

Univerzitet u Beogradu  
Fakultet organizacionih nauka  
Katedra za elektronsko poslovanje

# DOKUMENTACIJA ZA SEMINARSKI RAD

Cloud infrastruktura i servisi

Rbr	Ime	Prezime	Broj indeksa
1.	Aleksa	Živković	2022/0132
2.	Dušanka	Živanović	2022/0073
3.	Milan	Lalić	2022/0142

Beograd, 2024.

## Sadržaj

1	Korisnički zahtev .....	3
2	Opis tehnologija korišćenih u aplikaciji.....	4
2.1	React .....	4
2.2	Node.js .....	4
2.3	MySQL.....	4
2.4	Docker.....	5
3	Korisničko uputstvo.....	5
4	Objašnjenja karakterističnih delova koda.....	9
4.1	Frontend/public/js/form.js.....	9
4.2	Frontend/public/js/login.js.....	10
4.3	Backend/app.js.....	11
4.4	Db/init.sql.....	13
4.5	frontend/public/HTML i CSS .....	14
5	Prikaz procesa kontejnerizacije/dokerizacije i njegovih atributa .....	19
5.1	Detaljno objašnjenje compose.yaml fajla .....	19
5.2	Frontend/dockerfile.....	23
5.3	Backend/dockerfile.....	24
5.4	Postupak kontejnerizacije, pokretanja i ažuriranja aplikacije .....	24

## 1 Korisnički zahtev

Potrebno je napraviti web aplikaciju za piceriju Mamma Mia's pizza. Web aplikacija treba da predstavi piceriju korisnicima, da ih upozna sa menijem, istorijatom i katalogom picerije. Treba da pruži informacijski sistem za poručivanje pizza, gde će korisnici moći da izaberu vrstu i veličinu pize, kao i da popune lične podatke za dostavu. Picerija će preko svog interfejsa imati uvid u sve napravljene porudžbine, kako bi ih što pre servisirala. Sajt treba da ima 6 web stranica (Početna, O nama, Menu, Katalog, Kontakt, Login) od kojih svaka odgovara jednoj funkciji web sajta. Treba napraviti "Početnu" stranu sa koje se može klikom na dugme preći na stranicu sa slikama različitih pica (Menu). U desnom zaglavlju stranice je potrebno omogućiti brzi pristup svim web stranicama sajta. Stranica "O nama" treba da opisuje istorijat picerije, način pripremanja pizza i da ima sliku njenog osnivača. Stranica "Katalog" treba da prikaže fotografije svih pizza koje picerija ima u ponudi. Stranica "Menu" bi trebalo da prikazuje cene za različite veličine pica. Na stranici "Kontakt" treba omogućiti kupcima izbor vrste i veličine pize, kao i formu za ostavljanje ličnih podataka potrebnih za dostavu. Potrebno je napraviti dva dugmeta, jedno za poručivanje, a drugo za poništavanje porudžbine. Stranica "Login" nije namenjena kupcima, već isključivo administratoru informacijskog sistema picerije i na njoj posle logovanja treba da bude prikazana istorija porudžbina.

## 2 Opis tehnologija korišćenih u aplikaciji

Za razvoj ovog sajta korišćene su tehnologije React (frontend), Node.js (backend), MySQL (baza) i Docker (kontejnerizacija).

### 2.1 React

React je JavaScript biblioteka koja se koristi za izgradnju korisničkih interfejsa, posebno za web aplikacije. Razvila ju je Facebook i open-source zajednica. React omogućava programerima da kreiraju velike aplikacije sa dinamičkim podacima, koristeći komponentnu arhitekturu. Glavna ideja je da se aplikacija sastoji od malih, ponovno upotrebljivih delova (komponenti), koje se mogu kombinovati kako bi se izgradio kompleksniji interfejs. React koristi JSX (JavaScript XML) sintaksu koja omogućava pisanje HTML-a unutar JavaScript koda, što olakšava rad sa komponentama. Takođe, React omogućava efikasno upravljanje stanjem aplikacije kroz upotrebu jednosmerne veze podataka i može se integrisati sa drugim bibliotekama i framework-ovima.

### 2.2 Node.js

Node.js je serverska platforma bazirana na JavaScript-u, koja omogućava izvršavanje server-side aplikacija. Razvijena je na Chrome-ovom V8 JavaScript engine-u i popularizovana kao open-source projekat. Node.js se ističe svojom asinkronom (non-blocking) i event-driven arhitekturom, što omogućava visoku efikasnost i skalabilnost aplikacija. Omogućava programerima da koriste isti jezik (JavaScript) i na serverskoj strani aplikacija, što olakšava deljenje koda između front-end i back-end delova aplikacija. Node.js koristi asinkroni I/O model koji omogućava efikasno upravljanje operacijama koje zahtevaju dugotrajno izvršavanje, kao što su čitanje/pisanje fajlova ili pristup bazi podataka. Ovo doprinosi bržem odzivu aplikacija i boljoj iskorišćenosti resursa. Node.js podstiče modularni pristup razvoju aplikacija kroz Node Package Manager (npm), koji je jedan od najvećih registara softverskih paketa. Ovo omogućava laku integraciju različitih funkcionalnosti i komponenti u Node.js aplikacije. Baziran je na event-driven arhitekturi, gde se aplikacija gradi oko događaja (events) i njihovih callback funkcija. Ova arhitektura je idealna za aplikacije koje zahtevaju brz odziv na više istovremenih zahteva, kao što su real-time aplikacije (npr. chat aplikacije, streaming servisi). Node.js se često koristi za izradu serverskih aplikacija, API-ja, mikroservisa, web servera i drugih aplikacija koje zahtevaju efikasno i skalabilno obrtanje zahteva.

### 2.3 MySQL

MySQL je open-source sistem za upravljanje bazama podataka koji se koristi za skladištenje i upravljanje podacima u različitim vrstama aplikacija. Ovaj sistem koristi SQL (Structured Query Language) za manipulaciju podacima i pristup bazi. MySQL je relacionalna baza podataka, što znači da podaci u njoj postoje u obliku tabela koje su međusobno povezane prema definisanim relacijama. Koristi SQL za kreiranje, upravljanje i manipulaciju podacima. SQL je standardni jezik koji omogućava korisnicima da izvrše različite operacije nad bazom podataka, kao što su upiti (queries), dodavanje novih podataka, ažuriranje postojećih podataka i brisanje podataka. MySQL podržava transakcije, što omogućava izvršavanje grupa operacija nad bazom podataka u okviru jedne transakcije. Ovo obezbeđuje konzistentnost podataka i integritet baze. Često se koristi u web aplikacijama, e-commerce platformama, sistemima za upravljanje sadržajem, aplikacijama za analizu podataka i mnogim drugim vrstama aplikacija koje zahtevaju pouzdanu i efikasnu bazu podataka.

