Primjena uzoraka za sigurnost u razvoju web aplikacija

Slamić, Iva

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Iva Slamić

PRIMJENA UZORAKA ZA SIGURNOST U RAZVOJU WEB APLIKACIJA

DIPLOMSKI RAD

Varaždin, 2019.

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Iva Slamić

Matični broj:

Studij: ***Informacijsko i programsko inženjerstvo***

PRIMJENA UZORAKA ZA SIGURNOST U RAZVOJU WEB APLIKACIJA

DIPLOMSKI RAD

Mentor:

Prof. dr. sc. Dragutin Kermek

Varaždin, rujan 2019.

Iva Slamić

Izjava o izvornosti

Izjavljujem da je moj diplomski rad izvorni rezultat mojeg rada te da se u izradi istoga nisam koristio drugim izvorima osim onima koji su u njemu navedeni. Za izradu rada su korištene etički prikladne i prihvatljive metode i tehnike rada.

Autorica potvrdila prihvaćanjem odredbi u sustavu FOI-radovi

_____

Sažetak

U radu će se definirati pojam uzoraka i njihove osobine, a poseban naglasak će biti stavljen na uzorke za sigurnost u razvoju web aplikacija. Uzorci za sigurnost biti će kategorizirani prema nekoliko kategorija: uzorci za upravljanje identitetom, za autentikaciju, za kontrolu pristupa, uzorci za sigurnost mreže, za sigurnost web servisa, za kriptografiju web servisa, za sigurni aplikacijski sloj. Nadalje, biti će navedeni i uzorci za sigurnost za Java EE aplikacjie, web servise, upravljanje identitetom i pružanje usluga koje je razvio Sun Java Center, a dijele se u četiri kategorije odnosno četiri razine: razina weba, poslovna razina, razina web servisa i razina identiteta. Uz primjenu odabranih uzoraka za sigurnost, biti će prikazani dokumentacija i razvoj web aplikacije.

Ključne riječi: uzorci, uzorci za sigurnost, web aplikacija, kategorizacija, razvoj

Sadržaj

1. Uvod

Mnoge informacije danas imaju veliku vrijednost i ljudi često pokušavaju doći do njih kako bi ih iskoristili u različite svrhe poput financijske dobiti, širenja lažnih informacija i sličnog. Stoga nedostatak sigurnosti i određene kontrole može uzrokovati veliki poslovni gubitak firmama koje se oslanjaju na nedovoljno zaštićene sustave. Napadi na informacijske sustave se događaju iz raznih razloga dok su možda najčešći od njih popularnost neke web aplikacije zbog različitih motiva ili pak napadi koji ne dolaze izvana nego iznutra od strane zaposlenika neke organizacije koji rade sami ili pomažu nekome izvana. Bitno je procijeniti rizik koji se može dogoditi te osigurati imovinu, identificirati kontrolne točke i sve njihove slabosti pa odrediti one najisplativije i najefektnije. Svi digitalni uređaji, uz kompjutere, su izloženi napadima. Stoga se firme moraju osigurati da s povjerenjem mogu koristiti što im treba. Sigurnost tako zahtjeva tehničke, upravljačke i fizičke mjere od kojih će se u ovom radu obraditi one tehničke.

Sigurnost je sposobnost sustava da se spriječe razni napadi preko kojih bi se moglo doći do vrijednih informacija. Zato aplikacije moraju biti razvijane tako da se podaci zaštite od napada, nedozvoljenih pristupa i sličnog. Što se tiče primjene uzoraka sigurnosti njihov cilj i je poboljšanje rada i razvoja aplikacija u području sigurnosti.

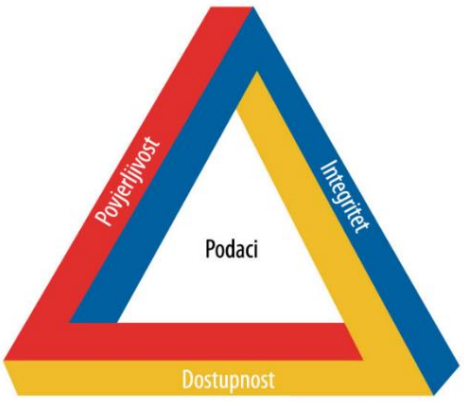
U radu će biti razrađeni uzorci sigurnosti prema 10 kategorija iz knjige *Security Patterns in Practice: Designing Secure Architectures Using Software Patterns*. Zatim navedeni i objašnjeni uzorci koji služe za implementaciju Java EE aplikacija, Web servisa, identiteta i usluga te će se kroz primjer izrade web aplikacije prikazati implementacija nekih od njih.

2. Sigurnost općenito

Potreba za sigurnim informacijskim sustavima dolazi iz želje da se očuva integritet osjetljivih podataka koje oni sadrže, a čiji broj sve više raste. Stoga je ljudima koji ih održavaju bitno zaštiti one podatke zbog kojih bi moglo doći do ozbiljnih posljedica poslovanja ako netko tko ima lošu namjeru dođe do njih. Sigurnost se odnosi na procese i metodologije kojima se nastoji informacije održati povjerljivima, dostupnima i kojima se nastoji osigurati njihov integritet (Technopedia, 2019). Njihovim korištenjem olakšavaju se otkrivanje i sankcije kršenja sigurnosnog protokola, kao i dokumentiranje takvih događaja.

Nadalje, sigurnost se odnosi i na kontrolu pristupa čime se osigurava da samo autorizirane osobe imaju pristup podacima ili sustavu općenito. Svakako je bitno zaštiti informacije i tijekom prijenosa, kao npr. slanje osjetljivih informacija putem e-pošte. Ovisno o kompleksnosti informacijskog sustava odnosno o sadržaju informacija koje posjeduje implementiraju se sigurnosne mjere za održavanje sigurnosti, što znači veći sustav veće sigurnosne mjere.

Prema autorima članka (Pressbook, n.d.), Daveu i Davidu T. Bourgeois, na web stranici Pressbook, kada se govori o sigurnosti podataka u sustavima bitno je naglasiti kako postoje tri glavne komponente sigurnosti odnosno sigurnosno trojstvo podataka, a to su: povjerljivost, integritet i dostupnost.



Slika 1 Tri komponente sigurnosti, Izvor: (Blog@Croz, 2014)

Povjerljivost označava ograničenje dostupnosti informacija svim osobama koje nemaju pravo znati što se u nekom sustavu ili određenom dijelu sustava nalazi. Jednostavan primjer povjerljivosti bio bi ograničenje dostupnosti specifičnih dokumenata koji se nalaze na sustavu, a do kojih mogu doći samo ovlaštene osobe odnosno one kojima je dano pravo pristupa tom dijelu sustava. Integritet se odnosi na konzistentnost i točnost podataka kojima se pristupa, znači da je informacija točno onakva kakva treba biti i da nije mijenjana. Do mijenjanja informacija može doći na razne načine kao npr. ljudskom greškom, zlonamjernim softverima, cyber napadima, kompromitiranim hardverom i slično. Zadnja komponenta, dostupnost, znači da informacije moraju biti na raspolaganju kad god ih netko zatreba. Sustavi bi trebali biti

dostupni cijelo vrijeme čak i ako je u pitanju nešto što može utjecati na zastoj poslovanja i sličnog kao npr. ažuriranje softvera ili nestanak struje pa čak i kvar hardvera.

Kako bi se osigurala funkcija navedenih komponenata potrebno je implementirati određene procese. Jedan od tih procesa je autentikacija koja se može implementirati raznim načinima kako bi se osiguralo da je osoba koja pristupa nekoj informaciji točno ona za koju se predstavlja. Danas se identitet osobe može odrediti na puno načina, ali je najčešći onaj gdje se osoba prijavljuje putem ID-a i lozinke koju samo ona zna. No, taj način je često vrlo lako probiti i podaci mogu postati kompromitirani stoga je sigurniji način određivanja identiteta npr. više-faktorska autentikacija.

Nakon što se korisniku odredi identitet dolazi na red sljedeći proces odnosno kontrola pristupa. Ovim procesom se osigurava povjerljivost podataka jer se određuje koji korisnik može čitati, ažurirati, dodavati i brisati određene podatke. Još jedan važan proces putem kojeg se čuva integritet podataka je enkripcija. Kao što je već prethodno spomenuto, ponekad se informacije šalju e-poštom odnosno internetom ili se mogu prenositi pomoću vanjskog medija. Kada su te informacije povjerljive bitno ih je zaštiti, zato postoji enkripcija. Služi kako bi se podaci kodirali i tako bili teže čitljivi. Osobe koje su u komunikaciji imaju specifične ključeve preko kojih mogu pristupiti tim informacijama.

Govoreći o napadima na web aplikacije posebno se često spominju sljedeća četiri: SQL Injection, XSS (Cross-Site Scripting), DDoS (Distributed Denial of Service Attacks) i CSRF (Cross Site Request Forgery). (Brune, n.d.) Kada netko vrši napad s SQL Injection cilj mu je dobiti pristup podacima u bazi podataka kako bi mogli mijenjati ili čak uništiti te podatke. Tako na primjer osoba koja vrši napad može doći do lozinki ili brojeva kreditnih kartica. Izvodi se na način da se u polja u obrascu unese zlonamjeran SQL kod kojim se mogu prikupiti informacije. Najbolji način zaštite od ovakvog napada je korištenje pripremljenih izjava (eng. prepared statements) kojima se SQL kod unaprijed definira. XSS je napad kojim se šalje zlonamjeran kod tako da se ubaci u aplikaciju koja se izvršava na klijentskoj strani. Zbog toga se računala mogu zaraziti različitim virusima, osjetljive informacije mogu biti ukradene i slično. Najbolje zaštita protiv ovakve vrste napada je imati programe koji mogu detektirati preuzete viruse i poduzeti preventivne mjere. Nadalje, postoje napadi koji usporavaju ili onemogućuju rad web stranica, a poznati su pod nazivom DDoS. Generiranjem zahtjeva s velikim brojem IP adresa nastoji se povećati promet na stranici toliko da se poslužitelju onemogući odgovaranje na zahtjeve. Četvrti i zadnji spomenuti napad je CSRF koji se događa kada je korisnik prevaren klikom na link ili skidanjem nekog kompromitiranog sadržaja kojim se pokreće neželjena akcija. Na taj način se ukrade korisnička sesija te se u ime ukradenog identiteta možete predstavljati kao kupac i obaviti kupnju. Iako postoje najbolje prakse koje su danas uvelike korištene, uvijek postoji određenja razina ranjivosti i izloženost napadima.

3. Uzorci

Općenito, uzorci predstavljaju opis ili predložak za rješenje uobičajenog problema u implementaciji softvera. Njihovo korištenje može spriječiti da manji problemi ne uzrokuju veće, te olakšava čitljivost koda svima koji su upoznati s uzorcima.

U ovom poglavlju biti će ukratko opisani uzorci dizajna, te uzorci sigurnosti koji će se kasnije detaljnije razraditi kroz ostala poglavlja u ovom radu.

3.1 Uzorci dizajna

Uzorci koji se primjenjuju u objektno-orijentiranom programiranju, poznati kao GoF (Gang of Four) uzorci. Detaljno su opisani u knjizi *Design Patterns: Elements of Reusable Object-Oriented Software*, a autori knjige su četvorka: Erich Gamma, Richard Helm, Ralph Johnson i John Vlissides. Knjiga opisuje 23 uzorka dizajna podijeljena u kategorije, te razne tehnike razvoja i moguće „zamke“. Uzorci dizajna podijeljeni su u tri kategorije: uzorci kreiranja, uzorci strukture i uzorci ponašanja. (Erich Gamma, 1994)

Uzorci kreiranja pružaju razne načine na koje se mogu instancirati jedan objekt ili grupa povezanih objekata. Čine proces kreiranja jednostavnijim i dinamičnijim. Postoji pet uzoraka koji pripadaju toj kategoriji: Builder, Factory Method, Abstract Factory, Prototype i Singleton.

Uzorci strukture pružaju načine da se definiraju veze između klasa ili objekata. Olakšavaju refaktoriranje na način da se ne mora mijenjati cijela struktura aplikacije nego samo jedan dio kojem je potrebna promjena. U ovu kategoriju ubraja se sedam uzoraka, a to su: Adapter, Bridge, Composite, Decorator, Facade, Flyweight i Proxy.

Uzorci ponašanja daju definiciju o tome kako bi klase i objekti trebali komunicirati te na taj način pruža fleksibilnost. Sljedeći uzorci pripadaju ovoj kategoriji: Chain of Responsibility, Command, Interpreter, Iterator, Mediator, Memento, Observer, State, Strategy, Template Method, Visitor.

Koristeći ove uzorke bitno je razumjeti koncept koji oni opisuju te ih na taj način i primijeniti, a ne učiti na pamet na koji način i kada se koriste. Svakako treba paziti i na to da se za određeni problem koristi točno uzorak koji je namijenjen za to jer, u suprotnom, koristeći krivi uzorak može nastati puno kompleksniji kod, koji je teži za održavanje, nego što bi zapravo trebao biti.

3.2 Uzorci sigurnosti

Kako autor knjige *Security Patterns in Practice: Designing Secure Architectures Using Software Patterns* navodi, uzorci za sigurnost pomažu razvojnim inženjerima implementirati sigurnosne mjere u aplikaciju iako nemaju dovoljno iskustva u području sigurnosti (Fernandez-Buglioni, 2013). Mogu se također koristiti za evaluaciju postojećih sustava kako bi se uvidjelo

sadrže li oni već potrebne uzorke. Da bi se uopće mogao razviti siguran sustav potrebno je identificirati sve prijetnje koje bi mu mogle naštetiti, a mogu biti namijenjeni u različite svrhe poput financijskih aplikacija, pravnih, medicinskih i mnogih drugih.

Prvi koji su predstavili ovakav pristup u razvoju informacijskih sustava bili su Joseph Yoder i Jeffrey Barcalow 1998. godine. Format je bio napravljen prema spomenutim uzorcima dizajna GoF-a. Yoder i Barcalow su u svom znanstvenom radu predstavili sljedeće uzorke (Romanosky, 2001):

Tablica 1 Početni uzorci Yodera i Barcalowa, Vlastita izrada prema (Romanosky, 2001)

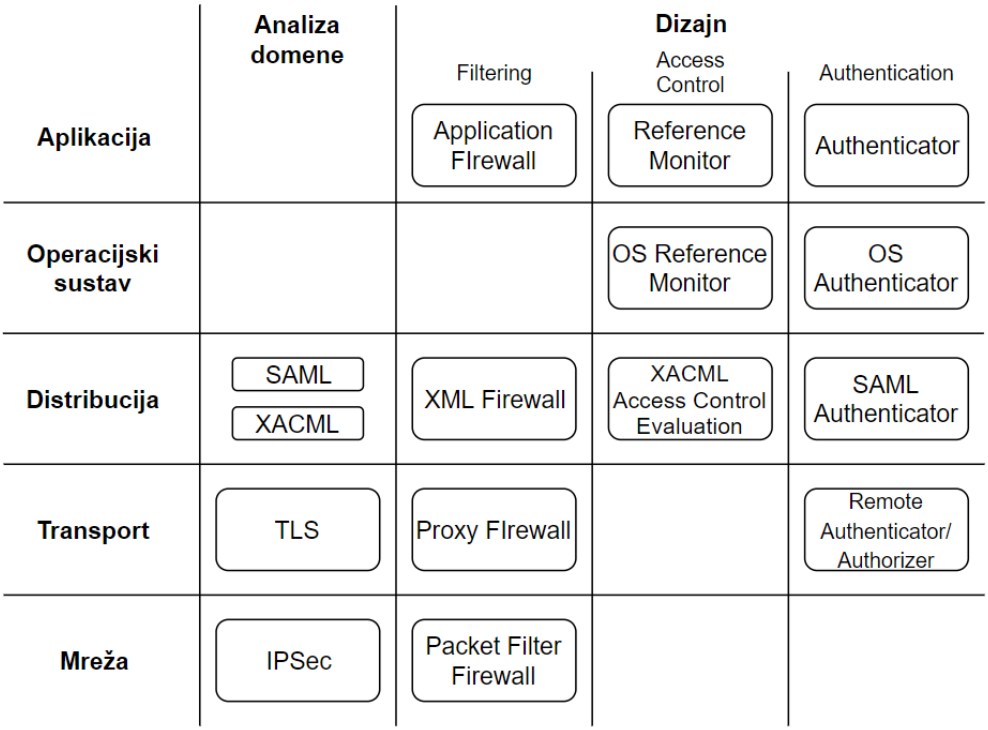
Patent	Svrha
Single Acces Point	pružanje sigurnosnog modula i načina kako se prijaviti u neki sustav
Check Point	organizacija sigurnosnih provjera i njihovih posljedica
Roles	podjela korisnika prema sličnim sigurnosnim privilegijama
Session	lokalizacija globalnih informacija u okruženjima s više korisnika
Full View with Errors	pružanje uvida u korisnike i prikaz potrebnih iznimki
Limited View	dopustiti korisniku da vide samo ono što je namijenjeno njima
Secure Access Layer	integriranje sigurnosti aplikacije s najnižom razinom sigurnosti

U radu je svaki od tih uzoraka detaljno opisan. Tako je za svaki naveden njegov pseudonim pod kojim je poznat i objašnjena je motivacija kao razlog zašto je uzorak zamišljen baš takav kakav je. Nadalje, naveden je problem kojemu je svaki uzorak namijenjen i njegove prednosti koje govore o tome zašto ga je dobro koristiti i po čemu je bitan. Za svaki postoji i rješenje u kojem je objašnjeno kako i što točno napraviti da bi se primijenio te je dan primjer i posljedice koje dolaze s njegovom primjenom od kojih su neke dobre, a neke loše. Također su objašnjene veze između tih uzoraka jer korištenje jednog se veže na korištenje drugog. Kao primjer se može uzeti uzorak *Roles* gdje informacije o korisniku spremamo u objekt uzorka *Session* kojem možemo pristupiti u bilo kojem trenutku kako bi došli do njih kada nam zatrebaju. Na kraju, daju se primjeri korištenja tih uzoraka bilo to kod nekih servera, baza ili aplikacija vezanih uz programske jezike (npr. Java apleti) ili bilo koje druge poznatije aplikacije (Joseph Yoder, 1998).

U knjizi *Security Patterns in Practice* (Fernandez-Buglioni, 2013) govori se o prirodi uzoraka sigurnosti odnosno opisano je nekoliko načina na koji se oni mogu gledati. Tako je prvi po redu opisan uzorak kao uzorak arhitekture. Mogu se promatrati tako jer često opisuju arhitekturalne koncepte softvera te se kao primjer navodi autentikacija između dvije podijeljene jedinice. Neki uzorke sigurnosti promatraju kao uzorke dizajna zbog činjenice što se sigurnost može promotriti kao aspekt softverskog podsustava. Također, mogu se gledati i sa strane

analize odnosno kao uzorci analize gdje se govori o tome da bi se sigurnosna ograničenja trebala definirati na vrlo visokoj razini. Na primjer, definiranje uloga za svakog korisnika i njihovih prava dok se koriste aplikacijom. Ovaj način gledanja autor koristi u svojoj metodologiji kako bi definirao zahtjeve koji su neovisni o sustavu. Zadnji pogled na uzorke sigurnosti je da su oni jedan specijalni tip uzoraka koji je koristan za istraživače i stručnjake sigurnosti, a ne toliko za razvojne inženjere.

Slika 2 prikazuje uzorke podijeljene u dva životna ciklusa: Analiza domene i Dizajn, te su raspoređeni prema različitim razinama arhitekture: Aplikacija, Operacijski sustav, Distribucija, Transport i Mreža. Važno je napomenuti kako na slici 2 nisu prikazani svi uzorci koji će biti opisani u sljedećem poglavlju, a većina ovih se odnosi na servisno orijentiranu arhitekturu (SOA) i samo neki na ostale domene.



Slika 2 Matrica uzoraka iz knjige Security patterns in practice,
Vlastita izrada prema (Fernandez-Buglioni, 2013)

Potrebno je dobro proučiti i razumjeti njihovu ulogu u sustavu i kako su međusobno povezani u čemu pomažu njihovi dijagrami, ali i ostali i alati i metode za njihovu primjenu. Većina uzoraka koji će biti opisanu u sljedećem poglavlju grupiranu su prema razini arhitekture u kojoj se koriste.

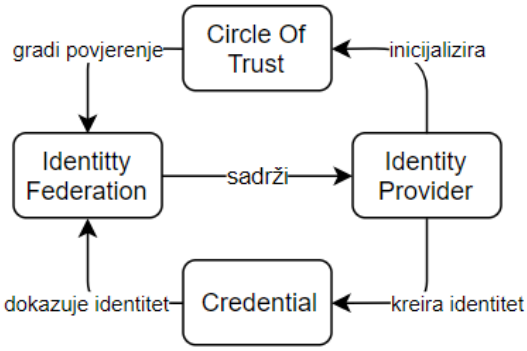
4. Kategorizacija uzoraka za sigurnost

U ovom poglavlju uzorci će biti opisani prema 10 kategorija koji se nalaze u knjizi *Security Patterns in Practice* (Fernandez-Buglioni, 2013). Svakoj kategoriji pripada određeni broj uzoraka za koje su opisani problemi koji oni rješavaju i način na koji ih rješavaju. Za svaki uzorak je dan kontekst u kojem se koristi, te nekakav problem i njegovo rješenje.

Iako će u svim kategorijama uzorci biti opisani, samo neki od njih mogu biti primijeniti u razvoju web aplikacije koja će biti opisana u zadnjem dijelu ovog rada iz razloga što nisu svi namijenjeni razvoju iste.

4.1 Upravljanje identitetom

Danas jedna osoba, s istim identitetom, aplikacijama može pristupiti s različitih uređaja što znači da su podložnije krađi, prisluškivanju i sličnom. Svaka odluka koja je donošena o odobravanju pristupa se mora temeljiti na pravilima o kontroli pristupa neke usluge i stupnju pouzdanosti korisnika. Ovi uzorci olakšavaju implementaciju za provjeru pouzdanosti (Fernandez-Buglioni, 2013).



