja koristi dio koda postojeće klase. Na taj način se ponovno koriste polja i metode postojeće klase.[4]

Polimorfizam je sposobnost objektno orijentiranih programskih jezika da učinkovito razlikuju entitete s istim imenom. Java to postiže kroz potpise i deklaracije tih entiteta. Mogućnost pojavljivanja u različitim oblicima naziva se polimorfizam. U Javi, polimorfizam se obično manifestira kroz 2 slučaja: preopterećenje i nadjačavanje.[4]

Preopterećenje metoda omogućuje različitim metodama da imaju isto ime, ali različite parametre, koje se mogu razlikovati prema broju ulaznih parametara, vrsti ulaznih parametara ili njihovoj kombinaciji.[5]

Nadjačavanje je značajka koja dopušta podklasi ili izvedenoj klasi da pruži specifičnu implementaciju metode koju već nudi jedna od njenih nadređenih klasa ili roditeljskih klasa. Kada metoda u podklasi ima isto ime, iste parametre ili potpis i isti povratni tip kao metoda u njenoj nadređenoj klasi, tada se kaže da metoda u podklasi nadjačava metodu u nadređenoj klasi.[4][6]

2.2.2 Neovisnost o platformi

Neovisnost o platformi za Javu znači da programi napisani u njoj moraju jednako raditi na različitim hardverskim platformama. Prema osnovnoj filozofiji Jave, program treba biti sposoban biti napisan jednom i izvršen bilo gdje. Većina Java prevoditelja ostvaruje ovo kompajliranjem Java koda u polu-bajt kod, što su pojednostavljene strojne instrukcije specifične za Java platformu. Kod se potom izvršava na virtualnom stroju koji tumači i izvršava generički Java bajt kod na host računalu. Također, postoje standardizirane biblioteke koje omogućuju jednoliki pristup značajkama domaćina, poput grafike, dretvi i umrežavanja. [2]

Napomena da, iako je faza kompilacije precizna, JIT (just-in-time, hrv. baš kad je potrebno) kompajler negdje tumači ili pretvara Java bajt kod u izvorne strojne naredbe. Java prevoditelji su također implementirani za kompajliranje izvornog objektnog koda, kao što je GCJ (GNU Compiler for Java), radi uklanjanja intermedijskih bajt kodova. Ipak, svaka arhitektura može pokretati ove prevoditelje. Sunova Java licenca zahtijeva "kompatibilnost" sa svim implementacijama, što je dovelo do pravnog spora s Microsoftom zbog neusklađenosti RMI (Remote Method Invocation) i

JNI (Java Native Interface) sučelja s Microsoftovom implementacijom. Microsoft više neće uključivati Javu sa sustavom Windows, a Internet Explorer neće podržavati male Java aplikacije bez dodatka treće strane u novijim verzijama sustava Windows. Unatoč tome, Sun i drugi su osigurali besplatne Java runtime sustave za te i druge verzije Windowsa. Rane implementacije jezika koristile su kodirani virtualni stroj za prenosivost, što je stvorilo programe koji su radili sporije od onih kompajliranih u izvorne izvršne datoteke kao što su C ili C++. Novije JVM (Java Virtual Machine) implementacije generiraju programe s višestrukim tehnikama koje rade značajno brže.[2]

Prva tehnika, prevođenje izvornog koda izravno u konvencionalni prevoditelj, potpuno preskače bajt kodove, što donosi dobre performanse ali uz smanjenu prenosivost. Druga tehnika, pravovremena kompilacija (just-in-time compilation), pretvara Java bajt kodove u izvorni kod tijekom izvođenja. To omogućuje programu da radi brže nego kod interpretacije i da se kompajlira dinamički tijekom izvođenja.[2]

Napredniji VM-ovi koriste dinamičku rekompilaciju kako bi selektivno optimizirali ključne dijelove softvera, koristeći okruženje za izvođenje i kolekciju učitanih klasa za optimizaciju. Dinamičko ponovno prevođenje omogućuje Java programima da iskoriste brzinu izvornog koda bez gubitka prenosivosti. [2]

2.2.3 Automatsko prikupljanje smeća

Jedan od ključnih koncepta Javinog modela automatskog upravljanja memorijom je olakšanje programerima od stresa ručnog upravljanja memorijom. U određenim jezicima, programer mora ručno dodijeliti memoriju za izgradnju objekata pohranjenih u gomili te zatim ručno upravljati njihovom memorijom kako bi ih uklonio. Moguće je da dođe do curenja memorije kada programer zaboravi osloboditi memoriju i ukloniti te objekte, što može dovesti do značajnog gubitka memorije. Također, program može postati nestabilan i "pucati" kada je programu nedostupna potrebna količina memorije.[2]

Automatsko prikupljanje smeća u Javi izbjegava ove potencijalne probleme. Odluka o stvaranju objekata ostaje na programeru, dok Java automatski upravlja životnim ciklusom objekata. Java skupljač smeća automatski briše objekte koji više nisu dostupni, oslobađajući memoriju i sprječavajući curenje memorije. Curenje memorije također može nastati ako programski kod sadrži referencu na objekt koji više nije potreban..[2]

3. SPRING

Spring je tipično opisivan kao lagani radni okvir za izradu Java aplikacija, no ovdje dolazimo do dvije zanimljive stvari kojih ćemo se dotaknuti kasnije.

Spring Framework je aplikacijski okvir i spremnik za inverziju kontrole za Java platformu. Osnovne karakteristike Spring-a su primjenjive u svim Java aplikacijama, s dodatnim mogućnostima za razvoj web aplikacija. Također podržava i funkcionalnosti Java Enterprise Edition platforme. Spring ne nameće specifičan model programiranja. U Java zajednici, Spring je postao popularan kao alternativa Enterprise JavaBeans modelu te se distribuira kao besplatan softver otvorenog koda. [7]

Što se tiče zanimljivih stvari oko Springa, prvo, Spring se može koristiti za izradu bilo koje aplikacije u Javi, na primjer, samostalne, web ili Java Enterprise Edition aplikacije, za razliku od mnogih drugih okvira kao što je Apache Struts, koji je ograničen na web aplikacije. Drugo, "lagani" dio opisa zapravo se ne odnosi na broj klasa ili veličinu distribucije nego definira načelo Spring filozofije u cjelini – to jest, minimalno invazivno. Spring je lagan u smislu da je potrebno napraviti vrlo malo ili ništa promjena u kodu aplikacije kako bi se iskoristile prednosti Spring Core-a (Jezgre Springa), a u slučaju odluke o prestanku korištenja Spring-a u bilo kojem trenutku, to je prilično jednostavno. Spring Framework je Java platforma koja pruža sveobuhvatnu infrastrukturnu podršku za razvoj Java aplikacija. Spring upravlja infrastrukturom tako da se programeri mogu usredotočiti na svoju aplikaciju.[8]

3.1 Zašto je nastao Spring

Prije pojave Enterprise Java Beansa, Java programeri su se morali oslanjati na JavaBeans za izradu web aplikacija. Iako su JavaBeans pomogli u razvoju korisničkog sučelja (UI) komponenata, nisu bili sposobni pružiti različite usluge kao što su upravljanje transakcijama i sigurnost, koje su ključne za robusne i sigurne poslovne aplikacije. Pojava Enterprise Java Beansa riješila je ovaj problem. Enterprise Java Beans proširuje Java komponente, uključujući web i poslovne komponente, te pruža usluge koje olakšavaju razvoj poslovnih aplikacija. Međutim, razvoj s Enterprise Java Beansom nije bio jednostavan, jer je programer morao obavljati zadatke poput stvaranja Home i Remote sučelja te implementacije metoda povratnog poziva u životnom ciklusu, što je dodatno otežavalo razvoj EJB-ova (Enterprise Java Beans).[7]

Okvir Spring je nastao kao rješenje za ove izazove. Ovaj okvir koristi moderne pristupe poput aspektno orijentiranog programiranja (AOP), običnih Java objekata (POJO) i ubacivanja ovisnosti (DI) za razvoj poslovnih aplikacija, što eliminira složenosti povezane s Enterprise Java Beansom. Spring je lagani okvir otvorenog koda koji omogućuje izgradnju jednostavnih, pouzdanih i skalabilnih poslovnih aplikacija. Fokusira se na pružanje različitih metoda za upravljanje poslovnim objektima, čime je značajno olakšao razvoj web aplikacija u usporedbi s tradicionalnim Java okvirima i API-ima (Application Programming Interface) kao što su Java Database Connectivity (JDBC), JavaServer Pages (JSP) i Java Servlets. [9]

Spring ima veliku i aktivnu zajednicu koja pruža stalne povratne informacije na temelju različitih slučajeva korištenja iz stvarnog svijeta. To je pomoglo Springu da se uspješno razvija tijekom jako dugog vremena.

Spring omogućuje gradnju aplikacije od "običnih Java objekata". Ova se mogućnost odnosi na programski model Java SE i na potpuni i djelomični Java Enterprise Edition.

Neki od primjera kako programer aplikacija može koristiti prednost platforme Spring:

- Izvršit Java metodu u transakciji baze podataka bez potrebe za korištenjem transakcijskih API-ja.
- Učiniti lokalnu Java metodu udaljenom procedurom bez rada s udaljenim API-ima.
- Učiniti lokalnu Java metodu operacijom upravljanja bez potrebe za radom s JMX (Java Management Extensions) API-ima.
- Učiniti lokalnu Java metodu rukovateljem porukama bez potrebe za radom s JMS (Java Message Service) API-ima.[10]

3.2 Značajke Springa

Postoji mnogo značajki Springa koje se mogu opisati u ovom radu, no počet ćemo prvo s onim glavnima, a to su inverzija kontrole, injektiranje ili umetanje ovisnosti i aspektno orijentirano programiranje. Navedene značajke Spring okvira čine ga jedinstvenim na popisu okvira.