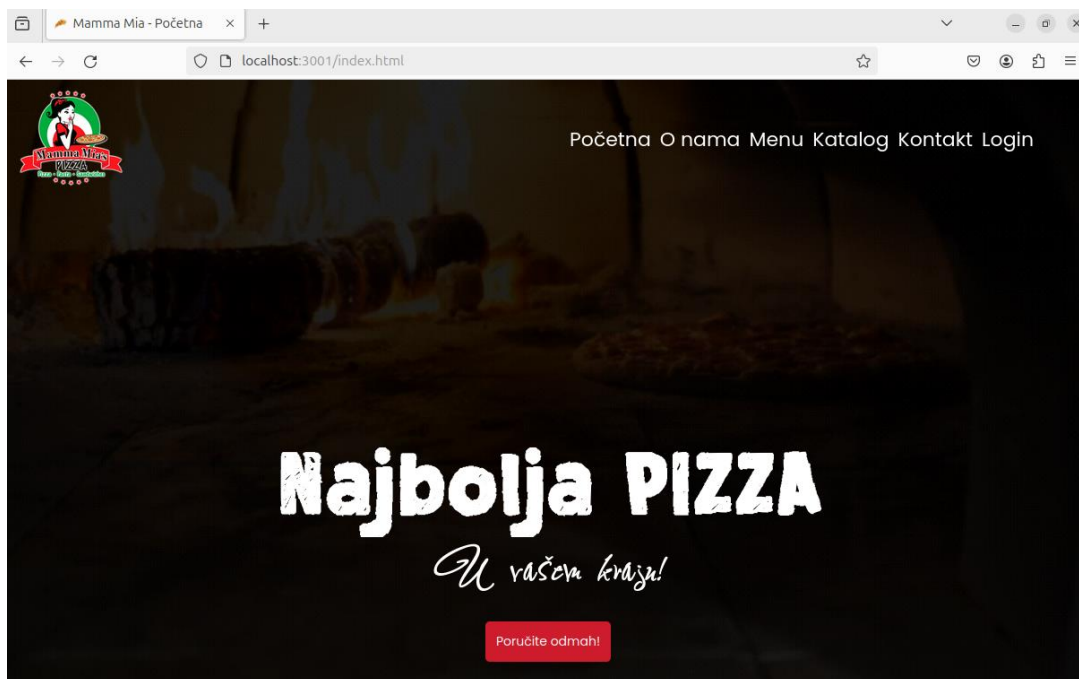
## 2.4 Docker

Docker je platforma za kontejnerizaciju koja omogućava razvoj, isporuku i pokretanje aplikacija u izolovanim okruženjima nazvanim kontejneri. Kontejneri sadrže sve potrebne komponente, uključujući kod, biblioteke i zavisnosti, što omogućava aplikacijama da rade dosledno bez obzira na okruženje u kojem se izvršavaju. Ovo pojednostavljuje proces razvoja, testiranja i implementacije softvera, jer eliminiše probleme sa kompatibilnošću između različitih sistema.

Jedna od glavnih prednosti Docker-a je njegova sposobnost da omogućava brzu i efikasnu skalabilnost aplikacija, jer kontejneri mogu biti brzo pokrenuti i zaustavljeni po potrebi. Takođe, Docker koristi manje resursa od tradicionalnih virtuelnih mašina, jer deli kernel operativnog sistema, čime se smanjuje overhead i povećava performanse. Docker je postao ključni alat u modernim DevOps praksama i mikroservisnoj arhitekturi, omogućavajući timovima da brže isporučuju softver i povećavaju efikasnost razvoja.

## 3 Korisničko uputstvo

Ovaj sajt predstavlja sajt picerije Mamma Mia's pizza. Sastoji se od 6 web stranica: Početna, O nama, Menu, Katalog, Kontakt i Login. Na svakoj od stranica, osim stranice Login, nalazi se navigacioni meni i logo picerije.



Na stranici "Početna" opisujemo naš proizvod i klikom na dugme "Poručite odmah" odlazite na stranicu Menu na kojoj možete da vidite ponudu pica.



## Probajte naš Novi Meni

### Margherita

390 680 940 kn

Pelat, kačkavalj, bosiljak, masline 24, 32, 42cm

### Vesuvio

400 710 980 kn

Pelat, kačkavalj, bosiljak, masline 24, 32, 42cm

### Fungi

400 710 980 kn

Pelat, kačkavalj, bosiljak, masline 24, 32, 42cm

### Vegetariana

440 760 1020 kn

Pelat, kačkavalj, bosiljak, masline 24, 32, 42cm

### Capricciosa

420 760 1020 kn

### Sweet Mamma

680 990 1400 kn

Odlaskom na stranicu "Katalog" možete da vidite slike svih jela u ponudi.



## Otkrijte naše Jedinstvene kreacije



Pelat, kačkavalj, šunka, šampinjoni, masline, origano

Capricciosa



Pelat, kačkavalj, kulen, masline, feferoni

Diavolo

Stranica "O nama" sadrži našu sliku i više informacija o piceriji.



# Upoznajte naš Majstorski tim



*Naš simpatični tim*

Mi smo porodična picerija smeštena u srcu Beograda. S ponosom donosimo autentične ukuse Italije u svaki zalogaj naših pizza. Naša priča počinje s dedom Rajkom, čija ljubav prema tradicionalnoj napuljskoj kuhinji nadahnula je svaki komad testa koji našama

Poručivanje je moguće preko stranice "Kontakt" na dva načina: pozivom na broj i popunjavanjem forme.



# Pozovite nas na 0633120807 ili Popunite web formu

- |  |   |  |
|--|---|--|
| <input type="checkbox"/> Margherita    | <input type="checkbox"/> Vesuvio          | <input type="checkbox"/> Fungi             |
| <input type="checkbox"/> Vegetariana   | <input type="checkbox"/> Capricciosa      | <input type="checkbox"/> Sweet Mamma       |
| <input type="checkbox"/> Pizza Tuna    | <input type="checkbox"/> Diavolo          | <input type="checkbox"/> Bresaola          |
| <input type="checkbox"/> Fat Mamma     | <input type="checkbox"/> Pizza BBQ        | <input type="checkbox"/> Pizza Bianca      |
| <input type="checkbox"/> Chicken Pizza | <input type="checkbox"/> Quattro Stagioni | <input type="checkbox"/> Quattro Formaggio |
- ☒ 24 cm  
☐ 32 cm  
☐ 42 cm



Početna O nama Menu Katalog **Kontakt** Login

- |  |   |   |
|--|---|---|
| <input type="checkbox"/> Margherita    | <input type="checkbox"/> Vesuvio          | <input type="checkbox"/> Fungi            |
| <input type="checkbox"/> Vegetariana   | <input type="checkbox"/> Capricciosa      | <input type="checkbox"/> Sweet Mamma      |
| <input type="checkbox"/> Pizza Tuna    | <input type="checkbox"/> Diavolo          | <input type="checkbox"/> Bresaola         |
| <input type="checkbox"/> Fat Mamma     | <input type="checkbox"/> Pizza BBQ        | <input type="checkbox"/> Pizza Bianca     |
| <input type="checkbox"/> Chicken Pizza | <input type="checkbox"/> Quattro Stagioni | <input type="checkbox"/> Quatro Formaggio |

☒ 24 cm

☐ 32 cm

☐ 42 cm

Ime i prezime

Adresa i broj

Broj telefona

Poruči

Poništi



Stranica "Login" – pristup istoriji narudžbina za administratora web aplikacije.