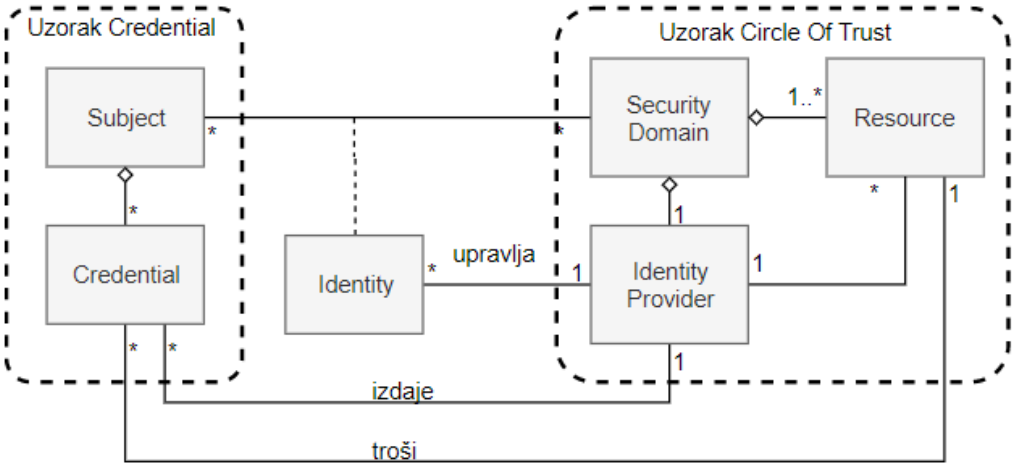
Slika 3 Odnos između uzoraka za Upravljanje identitetom. Vlastita izrada prema (Fernandez-Buglioni, 2013)

Circle of Trust

Uzorak kojim se daje mogućnost stvaranja povjerenja između pružatelja usluga da bi se njihovim subjektima dao pristup integriranom i sigurnijem okruženju. Može se još reći kako je to skup pružatelja usluga koji međusobno imaju poslovne i operativne odnose i vjeruju jedni drugima. Stavlja se u kontekst kod pružatelja usluga s velikim sustavima kao npr. internet. Postoje operativni dogovori, koji uključuju informacije o tome da li mogu davati informacije svojih subjekata, i poslovni koji opisuju način ponude drugih proizvoda vlastitim subjektima. Svaki pružatelj usluge ima tri važne komponente, a to su ime pružatelja usluge, operativni dogovor i poslovni dogovor. Implementacija podrazumijeva da operativni dogovori uključuju akcije poput dijeljenja tajnog ključa ili distribuciju certifikata, te se pružatelji moraju dogovoriti kako podijeliti akreditive da bi se međusobno mogli prepoznati i vjerovati jedni drugima.

Identity Provider

Uzorak koji omogućuje centralizaciju administracije za sigurnost informacija o identitetima. Primjenjuje se kod različitih resursa, kao npr. web servisa i drugih aplikacija, do kojih pristup imaju unaprijed određeni korisnici. Problem je u tome što svaka aplikacija može implementirati svoje rješenje za upravljanje informacijama zbog čega dolazi do nedosljednosti u organizacijskim jedinicama. Stoga je dio organizacije za upravljanje centraliziran u Identity Provideru koji ima zadatak spremati i pružati te informacije.



Slika 4 UML dijagram klasa uzorka Identity Provider. Vlastita izrada prema (Fernandez-Buglioni, 2013)

Ovaj uzorak kombinira prethodni, Circle of Trust, na način da sigurnosna domena koristi više resursa i prosljeđuje pružatelje usluga pružateljima identiteta. Dok, s druge strane, uzorak Credential (koji će biti opisan kasnije) pruža mogućnost identificiranja subjekta preko akreditiva koje su dobili od pružatelja identiteta. Obzirom da kombinira ova dva uzorka dan je pregled UML dijagrama klasa na slici 4.

Identity Federation

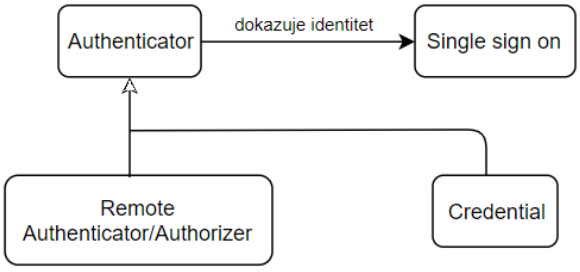
Omogućuje stvaranje dinamički kreiranog identiteta unutar federacije identiteta koja sadrži nekoliko pružatelja usluga kako bi se informacije korisnika transparentno prenosile između njih. Takva struktura je potrebna kako bi se centralizirani identitet mogao koristiti na više mjesta. Ovdje se kao primjer opet mogu navesti web servisi i ostale aplikacije u kojima administraciju sigurnosti vrši unikatni entitet gdje se i spremaju informacije o korisnicima. Koristi se zbog višestrukih registracija i autorizacija kroz više sigurnosnih domena. U sigurnosnoj domeni lokani identitet koji je povezan s korisnikom ima više atributa od kojih su neki označeni kao povjerljivi i ne smiju se slati drugim sigurnosnim domenama. Federalni identitet se postupno kreira skupljanjem atributa iz lokalnog identiteta u federaciji identiteta, te se na taj način informacije mogu prenositi među pružateljima s iste federacije identiteta. Subjekti imaju više lokalnih identiteta koji su opisani kao skup atributa jednog subjekta.

Liberty Alliance Identity Federation

Ovaj uzorak sigurnosti omogućuje spajanje identiteta u više organizacija pod federacijom identiteta. Kako federacija koristi različite pružatelje i stoga prati različite standarde dolazi do otežanog posjećivanja nekih mjesta pa se s ovim uzorkom nastoji to pojednostaviti. Koristi se kod problema kada dva pružatelja usluga nisu nikako povezana pa korisnik ima više nepovezanih računa. Pružatelji usluga formiraju federacije identiteta s operativnim sporazumima i među njima je jedan koji se ponaša kao pružatelj identiteta odgovoran za upravljanje federalnim identitetom. Obzirom da se ti identiteti sastoje od nekoliko lokalnih moguće je prenositi informacije korisnika pružateljima usluga kao što se to radi prema subjektima.

4.2 Autentikacija

Jednom kada je identitet potvrđen, sustav treba dokaz o autentičnosti tog identiteta. U ovom poglavlju će biti objašnjeno kako korisnicima dopustiti pristup korištenju određenim resursima nakon što se njihov identitet verificirao.



Slika 5 Odnos između uzoraka za Autentikaciju.
Vlastita izrada prema (Fernandez-Buglioni, 2013)

Authenticator

Uzorak kojim se implementira provjera da je subjekt, koji je zatražio pristup, upravo onaj za kojeg se predstavlja. Postoje informacijski sustavi koji sadrže vrijedne resurse kao npr. poslovne planove ili bilo kakvu drugu dokumentaciju, te potrebno je omogućiti pristup samo onim osobama kojima su ti podaci važni za rad i slično. Pitanje je kako spriječiti ilegalan pristup podacima. Potrebno je koristiti jednu točku pristupa koja će primati sve interakcije zatražene od strane subjekta i primijeniti protokol za provjeru identiteta. Funkcionira na način da autenticira korisnikov zahtjev putem provjere informacija za autentikaciju, te ako je ona uspješna stvara nekakav dokaz o korisnikovom identitetu kako bi je on mogao dalje koristiti.

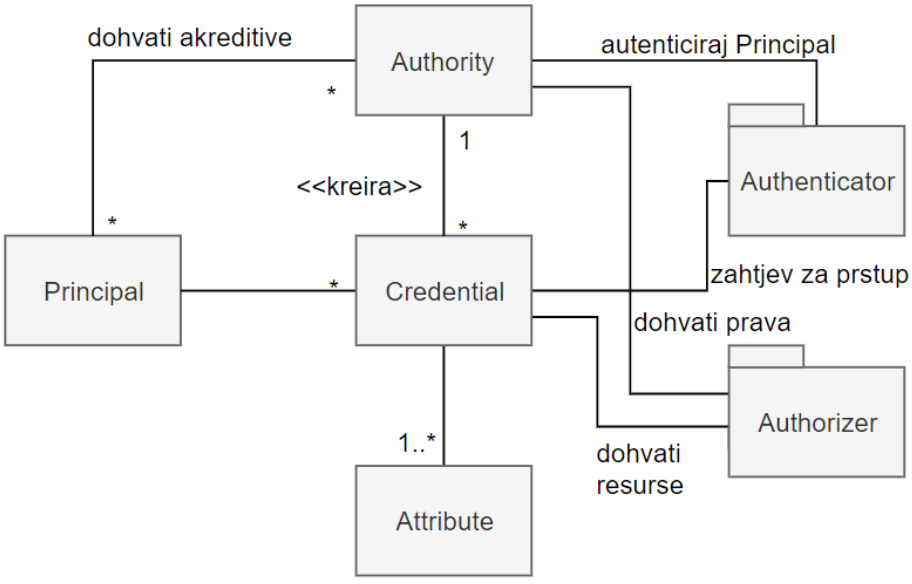
Remote Authenticator/Authorizer

Pruža mogućnost provjere autentičnosti i autorizacije kada se pristupa dijeljenim resursima u labavo-spojenom distribuiranom sustavu. Za primjer korištenja može se uzeti neka velika organizacija koja zaposlenike ima u više zemalja pa su informacije zaposlenika

podijeljene u odgovarajuće zemlje. Uzorak se koristi za problem kod autenticiranja i autoriziranja distribuiranih sustava, a rješenje je postaviti jednu ulaznu točku (npr. neki server) koja može preusmjeriti korisnike na točno onaj server koji sadrži podatke za prijavu. Tako klijent, osoba koja se želi prijaviti na udaljeni sustav, šalje zahtjev koji se preusmjerava na server s ulogom autentikatora. On sadrži podatke za prijavu i izvršava autentikaciju odnosno validira subjekta koji je poslao zahtjev.

Credential

Pruža sigurno sredstvo za bilježenje autentikacije i autorizacije informacija koje su korištene u distribuiranim sustavima. Koristi se u kontekstu sustava kada korisnik jednog sustava želi pristupiti resursima nekog drugog sustava što se bazira na temelju povjerenja koje dijele ta dva sustava. Da bi se dopustio pristup vanjskim korisnicima neke aplikacije potrebno je riješiti i neke probleme. Korisnik bi prvo trebao biti u mogućnosti dati dovoljan broj informacija, bez da je izložen nekakvom napadu. Također je potrebno da podaci koji se prenose budu zaštićeni i držani u privatnosti. Povjerenje između sustava koji prihvaća akreditive i onoga koji izdaje treba postojati. Način na koji se može dati pristup nekom vanjskom korisniku da koristi resurse sustava kojem ne pripada je da se podaci koji se stvaraju i koji su korišteni u sustavu, za autentikaciju i autorizaciju, pohrane u strukturi podataka izvan tog sustava. Određenom entitetu, koji predstavlja neku osobu ili proces, dodjeljuje se akreditiv, te on predstavlja identitet i prava te osobe ili procesa.



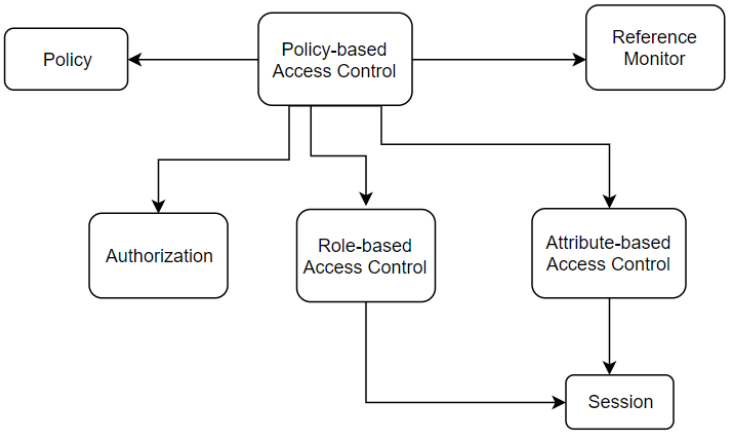
Slika 6 UML dijagram klasa uzorka Credential. Vlastita izrada prema (Fernandez-Buglioni, 2013)

Credential izdaje *Authority* kada mu osoba ili određeni proces pošalje zahtjev za izdavanjem. Provjerava se s uzorkom Authenticator koji je prethodno objašnjen. Ono po čemu je ovaj uzorak specifičan je to što se postiže kreiranjem objekta klase *Attribute* i sadrži imena i vrijednosti. Kreiranjem atributa pokazuje se da je subjekt autenticiran i identificiran kao

korisnik. Može se uočiti kako ovaj uzorak koristi druga dva uzorka koja služe za autentikaciju i autorizaciju.

4.3 Kontrola pristupa

Nakon prethodnih provjera slijedi kontroliranje pristupanja određenim resursima, a pristup ovisi o ulozi osobe koja šalje zahtjev za pristupanje. Raznim modelima se definiraju prava svakog korisnika. Sljedeći uzorci pomažu u implementaciji kontrole pristupa.



Slika 7 Odnos između uzoraka za kontrolu pristupa.
Vlastita izrada prema (Fernandez-Buglioni, 2013)

Authorization

Poznat i pod nazivom Matrica pristupa (eng. Access Matrix) je uzorak koji opisuje tko smije pristupati resursima sustava za koje je potrebno imati kontrolu pristupa. Koristi se u sustavima koji imaju vrijedne informacije te je stoga potrebna i kontrola pristupanja istima. Primjenjuje se na svakom aktivnom subjektu koji može pristupati resursima i definira kojim resursima može pristupiti i na koji način. Subjekt je aktivni entitet koji šalje zahtjev za pristup nekom resursu. Za svaki tip načina (eng. access type) pristupanja postoji opis načina odnosno popis prava za subjekta koji pokazuju kojim resursima on može pristupiti i na koji način.

Role-Based Access Control

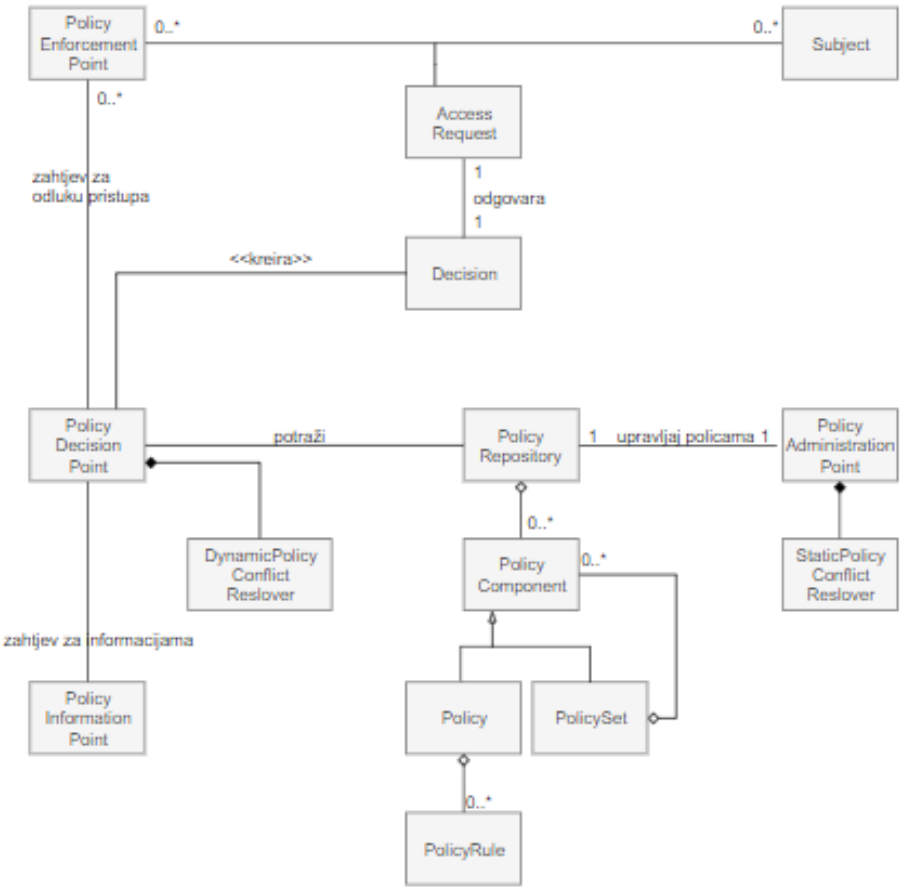
Ovaj uzorak opisuje kako nekome dodijeliti potrebna prava za pristup resursima. Koristi se u bilo kakvom okruženju koji zahtjeva kontrolirani pristup računalnim resursima gdje postoji veliki broj korisnika i informacija. Služi tome da se smanji broj pravila kojima se kontrolira pristup te njihovo značenje čini puno jasnijim. Obzirom da sustavi, pogotovo oni web bazirani, imaju različite uloge korisnika, poput zaposlenika, korisnika i partnera, potrebno je implementirati ovakav pristup kako bi se kontroliralo tko pristupa kojem resursu. Uvijek postoji neko mjesto gdje se čuvaju spremljene uloge za korisnike koje pokazuju točno onu ulogu korisnika koja mu pripada. Kada korisnik pokuša pristupiti sustavu te nakon validacije dodjeljuje mu se određena uloga.

Multilevel Security

Uzorak kojim se kategoriziraju informacije i sprječava njihovo otkrivanje. Prikazuje kako korisnicima dodijeliti klasifikacije i podacima razinu osjetljivosti. Primjenjuje se za one sustave čiji su podaci i dokumenti od velike vrijednosti i njihovo razotkrivanje može dovesti do ozbiljnih problema. Sam naziv uzorka nam govori na koji način se rješavaju problemi vezano uz prethodno. Osim dodjele klasifikacija i razine osjetljivosti bitno je raspodijeliti organizacijske jedinice u kategorije. Dodjelu klasifikacija korisnicima i podacima izvode pouzdani procesi koji imaju određena prava.

Policy-Based Access Control

Opisuje način odluke o tome da li je subjekt autoriziran za pristup, prema polici koja je definirana u centralnom repozitoriju polica. Uzorak se koristi u kontekstu centraliziranih ili distribuiranih sustava koji imaju veliki broj resursa i subjekata koji im mogu pristupiti, te postoje pravila koja definiraju kontrolu pristupa određena od strane organizacije. Potrebno je primijeniti pravila kontrole pristupa na način da se sva postojeća uzimaju u obzir i odluka donese na temelju svih njih. Kada subjekt pošalje zahtjev prvo ga presreće točka za provođenje politike pristupa. Ona prosljeđuje zahtjev sigurnosnoj infrastrukturi odnosno točki za politiku odluke koja donosi odluku o pristupanju.



Slika 8 UML dijagram klasa uzorka Policy-Based Access Control. Vlastita izrada prema (Fernandez-Buglioni, 2013)

Subjekt pošalje zahtjev kojeg presretne PEP (Policy Enforcement Point), koja ima ulogu primijeniti politike organizacije na pristup. Ona dalje prosljeđuje zahtjev PDP (Policy Decision Point) točki koja treba donijeti odluku o pristupu. Kako bi donijela tu odluku pretražuje repozitorij politika koji ima pohranjena sva pravila vezana uz te politike. *PolicyAdministrationPoint* klasa predstavlja jedinstvenu točku za upravljanje pravilima politika. Dio te klase je klasa *StaticPolicyCOnflictReslover* koja je odgovorna za identificiranje pravila u konfliktu koja se nalaze u repozitoriju polica. Sličnome služi i klasa i *DynamicPolicyConflictResolver* koja je odgovora za rješavanje konflikata kod kreiranja jedinstvene odluke o pristupu.

Access Control List

Omogućuje kontrolirani pristup objektima pokazujući koji subjekti mogu pristupiti objektu i na koji način. Isto se koristi u centraliziranim i distribuiranim sustavima. Sva prava za korisnike su definirana i modelirana kao matrica pristupa, gdje red predstavlja subjekta, a stupac objekta. Ona također specificira koje akcije objekt može provesti. Za ovaj uzorak potrebna je lista za kontrolu pristupa koja je sastavljena od prava vezana uz određenog subjekta. Zbog toga subjekt može pristupiti objektu ako njegovo pravo postoji u pristupnoj listi objekta.

Capability

Omogućuje implementaciju pružanja akreditiva ili „ulaznice“ subjektu kako bi mogao pristupiti nekom objektu, ali na točno određeni način. U distribuiranim sustavima postoji veliki broj subjekata i objekata pa bi zbog toga bilo dobro implementirati matricu pristupa, ali na način da bude efikasna u vremenu i prostoru. S ovim uzorkom se to radi na način da se implementira izdavanje skupa mogućnosti svakom subjektu koje definiraju pravo pristupa određenom objektu. Te mogućnosti sadrže skup prava (klasa *Rights*) koje subjekt ima nad objektom i to samo ako to pravo postoji u mogućnostima.

Reified Reference Monitor

Uzorak je još poznat pod nazivima Intercepting Filter i Application Controller. Opisuje kako prisiliti autorizaciju kada subjekt pošalje zahtjev zaštićenom objektu i tako omogući subjektu odluku. Koristi se u okruženjima s višestrukim procesima u kojima se pristupa zaštićenim objektima. Pitanje je kako kontrolirati akcije subjekta jer nekada mogu biti puno kompleksnije od toga da li dati ili ne dati pristup. Stoga je potrebno definirati proces koji presreće zahtjeve, sprema njih i njihove atribute, provjerava s autorizacijom usklađenost i donosi odluku. Kod ovog uzorka bitna stavka je definiranje skupa autorizacijskih pravila koja se pretražuju tijekom obrade korisničkih zahtjeva kako bi se donijela prava odluka.

Controlled Access Session

Pruža način implementacije kontrole pristupa tako da subjekt resursima može pristupiti s različitim pravima bez da se mora svaki put autenticirati. Koristi se u okruženjima gdje je potrebna kontrola pristupa u kojoj korisnici trebaju biti klasificirani ovisno o vrsti posla, odjelu, zadacima i slično. Kako bi se dobro implementirao ovaj uzorak potrebno je definirati jedinicu interakcije odnosno sesiju (eng. session) koja ima određeni životni vijek. Tijekom njezinog trajanja aktiviraju se određena prava korisnika koji se prijavio i samo su ona dostupna.

