УНИВЕРЗИТЕТ „СВ. КИРИЛ И МЕТОДИЈ “ – СКОПЈЕ
ФАКУЛТЕТ ЗА ИНФОРМАТИЧКИ НАУКИ И КОМПЈУТЕРСКО ИНЖЕНЕРСТВО
ОДДЕЛ: СОФТВЕРСКО ИНЖЕНЕРСТВО И ИНФОРМАЦИСКИ СИСТЕМИ
СКОПЈЕ

TO DO APPLICATION - АПЛИКАЦИЈА ЗА ЗАДАЧИ

Семинарски труд по предметот Напреден веб дизајн

Ментор:

Професор:

д-р Бобан Јоксимоски

Асистент:

Славе Темков

Кандидат:

Белма Хусеиноска 201227

Габриела Шурбеска 201185

Скопје, 08, 2023

Содржина

Апстракт

Апликацијата To-Do е веб-апликација изградена со помош на Deno, безбедно и модерно решение на работа за JavaScript и TypeScript. Оваа апликација е апликација за листа со задачи изградена со помош на Node.js, а потоа направенаа миграција во Deno. Овој проект после миграцијата вклучува создавање на веб-апликација за управување со задачи користејќи ја околината за извршување на Deno и рамката Oak. Апликацијата им овозможува на корисниците да додаваат, бришат и прегледуваат задачи преку кориснички интерфејс. Извештајот ја прикажува архитектурата, карактеристиките, деталите за имплементацијата, предизвиците со кои се соочуваат и идните подобрувања на проектот.

Клучни зборови: Deno, JavaScript, TypeScript, Node.js, Oak

1. Вовед

Апликацијата To-Do е веб-апликација изградена со помош на Deno, безбедно и модерно време на работа за JavaScript и TypeScript. Апликацијата им овозможува на корисниците ефективно да управуваат со нивните задачи преку обезбедување функции како што се додавање задачи, гледање задачи за денес и бришење задачи. Примарната цел на апликацијата е да ја демонстрира употребата на можностите на Deno за градење функционална веб-апликација со рендерирање од страна на серверот користејќи EJS шаблони и Bootstrap за стилизирање на предниот дел. Оваа апликација е апликација за список со задачи изградена со помош на Node.js, а потоа направенаа миграција во Deno. Апликацијата им овозможува на корисниците да додаваат, прегледуваат и бришат задачи. Секоја задача се состои од име, датум и опис.

Node е опишан како "an asynchronous event-driven JavaScript runtime". Node може да се користи за изградба на апликации од страна на серверот или десктоп, и е достапен за корисниците на JavaScript од 2009 година. Deno е понова рамка изградена од Рајан Дал, кој исто така го развил Node.js.Со цел да понуди едноставно, безбедно време за извршување, Deno користи стандарди за веб-платформи, испорачува со една извршна датотека и нуди вградена развојна алатка. Има алтернативен сет на стандардни модули за кои се гарантира дека ќе работат во рамките на траењето на Deno. Deno исто така може да ги извршува датотеките TypeScript природно без никаква дополнителна конфигурација.

Целта на оваа документација е да обезбеди сеопфатен водич за апликацијата To-Do, кој опфаќа различни аспекти како што се инсталација, конфигурација, имплементација на функции и најдобри практики за развој на Deno. Следејќи ја оваа документација, програмерите ќе стекнат подобро разбирање за тоа како да градат веб-апликации со Deno и да користат популарни библиотеки како EJS и Bootstrap за подобрени кориснички интерфејси.

2. Сличности помеѓу Node и Deno

На основно ниво, Node и Deno се прилично слични - и двата нудат управување со пакети, систем за увоз на модули и други кодови од трети страни и солидна стандардна библиотека.

