.NET sustav za prikupljanje i pohranu uređenih podataka u Azure okruženju s MySQL bazom podataka

Frčko, Luka

VELEUČILIŠTE U BJELOVARU
STRUČNI PRIJEDIPLOMSKI STUDIJ RAČUNARSTVO

.NET SUSTAV ZA PRIKUPLJANJE I POHRANU UREĐENIH PODATAKA U AZURE OKRUŽENJU S MYSQL BAZOM PODATAKA

Završni rad br. 17/RAČ/2023

Luka Frčko

Bjelovar, listopad 2023.

obrazac ZR - 001

Veleučilište u Bjelovaru

Trg E. Kvaternika 4, Bjelovar

1. DEFINIRANJE TEME ZAVRŠNOG RADA I POVJERENSTVA

Student: **Luka Frčko** JMBAG: **0314023009**

Naslov rada (tema): **.NET sustav za prikupljanje i pohranu uređenih podataka u Azure okruženju s MySQL bazom podataka**

Područje: **Tehničke znanosti** Polje: **Računarstvo**

Grana: **Programsko inženjerstvo**

Mentor: **Dario Vidić, mag. ing. el. techn. inf.** zvanje: **viši predavač**

Članovi Povjerenstva za ocjenjivanje i obranu završnog rada:

1. **Tomislav Adamović, mag. ing. el., predsjednik**
2. **Dario Vidić, mag. ing. el. techn. inf., mentor**
3. **Ivan Sekovanić, mag. ing. inf. et comm. techn., član**

2. ZADATAK ZAVRŠNOG RADA BROJ: 17/RAČ/2023

U sklopu završnog rada potrebno je:
1. razviti API metode za komunikaciju s Azurom, MySQL bazom podataka i SendGrid servisom za slanje potvrdne e-pošte.
2. dizajnirati i izraditi funkcionalnu bazu podataka koristeći migracijski sustav unutar .NET Entity Framework-a te koristeći Docker Image i Docker Hub javno objaviti najnovije nadogradnje API metoda.
3. ostvariti komunikaciju s Azure blob pohranom podataka te koristiti za pohranu, upravljanje i preuzimanje datoteka po određenim kriterijima i API pozivima.
4. implementirati sigurnu komunikaciju između Azure kontejnera, Sendgrid servisa i MySQL baze Podataka što uključuje hashiranje osjetljivih podataka, korištenje JSON web tokena, logiranje pozvanih metoda te prikazivanje samo potrebnih podataka ovisno o API pozivu.
5. koristiti C# za izradu REST API-a za detaljno upravljanje modelima unutar baze s sigurnosnim koracima i provjerama u sustavu s višerazinskim autorizacijskim pravima

Datum: 29.09.2023. godine Mentor: **Dario Vidić, mag. ing. el. techn. inf.**

Zahvala

Zahvalio bih se profesoru i mentoru Dariju Vidiću na izvrsnoj ideji i prilici za izradu završnog rada. Također na svim savjetima i pomoći prilikom izrade završnog rada.

Hvala profesoru i sjajnom voditelju smjera računarstva Tomislavu Adamoviću na vođenju smjera, studentskog inkubatora i naravno studentskog šaha.

Hvala profesoru Krunoslavu Husaku što me je podučio .NET programiranju što sam koristio za izradu završnog rada i čime se trenutno bavim na poslu.

Hvala svim ostalim profesorima na savjetima i pomoći pri izradi projekata za potrebe predmeta tijekom studija.

Hvala referadi Veleučilišta na korisnim i brzim informacijama tijekom studija.

Hvala svim kolegama i prijateljima koji su učinili da ovaj studij prođe što prije i zabavnije.

Hvala svim članovima moje obitelji koji su mi bili podrška tijekom studija.

Sadržaj

1. UVOD

.NET je danas jedan od najpopularnijih razvojnih okvira za razvoj aplikacija. Entity framework unutar .NET Core-a pruža jednostavno upravljanje nad podacima unutar baze podataka. MySQL je jedna od najčešće korištenih relacijskih baza podataka danas. Najveća prednost relacijskih baza podataka su SQL upiti koji dopuštaju dohvaćanje, stvaranje, ažuriranje i brisanje podataka na različite načine i uvjete. Cilj ovog završnog rada jest složiti funkcionalni .NET sustav koristeći .NET Web API aplikaciju kao temelj, povezati aplikaciju s Azure kontejnerom i Azure blob spremnikom podataka i uklopiti SendGrid servis za slanje verifikacijske e-pošte radi povećanja sigurnosti unutar aplikacije.

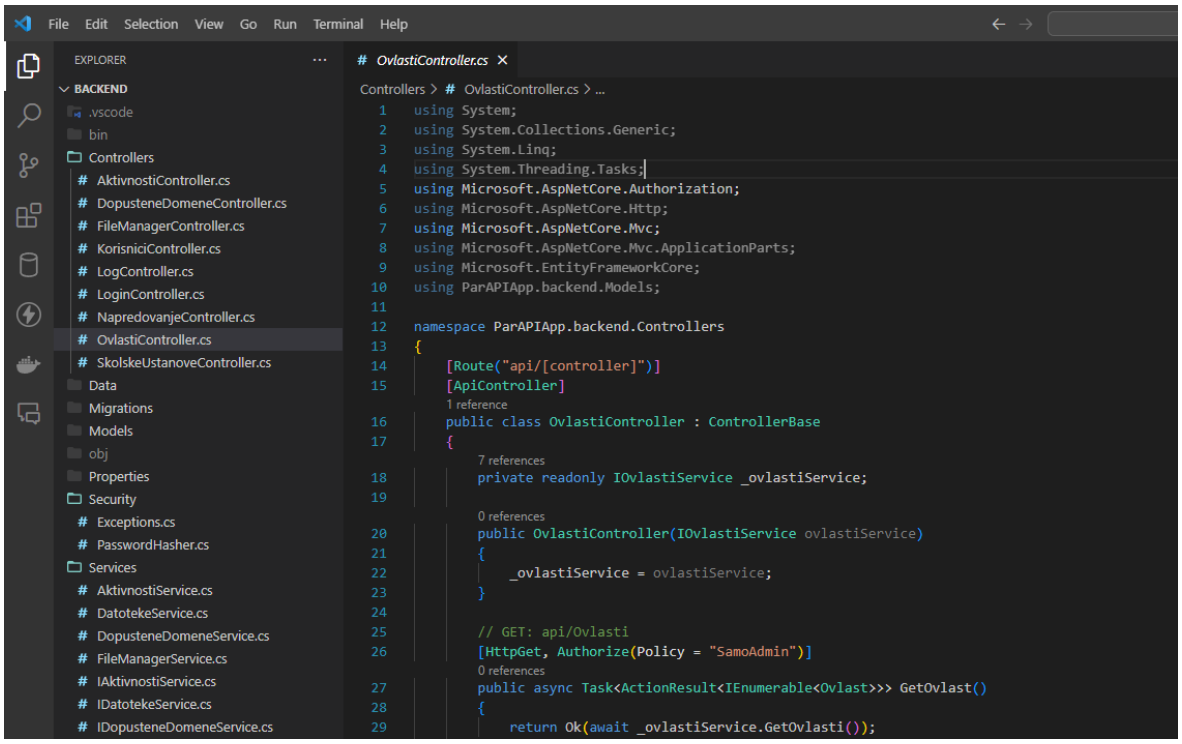
Drugo poglavlje govori o svim alatima, tehnologijama, proširenjima i servisima koji su korišteni za potrebe završnog rada. Također objašnjava korištenje gita za verzioniranje koda i docker-a za stvaranje slike (engl. *Image*) aplikacije koja se zatim koristi za javno korištenje aplikacije pomoću Azure kontejnera. U trećem poglavlju opisan je razvoj .NET sustava i kako spomenute tehnologije, alate i servise inicijalno postaviti za daljnji razvoj i održavanje sustava. Četvrto poglavlje govori kako koristeći .NET Core Entity Framework stvoriti tablice unutar baze podataka te kako stvoriti kontrolere i servise pomoću kojih se zatim može upravljati nad stvorenim tablicama. Peto poglavlje opisuje korištene metode zaštite podataka te kako ih primijeniti unutar aplikacije. Šesto poglavlje govori o registraciji i prijavi korisnika u sustav. Opisuje registraciju, prijavu i važnost istih, koji podaci su uobičajeno potrebni i sadrži primjere iz sustava kako je to riješeno unutar završnog rada. Sedmo poglavlje opisuje uz primjere koda unutar sustava kako se koristi Azure blob spremnik za učitavanje i preuzimanje datoteka s oblaka. Osmo poglavlje opisuje kako je aplikacija Postman korištena uz primjere za testiranje API ruta unutar .NET Web API aplikacije. U zaključku unutar posljednjeg dijela rada opisani su rezultati, mogućnosti i moguće nadogradnje.

2. KORIŠTENI ALATI

Ovo poglavlje pruža osnovne informacije tehnologija i servisa korištenih i implementiranih za izradu završnog rada. Cjelokupni kod metoda napisan je u uređivaču izvornog koda Visual Studio Code koristeći programski jezik C# odnosno .NET tehnologiju. Kao baza podataka korišten je MySQL. Za upravljanje podacima i objektima korišten je Entity Framework. Za testiranje i provjeru API metoda korišten je alat Postman i ekstenzija unutar VS Coda ThunderClient. Također je korišten git za upravljanje verzijama koda i docker za distribuiranje koda. Naposljetku je korišten servis SendGrid za slanje elektroničke pošte unutar API metoda prilikom registracije novog korisnika.

2.1 Visual Studio Code

Visual Studio Code je uređivač izvornog koda dostupan za Windows, MacOS i Linux operativne sustave koji sadrži proširenja za veliki broj jezika i okruženja. VS Code sadrži podršku za otklanjanje pogrešaka, isticanje sintakse, inteligentno dovršavanje koda i refaktoriranje koda. Izgled korisničkog sučelja koje pruža Visual Studio Code prikazan je na slici 2.1 [1].