Session-Based Role-Based Access Control

Uzorak kojim se primjenjuje kontrola pristupa ovisno o ulozi subjekta i daje prava koja se primjenjuju u danom vremenu što je definirano sesijom. Stavlja se u kontekst okruženja gdje je potrebna kontrola pristupa ovisno o poslu koji određena osoba radi i dodijeljenim ulogama te osobe. Kada se pojavi problem kako nekome narediti da koristi točno onu policu organizacije koju treba ovisno o ulozi primjenjuje se ovaj uzorak. Stoga, svaka uloga ima svoja prava i tijekom sesije koja kontrolira kako su ta prava korištena. Također, tijekom sesije samo neke od uloga mogu biti aktivirane.

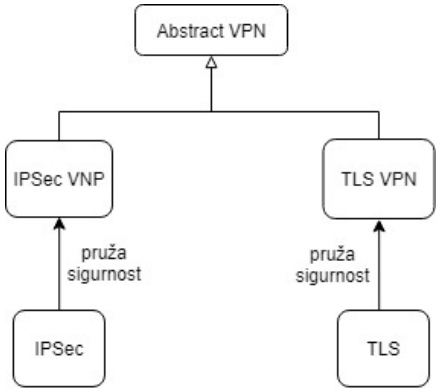
Security Logger and Auditor

Ovaj uzorak služi za praćenje akcija korisnika kako bi se točno znalo koji korisnik je napravio koju akciju i kada. Primjenjuje se u svakom sustavu s osjetljivim informacijama za koji je bitno znati tko im je pristupio i tko ih je koristio. Kada dođe pitanje kako se to sve može pratiti koristi se ovaj uzorak. Funkcionira na način da zapisuje sve operacije nekog korisnika. Može se odabrati bilo kakvo spremište za takve podatke, kao npr. baza podataka ili obična datoteka.

4.4 Sigurnost mreže

Postoje četiri razine TCP/IP mreže na koje se može primijeniti sigurnost, a to su: Aplikacijska, Transportna, Internet i Pristupna.

Za svaku razinu postoji uzorak koji se može iskoristiti za njezinu sigurnost. IPSec i TLS (Transport Layer Security) su dva najčešće korištena uzorka.



Slika 9 Odnos između uzoraka za sigurnost mreže.
Vlastita izrada prema (Fernandez-Buglioni, 2013)

Abstract VPN

Opisuje način na koji se može uspostaviti sigurnosti kanal između dvije krajnje točke, koje imaju nekakvu vrstu autentifikacije, kroz kriptografski tunel. Koristi se kod sigurne komunikacije između korisnika koji se nalaze na različitim mjestima zbog mogućnosti presretanja poruka. Problem se može riješiti, kako je već spomenuti, preko kriptografskog tunela između krajnjih točaka komunikacije i autentifikacijom na svakoj.

IPSec VPN

Uzorak kojim se osigurava implementacija sigurne veze s kriptografijom između dviju krajnjih točaka. Služi za sigurnu komunikaciju korisnika koji su na različitim mjestima. Sigurni kanal komunikacije može se uspostaviti preko kriptografskog tunela na IP razini.

TLS VPN

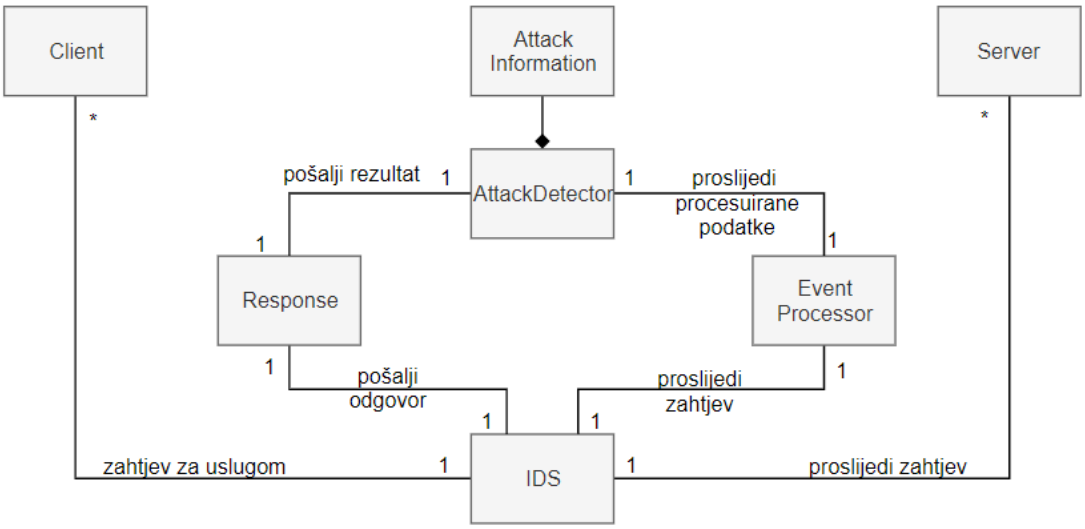
Ovim se uzorkom omogućuje implementacija sigurne komunikacije dviju krajnjih točaka preko kriptografskog tunela kroz transportni sloj. Velik broj korisnika danas komunicira putem interneta ili druge nesigurne mreže stoga je primjena uzorka bitna za to područje. Rješenje u osiguranju komunikacije transportnog sloja je u tome da se koriste TLS obrnuti proxy poslužitelji kako bi se povezali udaljeni korisnici.

Transport Layer Security

Omogućuje sigurnu komunikaciju klijenta i servera kada se poruke prenose preko transportnog sloja Interneta. Stavlja se u kontekst korištenja aplikacija gdje se šalju osjetljive informacije. U oba slučaja, klijenta i servera, moguće je da se predstavljaju kako netko drugi, stoga je potrebno implementirati sigurnu komunikaciju. Funkcionira na način da klijent šalje zahtjeve prema serveru, a svaki od njih sadrži svoje usluge koje klijenti mogu zahtijevati. TLS je protokol koji se brine o sigurnoj vezi između klijenta i servera, te izvodi akcije autentikacije i autorizacije.

Abstract IDS

Služi za implementaciju praćenja prometa koji prolazi kroz mrežu. Svi čvorovi lokalnih sustava koji komuniciraju putem interneta ili druge mreže trebaju biti zaštićeni. Stoga se svaki pristup mreži mora proanalizirati kako bi se napadi spriječili jer jednom kada se detektira podiže se alarm i poduzimaju se potrebne mjere. Zahtjeve koje klijent šalje serveru preuzima IDS (Intrusion Detection System) odnosno sustav za otkrivanje upada, koji ih dalje šalje procesorima za događaje gdje se procesuiraju i analiziraju. Ako dođe do napada kreira se određeni odgovor koji se šalje natrag IDS-u.



Slika 10 UML dijagram klasa uzorka Abstract IDS. Vlastita izrada prema (Fernandez-Buglioni, 2013)

Dvije su glavne uloge ovog UML dijagrama, klijent i poslužitelj. Klijent pošalje zahtjev poslužitelju za neku uslugu. IDS presreće taj zahtjev kako bi ga prvo obradio prije nego ga proslijedi poslužitelju. Predaje zahtjev *EventProessoru* koji procesuira podatke tako da *AttackDetector* može interpretirati događaj. U slučaju napada stvara se odgovor i šalje natrag IDS-u te on poduzima određene mjere, u suprotnom zahtjev se prosljeđuje poslužitelju.

Signature-Based IDS

Također poznat pod nazivima *Rule Based IDS* i *Knowledge-Based IDS* je uzorak koji pomaže u provjeri zahtjeva prema skupu postojećih potpisa napada. Koristi se za distribuirane sustave koji pružaju usluge udaljenim čvorovima. Kod distribuiranih sustava postoji mogućnost napada na lokalni čvor stoga je bitno otkriti napade tako da se usporedi potpis trenutnog s nekima što su se prethodno dogodili. Princip rada sličan je radu prethodnog uzorka. Razlika je što se tijekom analiziranja zahtjeva izvršava usporedba s podacima iz liste koja sadrži potpise svih napada koji su se dogodili kako bi se mogli usporediti s trenutnim.

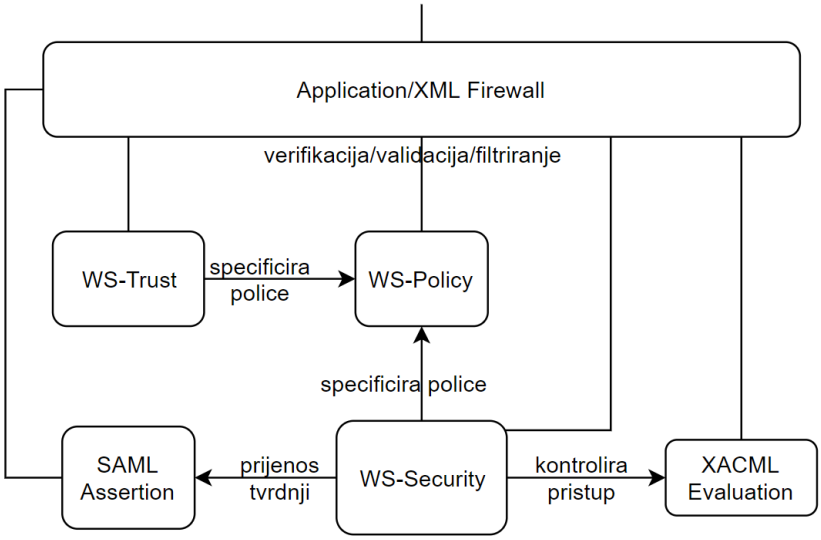
Behavior-Based IDS

Za razliku od prethodnog, ovaj uzorak uspoređuje zahtjeve s uzorcima prometa mreže kako bi otkrio nepravilnosti ako dođe do napada. Koristi se kod bilo koje aplikacije u kojoj se vremensko ponašanje prometa može predvidjeti i koje je ponovljivo. Napadi se mogu otkriti na nadziranjem prometa i ako promet odstupa od normalnog ponašanja tretira se kao napad. Kod ovog uzorka umjesto liste svih prošlih napada postoji lista koja ima prethodne profile ponašanja kako bi se trenutni mogli usporediti i tako na vrijeme detektirati napad.

4.5 Sigurnost web servisa

Za sigurnost web servisa postoje mnogi standardi koji mogu biti poprilično kompleksni, stoga se putem uzoraka pojednostavljuje njihovo razumijevanje i primjena. Također se mogu

bolje međusobno usporediti i otkriti njihova preklapanja i nedosljednosti. Sve te standarde definirale su razne organizacije poput W3C[1], OASIS[2] i IETF[3].

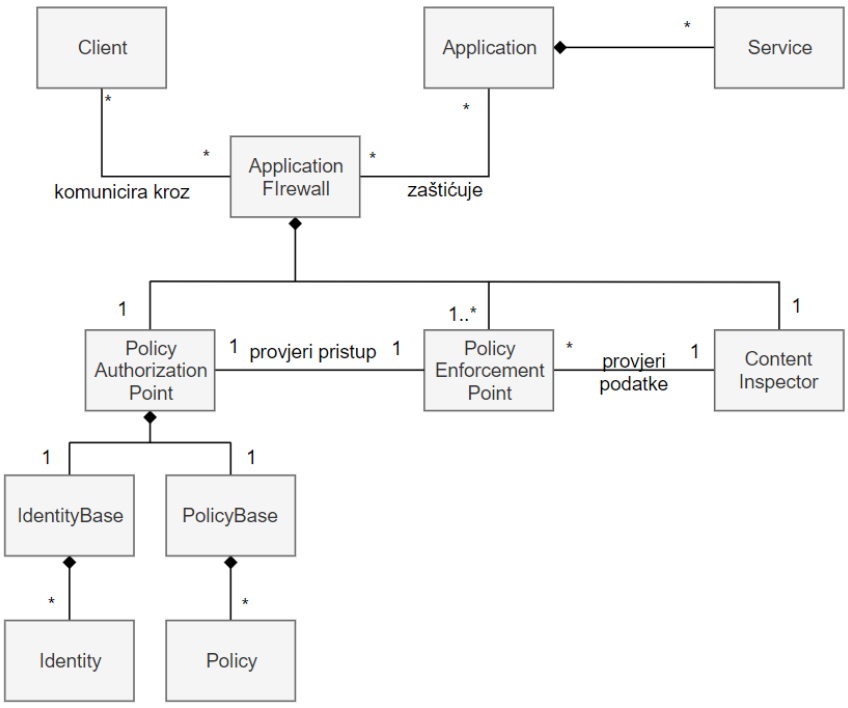


Slika 11 Odnos između uzoraka za sigurnost web servisa. Vlastita izrada prema (Fernandez-Buglioni, 2013)

Application Firewall

Definiraju se politike za kontrolu kojima se filtriraju pozivi i zahtjevi od i prema aplikacijama poduzeća. Uzorak se primjenjuje na sve aplikacije kojima se pristupa s lokalne i ostalih mreža. Kontroliranje akcija korisnika koji imaju lošu namjeru dok pristupaju aplikacijama rješava se na način da se vatrozidom (eng. firewall) analiziraju i autoriziraju zahtjevi koji dolaze. Ima ulogu zaštite same aplikacije koja pruža usluge. Također postoji lista u kojoj su definirana pravila o pristupanju.

[1] Međunarodna zajednica koja razvija web standarde (https://www.w3.org/Consortium/)
[2] Organizacija za unapređenje strukturiranih informacijskih standarda (https://www.oasis-open.org/)
[3] Široka zajednica koja se bavi razvojem internetske arhitekture i nesmetanim funkcioniranjem Interneta (https://www.ietf.org/)



Slika 12 UML dijagram klasa uzorka Application Firewall.
Vlastita izrada prema (Fernandez-Buglioni, 2013)

Sve počinje komunikacijom klijenta i aplikacije. Klijent s aplikacijom komunicira preko implementiranog vatrozida koji ima ulogu zaštite aplikacije. Jedino ako mu on dopusti može pristupiti usluzi koju aplikacije sadrži. Pristupi su kontrolirani putem pravila autorizacije i prikazani u klasi *Policy* koje se prikupljaju u klasi *PolicyBase*. Nadalje, vatrozid se sastoji od klase *PAP* koja centralizira police i identitete, nekoliko *PEP*-sa, te *ContentInspector*-a koji provjeravaju sve podatke koji se šalju preko vatrozida.

XML Firewall

Uzorak kojim se implementira filtriranje XML poruka koje dolaze prema i odlaze iz aplikacije, također prema određenim pravilima. Služi isto za aplikacije kojima se pristupa putem neke mreže, bilo lokalne ili vanjske. Iako mrežni vatrozidi imaju jako dobru sigurnosnu infrastrukturu, postaju beskorisni kada se koriste visoko-razinski protokoli i formati, zato je potrebno napraviti jedan koji će se pobrinuti za komunikaciju s XML porukama. Kod ovog uzorka također postoji lista u kojoj su definirana pravila o pristupanju, te se u točki autorizacije skupljaju informacije identiteta i autorizacije. Postoji i komponenta koja provjerava sadržaj XML poruka, a sadrži u sebi validator, verifikator odnosno potpisnik i komponentu za kriptiranje odnosno dekriptiranje.

XACML Authorization

XACML predstavlja jezik oznake za kontrolu pristupa koji predstavlja standardizirana pravila za autorizaciju. Uglavnom se koristi kod velikih poduzeća s puno vanjskih partnera i koji su povezani s ostalim poduzećima, a ona mogu pristupiti njihovim resursima. Potrebno je

ujediniti politike pristupa za cijelu organizaciju na način da se za njihovo pisanje koristi standardni format. Postoji tako točka administracije politika koja izvršava akcije dodavanja politike, brisanja, ažuriranja i sličnog. Za sve to koristi određena pravila.

XACML Access Control Evaluation

Ovaj uzorak se nadovezuje na prethodni na način da opisuje implementaciju odluke o tome da li je zahtjev autoriziran prema policama koje su definirane XACML Authorization uzorkom. Koristi se radi problema oko toga hoće li se netko držati onoga što je definirano policama. Rješenje ovoga je da se resursi zaštite bodovima za provođenje politike te se svi zahtjevi pristupa evaluiraju. Evaluiraju se na način da se putem zajedničkog formata pošalju točki za provođenje politike koja vraća svoju odluku. Na odluku može primijeniti skup definiranih politika.

Web Services Policy Language

Opisuje kako prezentirati politike kontrole pristupa i kako korisnik web servisa može iznijeti zahtjeve na standardizirane načine. Može se koristiti kod aplikacija koje koriste web servise. Opisivanje polica za kontrolu poziva web servisa rješava se na način da se svaki WSDL web servisa veže na XACML komponentu. Svaki WSDL ima svoju krajnju točku, poruku i operaciju. Uključuju nekoliko aspekata poput slanja poruka, pružanja privatnosti, validacije, verifikacije. Svaki od njih imaju svoju odgovarajuću politiku, a politika ima svoje ciljeve koji se moraju postići sigurnim pozivom. Ciljevi sadrže nekakve strategije definirane pravilima.

WS-Policy

Uzorak koji služi za implementaciju skupa tvrdnji koje naširoko opisuju razne zahtjeve i mogućnosti web servisa. Stavlja se u kontekst aplikacija koje međusobno komuniciraju putem web servisa u nesigurnom okruženju. Kako bi se osigurali pravilni zahtjevi i mogućnosti potrebno je primijeniti određene politike. Sastoji se od 3 dijela: dio koji predstavlja način na koje bi politike trebale biti napravljene, drugi koji prikazuje unutarnji tijek podataka kroz blokove polica, te zadnji dio koji predstavlja interakciju, odnos, polica s vanjskim korisnicima.

WS-Trust

Uzorkom se implementiraju sigurnosni token i alat za povjerenje kojima web servisi autenticiraju jedni druge i tako uspostavljaju povjerenje. Razne aplikacije koje međusobno komuniciraju putem web servisa mogu koristiti ovaj uzorak kako bi se uspostavila sigurna veza među njima. Sa sigurnosnim tokenom se odvija pouzdana komunikacijama među web servisima. Sadrži ista tri dijela kao i prethodni uzorak.

SAML Assertion

Uzorak kojim se implementira način sigurne komunikacije sa informacijama za sigurnost između sigurnosnih domena. U distribuiranim sustavim gdje sigurnosne domene

komuniciraju putem web servisa i razmjenjuju svoje resurse potrebno je definirati jedinicu za upravljanje identitetima u domeni subjekta koja izdaje tvrdnje o subjektima u toj domeni.

4.6 Kriptografija web servisa

Kako tijekom prijenosa informacija preko web servisa one mogu biti eksponirane bitno ih je na neki način zaštiti od toga. Kriptiranje pruža povjerljivost jer pretvara poruke u nečitljiv format tijekom prijenosa i može ih dekriptirati samo njihov primatelj. Dva su tipa kriptiranja: simetrično i asimetrično. U ovom poglavlju biti će opisani uzorci za sigurnost čiji je glavni dio kriptiranje.

Symmetric Encryption

Kriptira poruke koje se prenose i može ih dekriptirati samo ona osoba koja ima odgovarajući ključ. Za kriptiranje i dekriptiranje koristi se isti ključ. Koristi se u aplikacijama koje razmjenjuju osjetljive informacije i gdje postoji mali broj korisnika. Kada se poruka kriptira njezin primatelj primjenjuje obrnutu transformaciju pomoću valjanog ključa kako bi dekriptirao poruku. U komunikaciji sudjeluju dva tipa uloga, pošiljatelj i primatelj poruka. Pošiljatelj kriptira, a primatelj dekriptira dobivene poruke. Također, koriste isti algoritam kako bi de/kriptirali poruke.

Asymmetric Encryption

Ima istu svrhu kao i prethodno kriptiranje, ali se kod ovog koriste javni i privatni ključ za kriptiranje odnosno dekriptiranje poruka. Koristi se kod aplikacija koje razmjenjuju informacije preko nesigurnih mreža. Slanje poruka na taj način rješava se tako da za kriptiranje poruka primjene matematičke funkcije kako bi ih učinile nečitljivima. Uzorak se razlikuje od prethodnog po tome što postoji više parova javnih/privatnih ključeva. Ostalo sve ide po istom principu.