Дополнително, и Node и Deno можат да користат TypeScript, кој е популарен суперсет на JavaScript развиен од Microsoft. Сепак, малку потешко е да се започне со TypeScript кога се користи Node.
И Deno и Node користат V8 JavaScript мотор создаден од Google. Ова значи дека перформансите во прелистувачот треба да бидат скоро исти помеѓу двете рамки.

3. Разлики помеѓу Node и Deno

Она што Node и Deno се разликуваат е тоа што Node е изграден со C и C++, додека Deno е изграден со користење на поновиот јазик Rust. Rust е програмски јазик со повеќе парадигма, ниско ниво, првично изграден од Mozilla за да биде безбедна за меморија и статички напишана алтернатива на другите јазици на системско ниво. Ова и овозможува на основната архитектура на бинарната Deno да ги фати внатрешните грешки во времето на компајлирање. Овие подобрувања влијаат само на внатрешните работи на Deno, не и на перформансите на вашиот код. Перформансите на TypeScript и JavaScript кодот со користење на Deno се приближно исти како и кодот што се извршува во Node.js.
На повисоко ниво, Node користи npm за управување со пакети, додека Deno технички нема менаџер на пакети. За разлика од Node, Deno може да увезува пакети директно од нивниот извор - deno.land CDN, GitHub или која било URL адреса што се решава на компатибилен модул.

Category	Node.js	
Language	JavaScript	TypeScript, JavaScript
Packages	NPM	.ts or .JS modules
Security & Permissions	No Mechanisms	CLI Flags
Code Engine	V8 JavaScript	V8 JavaScript
Security	Full Access	Permissioned Access
TypeScript Support	Not Built In	Built In
Community Support	Strong and Huge	Relatively New
Importing Packages	Common Syntax	ES Modules
Async	Callbacks	Promises
Browser Support	Vague	Support Provided

4. Градба на апликација

1. Градба на Node.js апликација

Кодот започнува со линкување на експрес модул и креирање на пример од апликацијата Express. Потоа е дефинирана константата порта - PORT која е поставена на 3000, што ја означува портата на која серверот ќе слуша.

```
1   const express = require('express');
2
3   const app = express();
4   const PORT = 3000;
5
```

Слика бр:1 Дел од кодот во **index.js**

Среден софтвер:

Се користат две функции на среден софтвер:

express.urlencoded({ проширено: неточно }): Овој среден софтвер ги анализира дојдовните барања со товари шифрирани со URL (обично од HTML форми) и го пополнува req.body со анализираните податоци.

express.json(): Овој среден софтвер ги анализира дојдовните JSON товари и го пополнува req.body со анализираните JSON податоци.
Апликацијата е исто така поставена да го користи моторот за шаблони EJS (Вграден JavaScript) за прикажување прикази.

Надминување на методот Middleware:

Овој среден софтвер проверува дали телото на барањето содржи својство _method. Ако го прави тоа, соодветно го менува методот на барање. Ова се користи за симулирање на HTTP методи како PUT и DELETE, кои не се природно поддржани од HTML форми.

```
6     // Middleware
7     app.use(express.urlencoded({ extended: false }));
8     app.use(express.json());
9
10    // Set EJS as the view engine
11    app.set('view engine', 'ejs');
12
13    // Override method middleware
14    app.use((req, res, next) => {
15        if (req.body && req.body._method) {
16            req.method = req.body._method;
17            delete req.body._method;
18        }
19        next();
20    });
21
```

Слика бр:2 Дел од кодот во **index.js**

Низа задачи:

Создадена е низа за задачи за да се складираат објектите на задачите.