3.2.1 Inverzija kontrole

Inverzija kontrole predstavlja dizajnerski princip koji preokreće razne vrste kontrole u objektno orijentiranom dizajnu, omogućujući time labavu povezanost među objektima. Ove kontrole uključuju sve dodatne odgovornosti koje klasa može imati osim svoje glavne zadaće, poput upravljanja tokom aplikacije i stvaranja ili povezivanja objekata.[11]

Inverzija kontrole odnosi se na prepuštanje kontrole drugome. U svakodnevnom jeziku, to možemo objasniti ovako: zamislite da svakodnevno vozite auto do posla, što znači da sami kontrolirate vožnju. Prema principu inverzije kontrole, umjesto da vozimo, mogli bismo unajmiti taksi gdje vozač preuzima kontrolu. Na taj način, umjesto da mi upravljamo automobilom, to radi vozač taksija, a mi se možemo posvetiti svom glavnom poslu. Ovaj princip olakšava dizajniranje klasa koje su labavo povezane, što ih čini jednostavnijima za testiranje, održavanje i proširenje.[11]

No, za razliku od tradicionalnog programiranja, gdje naš prilagođeni kod poziva biblioteku, inverzija kontrole omogućava Spring radnom okviru preuzimanje kontrole nad tokom programa i poziva naš prilagođeni kod. Kako bi se to omogućilo, radni okvir koristi apstrakcije s ugrađenim dodatnim ponašanjem. Ako želimo dodati vlastito ponašanje, moramo proširiti klase radnog okvira ili uključiti vlastite klase.

Prednosti ove arhitekture su:

- odvajanjem izvršavanja zadatka od njegove implementacije, čime se olakšava prelazak između različitih implementacija,
- većom modularnošću programa,
- lakšim testiranjem programa izoliranjem komponente ili simuliranjem njezinih ovisnosti, te omogućavanjem komunikacije između komponenti putem ugovora.

Inverziju kontrole možemo postići kroz razne mehanizme, kao što su: uzorak dizajna strategije, uzorak lokatora usluge, obrazac tvornice i injektiranje ovisnosti. [12]

3.2.2 Injekcija ovisnosti

Injekcija ovisnosti je obrazac dizajna koji smanjuje povezanost među komponentama aplikacije. Pomoću injekcije ovisnosti, Spring okvir omogućuje labavu povezanost među

komponentama, čineći aplikaciju modularnijom i lakšom za održavanje. Spring okvir nudi kontejner za inverziju kontrole, koji stvara i upravlja instancama JavaBeana, te ApplicationContext, koji pruža cjelovit pregled konfiguracije aplikacije.[13]

Također, može se reći da je injekcija ovisnosti specifičan oblik inverzije kontrole, gdje objekti definiraju svoje ovisnosti (tj. druge objekte s kojima surađuju) samo putem konstruktorskih argumenata, metoda argumenata ili svojstava koja se postavljaju na instancu objekta nakon stvaranja ili vraćanja iz metode. Kontejner inverzije kontrole zatim ubrizgava te ovisnosti prilikom stvaranja beana (Spring objekta). Ovaj proces je zapravo obrnut (odakle i naziv, inverzija kontrole) od toga da bean sam upravlja instanciranjem ili pronalaženjem svojih ovisnosti koristeći izravno stvaranje klasa ili mehanizam poput obrazaca lokatora usluge.[13]

3.2.3 Spring kontejner inverzije kontrole

Sučelje "org.springframework.context.ApplicationContext" predstavlja Spring kontejner za inverziju kontrole, zadužen za instanciranje, konfiguriranje i sastavljanje bean-ova. Kontejner dobiva upute o komponentama koje treba instancirati, konfigurirati i sastaviti čitanjem konfiguracijskih metapodataka. Konfiguracijski metapodaci mogu biti prikazani kao označene klase komponenti, konfiguracijske klase s tvorničkim metodama ili vanjske XML datoteke ili Groovy skripte. Korištenjem bilo kojeg od ovih formata, možete sastaviti svoju aplikaciju i složene međuovisnosti između tih komponenti.[14]

Paketi org.springframework.beans i org.springframework.context čine temelj Spring kontejnera za inverziju kontrole. Sučelje BeanFactory nudi napredni mehanizam za konfiguraciju koji može upravljati bilo kojom vrstom objekta. ApplicationContext je proširenje BeanFactory sučelja. Ono omogućuje lakšu integraciju sa značajkama Springovog aspektno orijentiranog programiranja, kao što su rukovanje porukama, objavljivanje događaja i specifični konteksti sloja aplikacije poput WebApplicationContext za upotrebu u web aplikacijama. Ukratko, BeanFactory pruža osnovnu konfiguracijsku podršku i funkcionalnost, dok ApplicationContext dodaje dodatne funkcije specifične za poslovne aplikacije.[15]

Beanovi su Java objekti koje Spring kontejner za inverziju kontrole konfigurira u vrijeme izvođenja. BeanFactory je osnovni kontejner za inverziju kontrole i roditeljsko je sučelje za ApplicationContext. BeanFactory koristi metapodatke o beanovima i njihovim ovisnostima kako bi ih stvorio i konfigurirao tijekom izvođenja. BeanFactory učitava definicije beanova i njihove

ovisnosti iz konfiguracijske datoteke (XML), ili se beanovi mogu izravno vratiti kada su potrebni pomoću Java konfiguracije. Postoje i druge vrste konfiguracijskih datoteka, poput LDAP (Lightweight Directory Access Protocol), datoteka sa svojstvima itd. BeanFactory ne podržava konfiguraciju temeljenu na anotacijama, dok ApplicationContext to omogućava.[16]

U većini slučajeva programski kod nije potreban za instanciranje jedne ili više instanci Spring kontejnera za inverziju kontrole. Na primjer, u scenariju web aplikacije, nekoliko redaka standardnog XML web deskriptora u datoteci web.xml aplikacije obično je dovoljno. Ako koristite Spring Tool Suite integriran s razvojnim okruženjem Eclipse, ova se konfiguracija može jednostavno stvoriti s nekoliko klikova mišem..[13]

3.2.4 Aspektno orijentirano programiranje

Aspektno orijentirano programiranje (AOP) koristi aspekte u programiranju, kako sam naziv sugerira. Može se definirati kao proces razbijanja koda na različite module, poznato kao modularizacija, pri čemu je aspekt osnovna jedinica modularnosti. Aspekti omogućuju implementaciju poprečnih veza kao što su transakcije, logiranje i sigurnost, koje nisu središnje za poslovnu logiku, bez nepotrebnog natrpavanja ključnog koda. Ovo se postiže dodavanjem dodatnog ponašanja postojećem kodu. Na primjer, sigurnost je poprečna briga; sigurnosna pravila mogu se primijeniti na mnoge metode u aplikaciji, a ponavljanje tog koda u svakoj metodi bilo bi neučinkovito. Umjesto toga, funkcionalnost definiramo u zajedničkoj klasi i kontroliramo njezinu primjenu kroz cijelu aplikaciju.[17]

Kako aspektno orijentirano programiranje radi sa Springom: Možda bi se moglo pretpostaviti da pozivanje metode automatski implementira poprečne veze, ali to nije slučaj. Samo pozivanje metode ne pokreće posao koji treba obaviti. Spring koristi mehanizam temeljen na proxy (hrv. zamjenskim) objektima, tj. stvara proxy objekt koji će izvršavati posao povezan s pozivom metode. Proxy objekti mogu se ručno stvoriti putem proxy tvorničkog beana ili automatskom proxy konfiguracijom u XML datoteci, a uništavaju se nakon završetka izvršenja. Proxy objekti koriste se za obogaćivanje izvorne funkcionalnosti stvarnog objekta.[17]

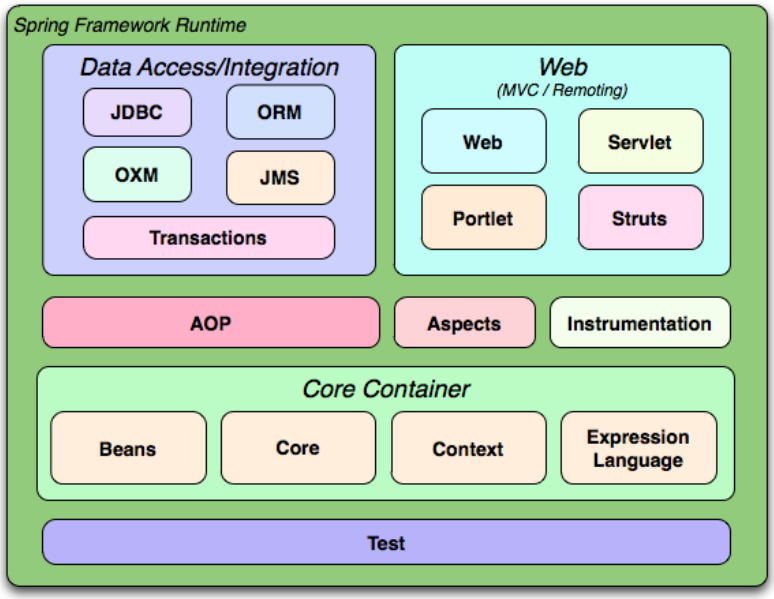
3.3 Spring moduli

Okvir Spring može se smatrati skupom pod-okvira, koji se nazivaju i slojevi, kao što je Spring AOP, Spring objektno-relacijsko mapiranje (Spring ORM), Spring Web Flow i Spring Web

MVC (Model-View-Controller). Bilo koji od ovih modula mogu se koristiti odvojeno tijekom izrade web aplikacije. Moduli se također mogu grupirati zajedno kako bi pružili bolje funkcionalnosti u web aplikaciji. Spring okvir je labavo povezan zbog umetanja ovisnosti. [9]

Spring okvir sadrži dvadesetak modula, koji se mogu podijeliti na:

- osnovni spremnik (eng. Core Container)
- pristup podacima i integracija (eng. Data Access and Integration)
- web
- aspektno orijentirano programiranje (eng. Aspect Oriented Programming)
- instrumentacija (eng. Instrumentation)
- aspekti (eng. Aspects)
- testiranje (eng. Test)