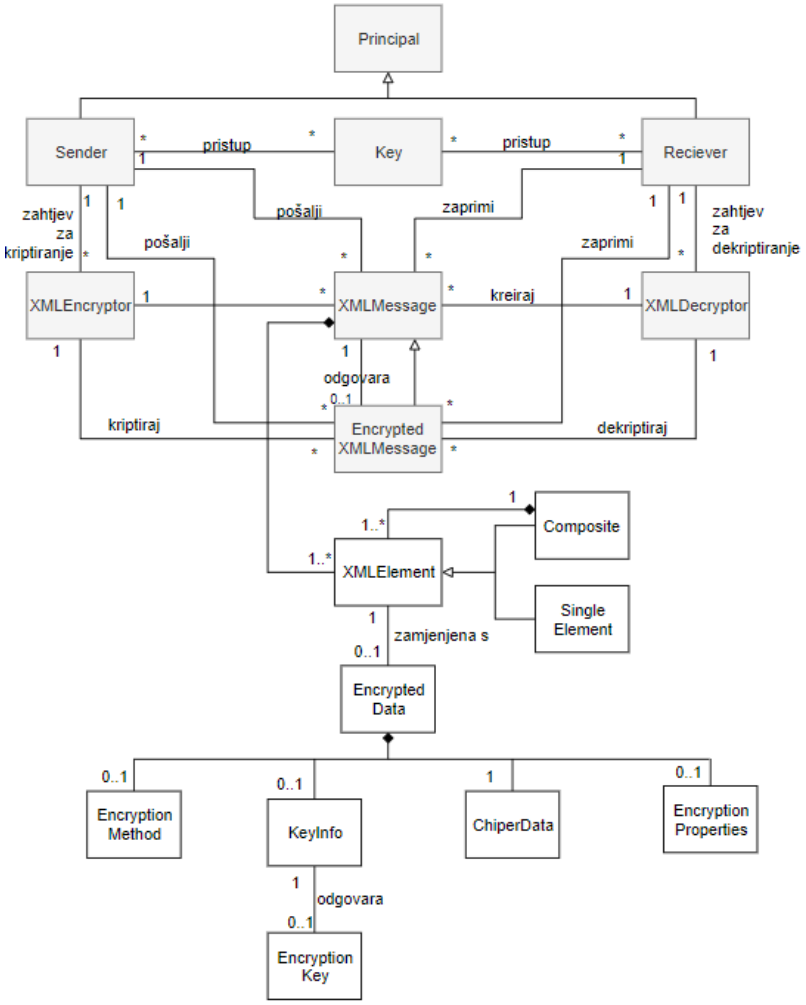
Digital Signature with Hashing

Uzorak kojim se implementira dokazivanje originalnosti poruke primatelju i njezin integritet. Koristi se jer ljudi često razmjenjuju informacije preko nesigurnih mreža pa je potrebno dokazati njihovu izvornost. Radi na način da se prvo napravi sažetak (eng. hash) poruke, te zatim kriptira koristeći privatni ključ pošiljatelja. Kada primatelj dobije poruku prvo verificira potpis koristeći javni ključ pošiljatelja za dekriptiranje. Zatim primatelj također napravi sažetak poruke, te ako odgovara dobivenom znači da se poruka nije mijenjala tijekom prijenosa. Pošiljatelj može poslati običnu poruku ili potpisanu. Isto tako postoji lista gdje su sadržani parovi ključeva: javni i privatni. Javnom ključu se pristupa u repozitoriju, dok je privatni kod njegovog vlasnika. Osobe koje potpisuju koriste algoritam za sažimanje kako bi kreirale potpis, a osobe koje verificiraju koriste algoritam za potpisivanje kako bi ga verificirale.

XML Encryption

Pruža implementaciju kriptiranja određenih dijelova poruke. Uzorak je potreban kada se šalju/primaju XML poruke kroz nesigurne mreže. Sprječavanje čitanja poruka od neautoriziranih ljudi rješava se na način da se XML poruke kriptiraju kako bi ih mogao pročitati samo legitiman primatelj. Podržava i simetrično i asimetrično kriptiranje. Pošiljatelji ili primatelj šalju/primaju obične ili kriptirane poruke koje se sastoje od svojih XML elemenata. Kriptirana poruka sadrži još dijelove poput metode kriptiranja, informacije o ključu, vrijednost šifre i postavke kriptiranja.

UML dijagram klasa se sastoji od dva dijela. Klase bez pozadinske boje predstavljaju strukturu XML poruke i prikazuje kako se dijelovi poruke i koji dijelovi mogu kriptirati. Dok, obojane klase predstavljaju proces komunikacije između pošiljatelja i primatelja, te izmjenu kriptiranih i nekriptiranih poruka između njih. Klasa *Principal* može biti bilo kakav korisnik organizacije koji šalje ili prima XML poruke odnosno ima ulogu pošiljatelja, koju predstavlja klasa *Sender,* ili primatelja, koji predstavlja klasa *Reciever*. Svaka XML poruka se sastoji od svojih XML elemenata, koji se mogu sastojati od svojih elemenata i tako dalje. Klase *XMLEncyptor* i *XMLDecryptor* služe za kriptiranje poruka kod pošiljatelja odnosno dekriptiranje kod primatelja. Kriptirani podaci sadrže elemente poput metode, ključa, podataka i ostalih postavki kriptiranja.



Slika 13 UML dijagram klasa uzorka XML Encryption. Vlastita izrada prema (Fernandez-Buglioni, 2013)

XML Signature

Uzorak kojim se implementira dokazivanje pošiljatelja da je upravo on poslao poruku. Pruža integritet poruka i zahtjeva standardiziranje prije nego se naprave sažetak i potpis. Služi kod slanja SOAP[4] poruka kroz nesigurne mreže. Potrebno je primijeniti kriptografiju kako bi se kreirao jedinstveni potpis za svaku poruku. Te su poruke standardizirane prije nego se primjeni algoritam za potpisivanje.

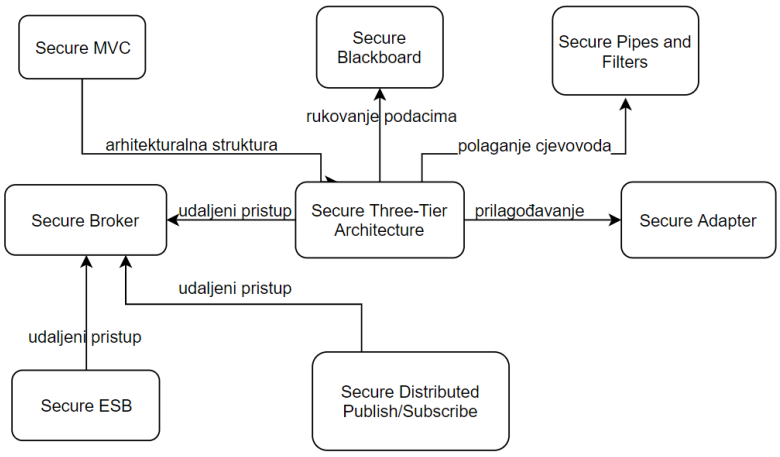
WS-Security

Uzorak koji opisuje kako ugraditi već postojeće sigurnosne mehanizme u SOAP poruke kako bi se pružilo povjerenje, integritet, autentikacija, ne-povreda podataka. Obzirom da postoje razna rješenja za navedene sigurnosti potrebno je napraviti selekciju kako bi se odabrao odgovarajući sigurnosni mehanizam. Stoga je potrebno definirati područje u formatu poruke s parametrima koji će točno specificirati što se koristi.

[4] Simple Object Access Protocol – protokol za razmjenu poruka u XML formatu

4.7 Aplikacijski sloj

Sloj srednjeg softvera općenito uključuje skup funkcija koje služe aplikacijama. Razne usluge koje te funkcije pružaju mogu biti podrška aplikacijama ili njihovom izvršavanju. U ovom poglavlju opisani su uzorci kojima se dodaje sigurnost srednjem sloju softvera.



Slika 14 Odnos između uzoraka za sigurnost aplikacijskog sloja. Vlastita izrada prema (Fernandez-Buglioni, 2013)

Secure Broker

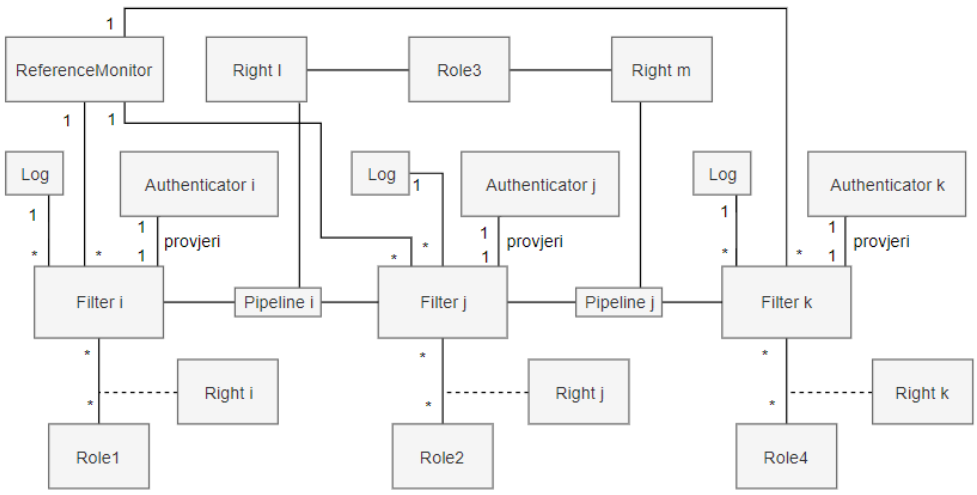
Ovaj uzorak proširuje uzorak dizajna Broker kako bi se osigurala sigurna komunikacija komponenata. Osiguravaju se računalni sustavi koji imaju više neovisnih komponenata. Sigurnost aktivnosti brokera rješava se na način implementiranja sigurnog posrednika (eng. broker) koji osigurava zajedničku autentikaciju između servera i klijenata. Subjekt ima zadatak pružiti identitet komponentama kako bi se međusobno mogle autenticirati. Ulogu autoriziranja ima uzorak Reference Monitor koji na temelju identiteta može dopustiti ili odbiti davanje usluge. Za kriptiranje prometa koji se prenosi između komponenata zaslužna je uzorak za sigurnost kanala.

Secure Pipes and Filters

Omogućuje implementaciju sigurnog rukovanja tokovima podataka transformacijom ili filtriranjem. Postoje paralelni softveri koji imaju distribuiranu platformu izvršavanja čije komponente trebaju određenu razinu sigurnosti tijekom razmjene podataka. Osiguravanje toka podataka rješava se dodavanjem osnovnih sigurnosnih mehanizama kako bi se izvršavale autentikacija, autorizacija, skrivanje i evidentiranje podataka.

Za uzorak je prikazan objektni dijagram jer se time lakše prikazuje i promatra se kao skup određenog broja faza. Dijagram tako prikazuje kako se za kontrolu pristupa u fazama primjenjuje uzorak Role-Based Access Control. Svaki korisnik je dio određene uloge i ima prava koja su dana toj ulozi, te se to i provjerava tijekom svake faze. Nadalje, postoje podsustavi *Authenticator* koji su instance uzorka Authenticator. Ovim uzorkom se filterima

omogućuje autenticiranje pošiljatelja podataka. Zatim postoje i instance uzorka Security Logger and Auditor koji pruža mogućnosti praćenja svakog pristupa nekom podatku.



Slika 15 UML dijagram klasa uzorka Secure Pipes and Filters.
Vlastita izrada prema (Fernandez-Buglioni, 2013)

Secure Blackboard

Služi za implementaciju sigurnosti podataka kada ploči (eng. blackboard) odnosno središnjem spremištu podataka pristupa kolekcija izvora znanja. Prava na čitanje i ažuriranje su predefinirani i evidentiraju se. Kako bi se riješio problem aktivnosti koji se obavljaju nad pločom potrebno je dodati sigurnosne mehanizme za kontrolu raznih prijetnji koji će kontrolirati komponente. Kod ovog uzorka koriste se još Security Logger and Auditor, Reference Monitor i Authenticator. Na kontrolu je povezan izvor znanja koji preko nje šalje zahtjeve za pristupanje crnoj ploči. Svakom izvoru znanja dodijeljena je određena uloga i tip pristupanja.

Secure Adapter

Općenito, uzorak dizajna Adapter prilagođava sučelje neke klase u pristupačnije sučelje, a ovaj uzorak služi tome da se prilagodi na siguran način. Koriste se u okruženjima gdje se treba koristiti neko sučelje nekompatibilno s klasom koja mu pristupa. Potrebno je identificirati moguće prijetnje za adapter i definirati politike i odgovarajuće mehanizme za sprječavanje. Zahtjevi klijenta idu preko adaptera te oni moraju biti autorizirani, a klijent pristupa prema određenoj ulozi i određenom tipu pristupa.

Secure Three-Tier Architecture

Proširuje Three-tier Architecture uzorak dizajna kojim se aplikacija dijeli na slojeve od kojih svaki ima svoju odgovornost. Uzorak omogućuje globalni pogled sigurnosti na sva tri sloja. Primjenjuje se na kompleksne i heterogene aplikacije koje rade nad bazama s osjetljivim informacijama. Da bi se osiguralo od vanjskih i unutarnjih prijetnji potrebno je zaštiti svaki sloj posebno. Tri sloja su: prezentacijski, poslovni i podatkovni. Prvi sloj uključuje korisnika i sučelje

sustava, drugi uključuje jedinstveni model podataka, a treći definira pristup podacima i mehanizmima za kriptiranje.

Secure Enterprise Service Bus

Uzorak kojim se na jednostavan i siguran način mogu integrirati različite usluge i komponente. Koristi se kod aplikacija koje imaju web servise, direktorije, baze podataka i ostale komponente. Rješenje se implementira na način da se uvede zajednička struktura sabirnice (eng. bus). Ona posreduje između komponenata i pruža osnovne usluge.

Secure Distributed Publish/Subscribe

Implementira se siguran način rada pretplate i objavljivanja tako što se odvajaju jedno od drugog. Kao i prethodni uzorak, također se koristi kod aplikacija koje imaju web servise, direktorije, baze podataka i ostale komponente. Organiziranje izdavača i pretplatnika da komuniciraju na siguran način izvodi se tako da se koristi sigurni kanal kroz koji se događaji mogu slati. U implementaciji ovog uzorka sudjeluju još Secure Channel, Authenticator i Signature Verifier.

Secure Model-View-Controller

Dodatak je MVC (Model-View-Controller) uzorku dizajna, a omogućuje sigurnu interakciju korisnika sa sustavom. MVC pruža modularnost aplikaciji razdvajajući je u komponente modela, pogleda i kontrolera. Pitanje je kako održati sigurnost između te tri komponente kako bi se zaštitile od potencijalnih napada. Ovaj uzorak, sigurni MVC, implementira siguran način pristupa i mijenjanja informacija koje se nalaze u komponenti modela.

Implementacija uzorka nije kompleksna jer se sastoji od implementacije nekih uzoraka koji su prethodno opisani iz ostalih kategorija. Uzorak Authenticator koristi se za autentikaciju kod modela i kontrolera, a Secure Channel služi za sigurnu komunikaciju između modela i pogleda, te modela i kontrolera. Za sigurnost modela koriste se još uzorci Secure Logger i Reference Monitor.

5. Java EE aplikacije, web servisi, identiteti i usluge

Ovo su uzorci koje je razvio tim inženjera Sun Java Centra, Ramesh Nagappan i Christopher Steel. Omogućuju implementaciju sigurnijih Web aplikacija što uključuje jednorazinsku autentikaciju, višerazinsku autentikaciju i dodjelu identiteta korisnicima.

Prema sukladnosti opće teorije uzoraka dizajna, autori su napravili predložak za uzorke sigurnosti, te podijelili sadržaj na sljedeći način: Problem, Snage, Rješenje, Struktura, Strategije, Posljedice, Sigurnosni faktori i rizici, Pregled, Srodni uzorci. Uzorci su također podijeljeni u četiri razine: web, poslovna, web servisi, identitet.

5.1 Razina weba

Razina weba je prva meta web aplikacija koju će netko iskoristiti za napad jer je to ujedno i prvi korak pristupa aplikaciji nekog korisnika. Uzorci vezani za ovu razinu omogućuju sigurnosne akcije poput autentikacije, autorizacije, privatnosti (Flylib, Web-Tier Security Patterns, 2017). Opisani su u tablici ispod preko problema do rješenja te njihovih prednosti, a detaljnije će neki biti razrađeni kroz praktični primjer.

Tablica 2 Prikaz uzoraka koji se primjenjuju na razini web sloja

Naziv uzorka	Problem	Rješenje	Prednosti
Authentication Enforcer	Slanje povjerljivih informacija korisnika tijekom prijave u sustav.	Kreiranje centralizirane autentikacije za korisnike koja sadrži detalje autentikacijskog procesa.	Korisnici su autenticirani na ispravan način, centralizirana autentikacija.
Authoriztion Enforcer	Korištenje dijelova aplikacije namijenjene određenoj ulozi korisnika.	Implementacija kontrolera za pristup koji za autorizaciju koristi standardne Java API klase za sigurnost.	Minimalna veza između pogleda i kontrolera sigurnosti, kontrola URL-a, centralizirana autorizacija.
Intercepting Validator	Nadzor i validacija nad podacima koji se prenose.	Potrebno je koristiti validator koji će pročistiti (filtrirati) i validirati podatke prije nego se koriste u aplikaciji.	Dinamička validacija različitih informacija, pravila validacije odvojena od prezentacijske logike.
Secure Base Action	Sastaviti u cjelinu interakciju svih komponenti koje su vezane za sigurnost.	Napraviti koordinaciju komponenti na jednom mjestu za administraciju sigurnosti.	Centralizirane funkcionalnosti sigurnosti, odvojene logike sigurnosti i prezentacije, koordinirane sigurnosne komponente.

Naziv uzorka	Problem	Rješenje	Prednosti
Secure Logger	Siguran način pohrane svih zapisa događaja aplikacije. Može dovesti do redundancije i kompleksnosti.	Implementacija sigurnog načina pohrane zapisa vezanih uz aplikaciju kojima se ne može lako pristupiti.	Povjerljivost i integritet pohranjenih zapisa, centraliziranost.
Secure Pipe	Pružanje privatnosti i sprječavanje neovlaštenog pristupa transakcijama korisnika.	Jednostavan i standardiziran način pružanja zaštite tijekom slanja podataka preko mreže njihovim kriptiranjem.	Sigurno korištenje aplikacija treće strane, zaštita samo nad bitnim i osjetljivim podacima.
Secure Service Proxy	Integracija novih i starih sigurnosnih protokola.	Vanjska autorizacija i autentikacija presretanjem zahtjeva.	Ne mijenja se postojeća aplikacija, krajnje točke web servisa su zaštićene.
Secure Session Manager	Sigurnost sesije	Centralizacija sigurnog rukovanja informacijama tijekom sesije.	Centralizira rukovanje informacijama.
Intercepting Web Agent	Skupa i kompleksna prilagodba sigurnosti u postojećoj web aplikaciji.	Presretanje zahtjeva za provjeru autentičnosti i autorizacije prije nego dođu do aplikacije.	Razdvajanje autentikacije i autorizacije od postojeće aplikacije, ne mijenja se postojeća aplikacije.

5.2 Poslovna razina

Poslovna razina sadrži sve one komponente koje služe za implementaciju poslovne logike u web aplikacijama. Ovisno o tome da li je ova razina povezana s web razinom koriste se određeni uzorci. (Flylib, Business Tier Security Patterns, 2017). Ovim uzorcima implementira se praćenje zahtjeva, validacija i verifikacija, nadzor te zaštita podataka koji se šalju kroz aplikaciju.

Tablica 3 Prikaz uzoraka koji se primjenjuju na razini poslovnog sloja

Naziv uzorka	Problem	Rješenje	Prednosti
Audit Interceptor	Presretanje zahtjeva/odgovora na poslovnoj razini.	Centralizacija funkcionalnosti praćenja i definiranje događaja koji će se pratiti.	Centralizirano praćenje zahtjeva/odgovora, odvojeno praćenje od aplikacije, odvija se prije i poslije procesa.
Container Managed Security	Potrebne validacija i verifikacija bez izmišljanja vlastitog sigurnosnog protokola.	Definiranje uloga na razini aplikacije za vrijeme razvoja i njihovo mapiranje za vrijeme ili nakon razvoja.	Jednostavan, deklarativni sigurnosni model temeljen na statičkim mapiranjima, nema zaobilaženja sigurnosnih zahtjeva.
Dynamic Service Management	Upravljanje i nadzor nad komponentama aplikacije.	Omogućiti fino-zrnato (eng. fine-grained) praćenje poslovnih objekata za vrijeme izvođenja (u Javi se radi pomoći JMX-a).	Pružanje sigurnog nadzora u realnom vremenu nad podacima i sprječavanje napada.
Obfuscated Transfer Object	Zaštita kritičnih podataka koji se šalju u aplikaciji preko razina.	Određivanje elemenata podataka koji će biti zaštićeni i implementacija objekta koji će ih zaštiti.	Objekt prijenosa je zaslužan za zaštitu podataka, određeni elementi podataka mogu biti zaštićeni.
Policy Delegate	Zaštita korisnika i administracija pravila njihovih zahtjevima.	Potrebno je implementirati posredovanje zahtjeva između korisnika i okvira za sigurnost.	Smanjen broj kompleksnih sučelja za sigurnost, upravljanje životnim ciklusom korisnikova sigurnosnog konteksta.
Secure Service Facade	Više pristupnih točaka više mogućnosti za sigurnosne propuste.	Sigurni prolazi kroz koji se zahtjevi validiraju i kojim se centralizira kompleksna	Razdvajanje sigurnosnih i poslovnih komponenti, nametnuta sigurnosna pravila.

Naziv uzorka	Problem	Rješenje	Prednosti
		interakcija poslovnih komponenti.	
Secure Session Object	Različite komponente imaju različite kontrole pristupa i provjere autentikacije.	Apstrahiranje učahurenih akreditiva koji se mogu slati i preko granica neke razine.	Struktura podataka koja obuhvaća autentikaciju i autorizaciju, postoji token, sigurni prijenos sigurnosnog konteksta kroz VM i adresne prostore.

5.3 Razina web servisa

Kada se aplikacija razvija s web servisima bitno se fokusirati na sigurnosne rizike do kojih može doći. Tako je potrebno definirati zaštitne mehanizme i protumjere prije nego što se započne s implementacijom aplikacije (Flylib, Web Services Security Patterns, 2017). Uzorcima se implementira sigurna komunikacija i nadzor nad svime što se šalje i prima putem web servisa.

Tablica 4 Prikaz uzoraka koji se primjenjuju na razini sloja web servisa

Naziv uzorka	Problem	Rješenje	Prednosti
Message Inspector	Kvaliteta mehanizama za sigurnost na razini poruka koji se primjenjuju na Web servise XML-a,	Modularno rješenje za integraciju servisnih komponenti koje obrađuju prije i poslije procesa ulazne i izlazne SOAP ili XML poruke.	Zajedničko rješenje za sve zadatke sigurnosti poruka, ograničenje poruka ovisno o sigurnosnom tokenu, praćenje i validacija kriptirane komunikacije, …
Message Interceptor Gateway	XML poruke koje imaju direktan pristup web servisima stvaraju mogućnost napada.	Kontroler kao infrastruktura koja osigurava jednu ulaznu točku koja obuhvaća pristup svim web servisima.	Blokiranje direktnog pristupa web servisima, jedna pristupna točka, verificiranje poruka, …
Secure Message Router	Sigurna komunikacija s ostalim krajnjim točkama.	Spajanje svih pristupa krajnjim točkama aplikacije koje sudjeluju u transakcijama web servisa.	Sigurnost elemenata i poruka koje se prenose, samo određeni dio poruke je otkriven primatelju, filtriranje dolazećih poruka, …

5.4 Razina identiteta

Kod upravljanja identitetima fokus je na podacima za autentikaciju i identifikaciju korisnika. Uzorci sigurnosti ove razine pružaju zajednički okvir dizajna, unificirani SSO (Single-Sign On) i mehanizam globalne odjave na raznim sustavima (Flylib, Identity Management Security Patterns, 2017). Time se smanjuje kompleksnost implementacije.