Насоки:

GET / route: го прикажува приказот „индекс“, пренесувајќи во низата задачи и тековниот датум како параметри.
POST / рута: Се справува со поднесување нова задача. Телото на барањето се очекува да содржи име на својства, датум на задача и опис. Датата на задачата е поделена на компоненти на година, месец и ден, а нов објект за задача се креира и додава во низата задачи.
DELETE / route: Се справува со бришење на задача. TaskIndex се добива од телото на барањето, а соодветната задача се отстранува од низата задачи.

```
// Routes
app.get('/', (req, res) => {
    const today = new Date().toLocaleDateString(); // Get today's date as a string
    res.render('index', { tasks, today });
});

app.post('/', (req, res) => {
    const { name, taskDate, description } = req.body; // Modified: Use "taskDate" instead of "Date"
    const [year, month, day] = taskDate.split('-'); // Split the date string into year, month, and day
    const task = {
        name,
        date: new Date(year, month - 1, day), // Create a new Date object using the extracted values
        description,
    };
    tasks.push(task);
    res.redirect('/');
});

// Delete route
app.delete('/', (req, res) => {
    const taskIndex = req.body.taskIndex;
    tasks.splice(taskIndex, 1);
    res.redirect('/');
});
```

Слика бр3 Дел од кодот во **index.js**

Почеток на серверот:

Серверот се стартува со повикување на функцијата app.listen(). Слуша на наведената порта и евидентира порака кога ќе се стартува серверот.
Накратко, овој код поставува основна веб-апликација користејќи Express.js за управување со задачите. Вклучува рути за прикажување список со задачи, додавање задачи и бришење задачи. Задачите се зачувани во лажна низа, а апликацијата користи EJS за прикажување прикази. Дополнително, кодот вклучува среден софтвер за справување со прескокнување на методот со користење на својството _method во телата на барањата.

Шаблони за преден дел

Апликацијата To-Do користи EJS (вграден JavaScript) како мотор за шаблони за да генерира динамична HTML содржина на серверот. EJS овозможува вметнување JavaScript код во HTML шаблони, овозможувајќи динамично прикажување врз основа на податоци од страна на серверот

Директориумот Views/ ја содржи датотеката index.ejs, која служи како главен EJS шаблон за прикажување на HTML содржината на почетната страница на апликацијата.

2. Градба на Deno апликација

Апликацијата ја следи архитектурата клиент-сервер. Клиентот, претставен преку веб-прелистувачи, комуницира со серверот хостиран на времетраењето на Deno. Серверот се справува со барањата и одговорите на HTTP, прикажувајќи HTML шаблони и управува со податоците за задачите.

Средниот софтвер рутер.allowedMethods() се користи за справување со дозволите за методот HTTP.

Постои сопствен интерфејс за задачи за да се дефинира структурата на задачата. Дено рутер се користи за дефинирање маршрути за различни дејства: наведување задачи, додавање задачи и бришење задачи.

Апликацијата користи глобална низа задачи за складирање задачи.

Рутата router.get се справува со root URL-то (/) и прикажува EJS приказ со задачи и денешниот датум.

Шаблони за EJS: Функцијата renderFileToString од библиотеката dejs се користи за прикажување на HTML шаблони. Шаблоните EJS содржат динамични поставки кои се заменуваат со вистински податоци пред да се рендерираат.

Рутер: Класата Router на рамката Oak се користи за дефинирање маршрути како што се „/“, „/add“ и „/delete“. Секоја рута одговара на одредена функционалност.

Рутата router.get се справува со root URL-то (/) и прикажува EJS приказ со задачи и денешниот датум.

Рутерот.пост рутата за додавање задачи (/додај) ги чита податоците од формуларот и додава нова задача во низата со задачи.

Рутерот.пост рутата за бришење задачи (/избриши) ги чита податоците од формуларот за да го идентификува индексот на задачи и ја отстранува задачата од низата задачи.

