Laboratorij za informacijske tehnologije

Dostop do podatkov v zaledju

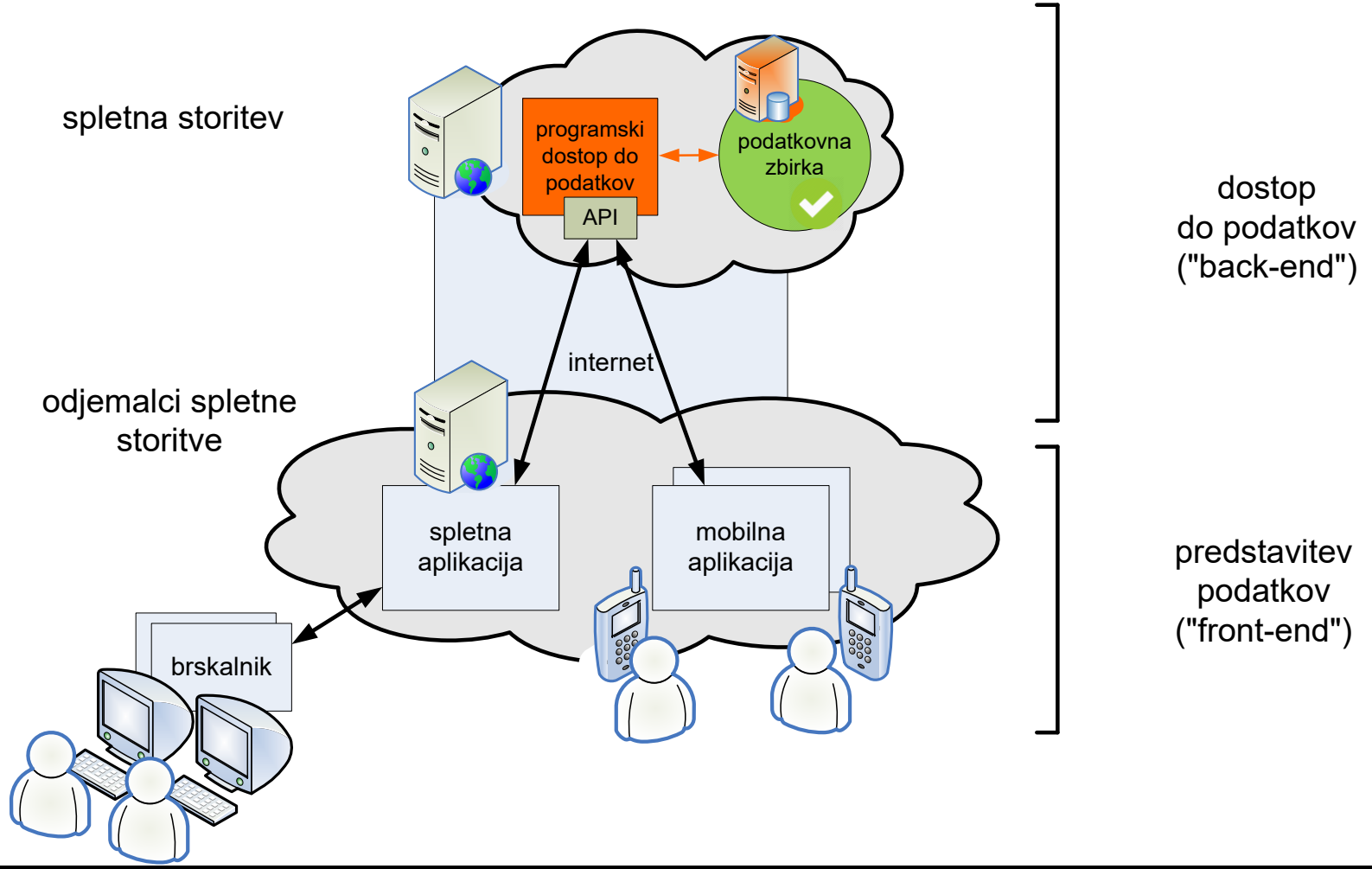
Grega Jakus, 2025/26

16. 10. 2025

Danes

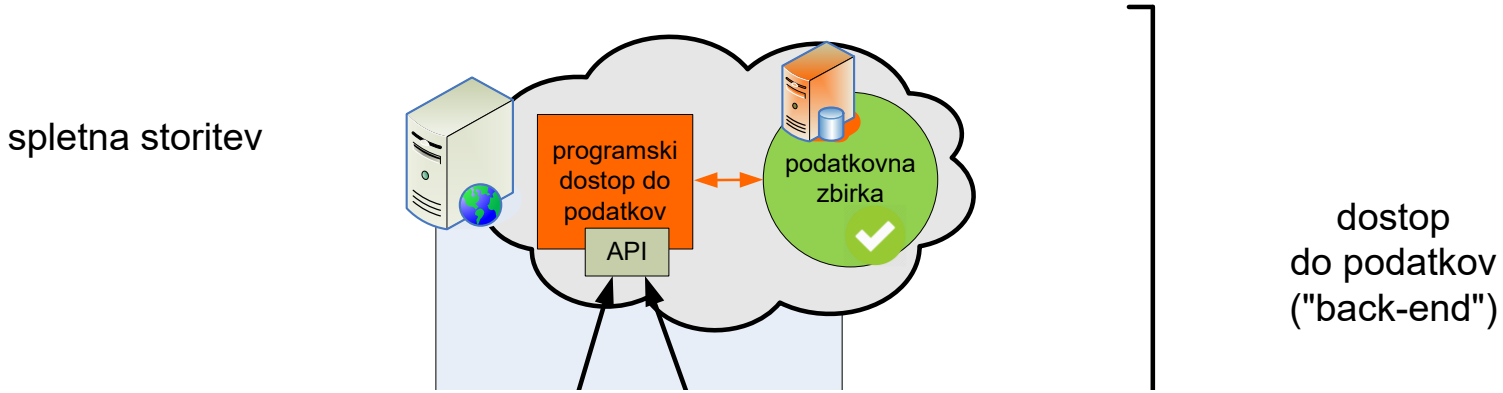
- Aplikacijski strežniki
- Node.js in Express
- Nekaj konceptov: MVC, sinhronost/asinhronost
- JavaScript
- Dostop do podatkov iz zaledja

Dostop do podatkov v arhitekturi sistema



Aplikacijski strežnik

- Aplikacijski strežnik je poleg podatkovnega strežnika najpomembnejši del zaledja
- Aplikacijski strežnik gosti in izvaja poslovno logiko internetne aplikacije
- Sprejema zahteve od odjemalcev in pripravlja dinamični odgovor, pri čemer dostopa do podatkov, ki mu jih streže podatkovni strežnik
- Lahko izvaja tudi druge naloge (gručenje (ang. *clustering*), varnostne storitve, transakcije, ...)