Tablica 5 Prikaz uzoraka koji se primjenjuju na razini sloja identiteta

Naziv uzorka	Problem	Rješenje	Prednosti
Assertion Builder	Pristup skupljanju sigurnosnih informacija tijekom prijave subjekta.	Apstrakcija slične logike za kontrolu kreiranja SAML tvrdnji.	Zajednička logika za slične izjave sigurnosti, fleksibilnost podrške zahtjeva klijenata.
Credential Tokenizer	Korištenje sigurnosnog tokena.	Enkapsulacija različitih tipova akreditacije u obliku tokena koji mogu koristiti više pružatelja usluga sigurnosti.	Komponenta koja se više puta može iskoristiti da se izdvoji procesna logika za kreiranje tokena, zaštita od složenog dizajna i implementacije.
Single Sign-on (SSO) Delegator	Kompleksna interakcija s različitim uslugama, komponentama, …	Odvajanje direktne interakcije klijenta sa sučeljem usluge za upravljanje identitetom.	Smanjena povezanost klijenta i usluge za upravljanje identitetom, nema reinženjeringa arhitekture, prikazivanje mrežnih iznimki u obliku aplikacijskih ili klijentskih.
Password Synchronizer	Sinkronizacija lozinki korisnika.	Centralizacija upravljanja sinkronizacije akreditacija putem sučelja u različitim sustavima.	Sučelje koje može upravljati različitim lozinkama, standardizacija procesne kontrole povratnih i kodova pogreške.

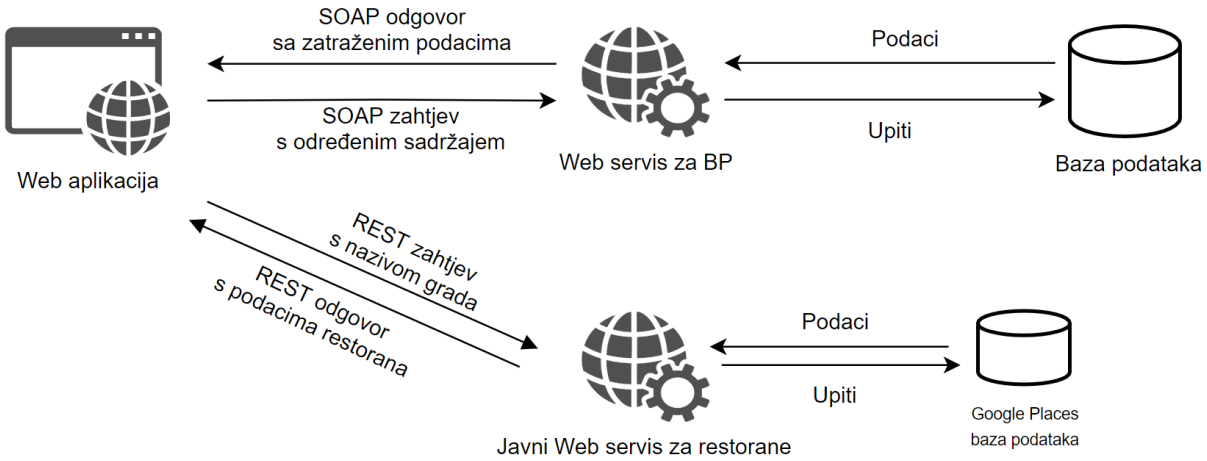
6. Primjena uzoraka u web aplikaciji

Odabranim skupom uzoraka iz prethodnih poglavlja biti će prikazana njihova primjena u razvoju web aplikacije. Sama domena web aplikacije ne utječe na primjenu, ali ju je ipak potrebno ukratko objasniti kako bi se bolje razumjela implementacija. Aplikacija se bavi rezervacijom mjesta u restoranima. Postoje obični korisnici koji mogu pregledati restorane, te rezervirati mjesto na željeni datum i vrijeme. Zatim imaju pregled rezervacija koje su aktivne i rezervacija koje su prošle. Nadalje, osim običnog korisnika postoji korisnik administrator koji upravlja restoranima, mogućnost dodavanja i brisanja. On također ima funkcionalnosti pregleda korisnika i zapisa aktivnosti aplikacije.

Implementacija web aplikacije napravljena je u razvojnom okruženju Visual Studio u programskom jeziku C#. Korišten je programski okvir .NET Core koji predstavlja najnoviju verziju ASP.NET programskog okvira za razvoj web aplikacija. U .NET Core-u se na jednostavan način može lako konfigurirati i upravljati sigurnost web aplikacije. Tako na primjer postoje značajke za autentikaciju, autorizaciju, upravljanje CORS[5]-om, itd. Također pruža razne alate i biblioteke koji podržavaju implementaciju sigurnosti.

OPIS SUSTAVA

Sustav sadrži dvije glavne komponente, jedna je aplikacija, a druga SOAP web servis koji komunicira s bazom podataka te vraća zatražene podatke natrag aplikaciji. Treća komponenta izvan implementiranog sustava je Google Places API REST servis koji sadrži podatke o restoranima.



Slika 16 Arhitektura implementiranog sustava

[5] Cross-Origin Resource Sharnig – dodatni HTTP header s kojim se dopušta pristup aplikaciji s drugog servera

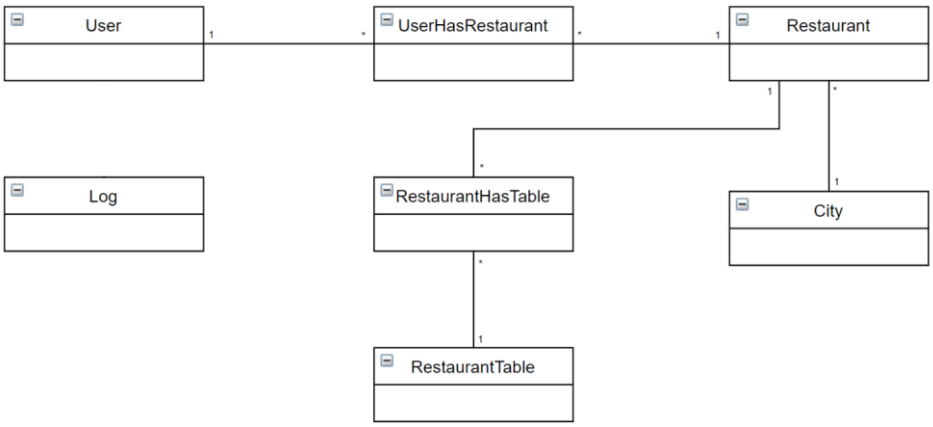
Za izgradnju prve komponente, aplikacije, korišten je MVC uzorak koji je jedan od arhitekturalnih uzoraka dizajna. Korišten je iz razloga što razdvaja aplikaciju na tri glavne komponente: modela, pogleda i kontrolera, te na taj način pomaže u postizanju principa „razdvajanje briga“. Također daje jednu lijepu strukturu aplikacije. Naravno, u jednom od prethodnih poglavlja spomenut je i uzorak Secure MVC, stoga je korišten i zbog toga. Tom uzorku se pridodaju i neki drugi uzorci sigurnosti koji su implementirani kroz aplikaciju, kao npr. Authenticator, Secure Logger ili Reference Monitor.

.NET Core MVC uzorak radi na način da kada korisnik pošalje zahtjev, on bude preusmjeren kontroleru koji je odgovoran za rad s modelom, te izvodi akcije ovisno o zahtjevima, dohvaća i vraća rezultate. Taj isti kontroler odabire pogled preko kojeg će korisniku prikazati rezultate i prikazuje ih putem modela koji ti podaci zahtijevaju.

Tako se kontroleri u MVC-u ponašaju kao REST API koji primaju HTTP zahtjeve i vraćaju HTTP odgovore, te ih je isto potrebno zaštititi. Također sadrže sve HTTP metode (GET, POST, PUT DELETE) preko kojih se mogu slati zahtjevi i vraćati odgovori. Potrebna je samo anotacija iznad odabrane metode u kontroleru koja će označiti za koju HTTP metodu je odgovoran, npr. [HttpGet] ili [HttpPost]. Većina implementiranih uzoraka veže se baš uz njih.

Drugu komponentu predstavlja SOAP web servis koji služi aplikaciji kako bih mogla komunicirati s bazom. Implementiran je iz razloga da se pokaže sigurnost SOAP servisa. Za korištenje tih servisa u sustav je dodan paket SoapCore koji je napravljen baš za potrebe implementacije SOAP servisa s .NET Core web aplikacijama. Također postoje i razni drugi paketi koji se mogu koristiti. Ovdje je korišten uzorak WS-Security odnosno implementiran je siguran način prijenosa zahtjeva putem certifikata koji koriste i strana koja šalje zahtjeve i strana koja ih prima i obrađuje.

Za bolje razumijevanje podataka s kojima se radi u aplikaciji prikazan je ERA model baze podataka na slici 17. Korisnici i restorani su najbitnije stavke modela. Može se vidjeti kako su međusobno povezani pomoćnom tablicom *UserHasRestaurant* gdje se bilježe zapisi o rezervacijama restorana određenog korisnika. Uz to kao atribute još se dodaju vrijeme rezervacije i stol koji je rezerviran. Svaki restoran ima grad kojem pripada jer se preko gradova pretražuju i dodaju restorani. Također, se mogu definirati i stolovi za restorane i dodavati mjesta, što je uloga korisnika koji ima administratorska prava. Tablica *Table* sadrži fiksne vrijednosti stolova koje mogu biti povezane na restorane. Na kraju, zasebna tablica *Log* služi za zapise aktivnosti korisnika koji izvršavaju akcije u aplikaciji.



Slika 17 ERA model baze podataka

6.1 Opis uzoraka

Većina uzoraka koji su korišteni za implementaciju nalaze se u knjizi (Fernandez-Buglioni, 2013), dok je mali dio uzoraka vezan uz Java EE uzorke za sigurnost. Razlog je prvenstveno taj što je aplikacija implementirana u programskom jeziku C#. Stoga su ti uzorci prilagođeni tom razvoju, a uzorci iz knjige su dosta apstraktni pa se mogu prilagoditi svakom programskom jeziku i OOP-u[6].

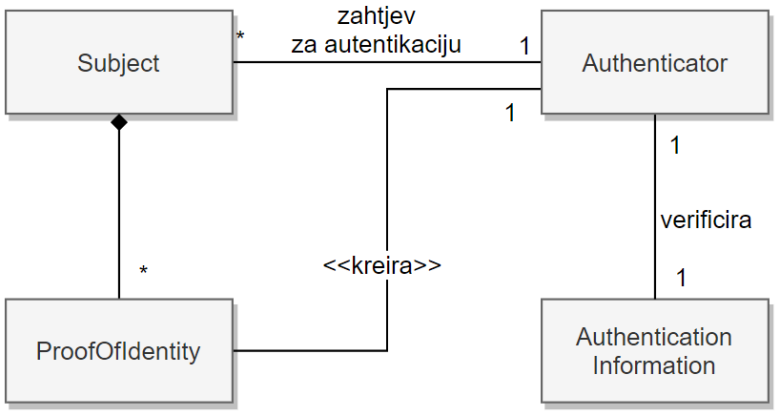
Authenticator

S obzirom da je domena aplikacije rezervacija restorana, te pregled trenutnih i povijest rezervacija, moguće je imati više korisnika iste aplikacije. Stoga je potrebno napraviti prijavu u sustav kako bi svaki korisnik mogao vidjeti svoje podatke. Potrebno je osigurati se kako korisnici ne bi vidjeli podatke od drugih korisnika istog sustava, što je bitno u sustavima s velikim brojem korisnika i velikom količinom podataka koje oni sadrže. Isto tako je moguće da se neka osoba predstavlja za drugu. Iz tih razloga treba napraviti verifikaciju korisnika koji se pokušavaju prijaviti u sustav s korisničkim imenom i lozinkom. Uzorak Authenticator rješava taj problem na način da provjerava korisnikov unos i uspoređuje ga s podacima u bazi podataka registriranog korisnika. Rješenje daje fleksibilnost, bolju performansu i centraliziranu autentikaciju sustava.

Struktura

Strukturu uzorka prikazuje slika 18, a sudjeluju Subject, Authenticator, AuthenticationInformation i ProofOfIdentity.

[6] Objektno orijentirano programiranje



Slika 18 UML dijagram klase za uzorak Authenticator.
Vlastita izrada prema (Fernandez-Buglioni, 2013), str. 63.

Subject predstavlja korisnika koji se pokušava prijaviti za korištenje aplikacije sa svojim korisničkim imenom i lozinkom. Zahtjev dolazi glavnoj klasi ovog uzorka, a to je *Authenticator* koja preuzima podatke te ih preko SOAP web servisa provjerava u bazi podataka, odnosno verificira informacije autentikacije. Kada je korisnik verificiran kreira se dokaz o provjeri identiteta, u ovom slučaju JWT token koji će biti objašnjen u nastavku. Ako korisnik nije verificiran vraća se natrag odgovor kako su uneseni krivi podaci za prijavu, te korisnik ostaje na login stranici.

Implementacija

Implementacijom ovog uzorka osigurava se autentikacija svim pristupnim točkama aplikacije. Glavnu ulogu ima kontroler AuthenticatiorController koji sadrži metodu *Authenticate()*. Kontroler se poziva nakon što se obavi validacija unosa koja je implementirana uzorkom Intercepting Validator koji će biti objašnjen kasnije u ovom radu.

```
public IActionResult Authenticate()
{
    string loginData = TempData["loginModel"].ToString();
    LoginModel loginModel = JsonConvert.DeserializeObject<LoginModel>(loginData);

    SOAPClient soapClient = new SOAPClient();

    User user = soapClient.AuthenticateUser(loginModel.Username, loginModel.Password);

    if (user == null)
    {
        ModelState.AddModelError("ErrorMessage", "Krivo uneseni podaci!");
        return View("Login",loginModel);
    }

    JWTCreator jwt = new JWTCreator(HttpContext);
    jwt.CreateJWT(user.Id.ToString(), user.Username, user.Email, user.Admin);

    if (user.Admin)
        return RedirectToAction("RestaurantManaging", "RestaurantManaging");
     else
        return RedirectToAction("Restaurants", "Restaurants");
}
```

Varijabla *loginData* služi za spremanje unesenih podataka za prijavu proslijeđenih iz validatora. Oni se zatim deserijaliziraju i spremaju u objekt modela prijave. Zatim slijedi validacija tih podataka pozivanjem SOAP servisa koji provjerava postoji li u bazi registrirani korisnik s tim podacima. Prosljeđuju se korisničko ime i lozinka. Ako podaci ne postoje, objekt *user* je null, stoga autentikacija nije prošla, te se vraća povratna poruka o grešci.

Dok, s druge strane, ako korisnik postoji i autentikacija je uspješna slijedi kreiranje JWT tokena koji predstavlja dokaz o identitetu prijavljenog korisnika. On se svaki put, kada korisnik pristupa nekom dijelu aplikacije, prenosi u zaglavlju zahtjeva kao dokaz da je prijavljen. To je definirano u Startup klasi aplikacije.

```
app.Use(async (context, next) =>
{
    var JWToken = context.Session.GetString("JWToken");
    if (!string.IsNullOrEmpty(JWToken))
    {
       context.Request.Headers.Add("Authorization", "Bearer " + JWToken);
    }
    await next();
});
```

JWT je standard za kreiranje tokena baziranih na JSON-u. Pomoću njega se na siguran način mogu dijeliti informacije među dvije strane. Može biti potpisan koristeći „tajnu“ (npr. HMAC) ili javnim/privatnim ključem (npr. RSA). (AuthO, n.d.) Može se još reći kako je on dokaz o tvrdnjama koje se nalaze u njemu odnosno čuva njihov integritet. Uglavnom se koristi za autorizaciju ili razmjenu informacija. Njegovo kreiranje biti će detaljnije objašnjeno u uzorku sigurnosti Credential Tokenizer kao i od čega se sastoji.

Također je bitno napomenuti kako je u Startup klasi aplikacije potrebno aktivirati odnosno dodati uslugu za autentikaciju uz koju se definiraju i opcije za token, što je prikazano sljedećim kodom.

```
string secret = Configuration.GetValue<string>("JwtSecret:Secret");
services.AddAuthentication(x =>
{
         x.DefaultAuthenticateScheme = JwtBearerDefaults.AuthenticationScheme;
         x.DefaultChallengeScheme = JwtBearerDefaults.AuthenticationScheme;
         })
         .AddJwtBearer(x =>
         {
                x.RequireHttpsMetadata = false;
                x.SaveToken = true;
                x.TokenValidationParameters = new TokenValidationParameters
                {
                    ValidateIssuerSigningKey = true,
                     IssuerSigningKey = new
                     SymmetricSecurityKey(Encoding.ASCII.GetBytes(secret)),
                    ValidateIssuer = false,
                    ValidateAudience = false
                };
            });
```

Uz sve prethodno, postoji još jedna važna stavka, a to je spremanje korisničkih lozinki u bazu podataka na siguran način. Kada bi osoba koja izvršava napad došla do baze podataka i svih informacija koje se nalaze u njoj s lakoćom bi mogla preuzeti podatke nekog korisnika i predstaviti se kao on. Stoga je na neki način bitno zaštiti podatke kako bi se smanjila ta opasnost. Iz tog razloga se prilikom registracije za lozinku koju korisnik upiše generira sažetak (eng. hash), te se on sprema u bazu kako ne bi nikome lozinka bila čitljiva i na taj način samo korisnik zna s kojim se podacima može ulogirati. U aplikaciji je to implementirano kod spremanja korisničkih podataka u bazu kod SOAP web servisa, te se sažetak koristi i prilikom autentikacije gdje se radi sažetak upisane lozinke kod prijave te se uspoređuje s onim u bazi podataka. Ako su sažetci jednaki, znači da je upisana lozinka ispravna i korisnik može biti autenticiran. Klasa *HashAndSalt* prikazuje pravljenje sažetka lozinke uz izgeneriranu sol.

```
public class HashAndSalt
{
   public static string ComputeSha256(string password, string salt)
   {
      MD5CryptoServiceProvider hash = new MD5CryptoServiceProvider();

      byte[] hashBytes = hash.ComputeHash(Encoding.ASCII.GetBytes(password +salt));

      string hashValue = Convert.ToBase64String(hashBytes);
      return hashValue;
   }
```

```
    public static string GenerateSalt()
    {
        using (var rand = new RNGCryptoServiceProvider())
        {
            var randomNumber = new byte[32];
            rand.GetBytes(randomNumber);
            return BitConverter.ToString(randomNumber);
        }
    }
}
```

Metode se pozivaju kod registracije i prijave korisnika. Generiranje soli izvršava se samo kada se korisnik registrira kako bi se upisala zajedno s njegovim podacima u bazu i kasnije poslužila za autentikaciju prilikom prijave. Korištenjem jedinstvene soli za svakog korisnika zapisi u bazi se razlikuju, iako postoji mogućnost da neki korisnici imaju istu lozinku.

```
public void RegisterUser(string username, string password, string email)
{

    DataBase db = new DataBase();
    db.Open();

    string salt = HashAndSalt.GenerateSalt();

    string hashpassword = HashAndSalt.ComputeSha256(password, salt);
    db.ExecuteQueries($"INSERT INTO User (username,password, salt,email,admin)
        VALUES ('{username}','{hashpassword}','{salt}','{email}',0)");

    db.Close();
}
```

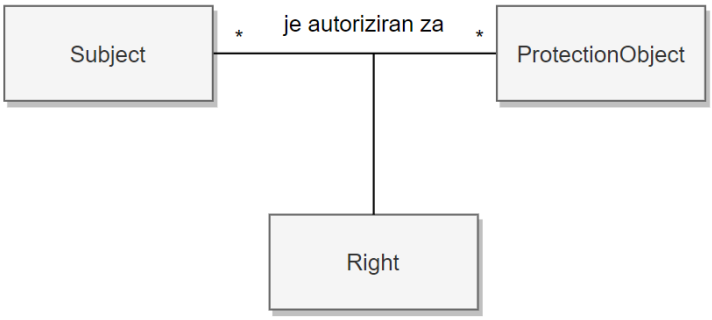
```
public UserModel Authenticate(string username, string password)
{
    DataBase db = new DataBase();
    db.Open();

    string salt = GetUserSalt(username);

    string hashpassword = HashAndSalt.ComputeSha256(password, salt);
    MySqlDataReader result = db.DataReader($"SELECT UserId,username,email,admin FROM
        User " +
        $"WHERE username = '{username}' AND password = '{hashpassword}'");

    UserModel user = null;
    if (result.Read())
    {
        user = new UserModel
        {
            Id = int.Parse(result.GetValue(0).ToString()),
            Username = result.GetValue(1).ToString(),
            Email = result.GetValue(2).ToString(),
            Admin = int.Parse(result.GetValue(3).ToString()) == 0 ? false : true
        };
    }

    db.Close();
    return user;
}
```

Authorization

Autorizacija u aplikaciji je potrebna iz razloga što se istom koriste obični korisnici, oni koji imaju pregled restorana, rezervacija i povijesti restorana, te administratori odnosno oni korisnici koji mogu upravljati aplikacijom kao npr. dodavanje i brisanje restorana. Kako obični korisnici ne bi mogli brisati restorane ili neke druge podatke iz baze podataka potrebno je dodijeliti određene uloge pri autentikaciji korisnika. Potrebna je kontrola nad resursima aplikacije kako ne bi došlo do gubitka ili mijenjanja podataka. Primjenom ovog uzorka se tako odvaja pogled običnog korisnika i administratora u samoj aplikaciji, te se zabranjuje neovlašteni pristup.