Slika 1: Podjela Spring modula

Osnovni kontejner sastoji se od modula Core (jezgra), Beans, Context (kontekst) i Expression Language (izrazi). Moduli Core i Beans pružaju osnovne komponente okvira, uključujući značajke kao inverzija kontrole i injekcije ovisnosti. BeanFactory je sofisticirana implementacija tvorničkog obrasca. Eliminira potrebu za programskim singletonima (jedina instanca) i omogućuje razdvajanje konfiguracije i specifikacije ovisnosti od stvarne logike programa.[10]

Context modul gradi na čvrstoj osnovi koju pružaju Core i Beans moduli. To je način pristupa objektima u stilu okvira, slično JNDI (Java Naming and Directory Interface, hrv. Java sučelje imenovanja i imenika) registru. Context modul nasljeđuje značajke od Beans modula i dodaje podršku za internacionalizaciju, širenje događaja, učitavanje resursa te transparentno stvaranje konteksta, kao što je pomoću servlet kontejnera. Također podržava Java EE (Enterprise Edition) značajke poput EJB, JMX i osnovnog udaljenog pristupa. ApplicationContext sučelje je središnji element Context modula.[10]

Expression Language modul nudi moćan jezik izraza za upite i manipulaciju objektima u vrijeme izvođenja. Proširen jezik ujedinjenog izraza (unified EL) kako je specificirano u JSP 2.1 specifikaciji. Podržava postavljanje i dohvaćanje vrijednosti svojstava, dodjeljivanje svojstava, pozivanje metoda, pristupanje kontekstu nizova, kolekcija i indeksa, logičke i aritmetičke operacije, imenovane varijable te dohvaćanje objekata po imenu iz Spring Inverzija kontrole kontejnera. Također podržava projekciju i odabir elemenata liste, kao i uobičajene agregacije lista.[10]

3.3.1 Pristup podacima i integracija

Sloj pristupa podacima/integracije sastoji se od modula JDBC, ORM, OXM (Object/XML), Java Messaging Service (JMS) i Transaction (hrv. transakcija). JDBC modul pruža apstrakcijski sloj koji eliminira potrebu za zamornim JDBC kodiranjem i rukovanjem specifičnim kodovima pogrešaka baze podataka.[10]

- ORM modul nudi integracijske slojeve za popularne API-je za objektno-relacijsko mapiranje, uključujući JPA (Java Persistence API), JDO (Java Data Objects) i Hibernate. Korištenjem ORM paketa možete kombinirati ove ORM okvire sa svim ostalim značajkama koje Spring nudi, poput jednostavnog deklarativnog upravljanja transakcijama.
- OXM modul pruža apstrakcijski sloj za mapiranje Object/XML, podržavajući implementacije poput JAXB, Castor, XMLBeans, JiBX i XStream.
- Modul Java Messaging Service sadrži značajke za proizvodnju i potrošnju poruka.
- Transakcijski modul podržava programsko i deklarativno upravljanje transakcijama za klase koje implementiraju specifična sučelja, kao i za sve vaše POJO (obične Java objekte) [10]

3.3.2 Web

Web sloj sastoji se od modula Web, Web-Servlet, WebSocket i Web-Portlet. Springov web modul pruža osnovne integracijske značajke orijentirane na web, kao što su funkcionalnost učitavanja višedijelnih datoteka i inicijalizacija spremnika inverzije kontrole pomoću servleta i konteksta aplikacije orijentiranog na web. Također, sadrži dijelove Springove daljinske podrške povezane s webom.[10]

Web-Servlet modul sadrži Springovu implementaciju model-view-controller za web aplikacije. Springov MVC okvir omogućuje jasno razdvajanje koda modela domene i web obrazaca te se integrira sa svim ostalim značajkama Spring Frameworka.[10]

Web-Portlet modul pruža MVC implementaciju za portlet okruženja, reflektirajući funkcionalnost Web-Servlet modula [10]

3.3.3 Aspektno orijentirano programiranje i instrumentacija

Springov AOP modul pruža aspektno orijentiranu programsku implementaciju usklađenu s AOP Allianceom koja vam omogućuje definiranje, na primjer, presretača metoda i isječaka za čisto odvajanje koda koji implementira funkcionalnost koja bi trebala biti odvojena. Koristeći funkcionalnost metapodataka na razini izvora, također možete ugraditi informacije o ponašanju u svoj kod, na način sličan onom s .NET atributima.

Odvojeni modul Aspects omogućuje integraciju s AspectJ.

Modul instrumentacije pruža podršku za instrumentaciju klase i implementacije učitavača klasa koje se koriste u određenim aplikacijskim poslužiteljima. [10]

Aspektno orijentirano programiranje nije specifično za Spring, a ni njegova terminologija nije jedinstvena za Spring. Ovdje su osnovni pojmovi:

- Aspekt: Modularizacija brige koja se proteže kroz više klasa. Primjer međusektorske brige u Java aplikacijama poduzeća je upravljanje transakcijama. U Spring AOP-u aspekti se

implementiraju korištenjem običnih klasa (pristup temeljen na shemi) ili klasa označenih @Aspect anotacijom (stil @AspectJ).

- Točka spajanja: Točka tijekom izvođenja programa, poput izvođenja metode ili rukovanja iznimkom. U Spring AOP-u točka spajanja uvijek se odnosi na izvođenje metode.
- Savjet: Akcija koju aspekt poduzima na određenoj točki spajanja. Postoje različite vrste savjeta, uključujući "oko", "prije" i "poslije" savjete. Mnogi AOP okviri, uključujući Spring, modeliraju savjet kao presretač i održavaju lanac presretača oko točke spajanja.
- Pointcut: Predikat koji odgovara spojnim točkama. Savjet se povezuje s pointcut izrazom i izvodi se na bilo kojoj točki spajanja koja odgovara pointcutu (npr. izvođenje metode s određenim nazivom). Koncept spojnih točaka usklađenih s pointcut izrazima je središnji za AOP, a Spring koristi AspectJ pointcut izrazni jezik kao zadani.
- Uvod: Deklariranje dodatnih metoda ili polja u ime tipa. Spring AOP omogućuje uvođenje novih sučelja (i odgovarajuće implementacije) za bilo koji preporučeni objekt. Na primjer, uvod možete koristiti da bi bean implementirao IsModified sučelje, čime se pojednostavljuje predmemorija. (Uvod je poznat kao međutipska deklaracija u AspectJ zajednici.)
- Ciljani objekt: Objekt kojemu savjetuje jedan ili više aspekata, također poznat kao "savjetovani objekt". Budući da je Spring AOP implementiran korištenjem izvršnih proksija, ovaj objekt je uvijek proksi objekt.
- AOP proxy: Objekt kreiran od strane AOP okvira za implementaciju aspekata (izvršenje metoda savjeta i slično). U Spring Frameworku, AOP proxy može biti JDK dinamički proxy ili CGLIB proxy.
- Tkanje: Povezivanje aspekata s drugim vrstama aplikacija ili objekata kako bi se stvorio preporučeni objekt. To se može dogoditi u vrijeme kompajliranja (koristeći AspectJ kompajler, na primjer), učitavanja ili izvođenja. Spring AOP, kao i drugi čisti Java AOP okviri, izvodi tkanje tijekom izvođenja [18]

3.3.4 Test

Modul Test podržava testiranje Spring komponenti s JUnit ili TestNG. Omogućuje dosljedno učitavanje Spring Application Contexts i predmemoriranje tih konteksta. Također nudi lažne objekte koje možete koristiti za testiranje vašeg koda u izolaciji. [10]

3.4 Spring security

Spring Security je jedan od projekata Spring tima izgrađen korištenjem Spring okvira u Javi. Cilj ovog projekta je olakšati programerima osiguranje web aplikacija protiv uobičajenih napada, kao što su napadi Cross-Site Request Forgery (CSRF). Sadrži kod koji se može prilagoditi ili koristiti takav kakav jest, ovisno o upotrebi. Njegova glavna funkcija je upravljanje autentifikacijom i autorizacijom na razini web zahtjeva i poziva metoda. Interno, Spring Security okvir sadrži niz servlet filtera koji upravljaju raznim aspektima sigurnosti. Iako se pridržava Springovih konvencija za postavljanje, programeri mogu birati između zadanih postavki i prilagoditi ih svojim specifičnim zahtjevima. Spring Security radi na sljedeća četiri osnovna koncepta:

- Autentifikacija – Je li korisnik stvarno onaj za koga se predstavlja?
- Autorizacija – Ima li korisnik odgovarajuću ulogu?
- Pohrana lozinki – Kako se lozinka pohranjuje? U memoriji ili u bazi podataka?
- Servlet filteri – Trebamo li dodati nove filtere ili jednostavno koristiti zadane koje pruža Spring tim?[19]

Spring Security je moćan i visoko prilagodljiv okvir za autentifikaciju i kontrolu pristupa za aplikacije temeljene na Javi. Jedna od značajki Spring Securityja je sposobnost osiguravanja metoda na razini metode, poznata i kao sigurnost na razini metode. To omogućava programerima da specificiraju sigurnosna ograničenja za pojedine metode unutar klase, umjesto da primjenjuju sigurnosna pravila na cijelu klasu ili aplikaciju. Tako se omogućava preciznija kontrola pristupa specifičnim dijelovima aplikacije. Za implementaciju sigurnosti na razini metode u Spring Securityju, programeri mogu koristiti anotacije @Secured i @PreAuthorize.[20]

Ukratko, sigurnost na razini metode u Spring Securityju omogućava programerima primjenu sigurnosnih ograničenja na specifične metode unutar klase umjesto na cijelu klasu ili aplikaciju. Ova značajka se implementira korištenjem anotacija @Secured i @PreAuthorize, omogućavajući finiju kontrolu pristupa specifičnim dijelovima aplikacije. [20]

3.5 Zašto koristiti Spring

Prethodno opisani građevni blokovi čine Spring logičnim izborom u mnogim scenarijima, od appleta do potpuno razvijenih enterprise aplikacija koje koriste Springovu funkcionalnost za upravljanje transakcijama i integraciju s web okvirima.