Управување со задачи: Податоците за задачите се зачувуваат во низа во рамките на серверот. Додавањето и бришењето задачи вклучува манипулирање со оваа низа. Информациите за задачата се прикажуваат на почетната страница со прикажување на шаблонот EJS со податоците од низата.

```
16  router.get("/", async (ctx: Context) => {
17      const today = new Date().toDateString();
18      const body = await renderFileToString("./views/index.ejs", { tasks, today, index: ctx.params.index });
19      ctx.response.body = body;
20  });
21
22
23  router.post("/add", async (ctx: Context) => {
24      const formData = await ctx.request.body().value;
25      const name = formData.get("name");
26      const taskDate = formData.get("taskDate");
27      const description = formData.get("description");
28
29      console.log("Received form data:", name, taskDate, description);
30
31          if (name && taskDate && description) {
32              const date = new Date(taskDate);
33              const task: Task = { name, date, description };
34              tasks.push(task);
35          }
36          ctx.response.redirect("/");
37      });
38  router.post("/delete", async (ctx: Context) => {
39      const formData = await ctx.request.body().value;
40      const taskIndex = formData.get("taskIndex");
41      console.log("Received delete request for taskIndex:", taskIndex);
```

Слика бр:4 Дел од кодот во **server.ts**

Кодот во датотеката deps.ts се користи за управување и организирање на надворешните зависности за апликацијата Deno. Поточно:
Зависностите на oak модулот (Application, Router, Context, send, Status) се користат за изградба на веб-апликации, дефинирање маршрути, справување со барања и одговори на HTTP и управување со кодови за статус на HTTP користејќи ја веб-рамката на Oak.
Зависноста од модулот dejs (renderFileToString) се користи за рендерирање на EJS шаблони, што овозможува генерирање и рендерирање на динамична содржина во рамките на апликацијата Deno.
Со извоз на овие зависности од deps.ts, остатокот од апликацијата лесно може да ги увезе и користи. Овој пристап промовира модуларност и го олеснува одржувањето и ажурирањето на зависностите на апликацијата

```
// deps.ts
export {
    Application,
    Router,
    Context,
    send,
    Status,
} from "https://deno.land/x/oak/mod.ts";
export {
    renderFileToString,
} from "https://deno.land/x/dejs/mod.ts";
```

Слика бр:5 Дел од кодот во **deps.ts**

Предизвици со кои се соочуваат:

Крива на учење: Бидејќи Deno е релативно нова средина за извршување, постоеше крива на учење во разбирањето на неговите карактеристики, безбедносниот модел и системот на модули.

Увоз на модули: Увозот на модули користејќи URL-адреси беше уникатен концепт. Обезбедувањето на компатибилност и достапност на модулите бара истражување. Конфигурација и поставување: Поставувањето на околината за извршување на Deno, управувањето со зависности и конфигурирањето на серверот бара темелно разбирање на алатките и екосистемот на Deno.

Идни подобрувања:

Проектот може да се прошири на различни начини:

Корисничка автентикација: имплементирајте автентикација на корисникот за да обезбедите безбеден пристап до управувањето со задачите. Сортирање и филтрирање задачи: Подобрете го управувањето со задачите со додавање опции за сортирање и филтрирање. Приоритетизација на задачите: Дозволете им на корисниците да постават приоритети и рокови на задачите. Потсетници за задачи: имплементирајте потсетници за е-пошта или известувања за претстојните задачи.

Шаблони за преден дел

Апликацијата To-Do користи EJS (вграден JavaScript) како мотор за шаблони за да генерира динамична HTML содржина на серверот. EJS овозможува вметнување JavaScript код во HTML шаблони, овозможувајќи динамично прикажување врз основа на податоци од страна на серверот.

Директориумот Views/ ја содржи датотеката index.ejs, која служи како главен EJS шаблон за прикажување на HTML содржината на почетната страница на апликацијата.