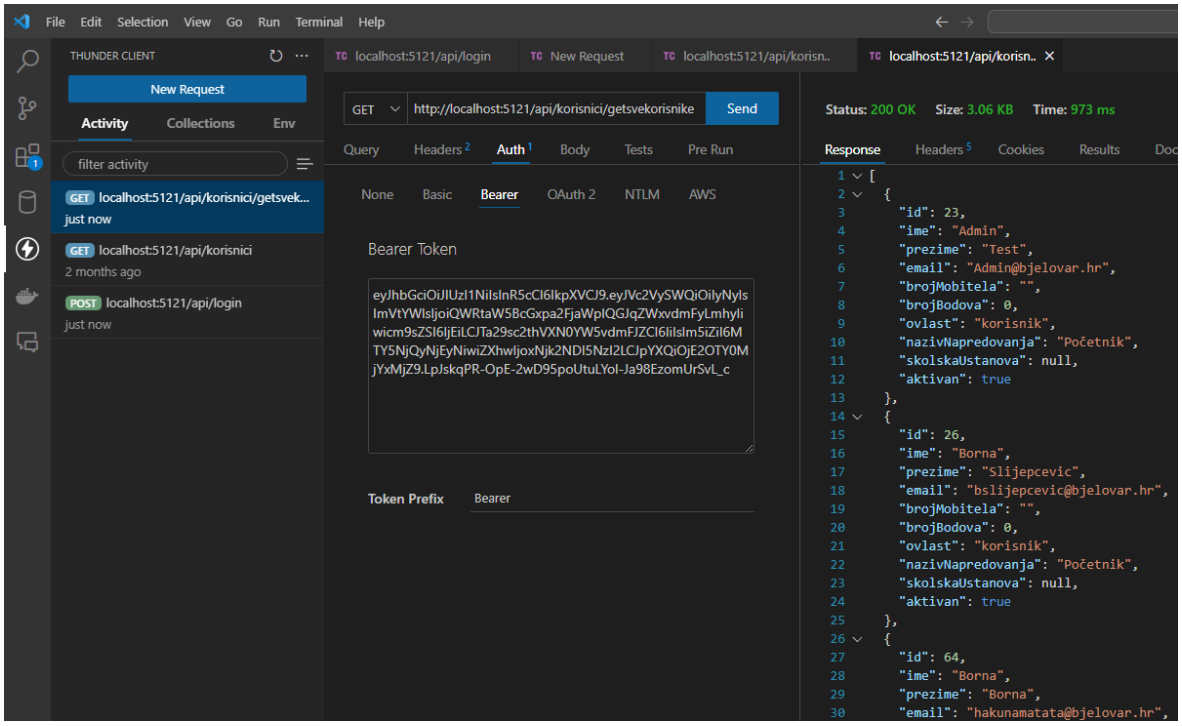
Slika 2.1: Korisničko sučelje razvojnog okruženja Visual Studio Code

2.1.1 *Entity Framework i ThunderClient*

Unutar VS Coda prilikom izrade servisa korišten je Entity Framework [2]. Razvija ga Microsoft i sadrži dosta paketa od kojih su neki potrebni dok drugi ovise o različitim parametrima poput modela baze podataka. To je objektno relacijski AutoMapper objekata koji omogućuje .NET programerima rad s relacijskim podacima pomoću objekata specifičnih za domenu. Entity Framework koristi LINQ upite za dohvaćanje podataka iz baze. LINQ upiti su imitacija SQL upita unutar C#. Entity Framework je iz tog razloga sposoban sam prevesti LINQ upite u SQL upite ovisno o vrsti baze podataka koja se koristi.

Unutar VS Coda korišten je RestClient za testiranje API metoda pod nazivom Thunder Client. Ovaj alat dolazi kao proširenje unutar VS Coda. Izgled ThunderClient sučelja unutar VS Coda prikazan je na slici 2.2.

ThunderClient omogućuje stvaranje API zahtjeva koji se pamte kroz razvijanje aplikacije i moguće ih je bilo kada pokrenuti u svrhu testiranja aplikacije.

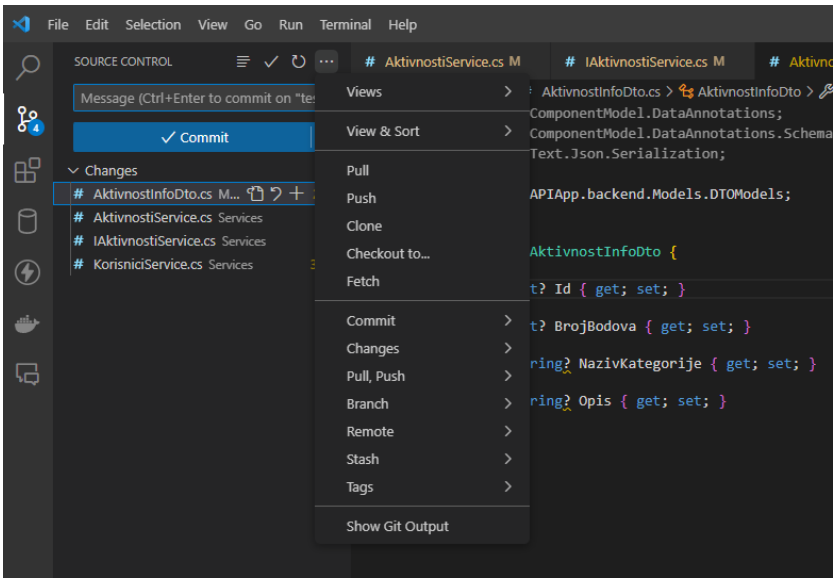


Slika 2.2: Sučelje ThunderClient odjeljka unutar VS Coda

2.1.2 *Git i docker*

Visual Studio ima ugrađenu podršku za git [3]. Git je alat koji se koristi za upravljanje verzijama izvornog koda. Glavna prednost Gita je što pruža da više programera može zajedno nelinearno razvijati aplikaciju. Git podržava stvaranje više grana koje mogu sadržavati istu aplikaciju u različitim verzijama. Na stvorene grane moguće je ubacivati datoteke koje se zatim mogu preuzeti na drugim uređajima koji imaju određena prava nad repozitorijem [3].

Za potrebe završnog rada git je korišten na način da su se promjene unutar aplikacije postavljale na testnu granu. Testna grana tako sadrži identičan kod aplikacije. Taj kod može a i ne mora biti ispravan. Također postoji glavna grana koja uvijek sadrži zadnju ispravnu verziju koda. Kada se uvjeri da su promjene na testnoj grani ispravne i da sve novostvorene ili izmijenjene API metode rade, promjene s testne grane se spajaju na glavnu granu. Slika 2.3 prikazuje git sučelje unutar Visual Studio Coda.



Slika 2.3: Sučelje git odjeljka unutar Visual Studio Coda

Docker se koristi za brzu izradu, testiranje i implementaciju aplikacija. On pakira softver u spremnike koji sadrže sve što je softveru potrebno za pokretanje. U radu je korišten docker hub koji se koristi za javno pronalaženje i dijeljenje slika spremnika.

Unutar završnog rada docker je korišten za stvaranje slike (engl. *image*) nakon promjena unutar koda i postavljanje (engl. *push*) slike (engl. *image*) na docker hub.

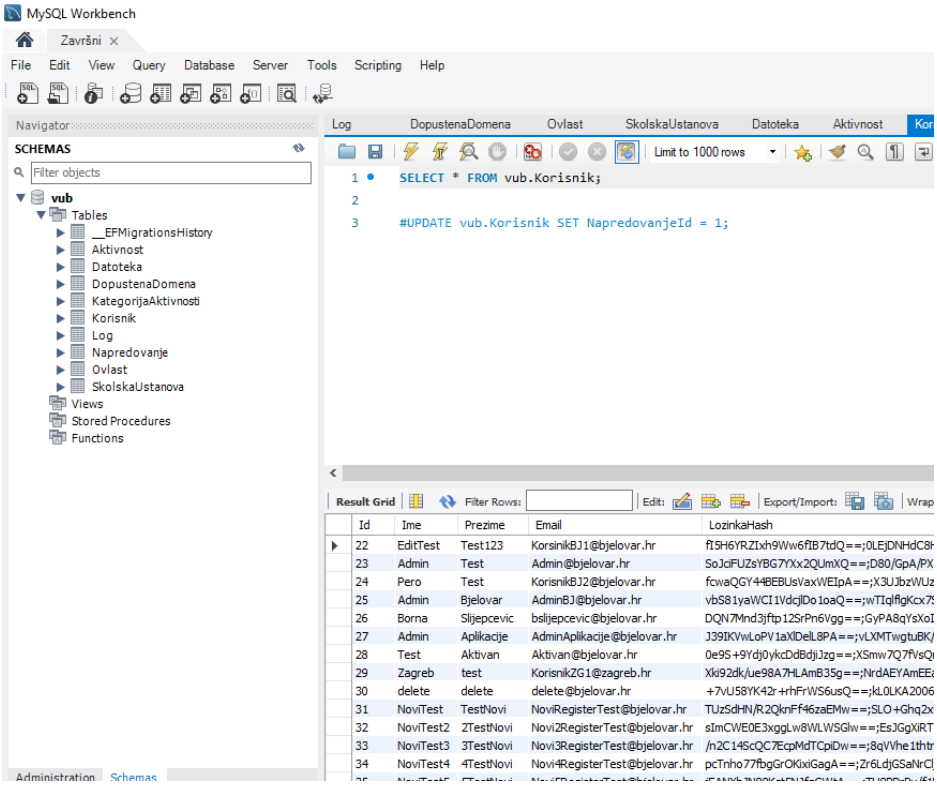
Git i docker sadrže unutar koda aplikacije ignore datoteke unutar kojih se nalaze nastavci datoteka ili nazivi direktorija koji će se zanemariti prilikom stvaranja slika (engl. *image*) kod dockera ili komitanja koda kod gita.

2.2 MySQL Workbench

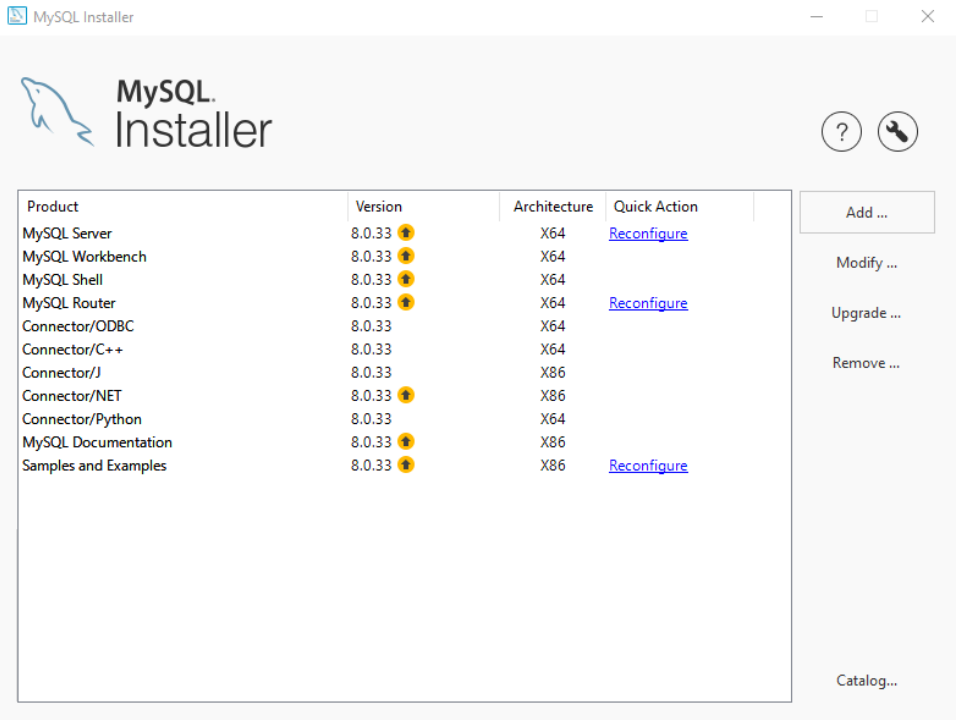
MySQL je najpopularniji Open Source SQL sustav za upravljanje bazom podataka kojega razvija Oracle Corporation. MySQL baze podataka su relacijskog tipa što znači da se podaci spremaju prema određenom logičkom modelu umjesto da su svi podaci unutar jednog velikog skladišta [4].