Springove značajke deklarativnog upravljanja transakcijama omogućuju web aplikaciji potpunu transakcijalnost, kao da koristite EJB upravljane transakcije. Vaša prilagođena poslovna logika može se implementirati koristeći jednostavne POJO-e i upravljati Springovim kontejnerom inverzije kontrole. Dodatne usluge uključuju podršku za slanje emailova i validaciju koja je neovisna o web sloju, omogućavajući vam odabir mjesta izvršavanja pravila validacije. Springova ORM podrška integrirana je s JPA, Hibernate i JDO; primjerice, pri korištenju Hibernatea, možete nastaviti koristiti postojeće mape datoteka i standardnu konfiguraciju Hibernate SessionFactory. Kontrole obrazaca besprijekorno integriraju web sloj s modelom domene, eliminirajući potrebu za ActionForms ili drugim klasama koje transformiraju HTTP parametre u vrijednosti za vaš model domene.[10]

Ponekad situacije ne omogućuju potpuni prijelaz na novi okvir. Spring Framework ne zahtijeva da koristite sve njegove komponente; nije to rješenje sve-ili-ništa. Postojeći front-endovi izgrađeni s okvirima poput Struts, Tapestry, JSF ili drugih UI okvira mogu se integrirati sa srednjim slojem temeljenim na Springu, omogućujući vam korištenje Springovih transakcijskih značajki. Potrebno je samo povezati poslovnu logiku putem ApplicationContext i koristiti WebApplicationContext za integraciju web sloja. Spring Framework također nudi sloj za pristup i apstrakciju za Enterprise JavaBeans, omogućujući vam ponovno korištenje postojećih POJO-a i omotavanje u stateless session beanove za korištenje u skalabilnim, sigurnim web aplikacijama koje možda trebaju deklarativnu sigurnost.[10]

Dakle, Spring Framework je moćan i popularan otvoreni okvir za Javu koji pruža sveobuhvatan model za programiranje i konfiguraciju poslovnih aplikacija. Njegova arhitektura je dizajnirana oko principa modularnosti, odvajanja odgovornosti i fleksibilnosti, pružajući programerima snažan skup alata za izgradnju robusnih, skalabilnih i održivih poslovnih aplikacija.[10]

4. MAVEN

Zamislimo da želimo stvoriti projekt koji se povezuje s MySQL bazom podataka. Takav projekt zahtijeva upotrebu vanjskih biblioteka kako bi naš program mogao uspostaviti vezu s bazom podataka. Uobičajeno je da veći projekti koriste više od jedne biblioteke, a svaku od njih moramo ručno preuzeti s web stranice i dodati u projekt. Ovdje se pojavljuje prvi problem prilikom stvaranja projekta. Za manje projekte s jednom ili dvije vanjske biblioteke, ovo možda nije problem, no kako smo već spomenuli, veći projekti često koriste stotine vanjskih biblioteka koje se moraju ručno dodati u projekt.

Međutim, zamislimo da jednog dana želimo ažurirati naše biblioteke na njihove modernije verzije. Morali bismo ukloniti sve postojeće biblioteke, ručno preuzeti nove verzije i ponovno ih dodati u projekt. Takve situacije ilustriraju važnost alata koji automatiziraju ovaj proces.

Maven je jedan od tih alata, namijenjen upravljanju projektima i olakšavanju razumijevanja njihove strukture. Cilj Maven-a je pojednostaviti proces izgradnje softvera i smanjiti složenost kroz deklarativnu konfiguraciju. Središnji element Maven-a je Project Object Model (POM.xml datoteka) koja centralizira sve informacije o projektu. Razvojni tim može brzo automatizirati infrastrukturu izgradnje projekta jer Maven koristi standardnu organizaciju direktorija i zadani životni ciklus izgradnje.[21]

Maven se često koristi u projektima temeljenim na Javi, olakšavajući upravljanje ovisnostima poput biblioteka ili JAR (Java Archive) datoteka. Alat pomaže u pronalaženju pravih JAR datoteka za svaki projekt, uzimajući u obzir različite verzije paketa koje mogu postojati.[21]

Nakon uvođenja Mavena, više nije potrebno ručno posjećivati web stranice različitih softverskih paketa za preuzimanje ovisnosti. Umjesto toga, možete posjetiti mvnrepository web stranicu kako biste pronašli biblioteke na različitim jezicima. Također, Maven pomaže u definiranju pravilne strukture projekta za struts, servlets, i druge tehnologije, što je ključno za uspješno izvođenje projekta..[21][23]

U slučaju okruženja s više razvojnih timova, Maven može postaviti način rada prema nekim standardima u vrlo kratkom vremenu. Budući da je većina postavki projekta jednostavna i višekratna, Maven olakšava život programera dok stvara izvješća, provjerava, gradi i testira postavke automatizacije. [22]

4.1 POM

POM, poznat kao Project Object Model, je osnovna jedinica rada u Maven alatu. To je XML datoteka smještena u korijenskom direktoriju projekta pod nazivom pom.xml. POM sadrži sve informacije o projektu i razne konfiguracijske detalje koje Maven koristi tijekom izgradnje. Uključuje ciljeve izgradnje i dodatke. Kada izvršava određeni cilj ili zadatak, Maven traži POM datoteku u trenutnom direktoriju, čita je, izvlači potrebne konfiguracijske informacije i zatim izvršava cilj.[24]

Važno je napomenuti da svaki projekt mora imati svoju jedinstvenu POM datoteku. Neke od konfiguracija koje se mogu navesti u POM-u uključuju:

- ovisnosti projekta
- dodatke
- ciljeve
- profile za izgradnju
- verziju projekta
- informacije o programerima
- adresu e-pošte

Prije stvaranja POM datoteke, bitno je definirati grupu projekta (groupId), njegovo ime (artifactId) i verziju, što pomaže jedinstvenom identificiranju projekta u repozitoriju.. [24]

4.2 Životni ciklus izrade

Maven koristi niz definiranih faza koje omogućuju automatizaciju procesa gradnje projekata. Ključne faze su:

- validate: Provjerava ispravnost projekta i svih potrebnih informacija prije početka gradnje.
- compile: Kompilira izvorni kod u binarne datoteke.
- test: Pokreće testove jedinica koji provjeravaju je li kôd ispravan. Ove testove izvršava bez potrebe za instalacijom aplikacije.
- package: Pakira kompilirane klase u formate poput JAR ili WAR (Web Application Resource) datoteka, spremne za distribuciju.
- verify: Provodi dodatne provjere i integracijske testove na paketu kako bi se osiguralo da sve funkcionira ispravno.

- install: Instalira izgrađeni paket u lokalni Maven repozitorij, omogućujući da ga drugi projekti koriste kao ovisnost.
- deploy: Distribuira konačni paket na udaljeni repozitorij, čime postaje dostupan drugim timovima i projektima.

Maven-ov životni vijek omogućuje dosljedan i ponovljiv proces izgradnje, pomažući timovima da učinkovito upravljaju složenim projektima. Svaka faza automatski ovisi o prethodnima, čime se osigurava da su svi koraci pravilno izvršeni prije prelaska na sljedeći.[25]

4.3 Maven dodaci i ovisnosti

Maven kao takav je zapravo skup Maven dodataka kojima alat upravlja. Dodaci obavljaju većinu operacija koje uključuju stvaranje JAR i WAR datoteka, kompiliranje koda, testiranje jedinica i generiranje dokumentacije projekta. Jedna od prednosti koje Maven dodaci omogućuju je ponovnu upotrebu zajedničke logike kroz više projekata. To se postiže izvršavanjem "akcija", kao što su stvaranje JAR datoteke ili provođenje testova u kontekstu opisa projekta u pom.xml datoteci. Ponašanje dodataka može se prilagođavati putem jedinstvenih parametara koji su izloženi u opisu svakog cilja dodatka. [26]

Mavenova ključna značajka je upravljanje ovisnostima. Upravljanje ovisnostima u jednom projektu je jednostavno, ali također se omogućuje upravljanje ovisnostima u više-modularnim projektima i aplikacijama koje mogu sadržavati stotine modula. Maven značajno olakšava definiranje, kreiranje i održavanje ponovljivih verzija s jasno definiranim putanjama klasa i verzijama biblioteka, a također automatski uključuje i tranzitivne ovisnosti, čime izbjegava potrebu za ručnim definiranjem biblioteka i njihovih ovisnosti.[27]

Ova funkcionalnost omogućena je čitanjem projektnih datoteka ovisnosti iz udaljenih repozitorija. Sve ovisnosti projekata koriste se u projektu, uključujući one koje projekt nasljeđuje od roditeljskih projekata ili drugih ovisnosti, bez obzira na razinu na kojoj se nalaze. Jedini problem može nastati ako se otkrije ciklička ovisnost. [27]

Cikličke ovisnosti nastaju kada dvije ili više komponente ovise jedna o drugoj, stvarajući petlju koju je teško prekinuti jer svaka komponenta treba drugu da ispravno funkcionira. To može dovesti do koda koji je teško održavati i može uzrokovati probleme prilikom pokušaja unosa izmjena. [28]

5. PRIKAZ APLIKACIJE

Aplikacija izrađena za potrebe ovog završnog rada je jednostavna konzolna aplikacija koja izvršava unos podataka u MySQL bazu podataka. Ona se bavi upisom zaposlenika u bazu podataka, te kreiranje korisničkog imena, lozinke i role unutar područja programa. Ovu vrstu aplikacije mogu koristiti tvrtke koje žele ograditi pristup podacima samo na određene role. Na primjer, admin može pristupiti svim podacima, dok korisnik može pristupiti samo određenim podacima ili metodama unutar programa.

5.1 Početak izrade aplikacije

Za pripremu aplikacije potrebno je napraviti dvije stvari: napraviti strukturu projekta i odabrati bazu podataka koja će se koristiti za spremanje podataka. Logično, za strukturu projekta će se pobrinuti Spring i Maven tehnologije, dok sam za bazu podataka izabrao MySQL bazu.