## 4 Objašnjenja karakterističnih delova koda

### 4.1 Frontend/public/js/form.js

```
var btn = document.getElementById("dugme");
btn.addEventListener("click", porucivanje);

function porucivanje() {
  var name = document.getElementById("ime").value;
  var adr = document.getElementById("adresa").value;
  var tel = document.getElementById("tel").value;

  var size = document.querySelector('input[name="radio"]:checked').value;
  var pizza = document.querySelector('input[class="pizza"]:checked').value;

  // Prepare the data to send
  const formData = {
    name: name,
    adr: adr,
    tel: tel,
```

```

    size: size,
    pizza: pizza,
  };

  try {
    axios
      .post("http://localhost:3000/orders", formData)
      .then((response) => {
        console.log("Order placed successfully:", response.data);
        document.getElementById("reset").click();
        alert("Porudžbina je uspešno poslata na obradu.");
      })
      .catch((error) => {
        console.error("Error placing order:", error);
        alert("Došlo je do greške prilikom slanja porudžbine.");
      });
  } catch {
    console.log("post nije izvršen");
  }
}

```

Nakon što korisnik izabere vrstu pice, veličinu i popuni lične podatke za dostavu, klikom na dugme Poruči, funkcija porucivanje se pokreće. Ona uzima vrednosti iz date forme, potom ih stavlja u dictionary, koji pomoću eksterne biblioteke Axios šalje POST request backendu na obradu. Potom prima odgovor backenda, u zavisnosti od njega korisniku ispisuje poruku koja ga obaveštava da li je porudžbina bila uspešna ili neuspešna.

#### 4.2 Frontend/public/js/login.js

```

var dugme = document.getElementById("logindugme");
dugme.addEventListener("click", logovanje);

function logovanje() {
  var uname = document.getElementById("uname").value;
  var psswd = document.getElementById("psswd").value;

  if (uname == "admin" && psswd == "admin") {
    document.getElementById("uname").style.visibility = "hidden";
    document.getElementById("psswd").style.visibility = "hidden";
    document.getElementById("logindugme").style.visibility = "hidden";
  }
}

```

```

try {
  axios.get("http://localhost:3000/orders").then((response) => {
    document.getElementById("pod").style.visibility = "visible";
    if (!Array.isArray(response.data)) {
      throw new Error("Data is not an array");
    }
    const tableBody = document.getElementById("orders-table-body");
    response.data.forEach((order) => {
      const row = document.createElement("tr");
      row.innerHTML = `
        <td>${order.id}</td>
        <td>${order.name}</td>
        <td>${order.adr}</td>
        <td>${order.tel}</td>
        <td>${order.pizza}</td>
        <td>${order.size}</td>
      `;
      tableBody.appendChild(row);
    });
  });
} catch {
  alert("doslo je do greske prilikom ispisa svih porudzbina");
}
}

```

Na stranici "Login" administrator web stranice može da vidi sve porudžbine. Međutim, kako bi onemogućili svakome pristup, administrator se prvo mora ulogovati. Klikom na dugme "Login" pokreće se funkcija logovanje() koja prvo proverava unesene kredencijale. Ukoliko su uneseni kredencijali ispravni, funkcija šalje GET request beckendu, proverava da li je dati odgovor u JSON formatu i ukoliko jeste za svaku porudžbinu kreira red tabele koji potom apenduje.

#### 4.3 Backend/app.js

```

const express = require("express");
const bodyParser = require("body-parser");
const mysql = require("mysql2");
const axios = require("axios");
const cors = require("cors");

```

```
const app = express();
const port = 3000;
app.use(cors());
app.use(bodyParser.json());

// MySQL connection setup
const connection = mysql.createConnection({
  host: "mysql-server",
  user: "root",
  password: "mypassword",
  database: "swfavorites",
});

connection.connect((err) => {
  if (err) throw err;
  console.log("Connected to the database.");

  const createTableSql = `
    CREATE TABLE IF NOT EXISTS orders (
      id INT AUTO_INCREMENT PRIMARY KEY,
      name VARCHAR(100) NOT NULL,
      adr VARCHAR(100) NOT NULL,
      tel VARCHAR(100) NOT NULL,
      pizza VARCHAR(100) NOT NULL,
      size VARCHAR(100) NOT NULL
    )
  `;

  connection.query(createTableSql, (err, result) => {
    if (err) throw err;
    console.log("Table 'orders' is ready.");
  });
});

// Endpoint to get
app.get("/orders", (req, res) => {
  connection.query("SELECT * FROM orders", (error, results) => {
    if (error) throw error;
    res.json(results);
  });
});

// Endpoint to add
```

```

app.post("/orders", (req, res) => {
  const { name, adr, tel, pizza, size } = req.body;
  if (!(name && adr && tel && pizza && size)) {
    throw Error;
  }
  const query =
    "INSERT INTO orders (name, adr, tel, pizza, size) VALUES (?, ?, ?, ?, ?)";
  connection.query(query, [name, adr, tel, pizza, size], (error, results) => {
    if (error) throw error;
    res.status(201).send(`Favorite added with ID: ${results.insertId}`);
    console.log("zahtev upisan");
  });
});

app.listen(port, () => {
  console.log(`Server running on port ${port}`);
});

```

Dati fajl kreira konekciju sa MySQL bazom podataka, kreira tabelu ukoliko ona već ne postoji, sluša na portu 3000 za nailazeće GET i POST requestove. Po primanju GET requesta na portu 3000, vraća se rezultat MySQL query-a "SELECT \* FROM orders" koja vraća sve narudžbine u JSON formatu. Po primanju POST requesta, vrši se MySQL naredba "INSERT INTO orders" koja u tabelu orders unosi red sa vrednostima: ime, adresa, telefon, pizza i veličina. Vraća se rezultat 201 – CREATED ili kod nastale greške.

#### 4.4 Db/init.sql

```

SET GLOBAL log_error_verbosity = 1;
ALTER USER 'novi'@'%' IDENTIFIED WITH caching_sha2_password BY 'novi';
GRANT ALL PRIVILEGES ON swfavorites.* TO 'novi'@'%';
FLUSH PRIVILEGES;
EXIT;

```

Ove SQL komande podešavaju MySQL server tako da prikazuje samo osnovne greške, menjaju autentifikaciju korisnika "novi" da koristi "caching\_sha2\_password" metodu sa lozinkom "novi" i dodeljuju sve privilegije datom korisniku na bazi podataka.