3. Споредба на двете апликации

Јазик и рамковна споредба
Node.js користи JavaScript и рамката Express.js.
Deno користи TypeScript и рамката Oak.
Споредба на структурата на апликацијата
И двете апликации имаат јасна поделба на грижите.
Модуларниот пристап на Deno овозможува пофлексибилна организација.
Споредба на среден софтвер и рутирање
Двете апликации користат среден софтвер за парсирање на податоците за барањето и префрлањето на методот.
Node.js користи Express.js за рутирање.
Deno користи Oak за рутирање.
Прикажи споредба на рендерирање
Двете апликации користат EJS шаблони за прикажување прикази.
Споредба за управување со задачи
Двете апликации користат низа за управување со задачи.
Споредба на ракување со формулари
И двете апликации ги анализираат податоците од формата и соодветно се справуваат со задачите.
Споредба на корисничко искуство
Двете апликации нудат слично корисничко искуство со можност за додавање и бришење задачи.
Интерфејсот на Deno е малку помодуларен поради употребата на насочувањето на Oak.
Споредба на среден софтвер и рутирање
И апликациите Node.js и Deno користат среден софтвер за справување со различни аспекти од обработката на барањата. Во апликацијата Node.js, Express.js ја поедноставува употребата на среден софтвер преку својот интуитивен интерфејс. Функциите на Middleware се регистрираат со помош на методот app.use(), што им овозможува на програмерите лесно да се справат со задачи како што се парсирање на податоците за барањето и надминување на методите за барање.
Слично на тоа, во апликацијата Deno, Oak обезбедува можности за среден софтвер за обработка на барања. Функциите на Middleware може да се поврзат заедно за да се справат со различни аспекти на барањето. Модуларниот пристап на Oak ја подобрува организацијата и читливоста на кодот, што го прави добро прилагоден за сложени апликации.
Прикажи споредба на рендерирање
Двете апликации го користат моторот за прегледување EJS (Вграден JavaScript) за прикажување на динамична HTML содржина. EJS шаблоните им овозможуваат на програмерите да вградат JavaScript код во HTML, овозможувајќи динамично генерирање содржина врз основа на податоци од страна на серверот. Овој пристап го поедноставува процесот на прикажување прикази и обезбедува конзистентно корисничко искуство низ различни правци.

Апликацијата Node.js го конфигурира EJS како мотор за прегледување користејќи ја изјавата app.set('view engine', 'ejs'). Ова ја поставува сцената за рендерирање на EJS шаблони како одговор на барањата на клиентите.
Апликацијата Deno користи сличен пристап со интегрирање на EJS за прикажување приказ. Функцијата renderFileToString обезбедена од модулот EJS на Deno овозможува прикажување на EJS шаблони. И покрај разликите во синтаксата и системот на модули, основниот концепт на користење EJS за динамично генерирање содржина останува конзистентен во двете апликации.
Споредба за управување со задачи
Пристапот за управување со задачи во двете апликации вклучува складирање задачи во низа, обезбедувајќи основен механизам за складирање податоци во меморијата. Иако овој пристап е погоден за мали апликации и цели за демонстрација, можеби не е погоден за производствени средини кои бараат постојано складирање на податоци.
Во апликацијата Node.js, задачите се додаваат и отстрануваат од низата задачи додека корисниците комуницираат со апликацијата. Употребата на едноставна низа ја илустрира основната функционалност на апликацијата без да внесува непотребна сложеност.
Слично на тоа, апликацијата Deno користи низа за управување со задачи, демонстрирајќи ја паралелната природа на управувањето со задачи во двете технологии. Сепак, важно е да се забележи дека на двете апликации им недостига истрајност на податоците надвор од траењето на апликацијата.
Споредба на ракување со формулари
Ракувањето со формулари е критичен аспект на двете апликации, овозможувајќи им на корисниците да комуницираат со апликацијата со испраќање податоци. Двете апликации гарантираат дека податоците од поднесените формулари се правилно анализирани и обработени за додавање или бришење задачи.
Во апликацијата Node.js, функциите за среден софтвер express.urlencoded() и express.json() се користат за анализирање на податоците од формата и товарите на JSON, соодветно. Овие податоци потоа се користат за создавање нови задачи или отстранување на постоечки. Функциите на среден софтвер обезбедени од Express.js го поедноставуваат процесот на работа со податоците од формуларот.
Апликацијата Deno ја користи функцијата ctx.request.body() обезбедена од рамката Oak за да ги анализира податоците од формуларот. Овие податоци потоа се користат за извршување слични дејства за додавање или бришење задачи. Додека синтаксата и имињата на методите се разликуваат помеѓу двете апликации, основниот концепт на парсирање и обработка на податоците од формата останува конзистентен.
Споредба на корисничко искуство
Двете апликации нудат корисничко искуство насочено околу управувањето со задачи, овозможувајќи им на корисниците да додаваат, прегледуваат и бришат задачи. Корисничките интерфејси вклучуваат форми за внесување детали за задачите, заедно со списоци за прикажување задачи категоризирани според датумот на доспевање.
Во апликацијата Node.js, задачите се категоризираат во две листи: задачи што не доспеваат денес и задачи што доспеваат денес. Оваа организација им обезбедува на корисниците ајасен преглед на нивните задачи, овозможувајќи ефективно управување со задачите.
Апликацијата Deno следи слична структура, обезбедувајќи им на корисниците можност да додаваат и бришат задачи. Како и апликацијата Node.js, задачите се категоризираат