5.1.1 Spring initializr

Spring Initializr[29] je web alat koji pruža korisničko sučelje za generiranje Spring projekata. Određivanjem metapodataka projekta, postavki sustava za izgradnju (Maven ili Gradle), ovisnosti i drugih konfiguracija, programeri mogu brzo generirati strukturu projekta prilagođenu njihovim potrebama. Ovaj pristup programerima omogućuje da se usredotoče na pisanje koda, a ne na često zamoran proces postavljanja. Prema slici 2, možemo vidjeti osnovne postavke koje nam omogućuje Spring Initializr, a uz to i odabrane postavke za našu aplikaciju. Kao kratki opis postavki:

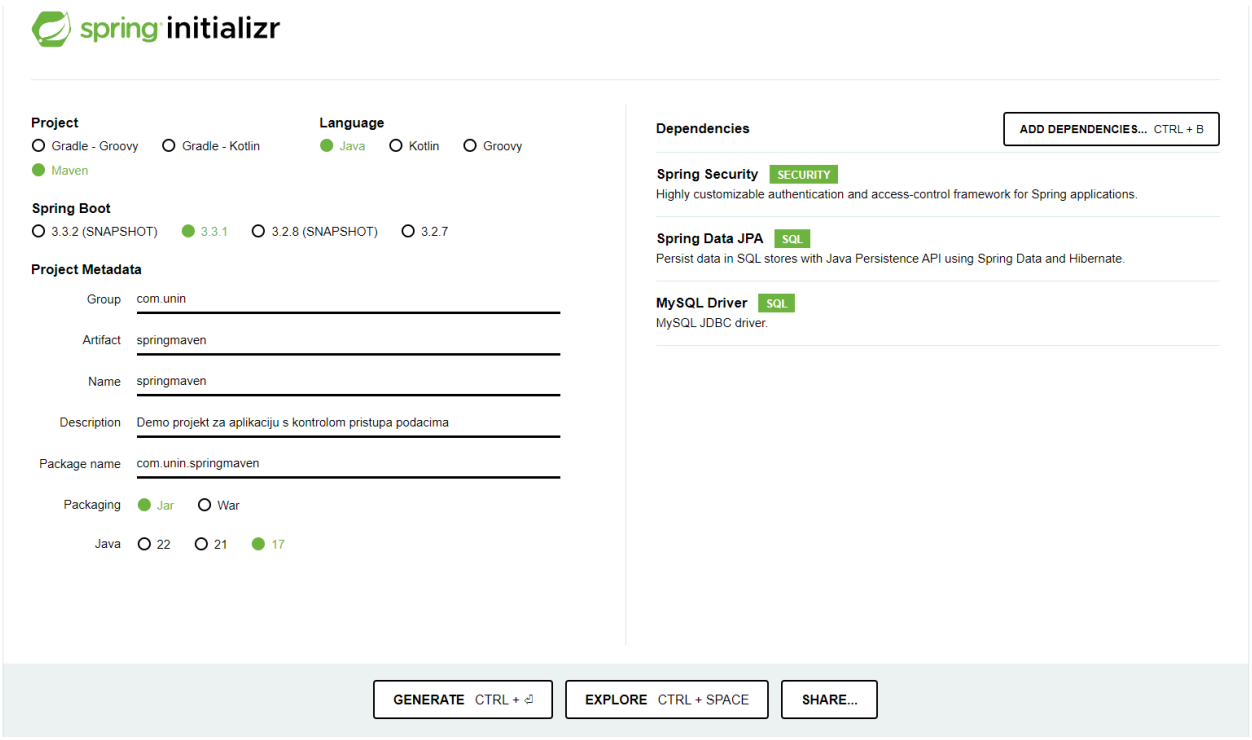
1. **Definiranje alata za izgradnju projekta**
   - Maven Projekt: Koristi Maven kao alat za izgradnju. Maven koristi XML datoteku (pom.xml) za upravljanje ovisnostima projekta, dodacima i konfiguracijama izgradnje.
   - Gradle Projekt: Koristi Gradle kao alat za izgradnju. Gradle koristi build.gradle datoteku napisanu u Groovy ili Kotlin jeziku za konfiguraciju izgradnje.
2. **Programski jezik:** Specifikacija programskog jezika (Java, Kotlin ili Groovy)
3. **Verzija Spring Boota**: Verzija Spring Boota koja će se koristiti za projekt. Preporučuje se odabrati najnoviju stabilnu verziju osim ako u svrhu pojedinog projekta nemamo specifične zahtjeve ili potrebe za drugom verzijom.
4. **Polja metapodataka projekta**

- **Group:** Tipično obrnuto ime domene (npr. com.example). Koristi se kao imenski prostor za vaš projekt kako bi se izbjegli sukobi.
- **Artifact**: Ime projekta (npr. demo). Ovo ime će se koristiti za JAR datoteku projekta i osnovni naziv paketa.
- **Naziv:** Prikazni naziv projekta (npr. demo). Ovo može biti isto kao i artefakt.
- **Opis**: Kratak opis projekta.
- **Ime paketa:** Osnovni naziv paketa (npr. com.example.demo). Uobičajeno je da se generira iz polja group i artifact.

5. **Pakiranje:** Definira kako će aplikacija biti upakirana.
   - Jar: JAR datoteka koja se može izvršiti s java -jar.
   - War: WAR datoteka koja se može postaviti na vanjski server.
6. **Java verzija:** Verzija Jave koja će se koristiti za projekt. Preporučuje se korištenje najnovije LTS (Long Term Support) verzije osim ako u svrhu pojedinog projekta nemamo specifične zahtjeve ili potrebe za drugom verzijom.
7. **Ovisnosti**: Spring Initializr omogućuje dodavanje ovisnosti vašem projektu. Ovisnost nije ništa drugo nego vanjska biblioteka koja pruža specifične funkcije koje možemo koristiti u našoj aplikaciji.
   - Spring Security: Za osiguranje web aplikacija. Omogućuje implementaciju autentifikacije i autorizacije za vašu aplikaciju.
   - SQL: Ovisnosti za rad s SQL bazama podataka.
   - Spring Data JPA: Za rad s JPA-baziranim repozitorijima. Omogućuje jednostavnu integraciju s različitim JPA implementacijama.
   - MySQL Driver: JDBC driver za MySQL bazu podataka. Omogućuje povezivanje i rad s MySQL bazama podataka.

5.1.2 Generiranje projekta

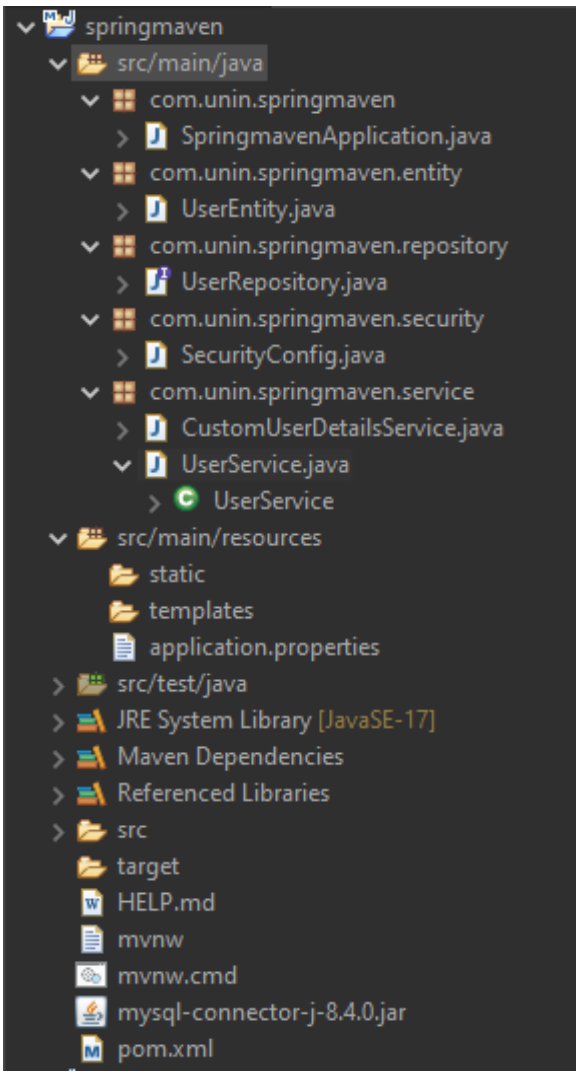
Nakon što odaberemo sve željene opcije i ovisnosti, možemo generirati projekt. Spring Initializr će spremiti projekt kao ZIP datoteku na naše računalo koja sadrži sve potrebne datoteke i konfiguracije za početak rada sa Spring Boot aplikacijom. Nakon toga, možemo pokrenuti bilo koje integrirano razvojno okruženje i uvesti naš projekt.



Slika 2: Osnovne postavke dostupne preko Spring Initializr web alata.

5.2 Struktura aplikacije

Projekt je strukturiran na način kako je tipični Maven ili Gradle projekt strukturiran, a sve datoteke mogu se vidjeti na slici 3.



Slika 3: Struktura projekta

1. SpringmavenApplication.java
   - Ovo je glavna klasa označena s @SpringBootApplication. Služi kao ulazna točka za aplikaciju Spring Boot. Inicijalizira Spring kontekst i pokreće aplikaciju.
2. Paket entiteta
   - UserEntity.java: Predstavlja korisnički entitet, obično preslikan u tablicu baze podataka. Uključuje polja poput korisničkog imena, lozinke, uloge i drugih korisničkih pojedinosti.
3. Repozitorij paket
   - UserRepository.java: sučelje koje proširuje JpaRepository (ili slično). Omogućuje CRUD operacije za UserEntity i prilagođene metode upita.
4. Sigurnosni paket
   - SecurityConfig.java: Konfigurira sigurnosne postavke za aplikaciju. Postavlja autentifikaciju, autorizaciju i druge sigurnosne aspekte poput kodiranja lozinke.

- CustomUserDetailsService.java: Implementira UserDetailsService. Učitava podatke specifične za korisnika za provjeru autentičnosti, obično traženjem korisnika u bazi podataka.