Struktura

Struktura uzorka se sastoji od tri komponente: *Subject*, *Right* i *ProtectionObject*.



Slika 19 UML dijagram klasa uzorka Authorization, Vlastita izrada prema (Fernandez-Buglioni, 2013), str. 87.

Subjekt predstavlja korisnika koji pokušava pristupiti određenoj stranici odnosno zaštićenom objektu. U aplikaciji su definirana prava koja ima korisnik s određenom ulogom, odnosno prava za običnog korisnika i prava za administratora. Tako vezu između subjekta i zaštićenog objekta definira autorizacija. Klasa *Right* definira pristupna prava koja ima korisnik s određenom ulogom gdje se ona i provjeravaju.

Implementacija

Ovaj uzorak implementiran je u .NET Core-u pomoću definiranja polica. Uz taj .NET Core sadrži i mnoge druge autorizacije poput onih baziranih samo na ulozi korisnika, na tvrdnjama, pogledu i slično. Ova je izabrana iz razloga što je najbolje odgovarala primjeni uzorka Authorization.

Kako bi se koristila autorizacija bazirana na policama potrebno je i kreirati jednu, koja u ovom slučaju ima samo jedan uvjet, da korisnik bude prijavljen u sustav.

```
public class LogedInUserRequirement : IAuthorizationRequirement
{
   public bool LogedIn { get; set; }
   public LogedInUserRequirement(bool logedIn)
   {
      LogedIn = logedIn;
   }
}
```

Da bi se mogla koristiti polica u .NET Core aplikaciji bitno je da svaki kontroler koji se želi zaštiti od neželjenih posjeta ima na početku klase anotaciju [Authorize(Policy = "LogedInUser")]. Također je potrebno u Startup klasi aplikacije registrirati uslugu i definirati uvjet za policu što prikazuje sljedeći kod.

```
services.AddAuthorization(options =>
{
    options.AddPolicy("LogedInUser",
                policy => policy.Requirements.Add(new LogedInUserRequirement(true)));
});
```

Kada je definirana i registrirana polica koja nam služi kao pomoć kod autorizacije potrebno je implementirati ostatak uzorka. U UML dijagramu je prikazana klasa *ProtectionObject* koju u ovom slučaju predstavlja klasa *Permission* s dvije definirane liste. Jedna lista sadrži objekte koji su dostupni samo administratorima, dok druga sadrži one kojima pristup imaju obični korisnici.

```
public class Permission
{
        private List<string> Admin = new List<string> { "RestaurantManaging",
       "UserManaging", "Log"};
        private List<string> User = new List<string> { "CurrentReservations" ,
       "HistoryReservations", "Restaurants" };

       ...
}
```

Glavna klasa ovog uzorka je *RightHandler* koja je definirana kao Singleton u Startup klasi aplikacije i nasljeđuje sučelje *IAuthorizationHandler*. Služi za provjeru uloge prijavljenog korisnika i tako mu definira određena prava putem klase *Permission* kako bi mogao pristupiti samo onim objektima koji su tu i definirani. Kada korisnik pokuša pristupiti objektu za kojeg nije autoriziran aplikacija ga preusmjerava na stranicu o pogrešci 403 Unauthorized.

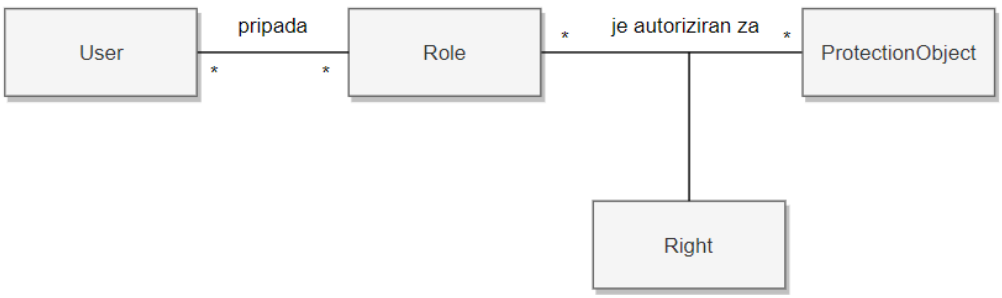
```
public class RightHandler : AuthorizationHandler<LogedInUserRequirement>
{
    IHttpContextAccessor _httpContextAccessor = null;

    public RightHandler(IHttpContextAccessor httpContextAccessor)
    {
        _httpContextAccessor = httpContextAccessor;
    }
    protected override Task HandleRequirementAsync(AuthorizationHandlerContext
    context, LogedInUserRequirement requirement)
    {
        Permission permission = new Permission();

        string requestedPath =
       _httpContextAccessor.HttpContext.Request.Path.ToString();

            if (context.User.IsInRole("Admin") &&
            permission.HasAdminRights(requestedPath))
                context.Succeed(requirement);
             if (context.User.IsInRole("User") &&
             permission.HasUserRights(requestedPath))
                context.Succeed(requirement);

            return Task.CompletedTask;
        }
    }
```

Role-Based Access Control

Uz prethodni uzorak implementiran je i uzorak Role-Based Access Control kojim se definira uloga korisnika koji pristupa određenom objektu. Korisnicima je potrebno definirati uloge kako bi aplikacija znala kakva prava korisnik ima dok koristi aplikaciju. U ovom slučaju uloga svakog korisnika je definirana u bazi podataka tablice User koja sadrži polje Admin te je tipa bool. Ako je korisnik administrator polje je postavljeno na true, a inače je false. Svakom korisniku se pri registraciji dodjeljuje uloga običnog korisnika.

Struktura

Struktura uzorka je slična prethodnom, a razlika je što je u ovom dodana klasa *Role*, što se vidi na slici 20.



Slika 20 UML dijagram klasa uzorka Role-Based Access Control,
Vlastita izrada prema (Fernandez-Buglioni, 2013), str. 94.

Korisnicima je potrebno definirati nekakve uloge kako bi se lakše moglo dodijeliti pravo nad određenim objektom kojem može pristupiti.

Implementacija

Obzirom da je dio ovog uzorka već implementiran prethodnim, te je dodana samo jedna klasa, implementacija nije kompleksna. Stoga je dodana klasa *Role* koja je tipa enum koji predstavlja tip podataka vrsta podataka. U njoj su definirane dvije uloge.

```
public enum Role
{
    Admin = 1,
    User = 0
}
```

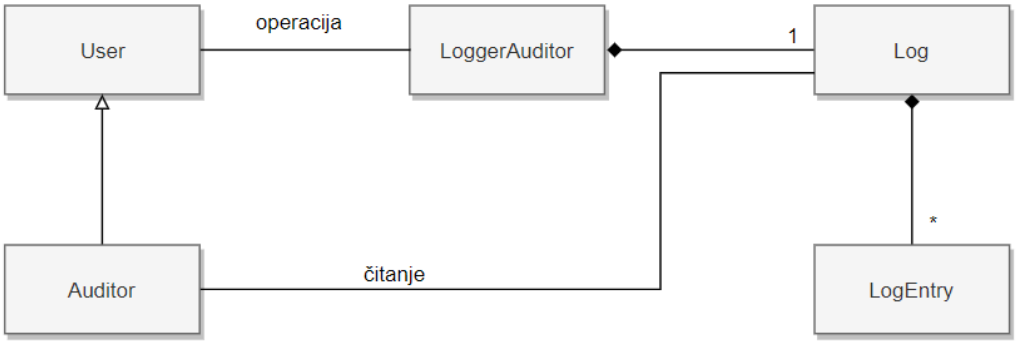
Uloga se dodjeljuje prilikom autentikacije kod kreiranja JWT tokena kada se definiraju zahtjevi (eng. claims) korisnika, te provjerava prilikom autorizacije.

Security Logger and Auditor

Ponekad se dogodi da neki korisnik pokuša napraviti nešto za što nije autoriziran ili se tijekom korištenja aplikacije dogodi neka greška. Ovim uzorkom se implementira bilježenje i praćenje svih mogućih zahtjeva koje korisnici izvršavaju. Uvijek je potrebno moći vidjeti tko je slao zahtjev, za što i kada, te da li je zahtjev prošao ili nije. Osobe koje imaju na to prava su u mogućnosti vidjeti zapise koji su u ovom slučaju zapisani u bazu podataka. Uzorak je prvenstveno dobar za sigurnost jer se može pratiti aktivnost svakog korisnika te uočiti u zapisima ako je netko možda pokušao izvršiti nešto za što nije zadužen.

Struktura

U strukturi uzorka sudjeluju: User, LoggerAuditor, Log, LogEntry i Auditor. U izvornom dijagramu postoji još i klasa SecAdmin koja je dio User klase, ali u ovom slučaju ona nije implementirana, jer nije toliko bitna za implementaciju uzorka u ovom dijelu, a služi kako bi se zapisi mogli aktivirati odnosno deaktivirati.



Slika 21 UML dijagram za uzorak Secure Logger and Auditor, Vlastita izrada prema (Fernandez-Buglioni, 2013), str. 143.

Svaki put kada korisnik pristupi nekom objektu akcija se zabilježi u bazu podataka. Informacije koje se bilježe su id korisnika, tip pristupa, objekt kojem je pristupano i vrijeme kada se to dogodilo i ostale informacije po izboru. Svaka operacija se zapisuje putem klase *LoggerAuditor* koji uvijek sadrži po jedan zapis. Tijekom zapisa stvara se objekt klase *LogEntry* čiji se podaci zapisuju u bazu podataka.

Implementacija

Kako bi se kreirali zapisi potrebno je imati model preko kojeg će se kreirati isti. U ovom slučaju to je klasa *LogEntry*, a implementirana je u aplikaciji kao klasa *LogModel* koja ujedno služi i za prikaz zapisa na pogledu zapisa. S njom se definiraju atributi koje će zapisi imati.

```
public class LogModel
{
    public int LogId { get; set; }
    public DateTime TimeStampUtc { get; set; }
    public string HostName { get; set; }
    public int UserId { get; set; }
    public string Action { get; set; }
    public string Path { get; set; }
    public bool Success { get; set; }
    public string FailureReason { get; set; }
}
```

Nadalje, klasa *Log* služi samo za zapisivanje i čitanje zapisa koji se nalaze u bazi, te kreira objekt klase *LogModel*. Poziva SOAP klijenta kako bi proslijedila podatke ulaznog zapisa da se spreme u bazu podataka.

```
public void AddEntry(int userId, string action, string path, DateTime timeStampUtc,
bool success, string failureReason)
{
    LogModel log = new LogModel
    {
        UserId = userId,
        Action = action,
        TimeStampUtc = timeStampUtc,
        Path = path,
        Success = success,
        FailureReason = failureReason
    };
     SOAPClient dBClient = new SOAPClient();
     dBClient.LogAction(log);
}

public List<LogModel> ReadEntry()
{
   SOAPClient dbClient = new SOAPClient();
   return dbClient.GetLogs();
}
```

Glavna klasa je *LoggerAuditor* koja je napravljena kao filter jer nasljeđuje sučelje IAsyncActionFilter. U Startup klasi aplikacije definirana kao filter za područje (eng. scope) što označava da je njezin objekt isti u isto zahtjevu, ali različit u drugima. Iz tog razloga se mogu raditi operacije nad zahtjevima prije izvršavanja kontrolera i nakon izvršavanja. Također je još bitno napomenuti da ako se zapisivanje akcija želi provoditi nad kontrolerom potrebno je staviti anotaciju, [ServiceFilter(typeof(LoggerAuditor))], koja će prvo provjeriti taj filter i po potrebi ga izvršiti.

```
public class LoggerAuditor : IAsyncActionFilter
{
   public async Task OnActionExecutionAsync(ActionExecutingContext context,
ActionExecutionDelegate next)
   {
      var result = await next();

      var request = result.HttpContext.Request;
      var response = result.HttpContext.Response;

      ClaimsPrincipal user = request.HttpContext.User;
      string path = request.Path.ToString();
      string method = request.Method;

      int statusCode = response.StatusCode;

      if (ApplyFilter(user, method, statusCode))
      {
          Log log = new Log();

          var arguments = context.ActionArguments;
          string argument = "";

          foreach (var item in arguments)
          {
             argument = item.Value.ToString();
              break;
          }

          var userId = int.Parse(user.FindFirst(ClaimTypes.NameIdentifier).Value);
          bool succes = (statusCode == 200 ? true : false);
          string status = (statusCode == 200 ? "/" : "ERROR " +
          statusCode.ToString());
          log.AddEntry(userId, method, path + argument, DateTime.Now, succes,
status);
      }
   }

   private bool ApplyFilter(ClaimsPrincipal user, string method, int statusCode)
   {
       if (user.FindFirst(ClaimTypes.NameIdentifier) == null) return false;
       if (statusCode >= 300 && statusCode <= 310) return false;
       if (user.IsInRole("Admin") && method != "DELETE") return false;
       return true;
   }
}
```

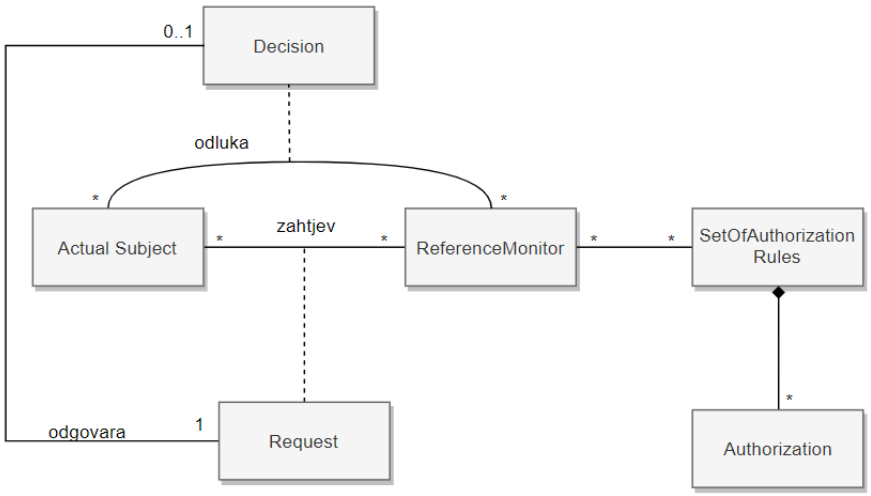
Također je bitno napomenuti kako se zapisi mogu i filtrirati, što je implementirano metodom *ApplyFilter()*. U ovom slučaju zapisi se ne spremaju ako korisnik nije prijavljen u sustav ili ako je http status kod neki od 3xx, jer su to preusmjeravanja. Isto tako zapisi se ne spremaju ako je korisnik administrator, a ne izvršava akciju koja ima veze s brisanjem podataka.

Reified Reference Monitor

Uzorak se nadovezuje na uzorak autorizacije, odnosno s definiranom policom za autorizaciju implementiran je i on. Opisuje kako „prisiliti“ autorizaciju kada korisnik želi pristupiti određenom objektu što se i implementira putem skupa polica. Kako je razvijena aplikacija vrlo jednostavna dovoljan je i primjer politike koja je definirana za autorizaciju, iako ih može biti i više.

Struktura

U strukturi uzorka sudjeluju: *ReferenceMonitor, ActualSubject, Request, Decision, SetOfuthorizationRules* i *Authorization*. Dio ovog uzorka je već implementirana autorizacija koja je prethodno opisana. Skup pravila u ovom slučaju predstavlja skup polica definiranih u aplikaciji koje su zapravo i sama pravila autorizacije.



Slika 22 UML dijagram klasa za uzorak Reified Reference Monitor, Vlastita izrada prema (Fernandez-Buglioni, 2013), str. 123.

Implementacija

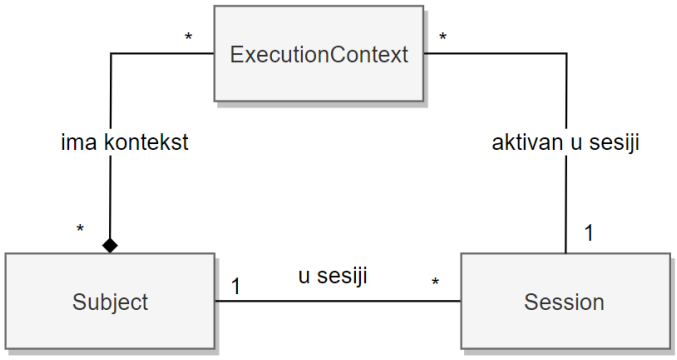
Uzorak je implementiran na vrlo jednostavan način, te u slučaju ove aplikacije povezan s autorizacijom koja se izvršava nad zahtjevima korisnika. Klasu *ReferenceMonitor* u ovom slučaju predstavlja Startup klasa koja sadrži metode u kojima se definiraju usluge i slične stvari te se pozivaju tijekom izvršavanja aplikacije i slanja zahtjeva. Dio koji je bitan za ovaj uzorak je već prije spomenut, a to je definiranje politike u Startup klasi, koja predstavlja pravilo autorizacije. Tu se može dodati više pravila ovisno o zahtjevima aplikacije. Odluka se donosi na temelju toga da li je pravilo autorizacije zadovoljeno ili ne, odnosno u ovom slučaju korisnik može pristupiti određenom objektu ako je validacija uspješna i naravno s određenom ulogom.

Controlled Access Session

Kontrola pristupa može se izvršiti na razne načine, te je potrebna kako bi korisnik mogao pristupiti bez da se svaki put zatraže autorizacija i autentikacija. Korisnici aplikacije imaju određena prava na akcije koje su im dodijeljene, a bitno je aktivirati samo ona prava koja

su bitna za tu sesiju. Potrebno je stoga na neki način ograničiti prava ovisno o potrebama aplikacije ili zadacima koje korisnik treba obaviti.

Struktura



Slika 23 UML dijagram klasa za uzorak Controller Access Session, Vlastita izrada prema (Fernandez-Buglioni, 2013), str. 127.

Struktura uzorka je vrlo jednostavna, a sudjeluju *Subject, ExecutionContext* i *Session.* Kada korisnik pošalje aplikaciji zahtjev za prijavu, prvo se aktivira autentikacija, te ako je ona uspješna stvara se sesija za aktivnog korisnika. Dok klasa *ExecutionContext* sadrži nekakav skup prava koje korisnik može koristiti tijekom korištenja aplikacije.

Implementacija

Obzirom da se koristi sesija, potrebno ju je i aktivirati odnosno reći aplikaciji kako će se ona koristiti. Dvije su bitne stavke aplikacije koje treba definirati, a nalaze se u Startup klasi. Jedna je dodavanje sesije u usluge, a druga je korištenje sesije.

```
public class Startup
{
       public void ConfigureServices(IServiceCollection services)
       {
            ...
            services.AddSession();
            ...
       }

       public void Configure(IApplicationBuilder app, IHostingEnvironment env,
       IConfiguration config)
       {
              app.UseSession();
       }
}
```

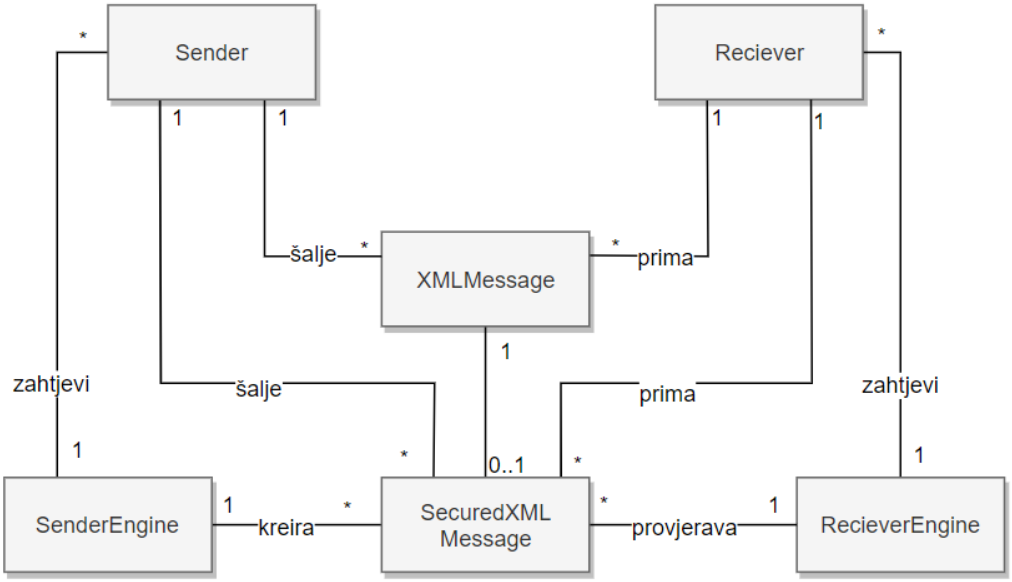
Sesiji se može pristupiti preko objekta klase HttpContext u svim kontrolerima aplikacjie zbog toga što oni primaju HTTP zahtjeve. Taj HttpContext predstavlja ExecutionContext klasu u ovom slučaju. Aktivna sesija u ovoj aplikaciji kao jedan od atributa sadrži i JWT token koji u sebi ima opis tvrdnji vezane uz korisnika, kao što su to identifikacija korisnika, ime, email i njegova uloga. Sve se to naravno veže i uz kontekst.

WS-Security

Uzorak koji je korišten kod sigurnosti zahtjeva prema SOAP web servisu. Već je spomenuto prije da služi za opis kako već neki postojeći mehanizam sigurnosti implementirati u aplikaciju. To mogu biti enkripcije, digitalni potpisi ili sigurnosni tokeni. Slanje podataka kroz nesigurne kanale izloženo je raznim napadima poput nelegalnog čitanja ili mijenjanja podataka. Obzirom da ovaj uzorak služi kao opis implementaciji raznih mehanizama, za ovaj slučaj je odabrana implementacija sigurnosti s certifikatom.

Struktura

Obzirom da je napravljena najjednostavnija implementacija vezana uz ovaj uzorak, a njegova struktura je dosta kompleksna, na slici 24 prikazane su neke od bitnijih klasa koje su u komunikaciji.