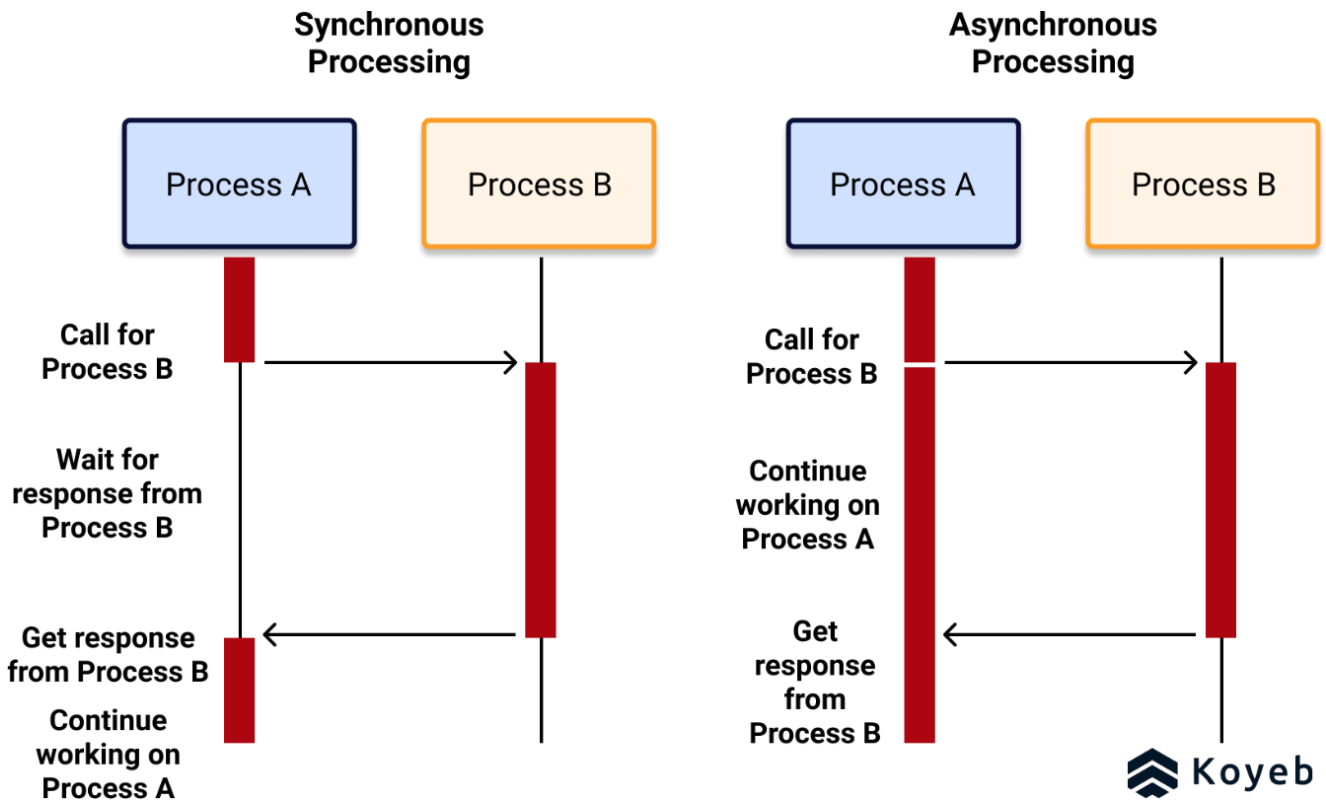
Aplikacijski strežniki

- Profesionalni („enterprise“) strežniki
  - večinoma uporabljajo javansko platformo
  - Oracle WebLogic Server, IBM WebSphere Application Server, WildFly, Apache Tomcat, ...
  - pogosto uporabljajo ločen spletni strežnik kot reverzni posrednik (ang. *reverse proxy*), npr. Nginx, Apache
- „Lahki“ strežniki
  - za hiter razvoj in mikrostoritve
  - kombinirajo vlogo aplikacijskega in spletnega strežnika
  - Node.js + Express
  - Django in Flask
  - Ruby on Rails
- Klasični spletni strežniki s podporo različnim izvajalnim okoljem
  - npr. Apache + {PHP/Perl/Python}
  - danes izgubljajo na veljavi

Node.js

- Node.js je **izvajalno okolje** (ang. *runtime environment*) za jezik **JavaScript**
- JavaScript se navadno izvaja v spletnem brskalniku – torej *v kontekstu spletnega odjemalca* (v ospredju)
- Node.js omogoča, da se JavaScript koda izvaja *v kontekstu spletnega strežnika* (v zaledju)
- Prednost: enoten jezik za celotno spletno aplikacijo
- Node.js se v kombinaciji z ogrodji, kot je Express.js, danes pogosto uporablja kot lahek aplikacijski strežnik za izvajanje mikrostoritev z aplikacijskimi programskimi vmesniki (API), predvsem po vzorcu REST
- Ostale ključne lastnosti
  - asinhrono delovanje – operacije (npr. dostop do podatkovne zbirke) se ne blokirajo, ker so izvedene vzporedno
  - obsežen ekosistem – upravljalnik paketov za Node.js (npm) je eden največjih
  - samostojnost in neodvisnost aplikacij – te niso npr. odvisne od različic knjižnic, ki jih uporabljajo druge aplikacije (ang. *dependency*)
  - relativno visoka hitrost izvajanja JavaScript kode