5. Paket usluga
   - UserService.java: Sadrži poslovnu logiku za upravljanje korisnicima. Uključuje metode za dodavanje, ažuriranje, brisanje i popis korisnika.
   - AdminService.java: Može uključivati funkcionalnost specifičnu za administratora ako je odvojena od općih korisničkih usluga.
6. Imenik resursa
   - static: Sadrži statičke elemente (npr. CSS datoteku) za aplikaciju. Za našu aplikaciju ne koristimo statične elemente
   - predlošci: Sadrži datoteke predložaka (npr. Thymeleaf, Freemarker) za prikaz prikaza u web aplikaciji.
   - application.properties: Konfiguracijska datoteka za aplikaciju. Uključuje postavke baze podataka, portove poslužitelja i druga konfigurabilna svojstva.

5.2.1 POM konfiguracijska datoteka

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
https://maven.apache.org/xsd/maven-4.0.0.xsd">
      <modelVersion>4.0.0</modelVersion>
      <parent>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-parent</artifactId>
            <version>3.3.1</version>
            <relativePath/> <!-- lookup parent from repository -->
      </parent>
      <groupId>com.unin</groupId>
      <artifactId>springmaven</artifactId>
      <version>0.0.1-SNAPSHOT</version>
      <name>springmaven</name>
      <description>Projekt za prikaz funkcionalnosti Spring i Maven
tehnologija</description>
```

Na početku naše pom.xml datoteke možemo vidjeti početne postavke, odnosno općenite informacije našeg projekta kao što je grupa ili organizacija kojoj ovaj projekt pripada, jedinstveni identifikator za projekt unutar grupe, verzija projekta, naziv projekta i kratak opis projekta. U

nastavku slijede važniji podaci, a to su ovisnosti. Svaka ovisnost je potrebna kako bi naš program funkcionirao, a to su:

```
<artifactId>spring-boot-starter-data-jpa</artifactId>
```

Pojednostavljuje pristup bazi podataka i ORM (Object-Relational Mapping) s JPA entitetima i spremištima.

```
<artifactId>spring-boot-starter-security</artifactId>
```

Naša glavna ovisnost u projektu. Pruža značajke kao što su kodiranje lozinke, korisničke uloge i sigurnosna konfiguracija za zaštitu vaše aplikacije.

```
<artifactId>mysql-connector-java</artifactId>
```

Omogućuje vašoj aplikaciji povezivanje s MySQL bazom podataka radi izvođenja CRUD operacija.

```
<artifactId>spring-boot-starter-test</artifactId>
```

Ako želimo testirati pravilno funkcioniranje naše aplikacije, ova ovisnost omogućuje alate kao što su JUnit, Mockito i Spring TestContext za pisanje jediničnih i integracijskih testova.

```
<artifactId>jakarta.persistence-api</artifactId>
```

Definira sučelja i komentare koji se koriste u JPA, omogućujući vašim entitetima interakciju s bazom podataka.

5.2.2 Application.properties datoteka

```
spring.application.name=springmaven
spring.datasource.url=jdbc:mysql://localhost:3306/administracija
spring.datasource.username=root
spring.datasource.password=
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver

spring.jpa.hibernate.ddl-auto=update

spring.main.allow-bean-definition-overriding=true
spring.main.web-application-type=none
```

Ovdje možemo vidjeti postavke unutar naše application.properties datoteke, a ona je potrebna za konfiguraciju Spring Boot aplikacije. Ovdje definiramo postavke poput:

- Naziv projekta: springmaven

- Podaci za povezivanje s bazom podataka, odnosno URL (Uniform Resource Locator), korisničko ime i lozinka, no u našem primjeru ne koristimo nikakvu lozinku za pristup bazi podataka. Osim toga, specificiramo upravljački program za bazu.
- Postavke Hibernatea okvira koji omogućuje programerima mapiranje Java objekta na zapise u relacijskoj bazi podataka.
- Vrsta aplikacije: u našem slučaju je to konzolna aplikacija.

Ove postavke omogućuju prilagodbu ponašanja aplikacije bez promjena u kodu.

5.3 Programski kod

```
@Entity
@Table(name = "employees", uniqueConstraints =
@UniqueConstraint(columnNames = "korisnicko_ime"))
public class UserEntity {


    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String ime;
    private String prezime;
    private String korisnickoIme;
    private String lozinka;
    private String rola;
```

UserEntity klasa je JPA (Java Persistence API) entitet koji predstavlja tablicu zaposlenika u bazi podataka. Sadrži polja za korisničke podatke, konstruktore i bilješke za preslikavanje tih polja u odgovarajuće stupce baze podataka, osiguravajući integritet podataka i jedinstvenost za korisnička imena što možemo vidjeti prema @UniqueConstraint parametru. Nakon toga, definirali smo našu klasu UserEntity koja pohranjuje podatke o ID-u korisnika, imenu, prezimenu, korisničkom imenu, lozinki i roli korisnika unutar aplikacije. @Id i @GeneratedValue anotacije služe za identificiranje našeg primarnog ključa u bazi, te način na koji će se on inkrementirati u bazi. Prva varijabla nakon @Id anotacije se uzima kao primarni ključ.

5.3.1 UserRepository.java sučelje

```
@Repository
public interface UserRepository extends JpaRepository<UserEntity, Long> {
    UserEntity findByKorisnickoIme(String korisnickoIme);
}
```

UserRepository sučelje se koristi u svrhu jednostavne i učinkovite interakcije s bazom podataka, pružajući metode za dohvaćanje i manipulaciju podacima korisnika, uz mogućnost definiranja dodatnih metoda za specifične upite. Ono pruža osnovne CRUD (Create, Read, Update, Delete) operacije bez potrebe za pisanjem SQL upita. Ovo smanjuje potrebu za ručnim pisanjem SQL upita i olakšava održavanje koda. Koristi se za pristup podacima i manipulaciju podacima unutar tablice employees, koja je mapirana entitetom UserEntity.

@Repository anotacija u Springu označuje klasu koja služi kao spremnik podacima. Ona omogućuje upravljanje iznimkama i olakšava interakciju s bazom podataka.

JpaRepository posebno je JPA specifično proširenje za Repository. U osnovi, Jpa Repository sadrži osnovne CRUD operacije, operacije za označavanje stranica i sortiranje. U svojim parametrima specificiramo tip entiteta kojim ovo spremište upravlja, te tip primarnog ključa entiteta.

U sučelju je definirana (ali ne i implementirana) i metoda koja omogućuje pronalaženje korisnika prema njegovom korisničkom imenu, a vraća objekt UserEntity koji odgovara traženom korisničkom imenu.

5.3.2 SecurityConfig.java klasa

```
@Configuration
@EnableMethodSecurity(securedEnabled = true)
public class SecurityConfig {

    @Bean
    public PasswordEncoder passwordEncoder() {
        return new BCryptPasswordEncoder();
    }

    @Bean
    public UserDetailsService userDetailsService() {
        return new CustomUserDetailsService();
    }

    @Bean
    public AuthenticationManager authenticationManager() throws
Exception {
        DaoAuthenticationProvider provider = new
DaoAuthenticationProvider();
        provider.setUserDetailsService(userDetailsService());
        provider.setPasswordEncoder(passwordEncoder());
        return new ProviderManager(provider);
    }
}
```

SecurityConfig je konfiguracijska klasa koja se obično koristi u aplikacijama sa Spring Security modulom. Njezina osnovna svrha je definirati kako aplikacija obrađuje sigurnosne aspekte poput autentifikacije i autorizacije. Autentifikacija je provjera pravog identiteta korisnika ili entiteta, dok autorizacija određuje kakvim podacima korisnik može pristupit. Autentifikacija je preduvjet za autorizaciju. U Spring aplikacijama, konfiguriranje sigurnosti je ključno za kontrolu pristupa resursima i zaštitu osjetljivih endpointa.

@Configuration je anotacija na razini klase koja označuje da je objekt izvor definicija beana. Klase @Configuration deklariraju bean-ove kroz @Bean anotaciju.

U Springu, objekti koji čine kostur aplikacije i kojima upravlja Spring spremnik inverzije kontrole nazivaju se Beanovi. Bean je objekt koji je napravljen i upravljan od strane Spring spremnika.

@EnableMethodSecurity omogućuje Spring Security podršku za sigurnost na razini metoda s podrškom za @Secured anotaciju. Ovo omogućuje zaštitu metoda na temelju uloga ili drugih kriterija definiranih u sigurnosnoj konfiguraciji.

U nastavku slijede Beanovi koji su gradivni blokovi Spring Security aplikacije. PasswordEncoder definira bean koji služi za enkripciju lozinki prije pohrane u bazu podataka. UserDetailsService definira sučelje koje Spring Security koristi za učitavanje podataka specifičnih za korisnika tijekom autentifikacije. AuthenticationManager je glavno Spring sučelje za autentifikaciju korisnika. Ove konfiguracije zajedno osiguravaju da bilo koja Spring aplikacija može sigurno autentificirati korisnike i provoditi kontrolu pristupa temeljenu na ulogama ili drugim kriterijima definiranim u aplikaciji.

5.3.3 CustomUserDetailsService.java klasa

```
@Service
public class CustomUserDetailsService implements UserDetailsService
{


    @Autowired
    private UserRepository userRepository;


    @Override
    public UserDetails loadUserByUsername(String username) throws
UsernameNotFoundException {
        UserEntity userEntity =
userRepository.findByKorisnickoIme(username);
        if (userEntity == null) {
            throw new UsernameNotFoundException("User not found");
        }


        return new User(
            userEntity.getKorisnickoIme(),
            userEntity.getLozinka(),
            Collections.singleton(new
SimpleGrantedAuthority(userEntity.getRola()))
        );
    }
}
```

Klasa CustomUserDetailsService implementacija je sučelja UserDetailsService Spring Security-a. Omogućuje dohvaćanje korisničkih detalja iz baze podataka tijekom provjere autentifikacije. Ova je klasa ključna za prilagođene mehanizme provjere autentičnosti korisničkog imena i lozinke u odnosu na pohranjene podatke u bazi podataka.

Spring @Service anotacija koristi se s klasama koje pružaju poslovne funkcije. Spring kontekst će automatski detektirati ove klase kada se koristi konfiguracija temeljena na anotacijama i skeniranje putanje klase.

@Autowired anotacija se koristi za uvođenje ovisnosti. Govori Spring-u da automatski ubaci UserRepository bean, koji je Spring Data JPA sučelje repozitorija za pristup korisničkim podacima u bazi podataka.

@Override anotacija govori Springu da se nadjačava metoda iz UserDetailsService što znači da možemo sami implementirati svoju loadUserByUsername metodu. U nastavku koda učitavamo UserEntity objekt iz baze podataka baziran na korisničkom imenu korisnika i vraćamo novu instancu User klase koju omogućuje Spring Security. Ovdje dobivamo 3 parametra: korisničko ime, lozinku i rolu koja je spremljena u UserEntity objektu.

5.3.4 UserService.java klasa

Klasa UserService koristi se u Spring aplikaciji za upravljanje operacijama povezanim s korisnicima, kao što je dodavanje novih korisnika u bazu podataka. Koristi Spring Security komentare za provođenje sigurnosnih pravila i interakciju s podatkovnim slojem kako bi se korisničke informacije sačuvale na siguran način.