Slika 24 UML dijagram klasa za uzorak WS-Security.
Vlastita izrada prema (Fernandez-Buglioni, 2013), str. 393.

Klasa *Sender* predstavlja pošiljatelja XML poruka odnosno aplikaciju koja šalje podatke SOAP servisu, a on je u ovo dijagramu predstavljen klasom *Reciever*. Nadalje, klase *SenderEngine* i *RecieverEngine* predstavljaju dijelove koji rade kriptiranje odnosno dekriptiranje podataka, potpisivanje odnosno verificiranje podataka i slično. Kod ove aplikacije u pitanju su potpisani certifikati.

Implementacija

Prvo i osnovno, kako bi veza prema SOAP servisu bila sigurna bitno je postaviti url servisa na HTTPS tako da se svi podaci prenose preko sigurnog sloja, tj. SSL-a (Secure Socket Layer-a). SSL je protokol koji omogućuje HTTPS i oslanja se na asimetrično kriptiranje. Radi na način da klijent, u ovom slučaju aplikacija koja šalje zatjeve, dobije javni ključ preko SSL certifikata i to koristi kako bi inicirao sigurnu komunikaciju s web servisom. Dok, s druge strane web servis svoj privatni ključ drži tajnim i s njime dekriptira dobivene zahtjeve.

```
public class Program
{
    public static void Main(string[] args)
    {
        var host = new WebHostBuilder()
            .UseKestrel()
            .UseUrls("https://*:5050")
            .UseContentRoot(Directory.GetCurrentDirectory())
            .UseStartup<Startup>()
            .Build();

        host.Run();
    }
}
```

Zatim je potrebno osigurati certifikati koji će imati i slati obje strane zajedno sa svojim zahtjevima odnosno odgovorima. Za potrebe ove aplikacije napravljen je lokalni certifikat na Windowsima. Kako bi se osiguralo da ga aplikacija može koristiti u Windows PowerShellu pokrenuta je komanda *donet dev-certs https –trust*, te se na taj način kreira certifikat koji mogu koristiti sve web aplikacije u razvoju pokrenute na localhostu.

Web aplikacija spaja se na servis putem *BasicHttpBinding* klase čijem se objektu dodjeljuju nekakve postavke koje su bitne za prijenos podataka. Najbitnija stvar ovdje je dohvaćanje certifikata iz „dućana“ kako bi aplikacija mogla potvrditi da može slati podatke servisu, a servis je na taj način autenticirati.

```
public class SOAPClient
{
   IAuthenticateService serviceClient;
   public SOAPClient()
   {
      var binding = new BasicHttpBinding();
      binding.Security.Mode = BasicHttpSecurityMode.Transport;
       binding.Security.Transport.ClientCredentialType =
       HttpClientCredentialType.Certificate;
      binding.TextEncoding = Encoding.UTF8;

      var endpoint = new EndpointAddress(new
      Uri(string.Format("https://localhost:5050/Service.svc",
      Environment.MachineName)));
      var channelFactory = new ChannelFactory<IUserManagingService>(binding,
      endpoint);

      X509Certificate2Collection results;
      X509Store store = new X509Store(StoreName.My, StoreLocation.CurrentUser);
      store.Open(OpenFlags.ReadOnly);
      try
      {
         string subject =
       MyAppData.Configuration.GetValue<string>("Certificate:Subjectname");
         string thumbprint =
       MyAppData.Configuration.GetValue<string>("Certificate:Thumbprint");
         results = store.Certificates
         .Find(X509FindType.FindBySubjectName, subject, true)
         .Find(X509FindType.FindByThumbprint, thumbprint, true);
       }
       finally
       {
          store.Close();
       }

        channelFactory.Credentials.ClientCertificate.Certificate = results[0];

        serviceClient = channelFactory.CreateChannel();
        }
```

Secure MVC

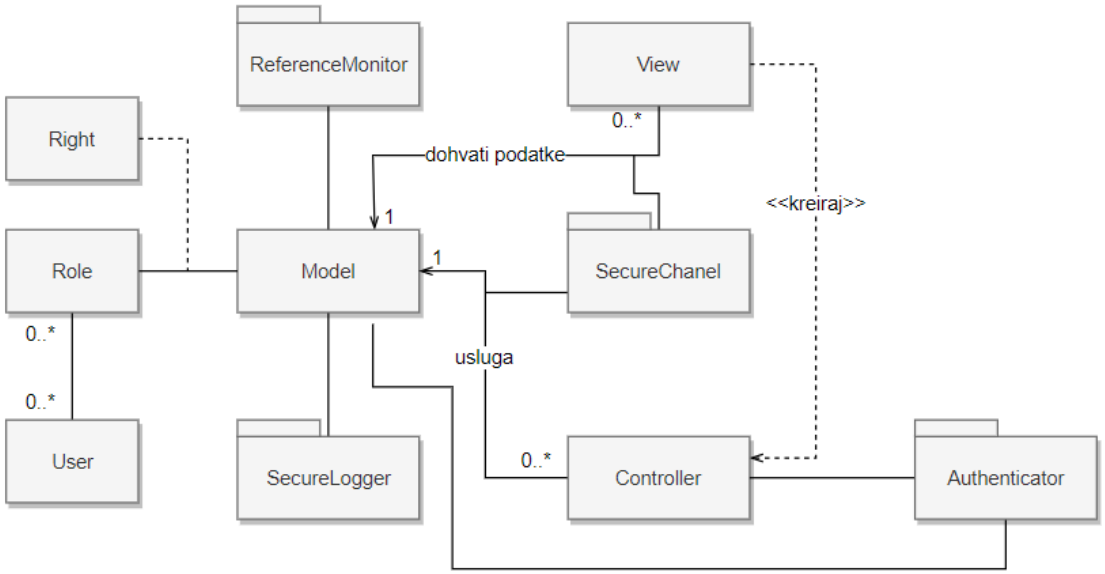
Obzirom da je .NET Core temeljen na MVC-u, iako se može i bez njega, ovaj je uzorak primijenjen kod implementiranja aplikacije. Svrha uzorka je zaštiti određene dijelove modela, pogleda i kontrolera. MVC uzrokom postiže se modularnost aplikacije razdvajajući je na tri komponente koje su prethodno spomenute.

Struktura

Kod klasičnog MVC uzorka strukturu čine četiri klase: *Model, View, Controller* i *Observer.* Prvo, kod .NET Core aplikacije, s obzirom da je to web aplikacija, ne postoji klasa *Observer*, pa struktura nije kao kod klasičnog modela. U njezinom slučaju, pogled šalje poruke kontroleru koji ažurira model putem poslovne logike, a on pokrene promjene koje prima pogled.

Drugo, ovaj uzorak, Secure MVC, uz temeljne klase MVC-a sadrži i ostale klase koje pripadaju uzorcima vezanima uz njegovu sigurnost. Uzorci su: SecureLogger, ReferenceMonitor vezani za Model, zatim SecureChannel koji daje sigurnu komunikaciju

između kontrolera i modela, te modela i pogleda. Uz model i kontroler veže se još uzorak Authenticator.



Slika 25 UML dijagram klasa uzorka Secure MVC, Vlastita izrada prema (Fernandez-Buglioni, 2013), str. 444.

Implementacija

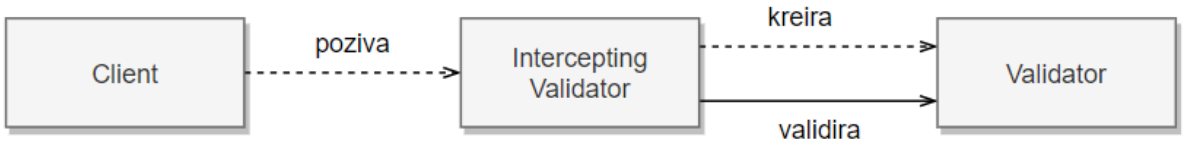
Implementacija ovog uzorka se više temelji na implementaciji ostalih uzoraka koji se brinu za sigurnost pojedinih modula aplikacije. Stoga je bitno aplikaciju podijeliti u logične cjeline i primijeniti ostale uzorke koji se vežu na uzorak Secure MVC.

Intercepting Validator

Prilikom prijave i registracije korisnici mogu unositi razne podatke pa je potrebno, prije nego krenu dalje u obradu, vidjeti da li uopće odgovaraju zahtjevima aplikacije. Ponekad korisnik može unijeti neke podatke koji sustavu ne odgovaraju te ga na taj način oštetiti. Isto tako, unosom napadači vrlo lako mogu naštetiti sigurnosti sustava stoga je bitno prvo validirati podatke koji prolaze kroz aplikaciju. Validacija pomoću uzorka Intercepting Validator se odvija na serverskoj strani. U tome mu pomažu .NET Core Razor stranice.

Struktura

U strukturi uzorka nalaze se *Client, Intercepting Validator* i *Validator*.



Slika 26 UML dijagram klasa uzorka Intercepting Validator, Vlastita izrada prema (Flylib, Web-Tier Security Patterns, 2017)

Klijent pošalje zahtjev za pristup određenom resursu. *Intercepting Validator* ponaša se kao filter te se prvo on poziva i prihvaća unos koji je poslan. Dobivene podatke validira prema uvjetima koje aplikacija zahtjeva. Ako je validacija prošla preusmjerava na sljedeću obradu

podataka što se u ovoj aplikaciji odnosi na autentikaciju korisnika odnosno registraciju korisnika u aplikaciji. Dok, s druge strane ako validacija nije uspješna vraća povratnu poruku.

Implementacija

Intercepting Validator u MVC web aplikaciji implementiran je kao kontroler. To je najlakši način da se može pozvati na određene zahtjeve korisnika. Implementiran je za dvije validacije. Jedna služi za funkcionalnost prijavljivanja korisnika u sustav, te prosljeđuje validirane podatke dalje autentikatoru u slučaju da je validacija uredno prošla. Inače vraća povratnu poruku kako su podaci krivo uneseni, odnosno moguće je da korisnik ne postoji.

Drugi slučaj za koji je validator implementiran je kod registracije korisnika. Kod .NET Corea, u modelima je moguće zadati određene anotacije atributima i s njima odrediti kako treba izgledati unos za svaki.

```
public class RegistrationModel
{
   [MinLength(5,ErrorMessage = "Korisničko ime mora sadžavati minimalno 5
   znakova")]
   [Required(ErrorMessage = "Korisničko ime mora biti popunjeno!")]
   public string Username { get; set; }

    [Required(ErrorMessage = "Lozinka mora biti popunjena!")]
    public string Password { get; set; }

     [Required(ErrorMessage = "Ponovljena lozinka mora biti popunjena!")]
     public string ConfirmPassword { get; set; }

     [RegularExpression("^[a-zA-Z0-9_.-]+@[a-zA-Z0-9-]+.[a-zA-Z0-9-.]+$",
     ErrorMessage = "Email nije ispravno upisan")]
     [Required(ErrorMessage = "Email mora biti popunjen!")]
     public string Email { get; set; }

     public string ErrorMessage { get; set; }
}
```

Ta dva slučaja prikazana su kodom u klasi *InterceptingValidatorController*.

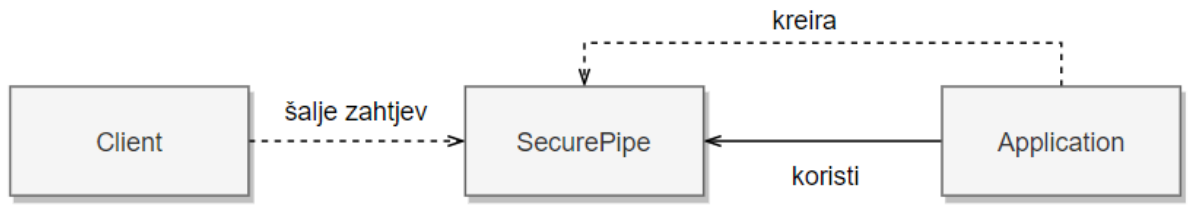
```
public class InterceptingValidatorController : Controller
{
   public IActionResult LoginValidator([Bind("Username, Password")] LoginModel
   loginModel)
   {
      if (ModelState.IsValid)
      {
         TempData["loginModel"] =
$"{{\"Username\":\"{loginModel.Username}\",\"Password\":\"{loginModel.Password}\"}}
";
         return RedirectToAction("Authenticate", "Authenticator");
       }
       return View("Login",loginModel);
    }

    public IActionResult RegisterValidator([Bind("Username,
    Password,Email,ConfirmPassword")] RegistrationModel regModel)
    {
       if (ModelState.IsValid)
       {
           if (regModel.Password != regModel.ConfirmPassword)
           {
              ModelState.AddModelError("ErrorMessage", "Lozinke nisu jednake!");
              return View("Registration", regModel);
           }
           TempData["regModel"] = $"{{\"Username\":\"{regModel.Username}\"," +
           $"\"Password\":\"{regModel.Password}\"," +
           $"\"Email\":\"{regModel.Email}\"}}";
           return RedirectToAction("Register", "Registration");
       }
       return View("Registration", regModel);
     }
}
```

Secure Pipe

Standardizirani način koji se koristi se kako bi se garantirao integritet i privatnost podataka koji se šalju putem nesigurnih veza. Njegova implementacija nije kompleksna, a zasniva se na kriptiranju i dekriptiranju podataka koji se prenose. Također, pomaže u sprječavanju „man-in-the-middle“ napada što se odnosi na presretanje poruka u prijenosu i njihovo izmjenjivanje, brisanje i slično. Kod web stranica napadima su na taj način često izložene transakcije koje se odvijaju unutar nje.

Struktura

Kao što ni implementacija nije kompleksna, tako nije ni struktura. Dio strukture su tri komponente: *Client, Secure Pipe* i *Application*.



Slika 27 UML dijagram klasa uzorka Secure Pipe. Vlastita izrada prema (Flylib, Web-Tier Security Patterns, 2017)

Klijent inicira prijavu na aplikaciju, a aplikacija kreira sigurnu vezu preko koje će komunicirati s njim. Klasa *SecurePipe* zaslužna je za stvaranje kriptiranje komunikacije koja pruža privatnost i integritet podataka između klijenta i aplikacije.

Implementacija

Bitno je napomenuti kako se kod .NET Corea, koristi server Kestrel, te se u glavnoj klasi aplikacije, *Program,* mogu namjestiti njegove opcije. Na taj način je implementiran sigurni kanal za komunikaciju. Isto tako, bitno je korištenje certifikata koji služi za vjerodostojnost aplikacije, a koji je napravljen lokalno u svrhu pokretanja web aplikacije u razvoju. Prikladna implementacija za konfiguraciju HTTPS protokola pronađena je na web stranici Microsofta (Daniel, 2017). Za konfiguriranje jer korištena klasa iz primjera.

Kod učitava HTTP krajnje točke konfiguracije servera i primjenjuje ih na server Kestrel. One sadrže i konfiguraciju za protokol HTTPS, kao na primjer koji certifikat treba koristiti što se u ovom slučaju koristi lokalno. Bitno je samo metodu *ConfigureEndpoints()* iz klase *KestrelServerOptionsExtensions* pozvati prilikom postavljanja Kestrela na lokalnom poslužitelju.

Klasa koja predstavlja Secure Pipe, te služi za preusmjeravanje s HTTP na HTTPS. Preuzeto sa (Daniel, 2017).

```
public static class KestrelServerOptionsExtensions
{
   public static void ConfigureEndpoints(this KestrelServerOptions options)
   {
      var configuration =
      options.ApplicationServices.GetRequiredService<IConfiguration>();
      var environment =
      options.ApplicationServices.GetRequiredService<IHostingEnvironment>();

       var endpoints = configuration.GetSection("HttpServer:Endpoints")
           .GetChildren()
           .ToDictionary(section => section.Key, section =>
           {
              var endpoint = new EndpointConfiguration();
              section.Bind(endpoint);
              return endpoint;
            });

            foreach (var endpoint in endpoints)
            {
                var config = endpoint.Value;
                var port = config.Port ?? (config.Scheme == "https" ? 443 : 80);

                 var ipAddresses = new List<IPAddress>();
                 if (config.Host == "localhost")
                 {
                    ipAddresses.Add(IPAddress.IPv6Loopback);
                    ipAddresses.Add(IPAddress.Loopback);
                 }
                 else if (IPAddress.TryParse(config.Host, out var address))
                    ipAddresses.Add(address);
                 else
                    ipAddresses.Add(IPAddress.IPv6Any);
```

```
                    foreach (var address in ipAddresses)
                    {
                        options.Listen(address, port, listenOptions =>
                        {
                            if (config.Scheme == "https")
                            {
                                var certificate = GetCertificate(config, environment);
                                listenOptions.UseHttps(certificate);
                            }
                        });
                    }
                }
            }

    private static X509Certificate2 GetCertificate(EndpointConfiguration config,
    IHostingEnvironment environment)
    {
        if (config.StoreName != null && config.StoreLocation != null)
        {
            using (var store = new X509Store(config.StoreName,
            Enum.Parse<StoreLocation>(config.StoreLocation)))
            {
                store.Open(OpenFlags.ReadOnly);
                var certificate = store.Certificates.Find(
                    X509FindType.FindBySubjectName,
                    config.Host,
                    validOnly: !environment.IsDevelopment());

                if (certificate.Count == 0)
                {
                    throw new InvalidOperationException($"Certificate not found for
                    {config.Host}.");
                }
                return certificate[0];
            }
        }

        if (config.FilePath != null && config.Password != null)
        {
            return new X509Certificate2(config.FilePath, config.Password);
        }

        throw new InvalidOperationException("No valid certificate configuration found
        for the current endpoint.");
          }
    }
}
```

Credential Tokenizer

Fleksibilni mehanizam koji kreira token kako bi se mogao koristiti u sigurnosne svrhe. Postoje razne vrste sigurnosnih tokena poput običnog korisničkog imena i lozinke, binarnih tokena, SAML tokena i slično. U ovom radu biti će opisan JWT token, koji je poslužio u svrhu autorizacije korisnika svaki put kada korisnik pošalje zahtjev za nekakav objekt, a kreira se pri uspješnoj autentikaciji korisnika.

Struktura

Ukratko je objašnjen kod autentikacije, ali bitno ga je malo bolje razumjeti. Sastoji se od tri dijela koji su odvojeni točkom: Zaglavlje (Header), Poruka (Payload), Potpis (Signature). Zaglavlje sadrži tip tokena i algoritam za potpisivanje, poruka sadrži izjave o entitetu i ostale proizvoljne podatke. Dok su za kreiranje potpisa potrebni kodirano zaglavlje, poruka, tajna i algoritam zapisan u zaglavlju koji se potpišu. (AuthO, n.d.)

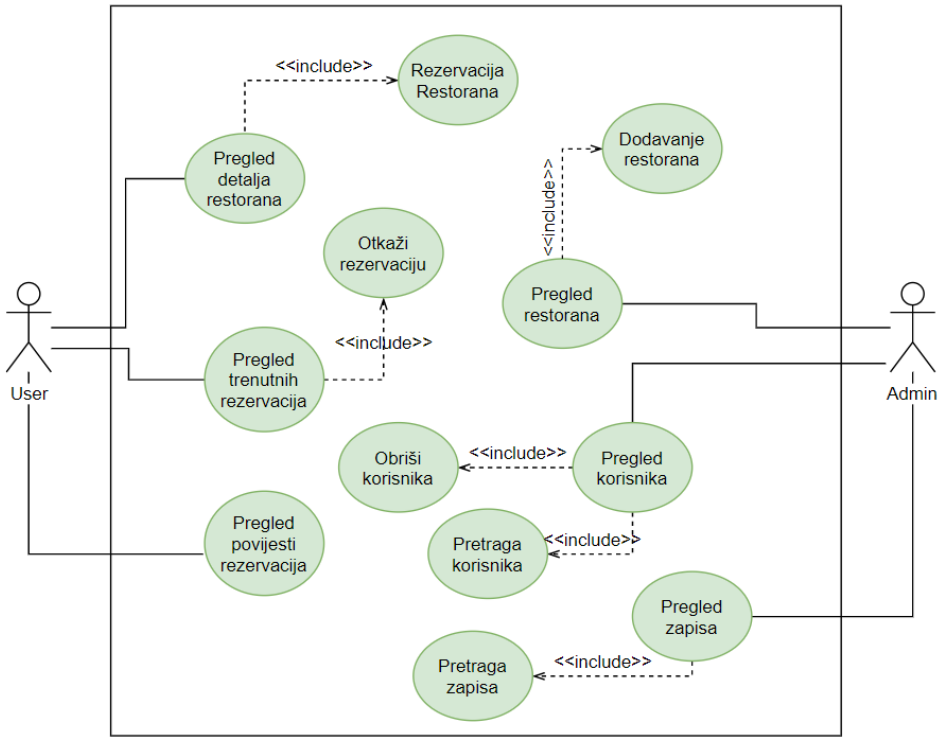
Implementacija

Objekt klase instancira se u klasi *AuthenticatorController* nakon uspješne autentikacije korisnika. Tajna koja je potrebna za generiranje potpisa može biti bilo kakav kombinacija znakova, po mogućnosti neka duža „riječ“. Kod .NET Corea postoji paket koji pomaže u generiranju tokena - System.IdentityModel.Tokens.Jwt. Prilikom stvaranja opisa tokena određuju se tvrdnje koje su vezane uz korisnika kojem se token dodjeljuje za trenutnu sesiju, te se one tijekom korištenja aplikacije mogu iskoristiti.

```
public class JWTCreator
{
   HttpContext _httpContext = null;

   public JWTCreator(HttpContext httpContext)
   {
      _httpContext = httpContext;
   }
   public void CreateJWT(string userId, string name, string email, bool admin)
   {
      string secret = MyAppData.Configuration.GetValue<string>("JWTSecret:Secret");

      var tokenHandler = new JwtSecurityTokenHandler();
      var key = Encoding.ASCII.GetBytes(secret);
      var tokenDescriptor = new SecurityTokenDescriptor
      {
          Subject = new ClaimsIdentity(new Claim[]
          {
             new Claim(ClaimTypes.NameIdentifier, userId),
             new Claim(ClaimTypes.Name, name),
             new Claim(ClaimTypes.Email, email),
             new Claim(ClaimTypes.Role, admin ? Role.Admin.ToString():
             Role.User.ToString()),
           }),
           Expires = DateTime.UtcNow.AddDays(1),
           SigningCredentials = new SigningCredentials(new SymmetricSecurityKey(key),
           SecurityAlgorithms.HmacSha256Signature);
       var token = tokenHandler.CreateToken(tokenDescriptor);

       _httpContext.Session.SetString("JWToken", tokenHandler.WriteToken(token));
   }
}
```

7. Prikaz rada aplikacije

Nakon prikazanih i detaljno opisanih uzoraka, može se pokazati kako aplikacija funkcionira u cjelini. Kako bi se bilo lakše snalaziti po aplikaciji potrebno je ukratko objasniti što se sve može napraviti. Pomoć u tome je UML dijagram slučajeva korištenja, gdje je prikazano što sve može raditi običan korisnik, te što sve može raditi administrator.