според датумот на доспевање, со што се подобрува разбирањето на корисниците и приоритизирањето на задачите.

5. Објаснување за миграција:

Апликацијата заснована на Deno е приклучок на апликацијата TODO базирана на Node.js до траењето на Deno. Ја користи рамката Oak за рутирање и справување со барања. Дено-кодот следи слична структура на кодот Node.js, но користи модули и синтакса специфични за Дено.

Миграцијата вклучува преведување на кодот Express.js во рамката Oak, прилагодување на среден софтвер и ракување со маршрутата и користење на Deno-специфични модули за различни функционалности како што се читање тела на барања, рендерирање на EJS шаблони и сервисирање статични датотеки.

Двете верзии на апликацијата постигнуваат слична функционалност, но се имплементирани со користење на алатките и конвенциите на нивните соодветни опкружувања за извршување.

6. Карактеристики на апликацијата

1. SRS

Функционални барања

1. Апликацијата треба да овозможи додавање на нови задачи.
2. Апликацијата треба да овозможи избирање на соодветен датум на задачата која што се внесува
3. Апликацијата треба да овозможи додавање на опис за задачата која што ја внесува корисникот
4. Апликацијата треба да им овозможи на корисниците бришење на задачите.
5. Апликацијата треба да овозможи прикажување на сите задачи кои што се внесени.
6. Апликацијата треба да овозможи поделба на задачите по датум.

Нефункционални барања

1. Апликацијата треба да биде разбирлива за употреба за сите луѓе.
2. Апликацијата треба да биде достапна за сите електронски уреди.

3. Апликацијата треба да им овозможи на корисниците приказ на задачите од денес и останатите денови.
4. Апликацијата ќе биде бесплатна за користење.
5. Веб апликацијата ќе биде достапна во секое време.
6. Веб апликацијата ќе работи на сите веб-прелистувачи.
7. Секое барање ќе се обработи во рок од 10 секунди.
8. Системот ќе подржува минимум 1000 корисници во еден момент.

2. Карактеристики

Додавање задача

За да додадете нова задача, следете ги овие чекори:

1. Внесете го името на задачата во полето за внесување „Внесете име на задачата“.
2. Изберете го датумот на задачата користејќи го внесувањето на избирачот датум.
3. Внесете опис за задачата во полето за внесување „Внеси опис“.
4. Кликнете на копчето "Додај задача".
5. Новата задача ќе биде додадена на списокот со задачи на почетната страница.

Прикажување на сите задачи

Сите задачи што не се закажани за денес се прикажани на почетната страница. Задачите се наведени во формат на картичка, прикажувајќи го името на задачата, датумот и описот. Секоја картичка со задачи вклучува копче „Избриши“ за да се отстрани задачата од списокот.