```
@Service
public class UserService {

    @Autowired
    private UserRepository userRepository;

    @Autowired
    private PasswordEncoder passwordEncoder;
```

@Service: Ova napomena označava klasu kao Spring uslugu, a omogućuje Springu da automatski otkrije i upravlja ovom klasom kao Spring beanom, olakšavajući uvođenje ovisnosti gdje je to potrebno.

@Autowired: Ova se anotacija koristi za automatsko uvođenje ovisnosti. Spring ubacuje potrebne bean-ove (UserRepository i PasswordEncoder) u UserService

UserRepository: Spring Data JPA repozitorij za interakciju s korisničkom bazom podataka.

PasswordEncoder: Bean za sigurno kodiranje lozinki, obično koristeći BCryptPasswordEncoder.

Sljedeća anotacija, @PreAuthorize("hasRole('ADMIN')"), je najbitnija uloga Spring Security modula za aplikaciju jer se njome osigurava da samo korisnici s ADMIN ulogom mogu izvršiti metodu koja dodaje korisnike u bazu. Svim ostalim korisnicima, odnosno korisnici koji u bazi podataka imaju rolu „Korisnik“, a ne „Admin“, nije dopušteno korištenje metode za dodavanje novih korisnika u bazu.

Ostale metode u ovoj klasi, odnosno metode kojima ažuriramo i brišemo korisnike iz baze, funkcioniraju na isti način, no metoda za izlistavanje svih korisnika je drugačija i zanimljiva kao metoda koja kontrolira pristup osjetljivim podacima korisnicima bazirano na njihovima rolama u aplikaciji.

```
@PreAuthorize("hasRole('ADMIN') OR hasRole('KORISNIK')")
    public void listAllUsers() {
        List<UserEntity> users = userRepository.findAll();
        System.out.println("Popis svih zaposlenika:");
        for (UserEntity user : users) {
            System.out.printf("Ime: %s\nPrezime: %s\n",
user.getIme(), user.getPrezime());
            if
(SecurityContextHolder.getContext().getAuthentication().getAuthoriti
es().stream()
                    .anyMatch(a ->
a.getAuthority().equals("ROLE_ADMIN"))) {
                System.out.printf("Korisničko ime: %s\n",
user.getKorisnickoIme());
            }
            System.out.printf("Rola: %s%n",
user.getRola().substring(5));
            System.out.println("--------------------------------");
        }
    }
```

Kao što vidimo iz koda, ovu metodu mogu koristiti korisnici s ADMIN rolom, no uz to, korisnici koji imaju rolu postavljenu kao KORISNIK također imaju pristup ovoj metodi. Metoda tad pomoću petlje ispisuje detalje za svakog korisnika spremljenog u bazu podataka, međutim, s jednom bitnom razlikom. Ako je korisnik aplikacije trenutno prijavljen kao admin, uz ime, prezime i rolu korisnika u bazi podataka, on može vidjeti i korisničko ime svakog korisnika. Kad bi korisnik aplikacije prijavljen kao korisnik htio izlistati sve korisnike iz baze, to mu je omogućeno, no ne bi

vidio korisnička imena korisnika. Ovim pristupom se osigurava da samo autorizirani korisnici vide osjetljive podatke.

5.3.5 SpringmavenApplication.java klasa

Posljednja klasa koju koristimo u ovoj aplikaciji je zapravo pokretačka klasa aplikacije. Kroz ovu klasu pozivamo sve metode koje koristimo u aplikaciji i radi se inicijalizacija svih potrebnih komponenti za rad aplikacije.

```
@SpringBootApplication
@ComponentScan("com.unin.springmaven")
public class SpringmavenApplication implements CommandLineRunner {

    @Autowired
    private UserService userService;

    @Autowired
    private AuthenticationManager authenticationManager;

    @Autowired
    private UserRepository userRepository;

    private Authentication authentication;

    public static void main(String[] args) {
        SpringApplication.run(SpringmavenApplication.class, args);
    }
```

@SpringBootApplication kombinira tri anotacije:

- @Configuration: Oznaka da je klasa izvor konfiguracije za Spring.
- @EnableAutoConfiguration: Omogućava automatsku konfiguraciju Spring Boot-a na temelju dodanih biblioteka.
- @ComponentScan: Pretražuje pakete za Spring Beans, omogućavajući automatsku detekciju komponenti.

@ComponentScan("com.unin.springmaven") specifično skenira pakete unutar "com.unin.springmaven" kako bi pronašao i registrirao sve Spring Beanove.

Sve @Autowired anotacije ovdje služe za injektiranje potrebnih beanova u našu aplikaciju.

```
private void authenticateUser(String username, String password) {

        this.authentication = authenticationManager.authenticate(new
UsernamePasswordAuthenticationToken(username, password));

SecurityContextHolder.getContext().setAuthentication(authentication)
;
    }
```

AuthenticateUser metoda je temelj za osiguranje da samo ovlašteni korisnici mogu pristupiti određenim dijelovima aplikacije, čime se štite podaci i aplikacijska logika. Ovdje se radi autentifikacija korisnika gdje se provjeravaju uneseni podaci i postavlja se sigurnosni kontekst, odnosno omogućuje se drugim dijelovima aplikacije pristup informacijama o trenutno prijavljenom korisniku.