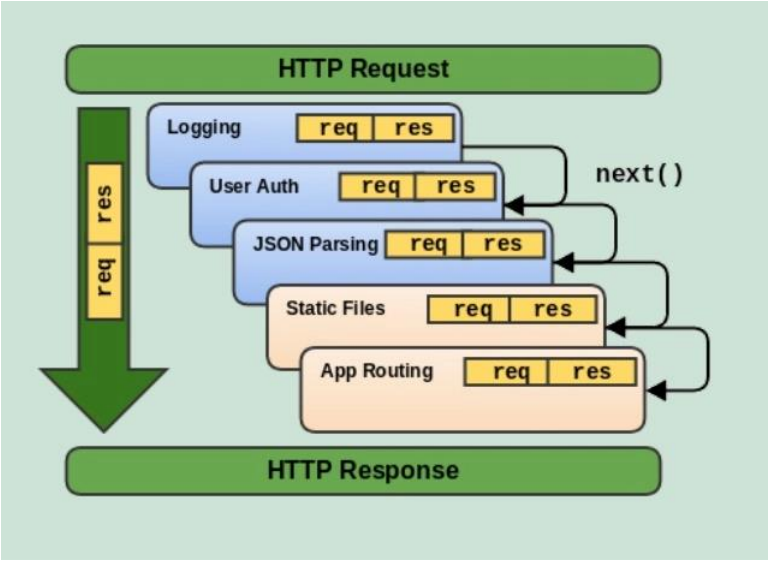
Sinhrono in asinhrono izvajanje



- Sinhrono izvajanje
  - za hitre in preproste operacije
  - za začetno nalaganje
  - če smo odvisni od rezultata prejšnjega koraka
- Asinhrono izvajanje
  - ko je izvajanje dolgotrajno
  - ko je izvajanje lahko vzporedno (neodvisno)
  - ko ne želimo blokirati uporabniške interakcije (v uporabniških vmesnikih)