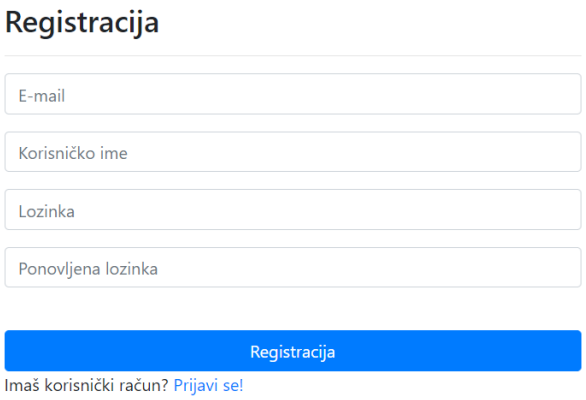


Slika 28 UML dijagrama slučajeva korištenja

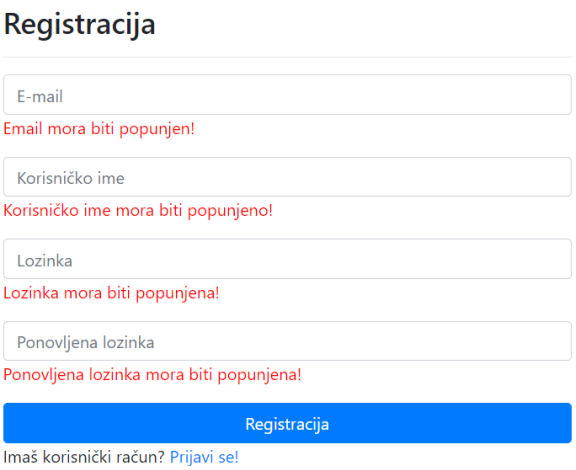
Funkcionalnosti običnog korisnika su: Pregled detalja restorana, Pregled trenutnih rezervacija i Pregled povijesti rezervacija. Kod pregleda detalja korisnik može vidjeti pojedinosti svakog restorana te rezervirati mjesto u restoranu tako da upiše broj stola, vrijeme koje će tamo provesti, te vrijeme i datum kada će doći. Kod trenutnih rezervacija je moguće otkazati rezervaciju. Administrator također ima tri glavne funkcionalnosti: Pregled restorana, gdje može dodati restorane pretragom po gradu, Pregled korisnika, gdje su mu omogućene pretraga i brisanje, te Pregled zapisa odnosno ima mogućnost pregleda svih akcija koje su korisnici proveli koristeći aplikaciju.

Sada kada je jasnije što sve mogu pojedine uloge, biti će prikazani i objašnjeni pojedini ekrani koje aplikacija sadrži. Na početku, tijekom učitavanja web aplikacije u web pregledniku, na mjestu za URL se prikazuje adresa http://localhost:8080/, no zbog implementiranog uzorka Secure Pipe preusmjerava se na sigurnu vezu putem SSL certifikata i HTTPS-a na adresu https://localhost:44340. Prvi ekran koji se prikazuje je prijava, ali je prije toga potrebno objasniti registraciju, jer ako korisnik nije registriran ne može se prijaviti u aplikaciju, te je ne može onda niti koristiti. Ovdje su prikazana dva primjera registracije na slikama 29 i 30. S time je na drugoj

slici pokazano kako točno uzorak Intercepting Validator radi. Kod krivog ili praznog unosa javlja određene greške i ne dopušta registraciju dok podaci ne odgovaraju zadanim uzorcima aplikacije. Zbog toga je korisnik primoran upisivati podatke na onaj način na koji aplikacija to zahtjeva.

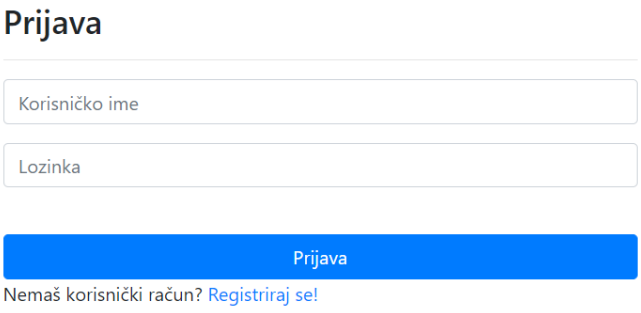


Slika 29 Obrazac registracije



Slika 30 Obrazac registracije s povratnim greškama

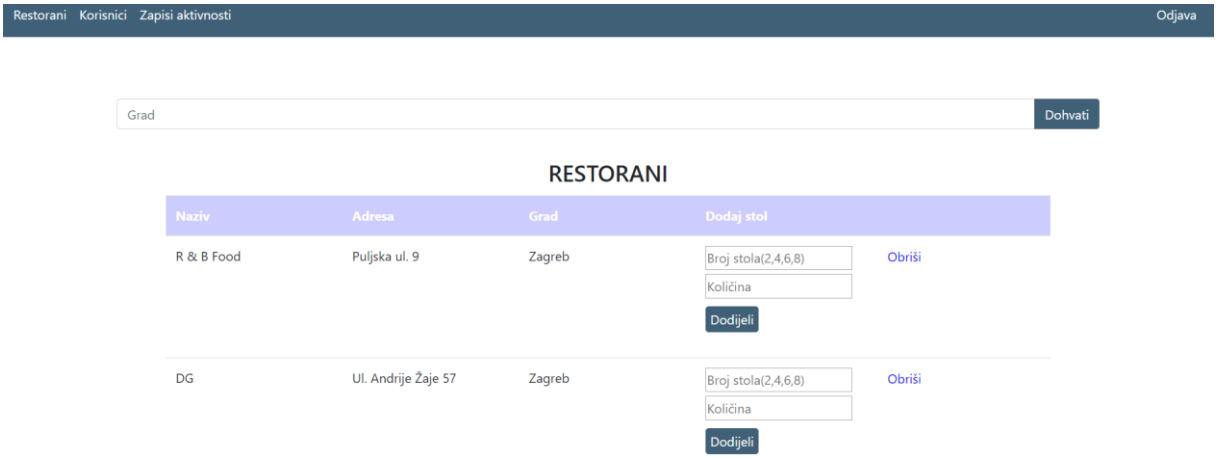
Kada se korisnik uspješno registrira aplikacija ga preusmjerava na formu prijave. Tu se isto odvija validacija unesenih podataka putem uzorka Intercepting Validator prije nego se nastavi s njihovom daljnjom obradom. Ovdje je koristan iz razloga da upozori korisnika, ako stisne na Prijava, kako nije popunio potrebne podatke ako je obrazac prazan.



Slika 31 Obrazac za prijavu

Tijekom prijave korisnika u sustav odvija se par akcija, tako prvo kreće autentikacija. Moraju se u bazi podataka provjeriti podaci koje je korisnik unio kako bi se kasnije mogli

pokazati i podaci koji pripadaju upravo njemu. Tijekom prijave i autentikacije provjeravaju se korisničko ime i lozinka, kojoj se uzima sažetak zajedno s izgeneriranom soli koju je korisnik dobio prilikom registracije. Kada je korisnik autenticiran, potrebno je dodijeliti JWT token kojim će se korisnik svaki put autenticirati prema aplikaciji kada šalje zahtjeve. Generiranje JWT tokena implementirano je jednostavnim uzorkom Credential Tokenizer. Nakon toga kreće autorizacija i dodjela prava kako bi mogao vidjeti samo ono što je namijenjeno njegovo ulozi. Slika 32 prikazuje početni zaslon prijavljenog korisnika u ulozi administratora koji je svoja prava dobio na temelju postavke uloge u bazi podataka i koja mu je dodijeljena tijekom prijave u aplikaciju putem implementiranog uzorka Authorization. Nadalje, tu se izvršava i uzorak Secure Logger And Auditor za zapisivanje korisnikovih akcija, kao npr. u ovom slučaju pregled restorana i mogućnost dodavanja.



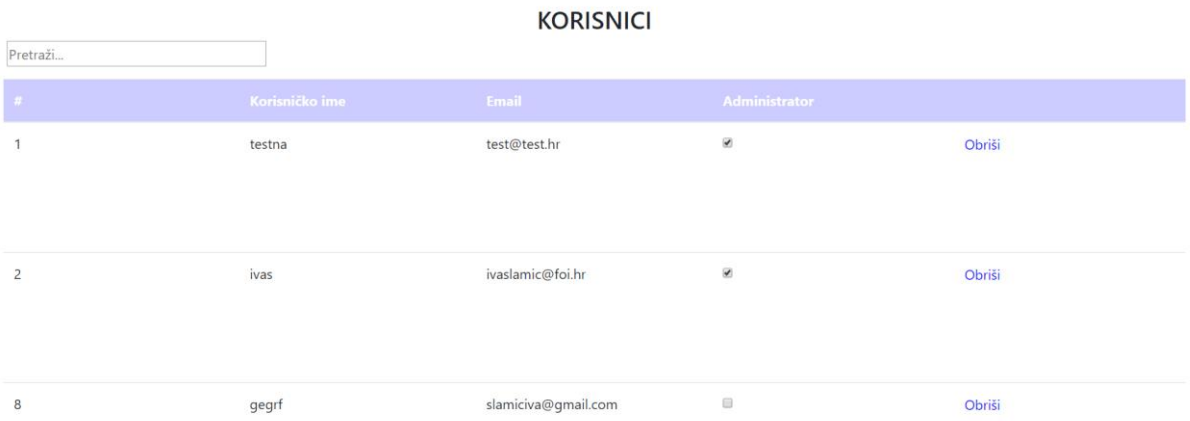
Slika 32 Početni zaslona za administratore

Administratori tako na ovo zaslonu imaju mogućnosti dodavanja novih restorana upisom željenog grada. Također, mogu dodavati stolove i specificirati brojku za količinu stolova koje ima neki restoran. Naravno, provjerava se unos ovisno o tome da li je stol dodan u bazu podataka za taj restoran, jer ako je onda će aplikacija javiti da kombinacija već postoji. Administratori imaju i mogućnost brisanja restorana. Slika 33 prikazuje dio izgleda zaslona kada se upiše grad i klikne gumb Dohvati, te su prikazani svi restorani dohvaćeni za taj grad.

Slika 33 Ispis restorana prema upisanom gradu

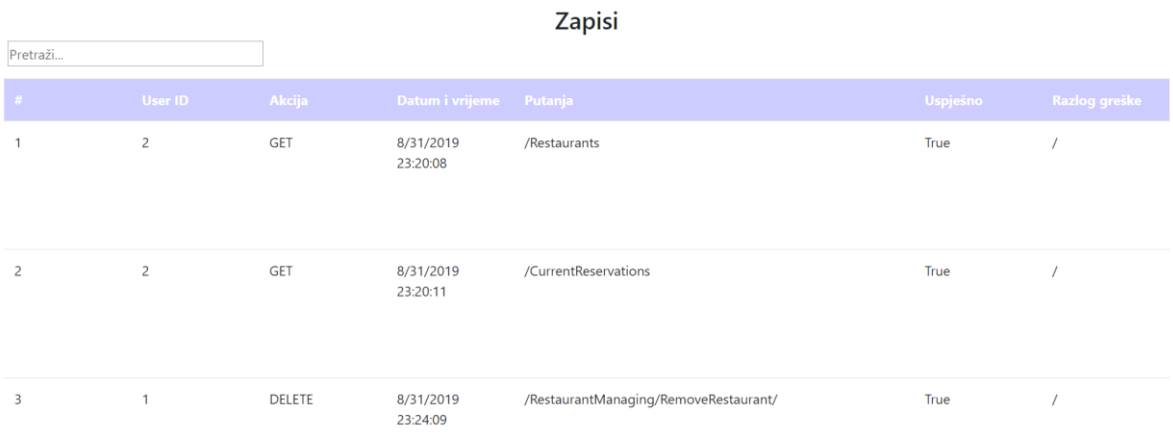
Kod administratora se zapis u log izvršava jedino ako izvrši akciju brisanja određenog entiteta odnosno restorana ili običnog korisnika, inače nije potrebno bilježiti. Taj dio filtrira se kod implementacije uzorka Secure Logger and Auditor. Nadalje, ima pregled korisnika koje može pretraživati, brisati ili im dodijeliti uloge: Administrator ili Korisnik. Potrebno je također naglasiti kako svaki dohvati podataka iz baze ide preko SOAP web servisa prema kojem su

pozivi osigurani SSL certifikatom, u ovom slučaju namijenjenom localhostu. Tome je namijenjen uzorak WS-Security.



Slika 34 Prikaz korisnika

Uz prethodne funkcionalnosti, svaki korisnik koji ima ulogu administratora može pregledavati zapise akcija koje su se izvršile. To je također dio uzorka Secure Logger And Auditor, gdje administratori imaju pristup podacima za pregledavanje (eng. audit).



Slika 35 Prikaz pregleda zapisa

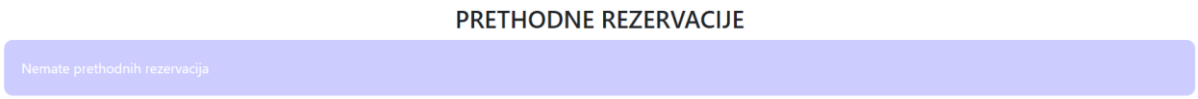
Običan korisnik ima prikaz potpuno druge navigacije, te ne može vidjeti stranice koje može vidjeti administrator. Ako ipak pokuša pristupiti nekoj od stranica na koju nema prava, javiti će mu se stranica s pogreškom kako nije autoriziran za pristup tom resursu s HTTP statusnim kodom 403 (Unauthorized). Ovo omogućava implementirani uzorak Authorization koji se pokrene svaki put kada bilo koji korisnik pokuša pristupiti nekom resursu provjeravajući njegovu ulogu.

Početni zaslon za običnog korisnika prikazuje popis restorana koje on može pregledavati i rezervirati.

Slika 36 Prikaz restorana za korisnika

Klikom na gumb Detalji kod pojedinog grada prikazati će se koje stolove ima i koliko slobodnog mjesta svaki stol ima. Uz to stoji obrazac gdje može upisati željene opcije i rezervirati stol u restoranu. Uz ovu funkcionalnost korisnik još ima mogućnost pregleda trenutnih rezervacija i povijest rezervacija. Slika 37 prikazuje trenutne, a slika 38 prikazuje povijest, odnosno može se vidjeti kako trenutno prijavljeni korisnik do sada nije rezervirao svoje mjesto.

Slika 37 Trenutne rezervacije prijavljenog korisnika



Slika 38 Povijest rezervacija prijavljenog korisnika

Na kraju, kao što se vidi na svakoj slici gdje je prikazana navigacija, postoji opcija za odjavu. Kada korisnik klikne na to, briše se sesija odnosno dodijeljeni token te se preusmjerava na stranicu za prijavu. Pokuša li se korisnik preko URL-a spojiti na neku od stranica koju je posjetio kada je bio prijavljen, prikazati će mu se stranica s pogreškom 401 (Not Found), te će pisati kako zatražena stranica ne postoji.

8. Zaključak

Prvenstveni zaključak tijekom izrade ove aplikacije je taj da su neki od uzoraka neizostavan dio aplikacije s aspekta sigurnosti, kao na primjer autentikacija ili autorizacija. Isto tako jedna od važnijih stvari koje uvijek treba provjeravati je korisnikov unos. Putem unosa u razne forme se često događaju napadi od kojih se mora zaštiti, što je zapravo vrlo lako. Također je uočeno kako .NET Core ima dobre temelje za konfiguriranje sigurnosnih mehanizama. Neke stvari se mogu s lakoćom implementirati, te pruža razne pakete koji mogu pomoći u kreiranju resursa potrebnih za implementaciju nekih uzoraka. Dobar primjer je spomenuti JWT token za kojeg se kreiranje olakšava paketom IdentityModel.Tokens.Jwt. Također sadrži i veliki broj klasa koje su od velike pomoći kada je u pitanju razvoj sigurnosti web aplikacije. Može se i uočiti kako su opisani uzorci iz knjige Security Patterns in Practice dosta apstraktni pa ih se može primijeniti u implementaciji s bilo kojim programskim jezikom, dok s druge strane Java programski jezik ima specificirane uzorke namijenjene Java programskom jeziku. Ti uzorci isto tako označavaju primjenu dobre prakse za sigurnost te pokrivaju sve slojeve aplikacije što je bitno za konzistentnost. Vidimo kako se ipak, malom prilagodbom neki od njih, mogu primijeniti i u ovom primjeru u slučaju programskog jezika C#.

Kada se razmišlja o primjeni sigurnosti u implementaciji potrebno je dobro razraditi plan kako bi se mogla primijeniti na svaki mogući sloj aplikacije da ona bude što bolje zaštićena od mogućih napada. Kada bi se dogodio propust sigurnosti u nekom od slojeva aplikacije, to bi utjecalo i na sigurnost ostalih slojeva, što znači da sva primjena sigurnosti na te slojeve „pada u vodu“. Postoje razne metode, rješenja i prakse koje se primjenjuju pri implementaciji web aplikacija koja su namijenjena specifičnim napadima.

Raznolikost resursa koji se danas koriste u poslovanju su duboko integrirani te su sve češća meta napadača. Tako i primjena sigurnosti raste proporcionalno napadima koji se često događaju. Isto kao što su danas napravljeni veliki koraci u samoj tehnologiji, tako je i sigurnost dio toga napretka. Tijekom izrade bilo kakvog većeg sustava posebnu pozornost treba namijeniti primjeni sigurnosti i njezinoj arhitekturi, te pokriti sve moguće slučajeve kako se ne bi događali propusti. Posebno zbog izgradnje većih sustava potrebni su ovakvi uzorci jer zbog njihove kompleksnosti može doći do pogrešaka u implementaciji koje se teže mogu naći, te su zbog toga i ranjiviji takvi sustavi. Implementacijom uzoraka se kasnije lakše može napraviti reinženjering sustava kako bi se npr. dodale neke sigurnosne značajke koje nedostaju i slično.

Na kraju, uvijek je bitno u dijelu životnog ciklusa razvoja aplikacije posebnu pozornost obratiti na implementaciju sigurnosti jer bez toga nema integriteta, povjerljivosti niti dostupnosti podataka, a s time dolazi i nepovjerenje korisnika. Isto kao i GOF uzorci dizajna, i uzorci sigurnosti prikazuju najbolje prakse za primjenu sigurnosti koje pomažu u dobrom strukturiranju koda pa se time postiže i bolja preglednost i bolja točnost programskog koda.

Popis literature

[1] AuthO. (n.d.). *Introduction to JSON Web Tokens*. Dohvaćeno iz jwt.io: https://jwt.io/introduction/

[2] Blog@Croz. (23. Listpoad 2014). *Zaštita od curenja podataka*. Preuzeto 1. Svibanj 2019 iz Blog: https://blog.croz.net/blog/zastita-od-curenja-podataka/

[3] Brune, E. (n.d.). *4 common web application security attacks and what you can do to prevent them*. Preuzeto 5. Srpanj 2019 iz Instart: https://www.instart.com/blog/4-common-web-application-security-attacks-and-what-you-can-do-prevent-them

[4] Daniel. (29. 11 2017). *Configuring HTTPS in ASP.NET Core across different platforms*. Dohvaćeno iz devblogs.microsoft.com: https://devblogs.microsoft.com/aspnet/configuring-https-in-asp-net-core-across-different-platforms/

[5] Erich Gamma, R. H. (1994). *Design Patterns: Elements of Reusable Object-Oriented Software.* Pearson Education.

[6] Fernandez-Buglioni, E. (2013). *Security Patterns in Practice: Designing Secure Architectures Using Software Patterns.* Wiley.

[7] Flylib. (2017). *Business Tier Security Patterns*. Preuzeto 21. Srpanj 2019 iz Flylib.com: https://flylib.com/books/en/3.211.1.132/1/

[8] Flylib. (2017). *Identity Management Security Patterns*. Preuzeto 22. Srpanj 2019 iz Flylib.com: https://flylib.com/books/en/3.211.1.142/1/

[9] Flylib. (2017). *Web Services Security Patterns*. Preuzeto 22. Srpanj 2019 iz Flylib.com: https://flylib.com/books/en/3.211.1.138/1/

[10] Flylib. (2017). *Web-Tier Security Patterns*. Preuzeto 21. Srpanj 2019 iz Flylib.com: https://flylib.com/books/en/3.211.1.127/1/

[11] Joseph Yoder, J. B. (1998). *Architectural Patterns for Enabling.* Dohvaćeno iz NTNU: https://www.idi.ntnu.no/emner/tdt4237/2007/yoder.pdf

[12] Pressbook. (n.d.). *Chapter 6: Information Systems Security*. Preuzeto 01. Svibanj 2019 iz https://bus206.pressbooks.com/chapter/chapter-6-information-systems-security/

[13] Romanosky, S. (21. Studeni 2001). *Security design patterns.* Preuzeto 4. Srpanj 2019 iz cgisecurity: https://www.cgisecurity.com/lib/securityDesignPatterns.pdf

[14] Technopedia. (2019). *Information system security (INFOSEC)*. Preuzeto 01. Svibanj 2019 iz https://www.techopedia.com/definition/24840/information-systems-security-infosec

Popis slika

Popis tablica