```
try {
      userService.addUser(scanner);
   } catch (AccessDeniedException e) {
      System.out.println("Nemate dozvolu za dodavanje novog
korisnika.");
}
break;
```

Svaka metoda koja se poziva okružena je try-catch (pokušaj-uhvati) blokom. Try-catch blok pokuša izvršiti metode koje se pozivaju od strane korisnika, no, ako korisnik nema potrebne dozvole za korištenje određene metode, poziva se AccessDeniedException izuzetak koji će informirati korisnika da nema potrebne ovlasti za tu metodu.

5.4 Prikaz aplikacije

Prije korištenja aplikacije, korisnici se moraju prijaviti sa svojim korisničkim podacima.

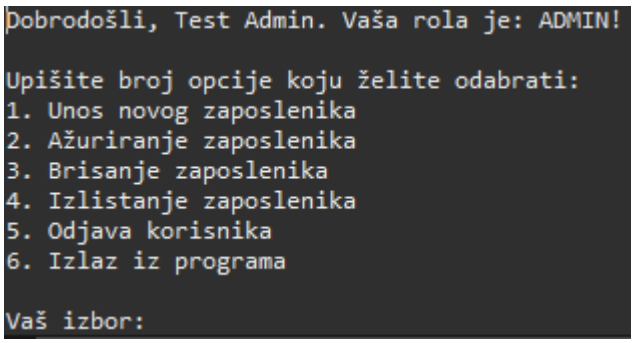
```
Prije korištenja ovog programa potrebna je autentifikacija korisnika.
Molimo Vas, upišite svoje korisničke podatke.
Vaše korisničko ime:
```

Slika 3: Prijava korisnika

Aplikacija u tom trenutku, iz baze podataka, pridodaje korisniku njegovu rolu u aplikaciji i pušta daljnje korištenje aplikacije. Korisnik ima šest izbora:

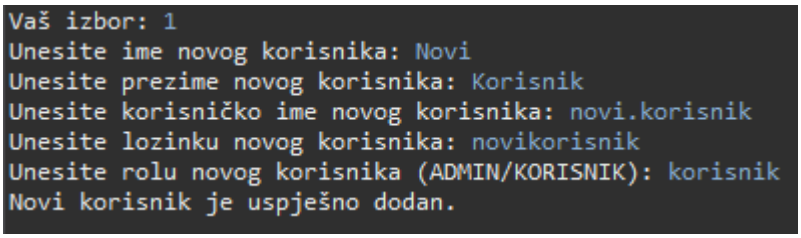
1. Unos novog zaposlenika
2. Ažuriranje zaposlenika
3. Brisanje zaposlenika
4. Izlistanje zaposlenika

5. Odjava korisnika
6. Izlaz iz programa



Slika 4: Glavni izbornik aplikacije

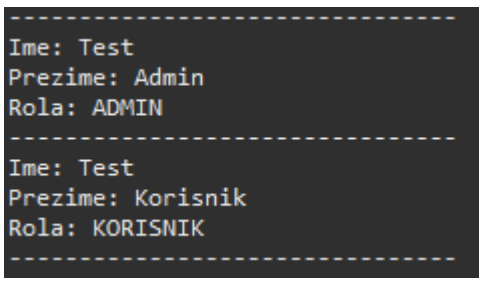
Slika 5 prikazuje odabir prve opcije, odnosno dodavanja novog korisnika u bazu podataka. Ovdje se također događa i provjera korisničkog imena unesenog korisnika, a ono mora biti unikatno za svakog korisnika. Također se provjerava upisana rola korisnika, gdje korisnik aplikacije točno mora upisati admin ili korisnik za dodavanje role korisnika u bazi podataka.



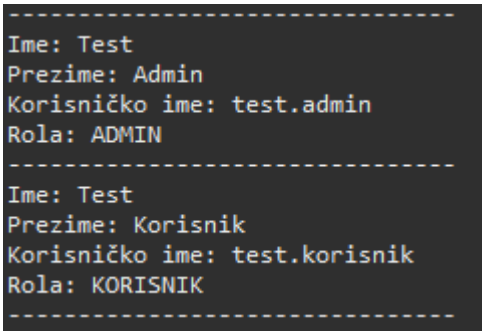
Slika 5: Unos novog korisnika

Ako se korisnik odluči odjaviti iz aplikacije, potrebno se ponovno prijaviti kako bi se mogla koristiti dalje, a ako se izabere šesta opcija, aplikacija se zatvara.

Još jedna bitna razlika između rola korisnika, odnosno kako se preko Spring Security modula može kontrolirati pristup osjetljivim podacima, može se vidjeti na slici 6 i 7.



Slika 6: Ispis zaposlenika za korisnika



Slika 7: Ispis zaposlenika za admina

Ovdje možemo vidjeti kako podaci ispisani za korisnika kojem Spring stavlja rolu „korisnik“ i korisnika kojem Spring stavlja rolu „admin“ nisu jednaki. Admin, kroz metodu ispisa zaposlenika, dobije još jedan podatak za svakog korisnika, a to je korisničko ime. Preko tog korisničkog imena, mogu se izmijeniti podaci u bazi ili čak izbrisati potpuno zaposlenika iz baze.

6. ZAKLJUČAK

Java, kao jedan od najpopularnijih programskih jezika, pruža stabilnu i sigurnu osnovu za razvoj aplikacija. Značajke filozofije Jave, kao što je platformska nezavisnost, zajedno s velikim ekosustavom biblioteka i okvira, omogućava programerima gradnju aplikacija koje su skalabilne i lako održive.

Također, Java ima objektno orijentirani pristup gdje se programiranje bazira na objektima. Spring radni okvir ovdje donosi drugi pristup, a to je aspektno orijentirano programiranje bazirano na aspektima. Uz to, ključna arhitektura Springa, inverzija kontrole i injekcija ovisnosti, donosi odvajanje odgovornosti i fleksibilnost, odnosno mijenja se tradicionalni tijek kontrole u aplikaciji. Umjesto da objekti stvaraju druge objekte, kontejner inverzije kontrole upravlja stvaranjem i životnim ciklusom objekata. Injekcija ovisnosti tad omogućuje injektiranje potrebnih ovisnosti u objekte, čime se omogućuje labavo povezivanje među komponentama i omogućuje lakše testiranje i održavanje aplikacija.

Kombinacija Spring i Maven tehnologija, uz Java programski jezik, pokazala se kao moćan alat u razvoju sigurnih i skalabilnih aplikacija. Spring nam ovdje služi kao sofisticirana kontrola pristupa podacima koji je omogućen Spring Security modulom, dok nam Maven ovdje koristi za efikasno upravljanje projektima i ovisnostima koje takvi projekti zahtijevaju.

Ovakve tehnologije omogućavaju programerima da se fokusiraju na poslovnu logiku i korisničke zahtjeve, dok se automatizirani alati brinu za sigurnost, integraciju i stabilnost aplikacija.

Literatura

[1] – Jezik Java: https://www.frgconsulting.com/insights/why-is-java-so-popular-developers (Pristupano 15.06.2024.)

[2] – Povijest Java programskog jezika: https://u-next.com/blogs/java/history-of-java/ (Pristupano 15.06.2024.)

[3] – Povijest Java programskog jezika: https://en.wikipedia.org/wiki/Java_(programming_language) (Pristupano 15.06.2024.)

[4] – Koncepti u Javi: https://www.geeksforgeeks.org/object-oriented-programming-oops-concept-in-java/ (Pristupano 16.06.2024.)

[5] – Preopterećenje metoda: https://www.geeksforgeeks.org/method-overloading-in-java/ (Pristupano 25.06.2024.)

[6] – Nadjačavanje metoda: https://www.geeksforgeeks.org/overriding-in-java/ (Pristupano 25.06.2024.

[7] – Spring radni okvir: https://en.wikipedia.org/wiki/Spring_Framework (Pristupano 17.06.2024.)

[8] - Iuliana Cosmina, Rob Harrop, Chris Schaefer, Clarence Ho – Pro Spring 5, Fifth edition - Stranica 1-1.

[9] – Spring Framework: https://www.geeksforgeeks.org/introduction-to-spring-framework/ (Pristupano 17.06.2024.)

[10] – Općenito o Springu: https://docs.spring.io/spring-framework/docs/4.0.x/spring-framework-reference/html/overview.html (Pristupano 17.06.2024.)

[11] – Inverzija kontrole https://www.tutorialsteacher.com/ioc/inversion-of-control (Pristupano 17.06.2024.)

[12] -Injekcija ovisnosti: https://www.baeldung.com/inversion-control-and-dependency-injection-in-spring (Pristupano 18.06.2024.)

[13] – Spring injekcija ovisnosti: https://docs.spring.io/spring-framework/docs/4.2.x/spring-framework-reference/html/beans.html (Pristupano 18.06.2024.)

[14] – Spring kontejner inverzije kontrole: https://docs.spring.io/spring-framework/reference/core/beans/basics.html (Pristupano 18.06.2024.)

[15] – Spring kontejner za inverziju kontrole: https://docs.spring.io/spring-framework/reference/core/beans/introduction.html (Pristupano 18.06.2024.)

[16] -Spring beans: https://www.geeksforgeeks.org/spring-beanfactory/ (Pristupano 19.06.2024.)

[17] – Aspektno orijentirano programiranje: https://www.geeksforgeeks.org/aspect-oriented-programming-and-aop-in-spring-framework/ (Pristupano 19.06.2024.)

[18] – Aspektno orijentirano programiranje: https://docs.spring.io/spring-framework/reference/core/aop/introduction-defn.html (Pristupano 20.06.2024.)

[19] – Spring Security: https://www.geeksforgeeks.org/introduction-to-spring-security-and-its-features/ (Pristupano 21.06.2024.)

[20] - Spring Security na razini metoda: https://www.geeksforgeeks.org/spring-security-at-method-level/ (Pristupano 21.06.2024.)

[21] – Maven: https://www.simplilearn.com/tutorials/maven-tutorial/what-is-maven (Pristupano 22.06.2024.)

[22] – Maven: https://www.tutorialspoint.com/maven/maven_overview.htm (Pristupano 22.06.2024.)

[23] – Maven: https://maven.apache.org/what-is-maven.html (Pristupano 22.06.2024.)

[24] – Maven POM: https://www.tutorialspoint.com/maven/maven_pom.htm (Pristupano 23.06.2024.)

[25] – Maven Lifecycle: https://maven.apache.org/guides/introduction/introduction-to-the-lifecycle.html (Pristupano 24.06.2024.)

[26] – Maven dodaci: https://maven.apache.org/guides/introduction/introduction-to-plugins.html (Pristupano 25.06.2024.)

[27] – Maven ovisnosti: https://maven.apache.org/guides/introduction/introduction-to-dependency-mechanism.html (Pristupano 25.06.2024.)

[28] – Cikličke ovisnosti: https://medium.com/@louismrc/fix-your-circular-dependencies-with-dependency-inversion-e22b6f4c9510 (Pristupano 25.06.2024.)

[29] – Spring Initializr: https://start.spring.io/ (Pristupano 14.06.2024.)

Popis slika:

Slika 6: Napravio autor uz pomoć Snipping Tool alata u Eclipse integriranom okruženju

Slika 7: Napravio autor uz pomoć Snipping Tool alata u Eclipse integriranom okruženju

Sveučilište
Sjever

SVEUČILIŠTE
SJEVER

IZJAVA O AUTORSTVU
I
SUGLASNOST ZA JAVNU OBJAVU

Završni/diplomski rad isključivo je autorsko djelo studenta koji je isti izradio te student odgovara za istinitost, izvornost i ispravnost teksta rada. U radu se ne smiju koristiti dijelovi tuđih radova (knjiga, članaka, doktorskih disertacija, magistarskih radova, izvora s interneta, i drugih izvora) bez navođenja izvora i autora navedenih radova. Svi dijelovi tuđih radova moraju biti pravilno navedeni i citirani. Dijelovi tuđih radova koji nisu pravilno citirani, smatraju se plagijatom, odnosno nezakonitim prisvajanjem tuđeg znanstvenog ili stručnoga rada. Sukladno navedenom studenti su dužni potpisati izjavu o autorstvu rada.

Ja, LUKA KUDUMIJA *(ime i prezime)* pod punom moralnom, materijalnom i kaznenom odgovornošću, izjavljujem da sam isključivi autor/ica završnog/diplomskog *(obrisati nepotrebno)* rada pod naslovom UPOTREBA SPRING I MAVEN TEHNOLOGIJA U RAZVOJU JAVA APLIKACIJA ZA KONTROLU PRISTUPA PODACIMA *(upisati naslov)* te da u navedenom radu nisu na nedozvoljeni način (bez pravilnog citiranja) korišteni dijelovi tuđih radova.

Student/ica:
(upisati ime i prezime)

(vlastoručni potpis)

Sukladno Zakonu o znanstvenoj djelatnost i visokom obrazovanju završne/diplomske radove sveučilišta su dužna trajno objaviti na javnoj internetskoj bazi sveučilišne knjižnice u sastavu sveučilišta te kopirati u javnu internetsku bazu završnih/diplomskih radova Nacionalne i sveučilišne knjižnice. Završni radovi istovrsnih umjetničkih studija koji se realiziraju kroz umjetnička ostvarenja objavljuju se na odgovarajući način.

Ja, LUKA KUDUMIJA *(ime i prezime)* neopozivo izjavljujem da sam suglasan/na s javnom objavom završnog/diplomskog *(obrisati nepotrebno)* rada pod naslovom UPOTREBA SPRING I MAVEN TEHNOLOGIJA U RAZVOJU JAVA APLIKACIJA ZA KONTROLU PRISTUPA PODACIMA *(upisati naslov)* čiji sam autor/ica.

Student/ica:
(upisati ime i prezime)

(vlastoručni potpis)