Express.js

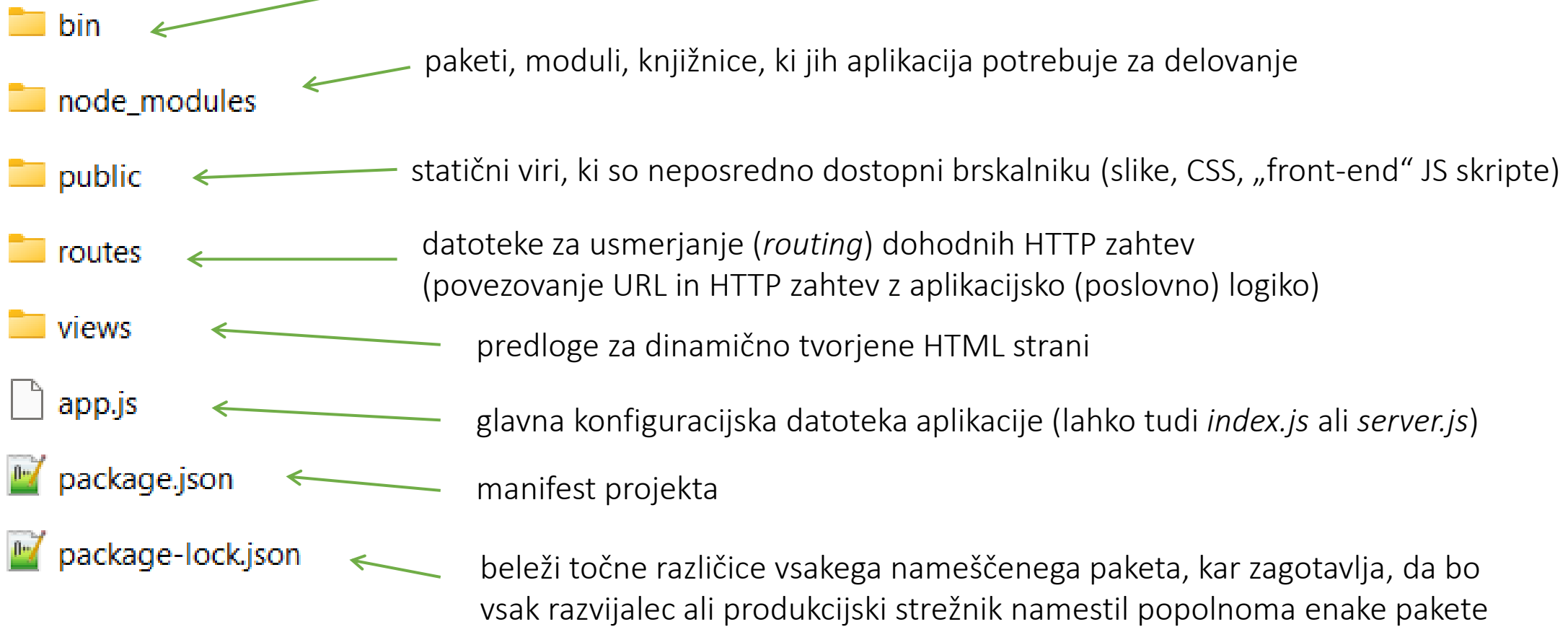
- Node.js je **izvajalno okolje** (ang. *runtime environment*) za jezik JavaScript
- Express.js je **spletno ogrodje za Node.js,** ki pomaga razvijalcem hitro in enostavno zgraditi spletne aplikacije
- Še posebej je primeren za izdelavo **aplikacijskih programskih vmesnikov** (API, Application Programming Interface)
- Glavne lastnosti
  - usmerjanje zahtev (*routing*) glede na URL naslov in HTTP metodo
  - vsiljuje delitev kode na logične bloke v skladu z modelom MVC
  - koncept vmesnih funkcij (*middleware*)
    - imajo dostop do objekta HTTP zahteve,
      preden ta doseže poslovno (aplikacijsko) logiko
    - uporabljajo se za izvajanje različnih nalog, kot so avtentikacija uporabnika, beleženje, razčlenjevanje vsebine telesa zahtev (JSON, XML), ...,
  - fleksibilnost – omogoča svobodo pri izbiri podatkovnih zbirk, sistema predlog (ang. *templating engine*), ...



Namestitev Express.js in ostalih orodij

- Ustvarite mapo za aplikacijo, zaženite konzolo (*cmd*) in se premaknite v ustvarjeno mapo
- `npm init` `// ustvari konfiguracijsko datoteko package.json`
- `npm install express` `// namesti express`
- `npx express-generator --view=ejs --force` `// namesti ogrodje aplikacije (s predlogami ejs)`
- `npm install` `// posodobitve`
- `npm audit fix --force` `// odpravljanje varnostnih ranljivosti`
- `npm install mysql2` `// namesti modul za mysql`
- `npm install --save-dev nodemon` `// modul za samodejni ponovni zagon strežnika ob spremembah`
- `npm install cors` `// modul za upravljanje deljenja virov med domenami(CORS)`
- `npm install bcrypt` `// modul za zgoščevanje gesel (hash)`
- V datoteki *package.json* dodajte v element *scripts* še vrstico: `"dev": "nodemon ./bin/www"`
- Poženite strežnik s pravkar ustvarjenim aplikacijskim ogrodjem z `npm run dev`
- Preizkusite delovanje v brskalniku (http://localhost:3000)

Struktura aplikacije



Glavna konfiguracijska datoteka (*app.js, index.js* ali server.js)

```
1  var createError = require('http-errors');
2  var express = require('express');
3  var path = require('path');
4  var cookieParser = require('cookie-parser');
5  var logger = require('morgan');
6
7  var indexRouter = require('./routes/index');
8  var usersRouter = require('./routes/users');
9
10 var app = express();
11
12 // view engine setup
13 app.set('views', path.join(__dirname, 'views'));
14 app.set('view engine', 'ejs');
15
16 app.use(logger('dev'));
17 app.use(express.json());
18 app.use(express.urlencoded({ extended: false }));
19 app.use(cookieParser());
20 app.use(express.static(path.join(__dirname, 'public')));
21
22 app.use('/', indexRouter);
23 app.use('/users', usersRouter);
24
25 // catch 404 and forward to error handler
26 app.use(function(req, res, next) {
27   next(createError(404));
28 });
29
30 // error handler
31 app.use(function(err, req, res, next) {
32   // set locals, only providing error in development
33   res.locals.message = err.message;
34   res.locals.error = req.app.get('env') === 'development' ? err : {};
35
36   // render the error page
37   res.status(err.status || 500);
38   res.render('error');
39 });
40
41 module.exports = app;
```