## 4.5 frontend/public/HTML i CSS

```
<link rel="shortcut icon" href="img/favicon.ico" type="image/x-icon">
```

Ovaj deo koda se nalazi u head-u html dokumenata, a prikazuje se kao ikonica u gornjem levom uglu pored naslova web stranice. Link element omogućava povezivanje dokumenta sa drugim izvorima, u ovom slučaju sa slikom ikonice.

```
  
<div id="background"></div>
```

Ovde smo kao pozadinu početne stranice ubacili video u formatu gif.

```
<nav>  
  <a href=""></a>
```

Ubacili smo sliku za logo. Element a definiše hiperlink koji se koristi da poveže jednu stranicu sa drugom. Njegov najvažniji atribut je href koji određuje destinaciju linka. Atribut src specificira URL slike.

```
<li><a href="kontakt.html" class="active">Kontakt</a></li>
```

class="active" obeležava aktivnu stranicu. U ovom primeru je kontakt, ali se takođe nalazi u htmlu svake stranice. Dalje je u index.css za tu klasu određena boja i važi za ceo kod.

```
<form onsubmit="return porucivanje()">
```

```
button:hover {  
  background-color: white;  
  color: #cf1c2c;  
}
```

button:hover čini da crveno dugme „Poručite odmah!“ prelaskom kursora postane belo.

```
#nav li:hover {  
  border-bottom: 4px solid white;  
}
```

#nav li:hover omogućava da se prelaskom kursora preko navigacionog menija ispod označene stranice pojavi bela donja crta.

```

@media only screen and (max-width: 650px) {
  #vid {
    visibility: hidden;
  }
  #background {
    background: □rgb(0, 0, 0);
  }
}

@media only screen and (max-width: 400px) {
  h1 {
    font-size: 80px;
  }
}

```

@media omogućava responzivnost sajta. U datom primeru na početnoj stranici, na širini manjoj od 650 piksela video nestaje i pozadina postaje potpuno crna. Na širini manjoj od 400 piksela smanjuje se i font h1.

```

#nav li a:hover {
  color: ■#cf1c2c;
}

```

Ostale stranice su za razliku od početne bele, tako da se prelaskom kursora preko navigacionog menija boja teksta menja u crvenu.

```

#nav li:hover {
  border-bottom: 4px solid ■#cf1c2c;
}

```

Prelaskom kursora preko imena stranice pojavljuje se donja crta u crvenoj boji.

```

nav {
  position: fixed;
  width: 100%;
  height: 125px;
  background-color: ■white;
  box-shadow: 0 3px 10px 0 □rgba(0, 0, 0, 0.19);
}

```

Stavili smo da je pozicija menija fiksna tj. da ostaje na vrhu ekrana bez obzira na listanje stranice.

```
@media only screen and (max-width: 650px) {
  nav {
    height: 200px;
    position: static;
  }
  #nav {
    position: static;
    margin-top: 20px;
  }
  h3 {
    margin-top: 20px;
  }
}
```

U cilju postizanja responzivnosti, smanjenjem širine stranice navigacioni meni postaje statičan, tj. neće se pomerati sa listanjem. Ovaj deo koda je ubačen na svim stranicama.

```
@media only screen and (max-width: 500px) {
  h1 {
    margin-top: 15px;
    font-size: 60px;
    margin-bottom: -150px;
  }
  #najjaci{
    visibility: hidden;
  }
}
```

Dodatnim smanjenjem veličine stranice, smanjuje se veličina fonta i margine, a slika nestaje.

```
hr {
  border: none;
  border-top: 2px dashed #cf1c2c;
  margin-bottom: 40px;
}
```

hr je horizontalna linija koja se nalazi ispod naziva pica.

```
.containerCheck input {
  position: absolute;
  opacity: 0;
  cursor: pointer;
  height: 0;
  width: 0;
}
```



Ovaj deo koda služi da se podrazumevani checkbox browser-a sakrije u cilju ubacivanja specifičnog checkbox-a. Height i width su podešeni na nulu i time je checkbox sakriven.

```
.checkmarkCheck {  
  position: absolute;  
  top: 0;  
  left: 0;  
  height: 25px;  
  width: 25px;  
  background-color: #eee;  
}
```

Ovim ubacujemo novi checkbox. Pozicija absolute je relativno pozicionirana u odnosu na najbližeg ancestor, tj. parent elementa. Ovde je data klasi checkmarkCheck koja je klasa za span čiji je parent element containerCheck.

```
.checkmarkCheck:after {  
  content: "";  
  position: absolute;  
  display: none;  
}
```

Checkmark je sakriven kada checkbox nije označen.

```
.containerCheck input:checked ~ .checkmarkCheck:after {  
  display: block;  
}
```

A kada je checkbox i checkmark se pojavljuje.

```
.containerCheck .checkmarkCheck:after {  
  left: 9px;  
  top: 5px;  
  width: 5px;  
  height: 10px;  
  border: solid white;  
  border-width: 0 3px 3px 0;  
  -webkit-transform: rotate(45deg);  
  -ms-transform: rotate(45deg);  
  transform: rotate(45deg);  
}
```

Ovim kodom smo stilizovali checkmark. Svojstvo CSS -webkit-transform omogućava veb autorima da transformišu element u prostoru. Elementi se mogu rotirati, skalirati, iskriviti itd. -ms-transform radi isto što i -webkit-transform samo za drugi browser.

```
.containerRadio input {
  position: absolute;
  opacity: 0;
  cursor: pointer;
}
```

Slično kao za checkbox, sakriva podrazumevani radio button.

```
.checkmarkRadio {
  position: absolute;
  top: 0;
  left: 0;
  height: 25px;
  width: 25px;
  background-color: #eee;
  border-radius: 50%;
}
```

Ovim ubacujemo novi radio button. Pozicija je absolute, kao i kod checkbox -a.

```
.checkmarkRadio:after {
  content: "";
  position: absolute;
  display: none;
}
```

```
.containerRadio input:checked ~ .checkmarkRadio:after {
  display: block;
}
```

Pokazuje indikator, u ovom slučaju kružić, kada se radio button obeleži. U suprotnom je sakriven.

```
.podaci:focus{
  border: none;
  outline: none;
  border-bottom: 2px solid #cf1c2c;
}
```

Kada upisujemo lične podatke pritiskom na polje se pojavljuje crvena linija, tj. donja ivica.

## 5 Prikaz procesa kontejnerizacije/dokerizacije i njegovih atributa

### 5.1 Detaljno objašnjenje compose.yaml fajla

```
cloud_seminarski > 📄 compose.yaml
```

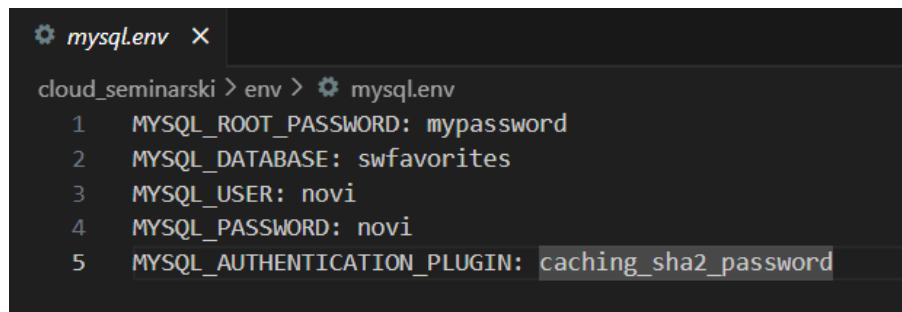
```
1  name: 'orders-app'
2  version: '1.0'
```

Top level element name je naziv projekta koji je u našem slučaju orders-app. Version navodi verziju Docker Compose specifikacije koja se koristi.

```
services:
  mysql: |
    image: mysql:latest
    container_name: mysql-server
    restart: always
    env_file:
      - ./env/mysql.env
    networks:
      - app-network
    volumes:
      - mysql-data:/var/lib/mysql
      - ./db/init.sql:/docker-entrypoint-initdb.d/init.sql
      - ./db/my.cnf:/etc/mysql/conf.d/my-custom.cnf:ro
    healthcheck:
      test: ["CMD", "mysqladmin", "ping", "-h", "localhost", "-u", "novi", "--password=novi"]
      interval: 10s
      timeout: 10s
      retries: 6
```