MySQL Workbench je alat za dizajniranje relacijske baze podataka otvorenog koda. Ovaj alat olakšava rad na razvoju MySQL i SQL baze podataka te pruža mogućnost izrade tablica i dizajniranje baze podataka bez spajanja na bazu. Korisničko sučelje koje pruža MySQL Workbench alat prikazan je na slici 2.4.

MySQL Workbench dolazi iz instalacijskog paketa MySQL Installer – Community koji pruža mogućnosti instalacije novih paketa, ažuriranje starih i izmjene nad paketima. Izgled MySQL Installer – Community prikazan je na slici 2.5.



Slika 2.4: Korisničko sučelje alata MySQL Workbench



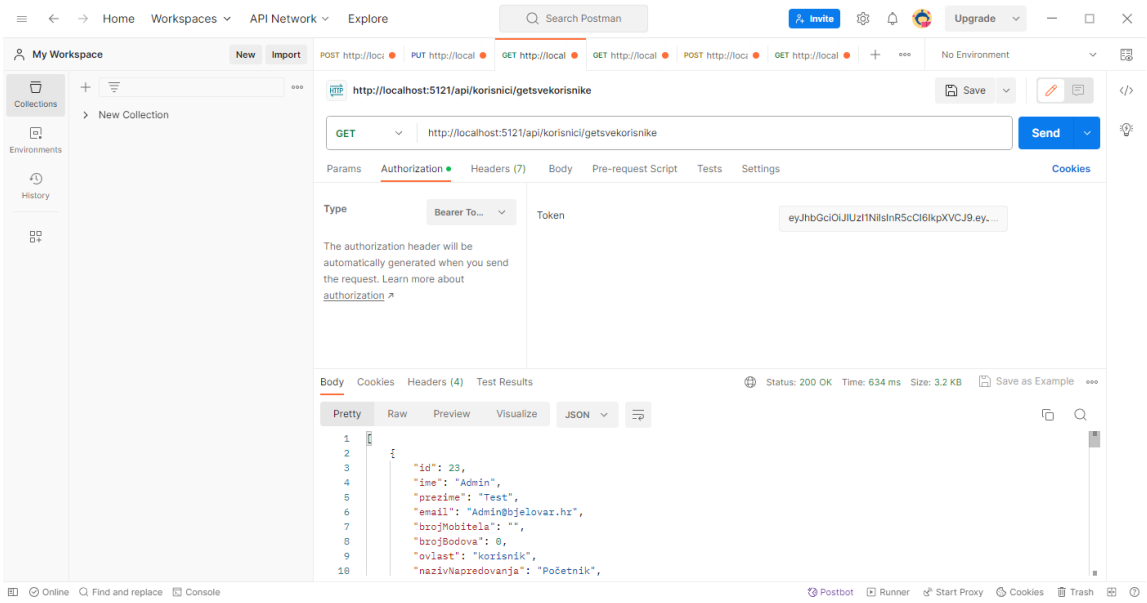
Slika 2.5: Početni ekran alata MySQL Installer – Community

2.3 Postman

Postman je API platforma za izradu i korištenje API zahtjeva. Pretežito se koristi za testiranje i poboljšanje API-ja [5].

Postman je sličan ThunderClientu, ali Postman nudi više mogućnosti kao stvaranje kolekcija. Kolekcija sadrži više API zahtjeva koji su međusobno povezani i obrađuju se zadanim redoslijedom. Također podržava stvaranje skripti koje očekuju određeni rezultat API zahtjeva ovisno kako su isprogramirane.

Kada se priča o REST API-ju postoje četiri glavne vrste API zahtjeva, a to su: GET, POST, PUT i DELETE. GET zahtjev se koristi prilikom dohvaćanja. Ovaj zahtjev može i ne mora sadržavati dodatne parametre koji se uobičajeno koriste u svrhu filtriranja dohvaćenih podataka. POST zahtjev se koristi za stvaranje novih zapisa prema podacima koji se nalaze u tijelu zahtjeva. Iako tijelo zahtjeva može sadržavati razne oblike zapisa uobičajeno se koristi JSON i XML oblik podatka. PUT zahtjev ažurira podatke sa podacima koji se nalaze unutar tijela zahtjeva. DELETE zahtjev briše podatke uobičajeno prema određenom uvjetu koji se pošalje.



Slika 2.6: Korisničko sučelje aplikacije Postman

2.4 SendGrid servis

SendGrid je servis za slanje elektroničke pošte. Za korištenje potrebno je stvoriti korisnički račun na njihovoj službenoj stranici [6]. SendGrid servis koristi vlastite API zahtjeve koji pružaju laku implementaciju u kod pomoću kojih se e-pošta šalje programski. Također pruža mogućnost stvaranja predložaka e-pošte.

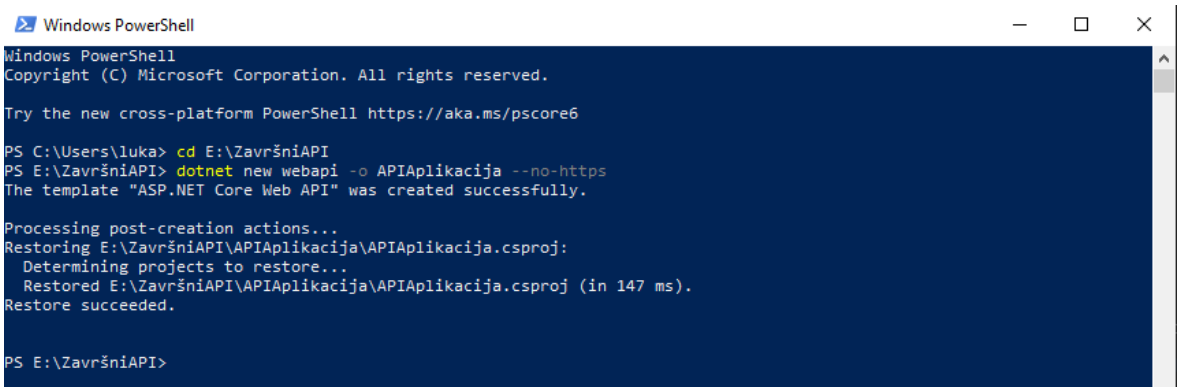
Što se tiče sigurnosnih aspekata, SendGrid servis sadrži alate za praćenje pošte pomoću kojih sprječava neželjenu poštu (engl. *spam*), a također pruža značajke koje pomažu kod autentifikacije i sigurnosti e-pošte.

3. POSTAVLJANJE ZAVRŠNOG RADA

Ovo poglavlje nudi informacije o stvaranju .NET Web API aplikacije i njenom povezivanje s MySQL bazom podataka. Također pruža informacije kako javno distribuirati aplikaciju koristeći Azure kontejner i kako uklopiti SendGrid servis za slanje elektroničke pošte.

3.1 Stvaranje API aplikacije

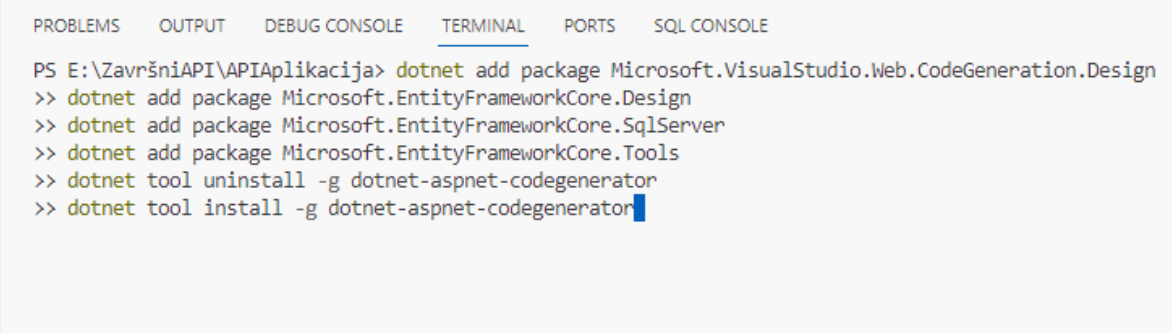
Za razvijanje API metoda potrebno je prvo inicijalno stvoriti web API aplikaciju unutar naredbenog retka (engl. *command prompt*) ili Windows Powershella. To se može ostvariti tako da prvo korištenjem naredbe cd i naziva direktorija se pozicionira unutar direktorija gdje se hoće stvoriti aplikacija. Nakon toga naredbom dotnet new webapi -o NazivAPIAplikacije –no-https stvaramo web API aplikaciju preko webapi predloška. Naznaka -o govori da naziv nakon toga govori u koji direktorij će se spremiti podaci o aplikaciji, a naznaka –no-https govori da se koristi HTTP protokol umjesto HTTPS protokola što se kasnije može promijeniti po potrebi. Spomenute naredbe prikazane su na slici 3.1.



Slika 3.1: Kod za stvaranje API aplikacije unutar Widnows PowerShell-a

Nakon uspješno odrađenih prethodno opisanih koraka moguće je pokrenuti aplikaciju. Pokretanje je moguće pomoću naredbe dotnet watch run unutar VS Code terminala. Naime unutar aplikacije postoje određene API metode unaprijed ugrađene ali u nastavku odjeljka biti će objašnjeno kako napraviti vlastite API metode. Važno je napomenuti kako sve radi samo lokalno i prilikom pokretanja aplikacije bitno je da unutar aplikacije nema greške inače neće biti moguće pokrenuti aplikaciju.

Nakon stvaranja inicijalne API aplikacije za stvaranje vlastitih API metoda potrebno je stvoriti početni model po volji i generirati kontroler (engl. *controller scaffolding*) prema stvorenom modelu. Kako bi mogli generirati kontroler potrebno je najprije instalirati nekoliko paketa koji su prikazani su na slici 3.2.



Slika 3.2: Paketi potrebni za generiranje kontrolera uneseni unutar VS Code terminala

Nakon dodavanja potrebnih paketa moguće je generirati kontroler pomoću sljedeće naredbe prikazane programskim kodom 3.1.

Programski kod 3.1: Naredba za stvaranje kontrolera unutar .NET Web API aplikacije

```
dotnet aspnet-codegenerator controller -name KorisniciController -async -api -m Korisnik -dc AppDbContext -outDir Controllers
```

Bitne stavke kod spomenute naredbe su KorisniciController što je naziv kontrolera koji će se generirat pomoću naredbe, Korisnik što je naziv modela po kojem će kontroler raditi i AppDbContert što je naziv sheme koja komunicira s bazom prilikom izmjena na Korisnik modelu.

Generiranjem kontrolera generiraju se bitne datoteke potrebne za povezivanje aplikacije s bazom podataka.

Za povezivanje aplikacije s bazom potrebna je MySQL baza podataka koja je unaprijed stvorena i postavljena. Moguće je stvoriti lokalnu MySQL bazu podataka koristeći MySQL Installer alat koja automatski sadrži root korisnika preko kojega se je moguće spojiti na bazu sa svim ovlastima. Također je potrebno konfigurirati datoteke appsettings.json i Program.cs koje se nalaze unutar aplikacije stvorene prema slici 3.1. Programski kod 3.2 prikazuje izgled konekcijskog stringa. Server i port se odnose na IP adresu i port na kojem se baza nalazi dok se database odnosi na naziv baze na koju se aplikacija treba spojiti. Uid i Pwd

se odnose na naziv i lozinku korisnika koji ima određena prava nad bazom. Prava mogu varirati od samo čitanja do mogućnosti izmjene svih mogućih podataka.