uvod modulov (*dependencies*)

inicializacija primerka aplikacije na osnovi ogrodja *express*

nastavitev mape s „pogledi“ (*views*) in sistema predlog za prikaz

dodajanje „middleware“ funkcij za predelavo HTTP zahtev pred njihovo obdelavo v aplikaciji

nastavitev usmerjevalnikov za določene poti

obravnava napake 404, če uporabljena pot nima usmerjevalnika

obravnava ostalih (posredovanih) napak

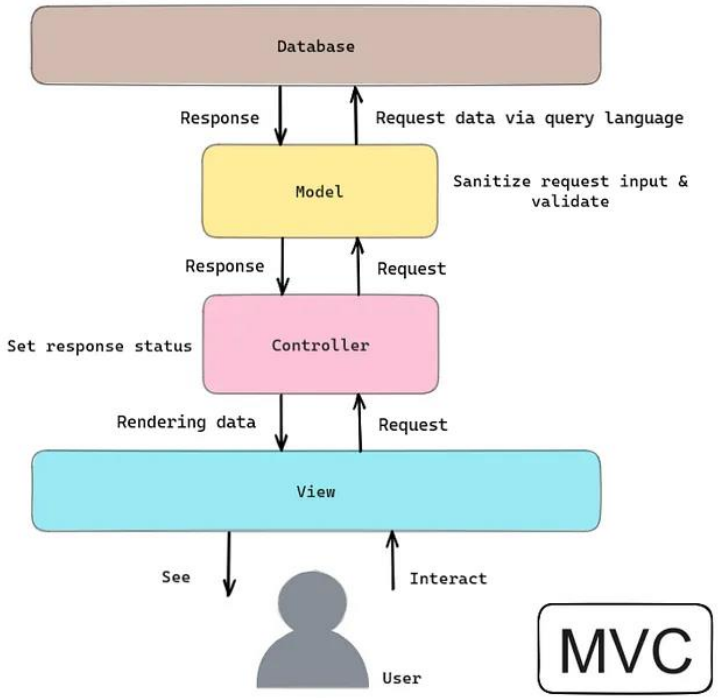
omogoči, da se primerek aplikacije uvozi v drugi datoteki (običajno v *bin/www*)

JavaScript

- **JavaScript (JS)** je eden izmed najbolj razširjenih programskih jezikov na svetu
- Prvotno je bil razvit za dodajanje **interaktivnosti** in **dinamike** statičnim spletnim stranem
- Danes se uporablja tako na strani odjemalca (brskalnika) kot na strani strežnika (Node.js)
- Visokonivojski, objektni, dinamično tipiziran, skriptni, interpretiran (tolmačen) jezik
- Osnove skladnje

Model MVC (v Node.js/Express)

- Eden najpomembnejših arhitekturnih vzorcev
- Glavna naloga je ločiti naloge znotraj aplikacije (ang. *separation of concerns*)
- Model (podatki in poslovna logika)
  - skrbi za interakcijo z zbirko podatkov (pridobivanje, shranjevanje, posodabljanje podatkov), preverjanje podatkov, uveljavljanje pravil nad podatki itd.
  - v Node.js je prepuščen zunanjim knjižnicam (v našem primeru npr. modulu *mysql2*)
- View (pogled, predstavitev podatkov)
  - odgovoren za uporabniški vmesnik
  - sprejme podatke od krmilnika in jih prikaže, omogoča uporabniško interakcijo
  - Express.js omogoča integracijo z različnimi sistemi za delo s predlogami (ang. *templating engines*), npr. EJS, Pug, ...
- Controller – krmilnik
  - posrednik med modelom in pogledom:
    1) prek pogleda sprejme zahtevo od uporabnika,
    2) kliče model, da pridobi/obdela podatke,
    3) odloči se za pogled, ki bo te podatke uporabil za prikaz rezultata in nadaljnjo interakcijo
  - vlogo krmilnika v Node.js ima sistem usmerjanja (routing)