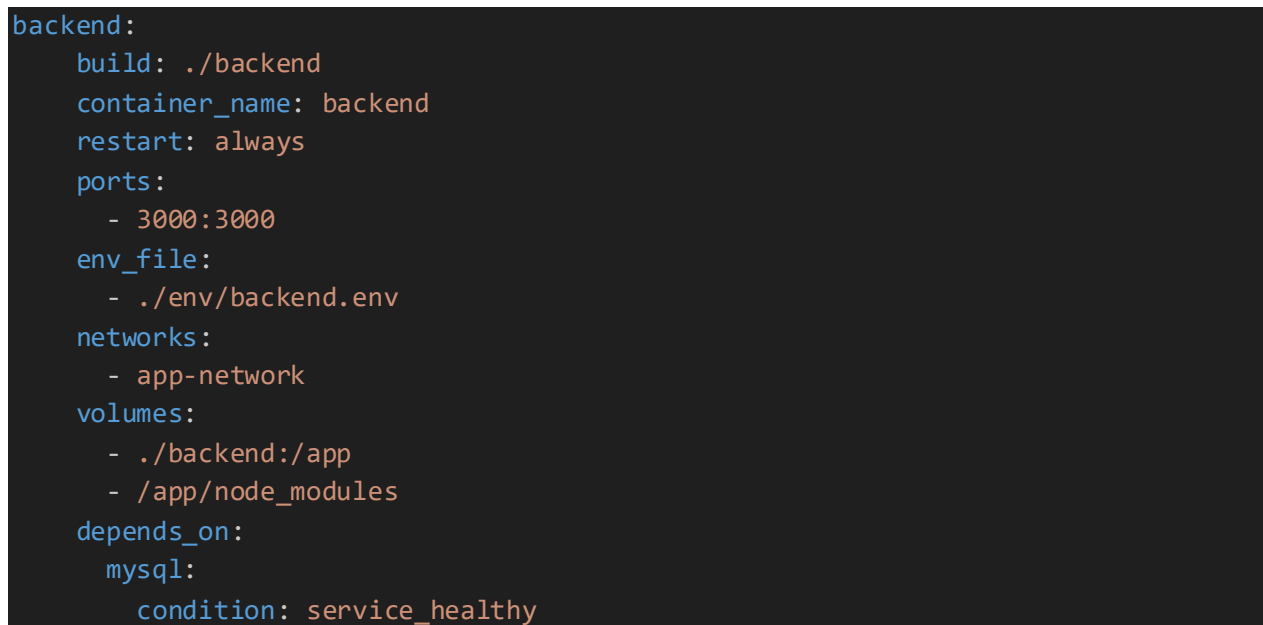
U delu **services** definišemo servise koji će se pokrenuti kao kontejneri. Prvi servis koji smo definisali je servis **mysql**. **image: mysql:latest** određuje da se koristi zvanična MySQL slika sa Docker Hub-a. **container\_name: mysql-server** definiše ime kontejnera koje će se koristiti za MySQL imidž. Kada se kontejner pokrene, dobiće ime mysql-server. Okruženje smo definisali u zasebnom **env fajlu** koji učitava environment promenljive iz datoteke (prikazan ispod). **networks: - app-network** Ova postavka dodaje kontejner u mrežu nazvanu goals-net, ako već ne postoji. Ovo omogućava komunikaciju između kontejnera u istoj mreži. U **volumes** definišemo jedan imenovani volumen data koji će biti mapiran na /var/lib/mysql unutar kontejnera MySQL-a. To je mesto gde će MySQL čuvati svoje podatke. Pored imenovanog volumena, dodaćemo i dva bind-mount volumena koji će fajlove iz folder db da postavie unutar mysql db konfiguracije. Prvi (.sql) fajl definiše komande koje se pokreći u interaktivnom mysql terminalu, dok drugi (.cnf) fajl isključuje upozorenja u okviru mysql-a. Da bismo automatizovali aplikaciju kroz compose.yaml fajl dodajemo **healthcheck** opciju unutar mysql-servis da se proveri da li je živ/zdrav. Od podopcija u healthchecku stavljamo **test**, **interval**, **timeout** i **retries**. Test proverava da li je servis dostupan. Pored test, ostale podopcije su interval (10s), odnosno vreme nakon pokretanja kontejnera kada će se provera pokrenuti. Ako proveru treba više od timeout sekundi (10s), smatraće se da je servis

failed. Ako za svaki definisani broj pokušaja (retries=6) servis fail-uje, smatraće se da je kontejner unhealthy.



```
mysqlEnv X
cloud_seminarski > env > mysqlEnv
1  MYSQL_ROOT_PASSWORD: mypassword
2  MYSQL_DATABASE: swfavorites
3  MYSQL_USER: novi
4  MYSQL_PASSWORD: novi
5  MYSQL_AUTHENTICATION_PLUGIN: caching_sha2_password
```

Opcija **MYSQL\_ROOT\_PASSWORD** u Docker Compose datoteci se koristi za postavljanje korisničke lozinke za "root" korisnika MySQL baze podataka unutar MySQL kontejnera koji se pokreće pomoću Docker Compose alata. **MYSQL\_DATABASE** se koristi za definisanje naziva MySQL baze podataka koja će se automatski kreirati prilikom pokretanja MySQL kontejnera. Opcija **MYSQL\_USER** i se koristi za definisanje korisničkog imena koje će se koristiti za kreiranje novog MySQL korisnika. **MYSQL\_PASSWORD** postavlja šifru za novog korisnika. **MYSQL\_AUTHENTICATION\_PLUGIN** je opcija u MySQL koja se koristi za odabir metode autentifikacije koju će MySQL server koristiti za proveru korisničkih identiteta prilikom prijavljivanja. Caching SHA-2 Plugin dodaje keširanje na vrh SHA-256 autentifikacije, čime se smanjuje opterećenje servera prilikom provere lozinke.



```
backend:
  build: ./backend
  container_name: backend
  restart: always
  ports:
    - 3000:3000
  env_file:
    - ./env/backend.env
  networks:
    - app-network
  volumes:
    - ./backend:/app
    - /app/node_modules
  depends_on:
    mysql:
      condition: service_healthy
```