Programski kod 3.2: Konekcijski string unutar appsettings.json

```
"ConnectionStrings": {
    "DefaultConnection":
"server=IP;port=1234;database=NazivBaze;Uid=NazivKorisnika;Pwd=Lozinka"
}
```

Kada se generira kontroler unutar Program.cs datoteke generira se dio koda koji je potreban za stvaranje veze s bazom podataka prilikom pokretanja aplikacije. Po zadanom, taj dio koda postavljen je da uspostavlja vezu s MSSQL bazom podataka. Kako bi mogli unijeti kod za povezivanje s MySQL bazom podataka potrebno je instalirati paket koji to podržava. Unutar završnog rada instaliran je sljedeći paket unutar VS Code terminala: „Install-Package Pomelo.EntityFrameworkCore.MySql“. Programski kod 3.2 prikazuje kod koji stvara vezu s bazom prilikom svakog pokretanja aplikacije.

Programski kod 3.3: Kod za stvaranje veze s bazom prilikom pokretanja aplikacije

```
builder.Services.AddDbContextPool<AppDbContext>(options =>
options.UseMySql(connectionString,
ServerVersion.AutoDetect(connectionString)));
```

3.2 Javno postavljanje API aplikacije pomoću Azure kontejnera

3.2.1 *Azure kontejner*

Azure kontejner jedan je od rješenja da se na API aplikaciju mogu slati API zahtjevi izvan lokalnog okvira. Azure kontejner jedan je od mnogih servisa koje Microsoft Azure nudi. Prilikom stvaranja računa na njihovoj službenoj stranici [7] potrebno je odabrati način pretplate. Razlog tomu jest što određeni Azure servisi nisu besplatni za korištenje. Postoji pretplatni model pod nazivom: „Pay as you go“ koja nudi određene servise besplatno do dvanaest mjeseci do određene razine korištenja.

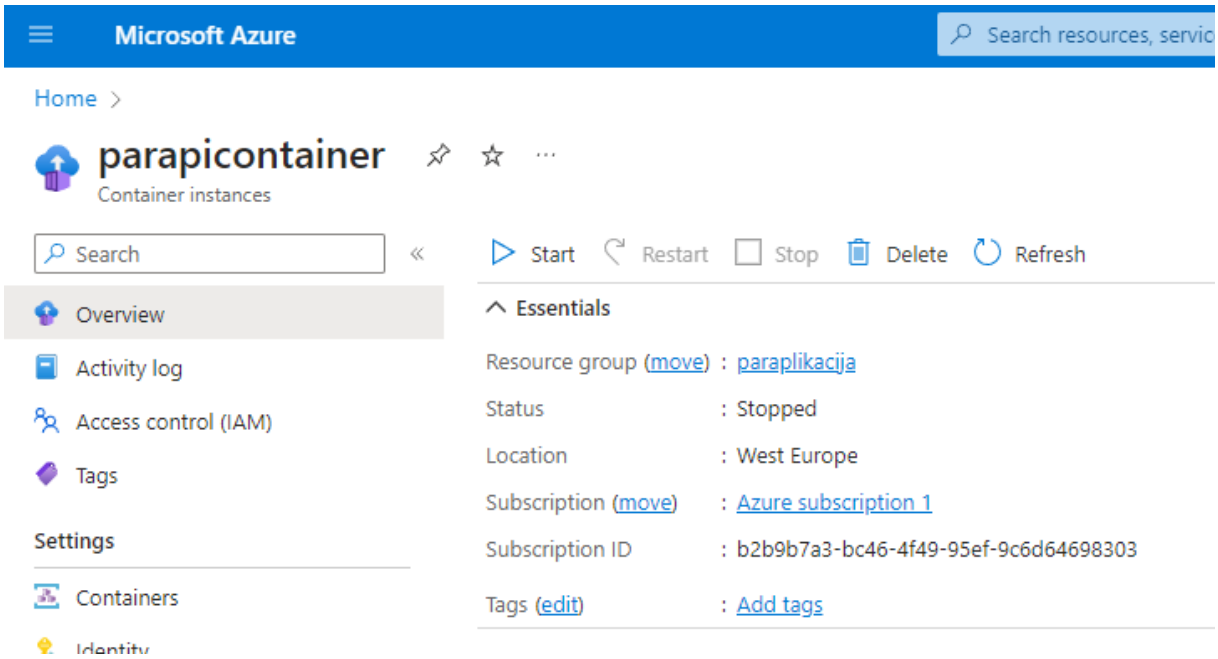
Za potrebe završnog rada na Microsoft Azuru korištena su dva servisa, a to su Azure kontejner i Azure blob servis. Azure kontejner nije besplatan servis i plaća se ovisno o tome koliko je kontejner dugo aktivan i kolika je jačina kontejnera u što se ubraja memorija i broj

jezgri. Azure blob storage također nije besplatan ali se ubraja u „Pay as you go“ i može se koristiti besplatno dvanaest mjeseci s do dvanaest gigabajta besplatnog prostora za pohranu podataka.

Prije nego što se stvori Azure kontejner potrebno je instalirati docker i ubaciti sliku (engl. *image*) aplikacije na docker hub.

Azure kontejner se može lako stvoriti. Potrebno mu je dodijeliti ime, pretplatu, DNS ime, sliku (engl. *image*) i veličinu kontejnera u memoriji i broju jezgri.

Nakon što se stvori novi kontejner potrebno je pričekati neko vrijeme da se postavi. Slika 3.3 prikazuje izgled sučelja za upravljanje novostvorenim kontejnerom. Kada se kontejner postavi potrebno ga je pokrenuti. Ukoliko sve prođe bez grešaka Azure će unutar odjeljka obavijesti javiti da je kontejner pokrenut. Sada se mogu slati API zahtjeve javno preko URL-a koji ovisi o DNS imenu koji je bio unesen prilikom stvaranja kontejnera.



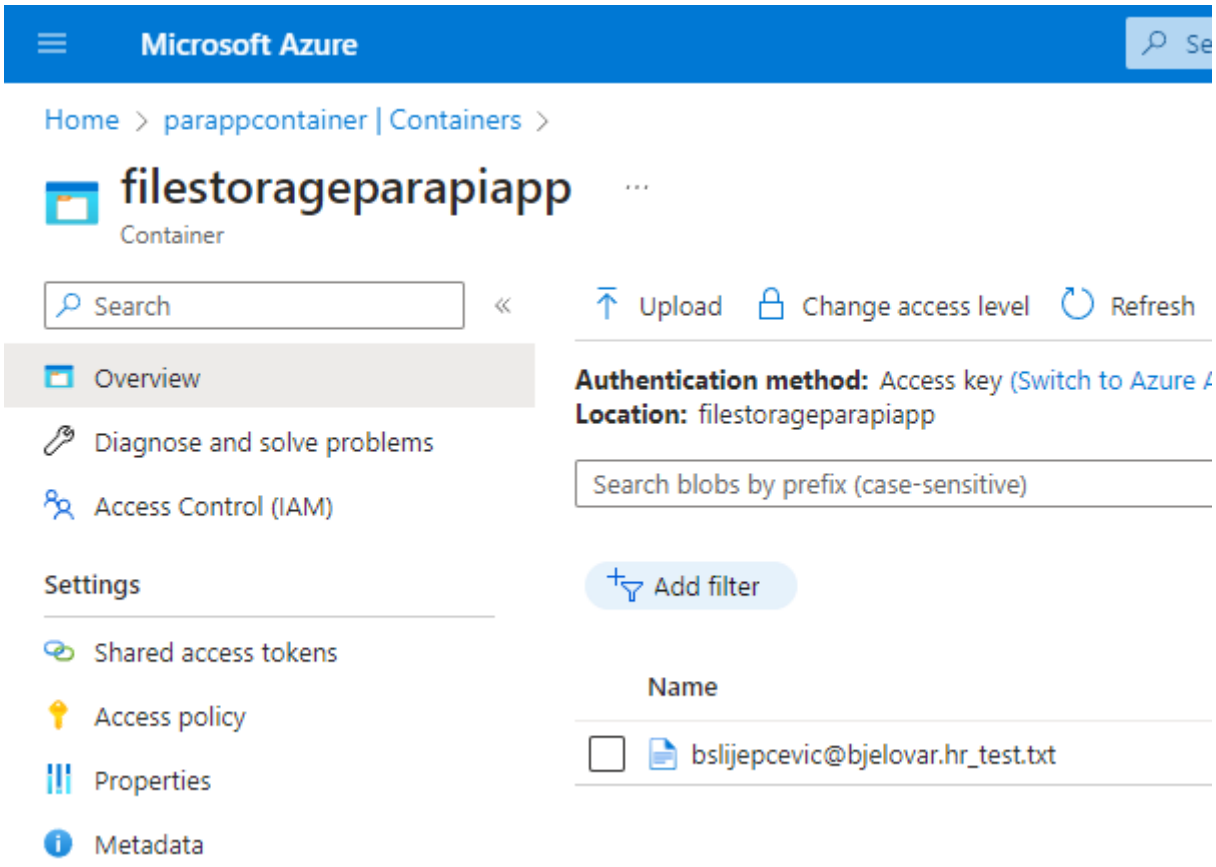
Slika 3.3: sučelje ekrana za upravljanje kontejnerom

3.2.2 *Azure blob servis*

Azure blob storage je usluga pohrane podataka temeljena na oblaku koju pruža Microsoft Azure. Dizajniran je za pohranu i upravljanje nestrukturiranim podacima, kao što su dokumenti, slike, videozapisi i slično [8].

Na sličan način na koji je stvoren Azure kontejner koji je spomenut u prethodnom odjeljku može se stvoriti i Azure Storage Account. Kao i kod kontejnera jednostavno se stvara i zahtijeva samo pretplatu, naziv, regiju, vrstu performanse i razinu redundancije podataka.

Novostvoreni Azure storage account nudi više servisa od kojih su najvažniji Blob servis i File servis. Blob servis će se koristiti za završni rad i služi za pohranu i preuzimanje podataka. File share služi za dijeljenje datoteka i nije potreban u svrhe završnog rada. Važno je napomenuti kako storage account još nudi queue servis, Table servis, security i networking servis. Slika 3.4 prikazuje sučelje ekrana za upravljanje Blob servisom unutar Azure storage account-a.



Slika 3.4: sučelje ekrana za pregled i upravljanje blob servisom

3.3 Implementiranje SendGrid servisa unutar aplikacije

Nakon što se stvori korisnički račun na SendGrid službenoj stranici [6] postoji odjeljak za opcije. Klikom na to otvara se padajući izbornik u kojemu se odabere Sender Authentication. Otvara se stranica na kojoj se može verificirati novog pošiljatelja. Za potrebe verificiranja pošiljatelja može se stvoriti novi elektronički poštanski račun. S verificiranog elektroničkog poštanskog računa kasnije se mogu slati e-pošte korisnicima unutar baze.

Sada je sve spremno i može se početi implementirati SendGrid servis unutar aplikacije [9]. To će se postići tako da se ode na odjeljak Email API. Odabere se Web API implementacija i na novom ekranu se odabere implementacija s C# programskim jezikom. Nakon ovih koraka dobivamo ekran koji sadrži detaljne upute kako implementirati SendGrid servis. Potrebno je prema uputama stvoriti ključ koji je poznat samo autoru aplikacije i nalazi se unutar koda i koristi se za stvaranje veze između aplikacije i SendGrid servisa. Također je potrebno instalirati paket za SendGrid servis unutar VS Code terminala pomoću naredbe: „dotnet add package SendGrid“. Sada je sve spremno za slanje elektroničke pošte preko aplikacije. Kako bi se poslala e-pošta potrebni su: ključ za stvaranje veze, verificirani pošiljatelj i primatelj.

4. OD MODELA DO OPERACIJA NAD TABLICOM U BAZI

Ovo poglavlje će pobliže objasniti kako stvoriti funkcionalno RESTful API okruženje krenuvši od modela s primjerom koda na Log modelu unutar aplikacije.

4.1 Model

Unutar .NET Web API-ja model predstavlja strukturu podataka koja se koristi za stvaranje tablice, kontrolera i prijenos podataka. Modeli se također koriste za serijalizaciju i deserijalizaciju podataka što omogućuje prijenos u različitim oblicima kao što su JSON ili XML. Programski kod 4.1 predstavlja primjer Log modela koji se koristi unutar aplikacije za logiranje pozvanih servisa ovisno o API zahtjevu.

Programski kod 4.1: Primjer modela unutar aplikacije

```
using System.ComponentModel.DataAnnotations;

namespace ParAPIApp.backend.Models;

public class Log {
    public int Id { get; set; }
    [Required]
    [MaxLength(10)]
    public string? Razina { get; set; } // razina loga (INFO, WARNING,
ERROR)
    [Required]
    public string? Poruka { get; set; } // Poruka koja se logira
    public int? KorisnikId { get; set; } // Korisnik koji je izvršio akciju
    public DateTime TimeStamp { get; set; }
}
```

Model također podržava validaciju podataka. Programski kod 4.1 sadrži linije koda [Required] i [MaxLength(10)]. Required govori da prilikom stvaranja novog zapisa taj podatak ne smije biti prazan, a MaxLength govori najveću veličinu podatka koja može biti spremljena na to mjesto.

4.2 Kontroler

Web API kontroler temeljna je komponenta koja obrađuje HTTP zahtjeve i vraća određeni HTTP odgovor. Služi kao okvir za izgradnju RESTful web usluga pomoću .NET-a.

Glavna karakteristika Web API kontrolera jest što sadrži HTTP krajnje točke kojima se pristupa preko različitih HTTP zahtjeva od kojih su glavne GET, POST, PUT i DELETE. Svaka krajnja točka ima svoj URI [10].

Kontroler se može jednostavno automatski stvoriti generiranjem kontrolera prema modelu kao što je opisano u odjeljku 3.1. Tako generiran kontroler sadrži svu logiku potrebnu za CRUD operacije nad modelom u bazi. Najprije je potrebno provjeriti je li novostvoreni model unesen unutar DbContext datoteke, ukoliko se nalazi potrebno je ažurirati bazu. Nakon ažuriranja baze stvara se tablica prema stvorenom modelu.

Unutar aplikacije također su korištene dodatne karakteristike kontrolera, a to su autentifikacija, autorizacija i rukovanje greškama. Kontroler može implementirati mehanizme za kontrolu pristupa određenim krajnjim točkama. Na ovaj način samo ovlašteni korisnici mogu pristupiti određenim resursima. Također kontroler može vraćati odgovore ovisno o grešci i također vratiti statusni kod koji može pobliže objasniti izvor greške. Programski kod 4.2 prikazuje primjer kontrolera unutar Web API aplikacije koji je povezan s Log modelom.

Programski kod 4.2: primjer kontrolera unutar aplikacije prema Log modelu

```
namespace ParAPIApp.backend.Controllers
{
    [ApiController]
    [Route("api/[controller]")]
    public class LogsController : ControllerBase
    {
        private readonly ILogService _logService;


        public LogsController(ILogService logService)
        {
            _logService = logService;
        }

        // GET: api/Logs
        [HttpGet, Authorize(Policy = "SamoAdmin")]
        public async Task<ActionResult<IEnumerable<Log>>> GetLogs()
        {
            return Ok(await _logService.GetLogs());
        }
    }
}
```

Programski kod 4.2 sadrži par zanimljivih informacija koje su bitne za API zahtjeve. [ApiController] je atribut koji označava klasu kao API kontroler što omogućuje određene karakteristike za izgradnju API-ja. [Route(„api/[controller]“)] je glavna ruta API zahtjeva za kontroler. GetLogs je akcijska metoda koja obrađuje HTTP GET zahtjev jer sadrži atribut [HttpGet]. Također sadrži atribut [Authorize(Policy = „SamoAdmin“)] što označava da pristup toj krajnjoj točki imaju korisnici koji su autorizirani sa zadanim pravilom. GetLogs metoda kao rezultat vraća status kod Ok sa zapisima koje mu _logService.GetLogs vrati. Kao što se vidi prema primjeru kontroler jedino obrađuje API zahtjeve i vraća određeni odgovor. Logika dohvaćanja podataka i komunikaciju s bazom odvija se u servisima koji vraćaju podatke kontrolerima ovisno o API zahtjevima.

4.3 Servis

Servis je jako bitna stavka API aplikacije. Oni uobičajeno sadrže svu poslovnu logiku aplikacije. Unutar završnog rada servisi komuniciraju s bazom te dohvaćaju, izmjenjuju ili brišu podatke ovisno o API zahtjevu. Servisi unutar API aplikacije omogućuju lakše upravljanje kodom, lakše testiranje koda i pregledniji kod. Također mogu upravljati greškama i vraćati kontroleru određeni odgovor ovisno o grešci po potrebi. Programski kod 4.3 prikazuje primjer log servisa unutar završnog rada.

Programski kod 4.3: primjer log servisa koji se poziva unutar log kontrolera

```
public class LogService : ILogService {
    private readonly AppDbContext _context;
    private readonly ILogiranKorisnikService _logiranKorisnikService;

    public LogService(AppDbContext context,
                      ILogiranKorisnikService logiranKorisnikService)
    {
        _context = context;
        _logiranKorisnikService = logiranKorisnikService;
    }

    public async Task<List<Log>> GetLogs()
    {
        return await _context.Log.ToListAsync();
    }

    public async Task<Log> CreateLog(Log log)
    {
        var korisnik = await _logiranKorisnikService.GetLoggedInUserInfo();
        if (korisnik != null) {
            log.KorisnikId = korisnik.Id;
        }
        else {
            log.KorisnikId = 0;
        }
        log.TimeStamp = DateTime.Now;
        _context.Log.Add(log);
        await _context.SaveChangesAsync();
        return log;
    }
}
```

5. ZAŠTITA PODATAKA

5.1 Hashiranje lozinke

Hashiranje lozinke je temeljna sigurnosna praksa koja se koristi za zaštitu osjetljivih korisničkih vjerodajnica. Hashiranje lozinke jest uzimanje čistog teksta lozinke i njezino pretvaranje u niz znakova fiksne duljine. Rezultat hashiranja je hash vrijednost. Hashiranje se obavlja pomoću određenog hash algoritma. Primarna svrha hashiranja jest da čak i ako su hash lozinke ugrožene haker ne može lako doći do čistog teksta lozinke.

Postoji dosta hash algoritama i glavne razlike između hash algoritama su u njihovoj brzini. Brzina algoritma uobičajeno je pokazatelj toga koliko je taj algoritam siguran. Ovisno o podatku koji se hashira treba razmisliti hoće li se radije koristiti brz ili spor algoritam. Ako se uzmu na primjer certifikati datoteka tu sigurnost nije toliko bitna i radi brzine rada bolje je iskoristiti široko korišteni algoritam SHA-256 koji je brz. Ukoliko je riječ o lozinki tu je bitna što veća sigurnost pa bi bilo bolje koristiti sporiji algoritam kao na primjer koji se koristi u završnom radu za hashiranje lozinke Bcrypt [11].

Primjer zašto je sporiji algoritam sigurniji jest što prvobitno uobičajeno znači da je njegov hash dulji odnosno kompliciraniji. Također znači da ako haker pokušava slučajnim stvaranjem hasheva pogodit hash sporiji algoritam će u jedinici vremena izračunat mnogo manje hasheva.

Bcrypt algoritam također sadrži takozvani faktor rada. Što veći faktor rada rezultira duljom i kompliciranijom hash vrijednosti. Naime to također znači da će hashiranje i potvrda hasha prilikom prijave dulje trajat. Stoga prilikom odabira faktora rada bitno je uzeti u obzir brzinu obrade podataka servera na kojem se aplikacija nalazi iz razloga što preveliki faktor rada može dovesti do dužeg vremena prijave u sustav. Programski kod 5.1 prikazuje klasu koja sadrži funkciju Hash za hashiranje lozinke i funkciju Verify koja služi za potvrdu lozinke prilikom prijave u sustav.

Programski kod 5.1: klasa za hashiranje i potvrdu lozinke

```
public class PasswordHasher
    {
        private const int WorkFactor = 12;

        public string Hash(string password)
        {
            return BCrypt.Net.BCrypt.HashPassword(password, WorkFactor);
        }

        public bool Verify(string passwordHash, string inputPassword)
        {
            return BCrypt.Net.BCrypt.Verify(inputPassword, passwordHash);
        }
    }
```

5.2 Data transfer object modeli

Dana transfer object modeli u kontekstu .NET razvoja koriste se za prijenos podataka između dijelova aplikacije. DTO modeli uobičajeno se koriste u kombinaciji s kodovima za pretvorbu objekata između modela domene i DTO-ova. Mapiranje jest kopiranje podataka iz modela domene u DTO i obrnuto. DTO modeli također mogu biti pojednostavljeni modeli od modela domene ili povezivati više modela domene ovisno o potrebi. DTO modeli na taj način olakšavaju upravljanje i održavanje aplikacije. DTO modele bi bilo dobro koristiti samo prema potrebi jer previše DTO modela može dovesti do težeg održavanja aplikacije.

Za potrebe završnog rada DTO modeli korišteni su za vraćanje samo bitnih podataka prilikom zvanja API zahtjeva. To doprinosi sigurnosti samih podataka unutar aplikacije. Programski kod 5.2 prikazuje primjer DTO modela prema korisničkom modelu unutar aplikacije. Kada bi se ovaj model usporedio s domenskim modelom korisnika vidjelo bi se da nedostaju informacije o tome je li račun potvrđen i lozinka.

Programski kod 5.2: primjer Data transfer object modela unutar aplikacije

```
public class KorisnikInfoDto {
    public int? Id { get; set; }
    public string? Ime { get; set; }
    public string? Prezime { get; set; }
    public string? Email { get; set; }
    public string? BrojMobitela { get; set; }
    public int? BrojBodova { get; set; }
    public string? Ovlast { get; set; }
    public string? NazivNapredovanja { get; set; }
    public string? SkolskaUstanova { get; set; }
    public bool? Aktivan { get; set; }
}
```

5.3 Zapisivanje pozvanih API zahtjeva

Zapisivanje (engl. *logging*) je važno kod bilo kakve aplikacije i igra ključnu ulogu u razvijanju aplikacije. Primarni razlog zapisivanja može biti radi testiranja i uklanjanja pogrešaka. Zapisi mogu dati ključne informacije gdje i kada je došlo do određene greške. Također pomaže kod sigurnosti. Tu zapisivanje može pomoći da uočimo neke sumnjive aktivnosti što je bitno kod održavanja sigurnosti i prevencije mogućih rizika unutar aplikacije. Također zapisivanje može pomoći kod analize brzine izvedbe API metoda i kod analize korisnika na način koje metode korisnici koriste. Programski kod 5.3 prikazuje primjer pozivanja metode za stvaranje zapisa unutar aplikacije. Prikazani kod zove logService servis i njegovu metodu CreateLog za stvaranje zapisa. Ovaj servis već je prethodno prikazan programskim kodom 4.3.

Programski kod 5.3: kod za poziv funkcije za stvaranje zapisa u bazi

```
await _logService.CreateLog(new Log() {Razina = "INFO",
      Poruka = "Pozvana metoda GetDatotekeBySkolskaUstanova."});
```

6. REGISTRACIJA I AUTORIZACIJA KORISNIKA

Ovo poglavlje će objasniti kod, metode i ostale zanimljivosti prilikom registracije korisnika u sustav. Također će biti opisan kod za prijavu korisnika u sustav i biti će objašnjen Jwt token koji se koristi za autorizaciju korisnika.

6.1 Registracija

Gotovo sve aplikacije i sustavi danas imaju neku vrstu registracije. Registracija se uobičajeno odnosi na stvaranje korisničkog računa. Registracija uobičajeno zahtijeva unos korisničkog imena, lozinke i adrese e-pošte. Dodatne usluge mogu također tražiti unos imena, prezimena i datum rođenja. Web stranice koje imaju nekakvu poslovnu logiku i uobičajeno su orijentirane prema nekom obliku trgovine ili usluga također mogu tražiti podatke o tvrtki u kojoj radimo i podatke o kartici.

Unutar završnog rada potrebne informacije prilikom registracije su ime i prezime, e-pošta, lozinka i OIB. Kako bi se korisnik unio u bazu potrebno je pozvati POST API zahtjev na api/korisnici krajnju točku. API kontroler zatim poziva AddKorisnik funkciju koja se nalazi unutar servisa pod nazivom KorisniciService.

Prije nego što se korisnik unese u bazu prvo se provjerava valjanost unesenih podataka. Provjere unesenih podataka uključuje provjeru jesu li poslani svi bitni podaci, unikatnost e-pošte, ispravnost lozinke, unikatnost i ispravnost OIB-a. Ako su podaci ispravni stvara se objekt korisnika koji se zatim unosi u bazu. Nakon unosa u bazu šalju se e-pošte na unesenu e-poštu i e-poštu voditelja ustanove ako osoba ima postavljeno da je član ustanove i ako ta ustanova ima voditelja. Na kraju servis vraća kontroleru informacije o novostvorenom korisniku koje zatim API kontroler vraća kao odgovor. Ukoliko dođe do određene greške ili pogreške prilikom unosa servis vraća određene odgovore kako bi se lakše utvrdilo gdje i zašto je došlo do greške. Programski kod 6.1 prikazuje dio API kontrolera koji se poziva prilikom registracije novog korisnika dok programski kod 6.2 prikazuje kod servisa koji se poziva unutar API kontrolera prilikom registracije. Kod 6.1 također prikazuje različite dohvate grešaka koje vraćaju informaciju o grešci koju dobiva od servisa. Programski kod 6.3 prikazuje funkciju koja se poziva nakon stvaranja korisnika za slanje verifikacijske e-pošte.

Programski kod 6.1: programski kod kontrolera za unos novog korisnika

```
// POST: api/Korisnici
[HttpPost]
public async Task<ActionResult<KorisnikInfoDto>>
PostKorisnik(KorisnikRegisterModel korisnik)
{
   try {
     KorisnikInfoDto result = await _korisniciService.AddKorisnik(korisnik);
     return StatusCode(StatusCodes.Status201Created, result);
       }
       catch (NeispravanUnosException e) {
          return BadRequest(e.Message);
       }
       catch (SkolskaUstanovaNijePronadenaException e) {
          return NotFound(e.Message);
       }
}
```

Programski kod 6.2: dio koda unutar servisa za provjeru unesene lozinke

```
bool sadrziVelikoSlovo = false;
bool sadrziMaloSlovo = false;
bool sadrziBroj = false;
bool sadrziSpecijalniZnak = false;
foreach (char znak in korisnik.Lozinka) {
         if (char.IsUpper(znak)) {
             sadrziVelikoSlovo = true;
         }
         if (char.IsLower(znak)) {
             sadrziMaloSlovo = true;
         }
         if (char.IsDigit(znak)) {
             sadrziBroj = true;
         }
         if (char.IsSymbol(znak) || char.IsPunctuation(znak)) {
             sadrziSpecijalniZnak = true;
         }
}
if (!sadrziVelikoSlovo || !sadrziMaloSlovo || !sadrziBroj ||
!sadrziSpecijalniZnak) {
         throw new NeispravanUnosException("Lozinka mora sadržavati veliko
slovo, malo slovo, broj i specijalni znak.");
        }
if (korisnik.Lozinka.Length < 8) {
            throw new NeispravanUnosException("Lozinka mora sadržavati
minimalno 8 znakova.");
        }
```

Programski kod 6.3: funkcija za slanje verifikacijske e-pošte

```
public async Task SendVerificationEmail(string destinationEmail, string
verificationLink)
    {
        var apiKey = "randomStringDobivenSaSendGrida";
        var client = new SendGridClient(apiKey);
        var from = new EmailAddress("parapplicationservice@gmail.com", "Par
Aplikacija");
        var to = new EmailAddress(destinationEmail);
        var subject = "Potvrda računa";
        var plainTextContent = "Molimo vas kliknite na poveznicu kako bi ste
potvrdili račun: " + verificationLink;
        var htmlContent = "<p>Molimo vas kliknite na poveznicu kako bi ste
potvrdili račun:</p><p><a href='" + verificationLink + "'>Potvrdi
račun</a></p>";
        var msg = MailHelper.CreateSingleEmail(from, to, subject,
plainTextContent, htmlContent);
        var response = await client.SendEmailAsync(msg);
    }
```

6.2 Prijava (autentifikacija) korisnika u sustav

Prijava u sustav je postupak koji je potrebno obaviti na većini internetskih stranica radi dobivanja pristupa određenim pravima i uslugama. Autentifikacija korisnika je vezana uz prijavu i osigurava da osoba koja traži pristup određenoj usluzi zaista jest ona kojom se predstavlja. Prijava korisnika u sustav uobičajeno zahtijeva samo korisničko ime i lozinku. Također postoje uobičajene alternative gdje se korisničko ime zamijeni s e-poštom ili je postavljeno da se može prijaviti s korisničkim imenom i lozinkom ili e-poštom po izboru. Kada se korisnici prijave stvara se neki oblik sjednice (engl. *session*) koji su uobičajeno spremljene informacije u kolačiće sjednice (engl. *session*) ili korištenje tokena. Upravljanje sjednicama (engl. *session*) na ovaj način pomaže da osoba može pristupiti svim uslugama sustava na određeno vrijeme bez dodatnog ometanja ili ponovnog traženja prijave od korisnika.

Prijava u sustav uobičajeno je mjesto kibernetičkih napada stoga je izrazito bitno da prijava korisnika u sustav bude što sigurnija. Najbolji načini zaštite kod prijave su korištenje sigurne komunikacije gdje se podaci šifriraju i zaključavanje računa prilikom više neuspjelih pokušaja prijave.

Unutar završnog rada za prijavu potrebno je unijeti e-poštu i lozinku. Nakon što se unesu podaci provjerava se jesu li uneseni podaci ispravni i provjerava se postoji li aktivan račun s zadanom e-poštom. Ukoliko je pronađen račun verificira se unesena lozinka s spremljenom hashiranom lozinkom unutar baze. Nakon što se verificira lozinka stvara se JWT token koji će biti detaljno objašnjen u idućem odjeljku. Programski kod 6.4 prikazuje dio koda koji poziva funkciju za generiranje tokena ukoliko verificiranje lozinke uspješno prođe.

Programski kod 6.4: dio koda za prijavu koji poziva funkciju za generiranje tokena

```
var result = _passwordHasher.Verify(user.LozinkaHash, model.Password);
if (!result)
{
      throw new KorisnikNijePronadenException("Email ili lozinka nisu točni.
Provjerite unesene podatke i pokušajte ponovno.");
}
var tokenString = GenerateJwtToken(user);
```

6.3 JSON web token

JSON web token kompaktan je i siguran način prijenosa informacija između dviju strana. Uobičajeno se koristi za provjeri autentičnosti i autorizaciju u web aplikacijama i API-jima. JWT-ovi su uobičajeno digitalno potpisani što osigurava integritet informacija pohranjen u njima [12].

JWT token sastoji se od zaglavlja, korisnog tereta i potpisa. Zaglavlje sadrži informacije o vrsti tokena i korištenom algoritmu potpisivanja. Korisni teret sadrži uobičajene informacije o korisniku za kojeg je token generiran. Potpis se koristi za sigurnosnu provjeru u slučaju neovlaštene izmjene tokena.

Dodatni sigurnosni koraci pri radu s JWT tokenom. Osjetljive informacije unutar tokena se mogu spremiti unaprijed šifrirane jer sam JWT token može lako biti dekodiran. Tokeni također mogu imati određeno vrijeme trajanja i ukoliko token istekne, on više nije ispravan. Programski kod 6.5 prikazuje kod za generiranje JWT tokena nakon što prijava u sustav prođe ispravno.

Programski kod 6.5: funckija za generiranje JWT tokena

```
private string GenerateJwtToken(Korisnik user)
        {
            var tokenHandler = new JwtSecurityTokenHandler();
            var key = Encoding.ASCII.GetBytes(_configuration["Jwt:Key"]);

            var tokenDescriptor = new SecurityTokenDescriptor
            {
                Subject = new ClaimsIdentity(new[]
                {
                    new Claim("UserId", user.Id.ToString()),
                    new Claim(ClaimTypes.Email, user.Email),
                    new Claim(ClaimTypes.Role, user.OvlastId.ToString())
                }),
                Expires = DateTime.UtcNow.AddHours(1),
                Issuer = "https://parapiapp.azurewebsites.net",
                Audience = ".NET Core Web API for PAR",
                SigningCredentials = new SigningCredentials(new
SymmetricSecurityKey(key), SecurityAlgorithms.HmacSha256Signature)
            };

            var token = tokenHandler.CreateToken(tokenDescriptor);
            return tokenHandler.WriteToken(token);
        }
```

7. UPRAVLJANJE DATOTEKAMA S AZUROM

U ovom poglavlju biti će opisan kod zaslužan za spremanje i preuzimanje datoteka uz pomoć Azure-ovog blob spremnik servisa.

Azure blob spremnik nudi mogućnost stvaranja više naziva kontejnera za pohranu datoteka ovisno o potrebi. Azure blob spremnik nalazi se u oblaku (engl. *Cloud service*). Za potrebe završnog rada korišten je jedan kontejner za spremanje datoteka. Programski kod 7.1 prikazuje kod za učitavanje datoteke na Azure blob spremnik dok programski kod 7.2 prikazuje kod za preuzimanje datoteke s Azure blob spremnika.

Datoteke prije učitavanja prvo prolaze kroz provjere o ispravnosti naziva datoteka i je li nastavak datoteke podržan. Ukoliko ne prođe provjeru dolazi do greške koja se vraća i tekst koji pobliže objašnjava što nije bilo točno predano za učitavanje datoteke.

Prilikom preuzimanja datoteke unutar završnog rada prvo se pregledava postoji li datoteka prema zadanim kriterijima. Ukoliko postoji provjerava se ovlast računa preko kojega se traži preuzimanje datoteke. Korisnici s običnim pravima mogu preuzeti samo vlastite datoteke, voditelj ustanove ima pravo preuzet datoteke od svih korisnika unutar njihove ustanove i admin ima prava preuzeti bilo koju datoteku.

Programski kod 7.1: dio koda zaslužan za učitavanje datoteka na Azure blob spremnik

```
var connectionString =
_configuration.GetConnectionString("AzureBlobStorage");

Debug.WriteLine($"Connection String: {connectionString}");
var containerName = "filestorageparapiapp";

BlobServiceClient blobServiceClient = new
BlobServiceClient(connectionString);

lobContainerClient containerClient =
blobServiceClient.GetBlobContainerClient(containerName);

using (var stream = file.OpenReadStream())
{
   await blobClient.UploadAsync(stream, true);
}
```

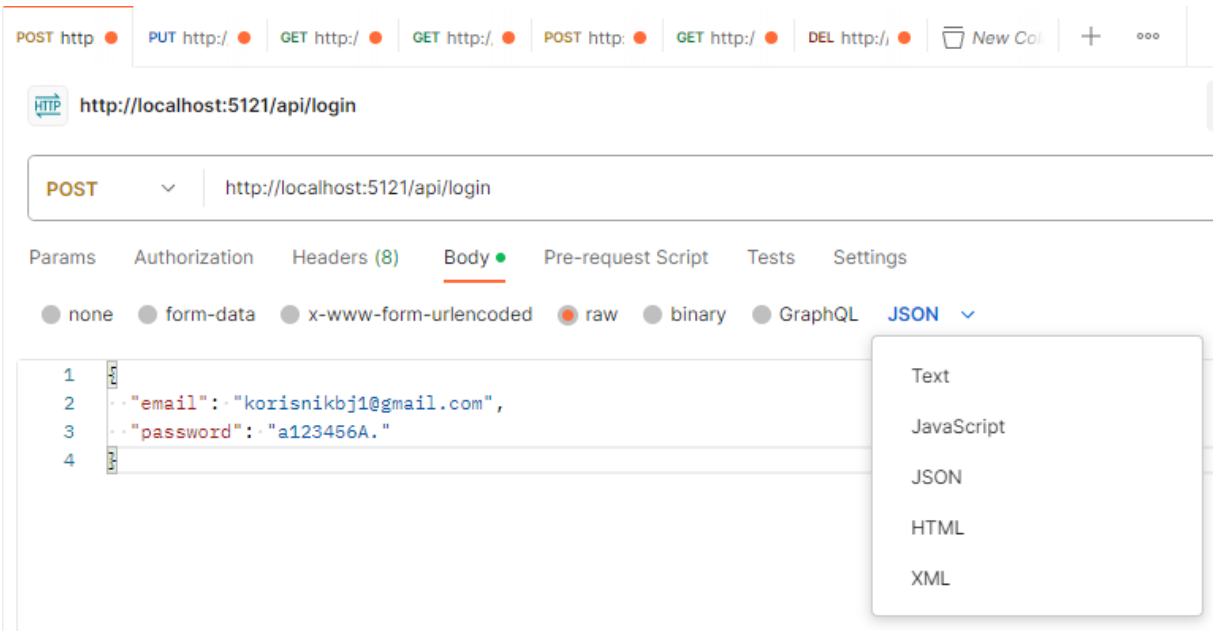
Programski kod 7.2: programski kod za preuzimanje datoteke s Azure blob spremnika

```
var connectionString =
_configuration.GetConnectionString("AzureBlobStorage");

var containerName = "filestorageparapiapp";
BlobServiceClient blobServiceClient = new
BlobServiceClient(connectionString);

BlobContainerClient containerClient =
blobServiceClient.GetBlobContainerClient(containerName);

BlobClient blobClient = containerClient.GetBlobClient(dbDatoteka.Naziv);
var blobStream = await blobClient.OpenReadAsync();

return File(blobStream, blobClient.GetProperties().Value.ContentType,
dbDatoteka.Naziv);
```

8. TESTIR ANJE APLIKACIJE POMOĆU POSTMANA

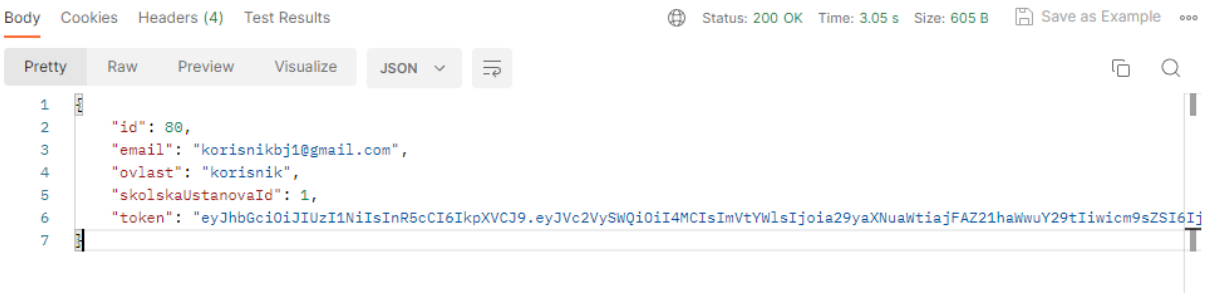
Ovo poglavlje opisuje korištenje Postman aplikacije za testiranje .NEZ Web API aplikacije i njenih API ruta.

Većina API ruta unutar završnog rada zahtijeva autorizaciju korisnika prilikom poziva HTTP zahtjeva. Iz toga razloga prvo će se poslati HTTP zahtjev za prijavu na rutu /api/login i unutar zahtjeva potrebno je uključiti tijelo (engl. *Body*). Unutar tijela prilikom prijave preporuka je koristiti opciju *raw* i u plavom padajućem izborniku odabrati JSON format podataka. Slika 8.1 prikazuje spomenuti postupak.



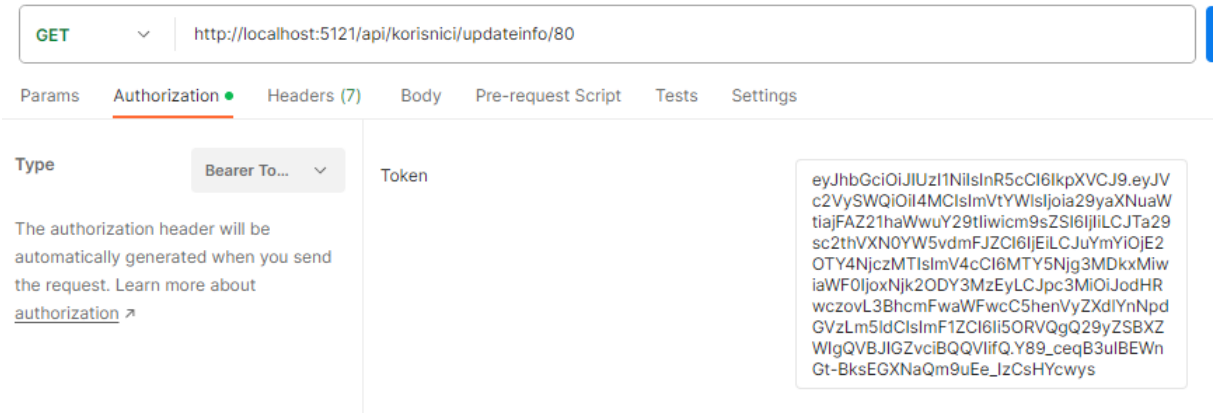
Slika 8.1: API zahtjev za prijavu korisnika

Ukoliko su unesena email adresa i lozinka ispravni kao rezultat dobiva se model koji sadrži informacije o korisniku s generiranim tokenom što je prikazano na slici 8.2.