Programski dostop do podatkov („Model“)

- Za dostop do podatkovne zbirke iz aplikacije moramo imeti možnost vzpostavitve povezave s sistemom za upravljanje z zbirkami (npr. v obliki gonilnikov in omrežne povezave)
- „Recept“ za uporabo podatkov iz zbirke je vedno enak
  - najprej ustvarimo povezavo s podatkovnim strežnikom
  - prek ustvarjene povezave dostopamo do podatkov izbrane podatkovne zbirke
  - podatke programsko obdelamo in po potrebi v zbirko podatkov dodamo nove podatke
  - sprostimo povezavo s podatkovnim strežnikom

Dostop do podatkov iz Node.js

```
const mysql = require('mysql2/promise');
```

Uvozimo modul za povezavo z izbranim podatkovnim strežnikom.

```
const pool = mysql.createPool({
    host: 'localhost',
    user: 'root',
    password: '',
    database: 'igra',
    waitForConnections: true,  // Čaka, če ni prostih povezav
    connectionLimit: 10,       // Največje število povezav
    queueLimit: 0              // Največje število čakajočih zahtev
});
```

Ustvarimo nabor povezav (pool), ki se jih da ponovno uporabiti.

Izvedemo poizvedbo in shranimo rezultat.

```
try {
    const [rows, fields] = await pool.execute('SELECT vzdevek, ime, priimek, email FROM igralec');
    //obdelamo podatke in vrnemo odgovor
    }
catch (err) {
    next(err);
}
```

V prestrezniku izjem morebitno izjemo prepustimo posebej določeni funkciji za obravnavo napak.

Ključna beseda *await* zaustavi izvajanje kode, dokler asinhrona funkcija ne vrne vrednosti. Uporabimo jo lahko le znotraj funkcije, ki je označena s ključno besedo *async* (asinhrona funkcija).

Naloge

- Za vsako izmed pripravljenih (8) poizvedb iz prejšnje vaje pripravite skripto, ki bo izvedla posamezno poizvedbo in po potrebi brskalniku posredovala rezultate

Poenostavitve in nadgradnje

- Da bi izpostavili bistvene stvari, smo se poslužili nekaj poenostavitev
- Overjanje: **shranjevanje gesel neposredno v kodi je veliko varnostno tveganje**, zlasti ko je aplikacija na produkcijskem strežniku. Varen način za shranjevanje občutljivih podatkov je uporaba **okoljskih spremenljivk**, npr. s pomočjo paketa dotenv
- Avtorizacija dostopa do zbirke
  - v sistemu za upravljanje s podatkovnimi zbirkami izdelamo novega uporabnika z geslom in ustreznimi pravicami
  - nikoli ne uporabljamo "vgrajenega" uporabnika "root" s praznim geslom
- Izvajanje poizvedb
  - namesto izvajanja poizvedb iz programske kode, se lahko poslužimo t.i. shranjenih postopkov (angl. *stored procedures*)
    - procedure ali funkcije, definirane v okviru sistema za upravljanje z zbirkami, ki ob klicu izvedejo v njih zapisane ukaze
  - če vseeno izvajamo poizvedbe neposredno iz programske kode, se zaradi varnosti (preprečitev napadov z vrivanjem poizvedb, angl. *query injection*) **nujno** poslužimo t.i. pripravljenih izrazov (angl. *prepared statements*)

```
... FROM igralec WHERE vzdevek=?', [vzdevekIgralca])
```

Domača naloga

- Namestite Postman

- Namesto Notepad++ si lahko si namestite tudi pravo razvojno okolje, predlagam Visual Studio Code