Ovde definišemo servis **backend**. **build: ./backend**: Za kreiranje imidža koristimo naš dockerfile definisan unutar backend foldera. build opcija se piše korišćenjem stringa koji definiše putanju kao dockerfile-u. **container\_name: backend** Ovde se definiše ime kontejnera koje će se koristiti za node aplikaciju. Kada se kontejner pokrene, dobiće ime backend. **ports: - "3000:3000"** definiše kako će se mrežni portovi mapirati između host sistema i servisa (kontejnera). Konkretno, ports se koristi za mapiranje portova

između host sistema i servisa u Docker kontejneru. `3000:3000`, znači da se port 3000 na host sistemu mapira na port 3000 u Docker kontejneru. Okruženje je definisano u zasebnom **env fajlu** koje učitava environment promenljive iz datoteke `backend.env` (prikazano ispod). Servis backend je dodat u istu mrežu **networks**, kao i mysql servis, kako bi mogli da komuniciraju. **volumes**: - `./backend:/app` - `/app/node_modules` definiše dva volumena: **o `./backend:/app`**: je bind-mount koji mapira lokalni direktorijum `./backend`, odnosno sadržinu našeg backend foldera, na putanju `/app` kreiranog backend kontejnera. Ovaj volumen će nam dozvoliti menjanje aplikacije na lokalnom računaru, bez potrebe da se novi imidž kreira da bi se videle promene. **o `/app/node_modules`**: je neimenovani volumen. Definiše `node_modules` direktorijum unutar kontejnera kao volumen bez eksplicitnog mapiranja, što znači da će bilo kakve izmene u `node_modules` biti zanemarene i `node_modules` će biti dostupan samo unutar kontejnera. **depends\_on**: - `mysql` Definiše zavisnost ovog servisa od servisa `mysql`, što znači da će servis `mysql` biti pokrenut pre nego što se pokrene servis backend.

```
cloud_seminarski > env > backend.env
1 DB_HOST=mysql-server
2 DB_USER=novi
3 DB_PASSWORD=novi
4 DB_NAME=swfavorites
5
```

**DB\_HOST**: Ova promenljiva definiše host ime ili IP adresu MySQL servera do kojeg aplikacija treba da se poveže. **DB\_USER**: Ova promenljiva definiše korisničko ime koje aplikacija koristi za autentifikaciju pri pristupu MySQL bazi podataka. **DB\_PASSWORD**: Ova promenljiva definiše lozinku (password) koja se koristi zajedno sa korisničkim imenom (`DB_USER`) za autentifikaciju pri pristupu MySQL bazi podataka. **DB\_NAME**: Ova promenljiva definiše naziv MySQL baze podataka kojoj aplikacija treba da pristupi ili koju treba da koristi za operacije čitanja i upisivanja podataka.

```
frontend:
  build: ./frontend
  container_name: frontend
  restart: always
  ports:
    - "3001:3000"
  networks:
    - app-network
  volumes:
    - ./frontend:/app
    - /app/node_modules
  depends_on:
    - backend
```

Ovde definišemo servis frontend. **build: ./frontend:** Ovde se definiše putanja do Dockerfile-a koji će se koristiti za izgradnju Docker imidža za frontend aplikaciju. Docker Compose će potražiti Dockerfile unutar direktorijuma ./frontend kako bi izgradio imidž za frontend servis. **container\_name: frontend:** Ovde se definiše ime kontejnera koje će se koristiti za frontend aplikaciju. Kada se kontejner pokrene, dobiće ime frontend. **restart: always:** Ova opcija definiše da Docker treba automatski ponovo pokrenuti kontejner ako se zaustavi, bez obzira na razlog zaustavljanja (na primer, nakon restarta Docker servisa ili host sistema). Osigurava da frontend aplikacija bude uvek dostupna, čak i u slučaju neočekivanog zaustavljanja kontejnera. **ports: - "3001:3000":** Definiše kako se mrežni port 3000 u frontend kontejneru mapira na port 3001 na host sistemu. **networks: - app-network:** Povezuje frontend servis sa mrežom nazvanom app-network. **volumes:** Definiše dva volumena za frontend kontejner: **./frontend:/app:** Bind-mount koji mapira lokalni frontend direktorijum ./frontend na putanju /app unutar frontend kontejnera. Ovo omogućava promene u kodu frontend aplikacije na host sistemu bez potrebe za ponovnom izgradnjom Docker imidža. **/app/node\_modules:** Neimenovani volumen koji definiše node\_modules direktorijum unutar kontejnera kao volumen bez eksplicitnog mapiranja. Ovo se koristi za ignorisanje node\_modules direktorijuma iz lokalnog sistema, čime se sprečava da lokalne promene u node\_modules utiču na kontejnersku verziju.

**depends\_on: - backend:** Definiše zavisnost frontend servisa od backend servisa. Ovo znači da će Docker Compose prvo pokrenuti backend servis pre nego što pokrene frontend servis.

```
networks:
  app-network:
    driver: bridge
```

**networks:** Ovde definišemo networks za compose.yaml koje će se koristiti za komunikaciju između Docker kontejnera. **app-network:** je naziv mreže koju definišemo. Ovaj naziv smo koristili da povežemo naše servise sa mrežom. **driver: bridge:** je vrsta mrežnog drivera koji se koristi za app-network. bridge je podrazumevani driver za Docker mreže. **Bridge network:** su privatne mreže koje Docker automatski kreira kada se ne definiše drugačije. Svaki kontejner povezan na bridge mrežu može komunicirati sa drugim kontejnerima na istoj mreži. Svaki kontejner na bridge mreži dobija svoju IP adresu. **Driver:** Driveri mreže u Dockeru pružaju apstrakciju za mrežne operacije. bridge driver je najčešće korišćeni driver za kontejnere koji treba da komuniciraju na istoj mašini.

```
volumes:
  mysql-data:
```

Ovde se definiše imenovani volumen **mysql-data** koji je nezavisan od kontejnera i može se ponovo koristiti između različitih kontejnera. Onda referenciramo taj volumen u odgovarajućem servisu pod volumes ključem (u našem slučaju to je volumes: - mysql-data:/var/lib/mysql ).

## 5.2 Frontend/dockerfile

```
cloud_seminarski > frontend > dockerfile > FROM
1 FROM node
2 WORKDIR /app
3 COPY package.json ./
4 RUN npm install
5 COPY src/ src/
6 COPY public/ public/
7 RUN npm run build
8 EXPOSE 3000
9 CMD ["npm", "start"]
```

**FROM node** - postavlja osnovnu Docker sliku na Node.js.

**WORKDIR /app** - postavlja radni direktorijum u Docker slici na "/app". Sve naredne komande će se izvršavati unutar ovog direktorijuma.

**COPY package.json ./** - Kopira lokalni package.json fajl u direktorijum "/app" u Docker slici. Ovo se radi odvojeno kako bi se omogućila optimalna upotreba keša tokom izvršavanja npm install koraka.

**RUN npm install** - Izvršava npm install komandu unutar "/app" direktorijuma. Ova komanda instalira sve zavisnosti navedene u package.json fajlu i takođe izvršava unutar Docker slike.

**COPY src/ src/** - Komanda COPY src/ src/ kopira sve fajlove i direktorijume iz direktorijuma src/ na host sistemu u isti direktorijum src/ unutar Docker kontejnera.

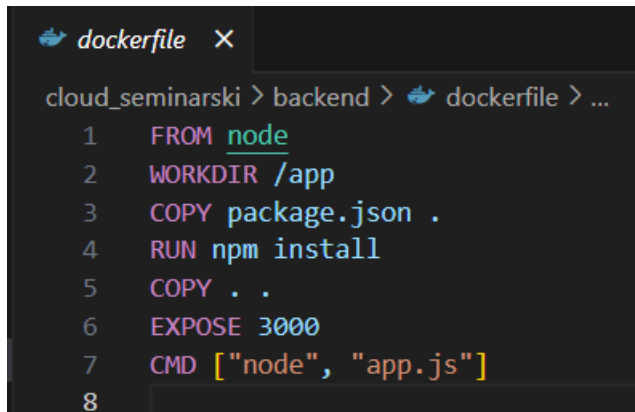
**COPY public/ public/** - Komanda COPY public/ public/ kopira sve fajlove i direktorijume iz direktorijuma public/ na host sistemu u isti direktorijum public/ unutar Docker kontejnera.