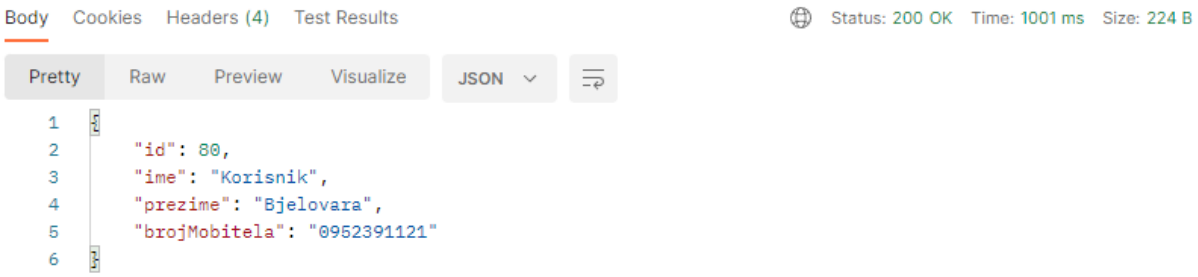


Slika 8.2: Rezultat API zahtjeva prilikom ispravne prijave

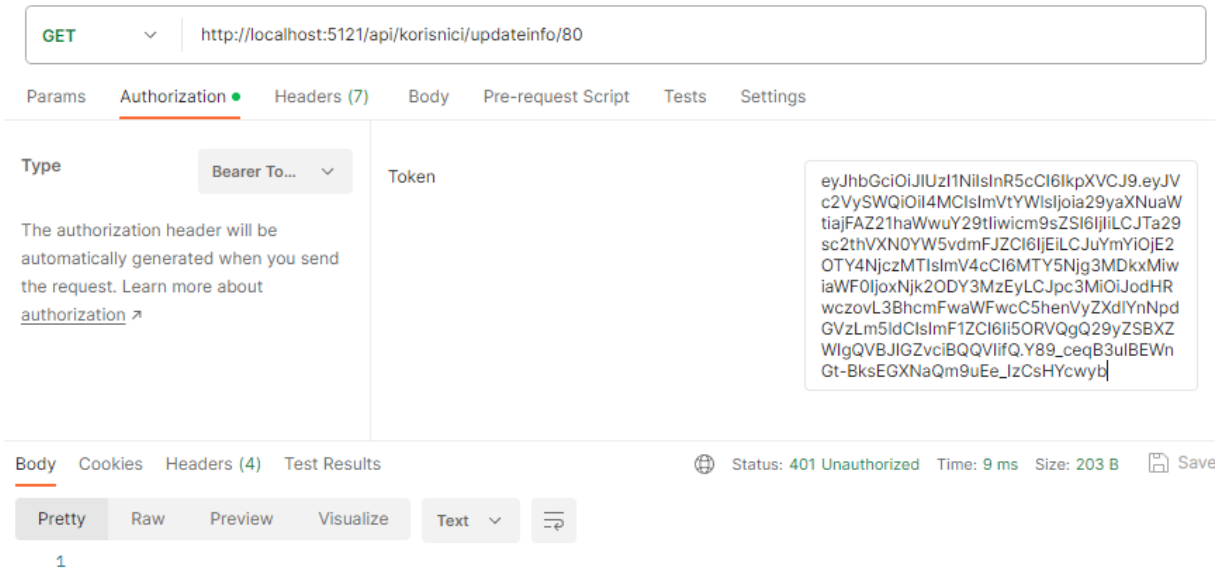
Pomoću generiranog tokena sada korisnik ima pristup puno više API ruta. Kako bi se pristupilo rutama unutar završnog rada za koje je potreban token to se može uraditi tako da se unutar odjeljka *Authorization* odabere tip autorizacije pod nazivom *Bearer Token* i u polje za unos teksta prekopira token koji smo dobili kao rezultat na slici 8.2. Slika 8.3. prikazuje izgled Postman zahtjeva koji zahtijeva autorizaciju. Slika 8.4 prikazuje odgovor API zahtjeva ukoliko je uneseni token ispravan, a slika 8.5 prikazuje odgovor API zahtjeva ukoliko token nije ispravan. Na slici 8.5 je vidljivo kako je samo zadnje slovo unutar tokena promijenjeno iz s u b te token odmah više nije ispravan i API javlja kako korisnik nema prava pristupa podacima.



Slika 8.3: Izgled API poziva za kojega je potrebna autorizacija pomoću tokena



Slika 8.4: Rezultat API poziva ukoliko je token ispravan



Slika 8.5: Rezultat ukoliko token nije unesen ili token nije ispravan

9. ZAKLJUČAK

Ovim se radom istražuje .NET Web API aplikacija i implementacija MySQL baze podataka, Azure kontejnera za javno objavljivanje koda, Azure blob storage-a za pohranu datoteka, SendGrid servisa za slanje validacijskih e-pošti i alata za testiranje API poziva kao što je Postman. Cilj rada bio je stvoriti ispravan .NET sustav koristeći sve spomenute tehnologije, alate i servise uzimajući u obzir određenu poslovnu logiku i implementaciju sigurnosnih koraka.

Kao rezultat istraživanja postignut je razvijen .NET sustav koji koristi sve spomenute tehnologije, alate i servise. Postignuta ja API aplikacija koja unutar sebe sadržava poslovnu logiku koja dalje može samo rasti. Korištenjem modela i Entity Framework-a stvorene su uspješne relacijske tablice unutar MySQL baze podataka. Također su stvoreni jednostavni kontroleri za učinkovito upravljanje podacima unutar tablica. Korištenjem Azure kontejnera postignuto je uspješno javno objavljivanje najnovijih API metoda unutar aplikacije koje zatim korisnici mogu koristiti po potrebi. Pomoću Azure blob storage servisa postignuto je spremanje i preuzimanje datoteka s oblaka po određenim kriterijima. Korištenjem SendGrid API zahtjeva uspješno je implementirano slanje verifikacijske e-pošte kako bi se spriječilo stvaranje lažnih računa. Alati Postman i nastavak ThunderClient unutar VS Coda uspješno su korišteni za testiranje API poziva prije javnog objavljivanja istih.

Ovaj .NET sustav sa svim spomenutim implementiranim sigurnosnim koracima i dalje nije potpuno siguran. Važno je napomenuti kako se za API pozive koristi ne siguran HTTP protokol koji se može zamijeniti s veoma sigurnim HTTPS protokolom. Prednost ovog .NET sustava jest što je funkcionalan, učinkovit i podržava određenu poslovnu logiku.

Uporaba ovog .NET sustava zahtijeva određeno stručno i poslovno znanje za učinkovito korištenje i održavanje aplikacije.

Ovaj rad pruža primjer kako stvoriti funkcionalni sustav sa različitim implementacija tehnologija, alata i servisa. Izrada sustava korištenjem više različitih alata i tehnologija pruža razne mogućnosti izvedbe, implementacije i nadogradnje sustava. Korištenje različitih alata unutar različitih dijelova sustava doprinosi većoj učinkovitosti izrade i korištenja sustava.

10. LITERATURA

[1] Microsoft Corporation. Documentation for Visual Studio Code [Online]. 2023. Dostupno na: https://code.visualstudio.com/docs. (12.9.2023.)

[2] Entity Framework Tutorial. What is Entity Framework? [Online]. 2023. Dostupno na: https://www.entityframeworktutorial.net/what-is-entityframework.aspx. (12.9.2023.)

[3] Simplilearn Solutions. What is Git: Features, Commands, Branch and Workflow in Git [Online]. 2023. Dostupno na: https://www.simplilearn.com/tutorials/git-tutorial/what-is-git. (13.9.2023.)

[4] Oracle Corporation. MySQL :: MySQL 8.0 Reference Manual :: 1.2.1 What is MySQL? [Online]. 2023. Dostupno na: https://dev.mysql.com/doc/refman/8.0/en/what-is-mysql.html. (14.9.2023.)

[5] Postman, Inc. Postman API Platform [Online]. 2023. Dostupno na: https://www.postman.com/. (15.9.2023.)

[6] Twillio Inc. Email Delivery, API, Marketing Service | SendGrid [Online]. 2023. Dostupno na: https://sendgrid.com/. (16.9.2023.)

[7] Microsoft Corporation. Cloud Computing Services | Microsoft Azure [Online]. 2023. Dostupno na: https://azure.microsoft.com/en-us. (19.9.2023.)

[8] Microsoft Corporation. Azure Blob Storage | Microsoft Azure [Online]. 2023. Dostupno na: https://azure.microsoft.com/en-us/products/storage/blobs. (19.9.2023.)

[9] Twillio Inc. Email API Quickstart for C# [Online]. 2023. Dostupno na: https://docs.sendgrid.com/for-developers/sending-email/email-api-quickstart-for-c. (21.9.2023.)

[10] Microsoft Corporation. Create web APIs with ASP.NET Core [Online]. 2023. Dostupno na: https://learn.microsoft.com/en-us/aspnet/core/web-api/?view=aspnetcore-7.0. (23.9.2023.)

[11] Nord Security. What is Bcrypt and how does it work? [Online]. 2023. Dostupno na: https://nordvpn.com/blog/what-is-bcrypt/. (25.9.2023.)

[12] Okta. JSON Web Token Introduction [Online]. 2023. Dostupno na: https://jwt.io/introduction. (27.9.2023.)

11. POPIS OZNAKA I KRATICA

VS Code – *Visual Studio Code*

SQL – *Structured Query Language* (strukturni upitni jezik)

API – *Application Programming Interface* (aplikacijsko programsko sučelje)

HTTP – *Hyper Text Transfer Protocol*

HTTPS – *Hyper Text Transfer Protocol Secure*

MSSQL – *Microsoft SQL Server*

LINQ – *Language Integrated Query*

REST API – *RESTful API*

JSON – *JavaScript Object Notation*

HTML – *HyperText Markup Language*

XML – *Extensible Markup Language* (jezik za označavanje podataka)

URL – *Uniform Resource Locator*

DNS – *Domain Name System* (sustav domenskih imena)

URI – *Uniform Resource Identifier*

CRUD – *CREATE, READ, UPDATE and DELETE*

DTO – *Dana Transfer Object*

OIB – Osobni identifikacijski broj

12. SAŽETAK

Naslov: .NET sustav za prikupljanje i pohranu uređenih podataka u azure okruženju s MySQL bazom podataka

Ovaj završni rad govori o izradi .NET sustava koji uključuje .NET Web API aplikaciju, MySQL bazu podataka, Azure kontejner, Azure blob spremnik i SendGrid servis. Također se govori o načinu testiranja API ruta pomoću aplikacije Postman i ekstenzije unutar VS Coda ThunderClient. Završni rad također opisuje izradu modela, kontrolera i servisa unutar .NET aplikacije pomoću kojih se mogu upravljati podaci unutar baze i opisane su mjere zaštite podataka unutar aplikacije. Unutar završnog rada opisan je način, provjere i sigurnosne mjere prilikom registracije i prijave korisnika. Završni rad koristi autorizaciju korisnika za pristup većini API ruta preko JSON web tokena koji je također opisan unutar završnog rada.

Ključne riječi: sustav, aplikacija, mode, kontroler, servis, token

13. ABSTRACT

Title: .NET system for collecting and storing structured dana ina n Azure enviroment with a MySQL database.

This thesis is about creating a .NET system that includes a .NET Web API application, a MySQL database, an Azure container, an Azure blob container and SendGrid service. It also shows how to test API routes using the Postman application and the extension within VS Code ThunderClient. The final thesis also describes the creation of models, controllers and services within the .NET application that can be used to manage dana within the database. Security measures within the .NET system with examples inside the system are also shown and described. It also describes the method, checks and security measures taken during user registration and login. .NET system uses user authorization to access most API routes via a JSON web token which is described within the thesis.

Keywords: system, application, method, controller, service, token

IZJAVA O AUTORSTVU ZAVRŠNOG RADA

Pod punom odgovornošću izjavljujem da sam ovaj rad izradio/la samostalno, poštujući načela akademske čestitosti, pravila struke te pravila i norme standardnog hrvatskog jezika. Rad je moje autorsko djelo i svi su preuzeti citati i parafraze u njemu primjereno označeni.

Mjesto i datum	Ime i prezime studenta/ice	Potpis studenta/ice
U Bjelovaru, 29.09.2023.	Luka Frčko	Luka Frčko

U skladu s čl. 58, st. 5 Zakona o visokom obrazovanju i znanstvenoj djelatnosti, Veleučilište u Bjelovaru dužno je u roku od 30 dana od dana obrane završnog rada objaviti elektroničke inačice završnih radova studenata Veleučilišta u Bjelovaru u nacionalnom repozitoriju.

Suglasnost za pravo pristupa elektroničkoj inačici završnog rada u nacionalnom repozitoriju

Luka Frčko
ime i prezime studenta/ice

Dajem suglasnost da tekst mojeg završnog rada u repozitorij Nacionalne i sveučilišne knjižnice u Zagrebu bude pohranjen s pravom pristupa (zaokružiti jedno od ponuđenog):

a) Rad javno dostupan
b) Rad javno dostupan nakon _____ (upisati datum)
c) Rad dostupan svim korisnicima iz sustava znanosti i visokog obrazovanja RH
d) Rad dostupan samo korisnicima matične ustanove (Veleučilište u Bjelovaru)
e) Rad nije dostupan

Svojim potpisom potvrđujem istovjetnost tiskane i elektroničke inačice završnog rada.

U Bjelovaru, 29.09.2023.

Luka Frčko
potpis studenta/ice