Погледнете ги задачите за денес

Задачите закажани за тековниот датум (денес) се прикажуваат посебно на почетната страница. Овие задачи се исто така наведени во формат на картичка со име, датум и опис на задачата. Секоја картичка со задачи вклучува копче „Избриши“ за да се отстрани задачата од списокот.

Избришете ја задачата

За да избришете задача, кликнете на копчето „Избриши“ на картичката со задачи или во делот „Сите задачи“ или „Задачи за денес“. Задачата ќе биде отстранета од списокот.

3. Предизвици со кои се соочуваат апликациите изградени во Deno

Deno е платформа за извршување на JavaScript и TypeScript код на серверска страна. Иако е нова и интересна технологија, сè уште може да се соочи со некои предизвици. Некои од нив вклучуваат:

Адоптирање на заедницата:

Deno има подобра заедница во споредба со почетокот, но уште се развива. Ова може да предизвика проблеми со наоѓање одговори на прашања, решавање на проблеми и наоѓање на одговарачи за развој на проект.

Библиотеки и модули:

Сеуште нема таква голема понуда на библиотеки и модули како во Node.js. Ова може да создаде предизвик при изборот на соодветни библиотеки за вашиот проект.

Компатибилност со постоечки код:

Доколку имате постоечки код напишан во Node.js, треба да го преработите за да работи во Deno. Ова може да биде времетрошно и предизвикувачки, особено ако кодот е комплексен.

Безбедност и дозволи:

Deno се фокусира на безбедноста и воведува модел со дозволи за пристап до ресурси како датотеки, мрежни барања итн. Ова може да биде предизвик за развојниците кои не се навикнати на овој начин на работа.

Инструменти и екосистема:

Иако Deno има некои инструменти како Deno CLI, уште се развиваат инструменти и екосистема околу платформата. Ова може да создаде предизвик при изградба, тестирање и деплојување на апликации.

Изведба:

Во зависност од видот на апликацијата, изведбата може да биде предизвик поради различни фактори како начинот на компајлирање и извршување на кодот.

Документација и ресурси:

Документацијата и учебните ресурси за Deno уште се развиваат и прошируваат. Ова може да создаде предизвик при учење на нови концепти и пристапи.

Корпоративна поддршка:

Сеуште има предизвици во добивањето на корпоративна поддршка и прифаќање на Deno како важна технологија во бизнис окружението.

4. Понатамошно истражување и напредни концепти

Напредно користење на среден софтвер

И апликациите Node.js и Deno користат среден софтвер за вообичаени задачи како што се парсирање на податоците за барањето и отфрлања на методот на ракување. Сепак, напредната употреба на среден софтвер може да обезбеди уште помоќни способности. На пример, може да имплементирате среден софтвер за автентикација за да го ограничите пристапот до одредени маршрути засновани на корисничките улоги или да имплементирате среден софтвер при справување со грешки за благодатно фаќање и справување со исклучоците.

Во апликацијата Node.js, средниот софтвер може да се прошири за да вклучи проверки за автентикација користејќи стратегии како што се JSON Web Tokens (JWT) или OAuth. Ова осигурува дека само овластените корисници можат да пристапат до одредени правци и да вршат конкретни дејства. Овој додаден слој на безбедност ја подобрува целокупната робусност на апликацијата.

Во апликацијата Deno, средниот софтвер на Oak може да се прошири за да вклучи дополнителни слоеви на безбедност или да имплементира сопствена логика за авторизација. Ова може да вклучува проверка на улогите или дозволите на корисникот пред да се дозволи пристап до одредени маршрути. Со интегрирање на напреден среден софтвер, програмерите можат да го приспособат однесувањето на апликацијата за да одговара на специфичните безбедносни барања.

Упорност на податоците со бази на податоци

Додека апликациите моментално користат низи во меморијата за складирање задачи, интегрирањето на базата на податоци може значително да го подобри упорноста и управувањето со податоците. Со поврзување на апликациите во базата на податоци, можете да се осигурате дека задачите се складирани сигурно дури и при рестартирање или паѓање на серверот.

Во апликацијата Node.js, можете да користите популарни бази на податоци како MongoDB, MySQL или PostgreSQL за складирање задачи. Ова ќе овозможи складирање на задачите низ сесиите и рестартирање на серверот, обезбедувајќи поцврсто и посигурно решение за складирање податоци.

Слично на тоа, апликацијата Deno може да има корист од интеграцијата на базата на податоци. Deno поддржува различни драјвери за бази на податоци кои овозможуваат интеракција со бази на податоци како MongoDB, PostgreSQL и SQLite. Со интегрирање на базата на податоци, можете да ги искористите функциите како што се индексирање на податоци, барање и трансакции со ACID, обезбедувајќи интегритет на податоците и приспособливост.

RESTful API и раздвојување од предниот дел

Додека апликациите првенствено се фокусираат на прикажување од страна на серверот, додавањето на RESTful API може да отвори нови можности за развој на предниот дел. Со креирање на посебен API, овозможувате развој на динамични и

интерактивни апликации од предниот дел со користење на современи JavaScript рамки.
Покрај сервирањето HTML прикази, апликациите би можеле да изложат крајни точки на API за задачи, дозволувајќи им на предните рамки како React, Angular или Vue.js да преземаат и прикажуваат податоци за задачите. Оваа поделба на грижи промовира помодуларна и поодржлива база на кодови.
Пристапот RESTful API овозможува и развој на мобилни апликации кои можат да комуницираат со истиот извор на податоци, проширувајќи го досегот на апликацијата до поширок опсег на уреди и платформи.
Контејнеризација и распоредување
Контејнеризацијата стана стандардна практика во современиот развој на софтвер. И апликациите Node.js и Deno можат да имаат корист од контејнеризацијата користејќи технологии како Docker. Контејнеризацијата обезбедува неколку предности, вклучувајќи конзистентни околини, полесно распоредување и приспособливост.
Со контејнеризирање на апликациите, ги инкапсулирате сите зависности, конфигурации и опкружувања за траење во еден пакет. Овој пакет потоа може да се распореди низ различни средини, обезбедувајќи постојано однесување и намалувајќи ги проблемите поврзани со распоредувањето.
Дополнително, распоредувањето на апликациите на облак платформи како AWS, Google Cloud или Heroku може да обезбеди автоматско скалирање, балансирање на оптоварување и управување со инфраструктурата. Распоредувањето на облак дополнително го поедноставува процесот на правење на апликациите достапни за корисниците додека одржува висока достапност и перформанси.

7. Заклучок

Во оваа документација, истражувавме и споредивме две To-Do апликации изградени со помош на Node.js и Deno. Испитавме различни аспекти на секоја апликација, вклучувајќи ја нивната архитектура, користење на среден софтвер, рутирање, прикажување приказ, управување со задачи, ракување со форми и корисничко искуство.
Додека апликацијата Node.js го користи зрелиот и добро воспоставен екосистем Node.js, апликацијата Deno ги прикажува современите безбедносни карактеристики на Deno и домашната поддршка за TypeScript. Двете апликации ја демонстрираат основната функционалност на апликацијата To-Do и обезбедуваат основа за идни подобрувања.
Како што пејзажот за развој на веб продолжува да се развива, програмерите имаат на располагање низа технологии. Изборот помеѓу Node.js и Deno зависи од проектните барања, стручноста на тимот и преференциите за јазичните карактеристики и безбедносните модели. Без разлика дали го користат добро воспоставениот Node.js или поновиот Deno, програмерите можат да креираат робусни и прифатливи веб-апликации за да ги задоволат потребите на нивните корисници.