**RUN npm run build** - Komanda RUN npm run build u Dockerfile-u se koristi za automatsko izvršavanje npm run build komande tokom izgradnje Docker imidža, što omogućava prethodnu obradu ili kompilaciju resursa potrebnih za aplikaciju.

**EXPOSE 3000** - Ova komanda označava da će kontejner, zasnovan na ovoj slici, biti dostupan na portu 3000.

**CMD ["npm", "start"]** - CMD ["npm", "start"] u Dockerfile-u postavlja podrazumevanu komandu koja se automatski izvršava prilikom pokretanja Docker kontejnera, u ovom slučaju pokreće Node.js aplikaciju koristeći npm start skriptu.

### 5.3 Backend/dockerfile



```
cloud_seminarski > backend > dockerfile > ...
1 FROM node
2 WORKDIR /app
3 COPY package.json .
4 RUN npm install
5 COPY . .
6 EXPOSE 3000
7 CMD ["node", "app.js"]
8
```

**FROM node** - Ova komanda postavlja osnovnu Docker sliku na Node.js.

**WORKDIR /app** - Postavlja radni direktorijum u Docker slici na "/app". Sve naredne komande će se izvršavati unutar ovog direktorijuma.

**COPY package.json /app** - Kopira lokalni package.json fajl u direktorijum "/app" u Docker slici. Ovo se radi odvojeno kako bi se omogućila optimalna upotreba keša tokom izvršavanja npm install koraka.

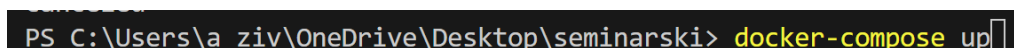
**RUN npm install** - Izvršava npm install komandu unutar "/app" direktorijuma. Ova komanda instalira sve zavisnosti navedene u package.json fajlu i takođe izvršava unutar Docker slike.

**COPY . .** - kopira sve fajlove i direktorijume iz trenutnog direktorijuma na host sistemu u radni direktorijum kontejnera.

**EXPOSE 3000** - Ova komanda označava da će kontejner, zasnovan na ovoj slici, biti dostupan na portu 3000.

**CMD ["node", "server.js"]** - Postavlja podrazumevanu komandu koja će se izvršiti kada se kontejner pokrene. U ovom slučaju, pokreće se Node.js aplikacija "server.js" pomoću node komande.

### 5.4 Postupak kontejnerizacije, pokretanja i ažuriranja aplikacije

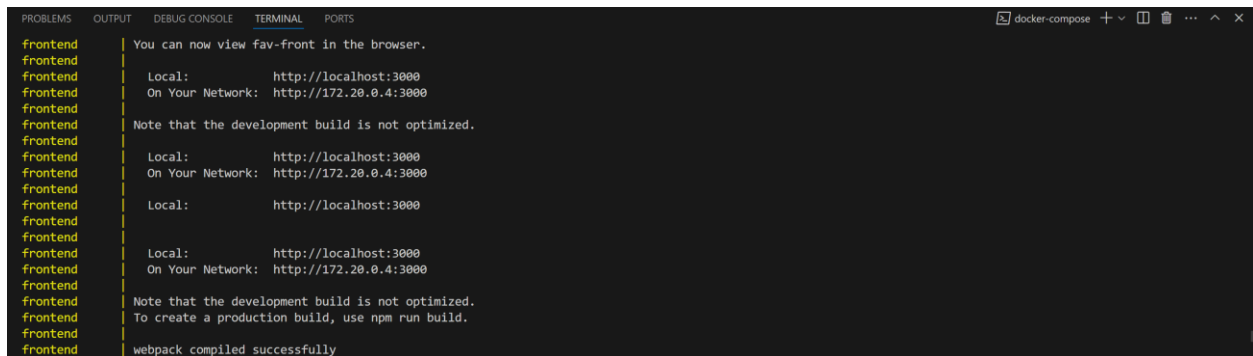


```
PS C:\Users\ziv\OneDrive\Desktop\seminarski> docker-compose up
```

Komanda "docker-compose up" pokreće definisane servise u Docker Compose fajlu (compose.yml). Ona kreira i startuje kontejnere prema specifikacijama u Compose fajlu, rešava zavisnosti među servisima,



kreira mreže i mount-uje volumene po potrebi. Ako kontejneri ili mreže već postoje, komanda će ih restartovati i aplicirati bilo kakve promene u konfiguraciji.



```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
frontend | You can now view fav-front in the browser.
frontend | Local: http://localhost:3000
frontend | On Your Network: http://172.20.0.4:3000
frontend | Note that the development build is not optimized.
frontend | Local: http://localhost:3000
frontend | On Your Network: http://172.20.0.4:3000
frontend | Local: http://localhost:3000
frontend | Local: http://localhost:3000
frontend | On Your Network: http://172.20.0.4:3000
frontend | Note that the development build is not optimized.
frontend | To create a production build, use npm run build.
frontend | webpack compiled successfully
```

```
mysql-server | 2024-07-12T15:40:38.270807Z 0 [System] [MY-010931] [Server] /usr/sbin/mysqld: ready
for connections. Version: '9.0.0' socket: '/var/run/mysqld/mysqld.sock' port: 3306 MySQL Community
Server - GPL.
```

...

```
backend | Server running on port 3000
```

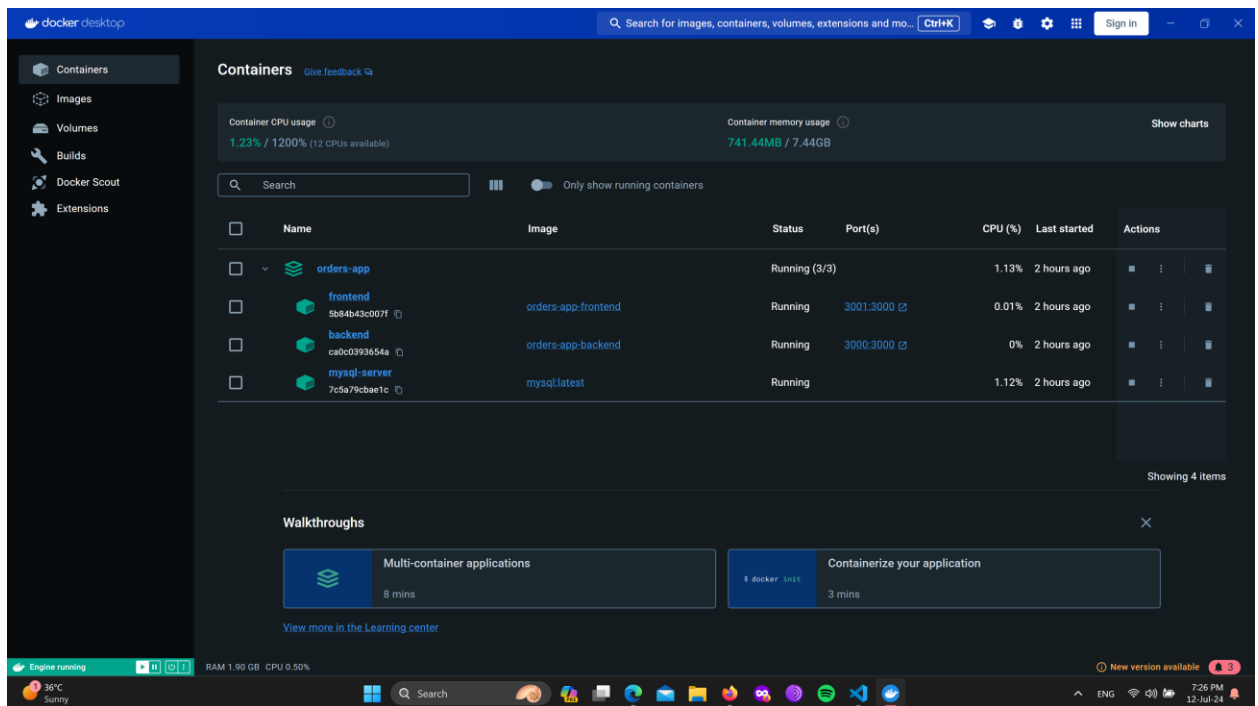
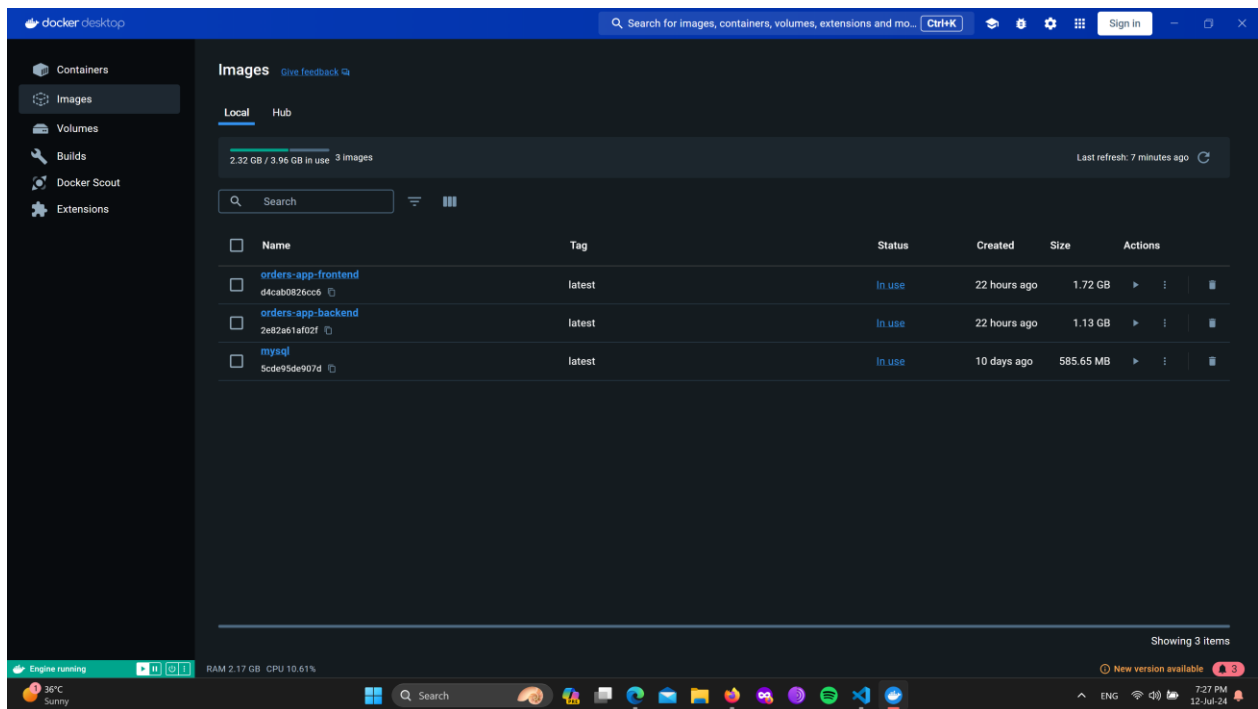
```
backend | Connected to the database.
```

```
backend | Table 'orders' is ready.
```

...

```
frontend | webpack compiled successfully
```

Po završetku izvršavanja navedene komande, ukoliko nije došlo do grešaka, u kozoli će biti ispisan dati output. Slika gore, predstavlja deo outputa.



Na gorenavedenim slikama prikazan je GUI Docker Desktop okruženja. Tu su prikazani pokrenuti kontejneri (frontend, backend i my-sql server), kao i image-i koji su trenutno u upotrebi.

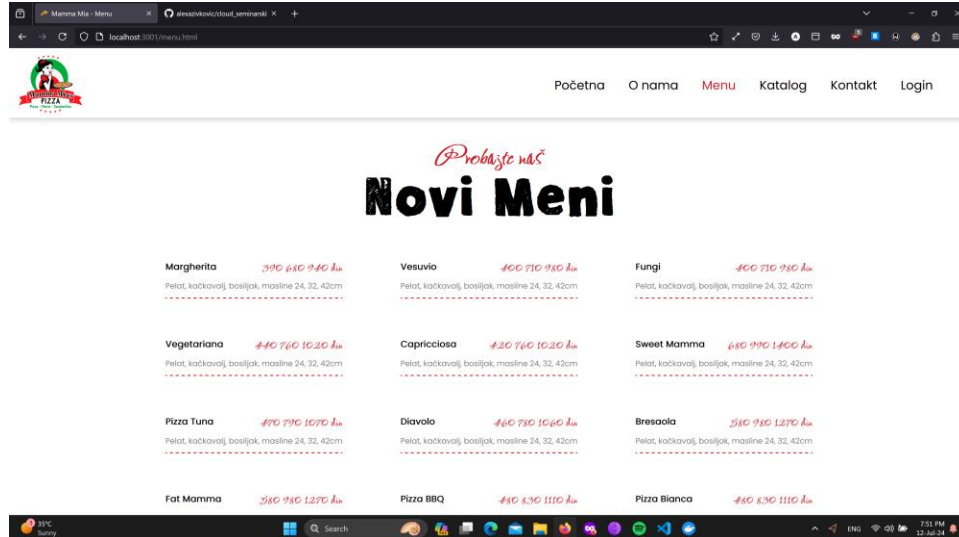
Sad ćemo prikazati proces ažuriranja aplikacije. Modifikacija će biti izvršena nad fajlom frontend/public/menu.html. Umesto:

```
<h1>Novi Meni</h1>
```

Zamениćemo sa:

```
<h1>Stari Meni</h1>
```

Izgled web stranice pre izmena je sledeći:



U Visual Studio Code-u izvršena je izmena datog HTML fajla. Bez ponovnog pokretanja komande "docker-compose up" kada "refresh-ujemo" web browser izgled stranice je sledeći:

