

Backend AdonisJS

kurz Vývoj progresívnych
webových aplikácií

Eduard Kuric



Node.js

- prostredie (runtime systém) umožňujúce **vykonávať JavaScript na strane servera**
 - založený na [Google V8 engine](#) (ako Chrome)
- otvorený zdrojový kód (open source, MIT licencia)
- asynchrónne, udalosťami riadené API (event-driven)
- zabudované moduly (napr. http, url, fs, events)
- <https://nodejs.org/>
- <https://kangax.github.io/compat-table/es2016plus/>



Node.js – HTTP modul

- Umožňuje vytvoriť HTTP server
 - HTTP protokol
 - počúvajúci na určenom porte

```
var http = require('http');

http.createServer(function (req, res) {
  res.writeHead(200, {'Content-Type': 'text/html'});
  res.end('Hello World!');
}).listen(8080);
```

> node myFirstNodeApp.js



Node.js – URL modul

- Uniform Resource Locator
- je referencia, ktorá určuje umiestnenie webového zdroja na Internete

scheme : [// [user[:password]@]host[:port]] [/path] [?query] [#fragment]

<https://www.eshop.com/smartphones/apple/list?order=ASC#pagination>



Node.js – URL modul

- Pársovanie URL adresy

```
var http = require('http');
var url = require('url');
```

```
http.createServer(function (req, res) {
  let adr =
    url.parse('http://localhost:8080/products?year=2022&month=february');
  res.writeHead(200, { 'Content-Type': 'text/html' });
  res.end('<pre>' + JSON.stringify(adr, null, 4) + '</pre>')
}).listen(8080);
```

```
> node URLModuleExample.js
```



Node.js – FS modul

- Vytváranie, vymazanie, zmena, premenovanie, čítanie zo súborov

```
var http = require('http');
var fs = require('fs');
http.createServer(function (req, res) {
  fs.readFile('demofile.html', function(err, data) {
    res.writeHead(200, {'Content-Type': 'text/html'});
    res.write(data);
    res.end();
  });
}).listen(8080);
```



Node.js – FS modul /2

```
// ak neexistuje vytvorí subor  
fs.appendFile('mynewfile.txt', 'Hello  
content!', function (err) {});
```

```
// ak neexistuje vytvorí subor, w - iba na zapis  
fs.open('mynewfile2.txt', 'w',  
function (err, file) {});
```

```
// nahradí existujúci subor/obsah, ak existuje  
fs.writeFile('mynewfile3.txt', 'Hello  
content!', function (err) {});
```



Node.js – FS modul /3

```
// vymazanie suboru  
fs.unlink('mynewfile2.txt', function (err)  
{ } );  
  
// premenovanie suboru  
fs.rename('mynewfile1.txt',  
          'myrenamedfile.txt',  
function (err) {});
```



Node.js – EVENTS modul

```
var events = require('events');
var eventEmitter = new events.EventEmitter();

// vytvorenie obsluhy pre udalost
var myEventHandler = function () {
    console.log('I hear a scream!');
}

// zaregistrovanie obsluhy k udalosti
eventEmitter.on('scream', myEventHandler);

// vyvolanie udalosti
eventEmitter.emit('scream');
```

AdonisJS



- Komplexný BE rámec na vývoj webových aplikácií
 - **Server Side Rendering (SSR)**
 - alebo ako **Web API server**
- Nad Node.JS
- Značne inšpirovaný rámcom Laravel
- Model-View-Controller (MVC) pattern (SSR)
- TypeScript ako natívny jazyk rámca

<https://adonisjs.com/>

AdonisJS – projekt



- **Vytvorenie projektu**
- `npm init adonisjs@latest hello-world`
- Projektová štruktúra:
 - web - klasické ssr appky
 - api - API server
 - slim - v podstate iba jadro rámca
- **Spustenie v dev režime:**
- `node ace serve --watch`
// --watch – automaticky reštartuje server pri zmene súborov

AdonisJS – koncepty



- Routing
- Context, request, response
- Controller
- Middleware
- Database
- ORM
- Auth
- ...

Routing



- start/routes.ts

```
Route.get('/', () => {
    return 'Hello world'
})
```

```
// /posts, obsluha controller, metoda index
Route.get('posts', 'PostsController.index')
```

Routing – HTTP metódy



- HEAD
- OPTIONS
- **GET**
- **POST**
- PUT
- PATCH
- DELETE

Routing – GET



- **na získanie dát** (zdroja - dokumentu...)
- používa URL, odovzdáva parametre
- má limitovanú dĺžku
- môže byť cachovaná, uložená v histórii prehliadača
- **nikdy by nemala byť použitá na prenos citlivých údajov**

Routing – POST



- **na posielanie/vytváranie dát (často formulárové dát)**
- „nemá“ obmedzenie na dĺžku dát
- dáta nie sú cachované, nezostávajú v histórii prehliadača
- parametre z URL sa odosielajú v tele správy

Routing CRUD



- `router::resource('users', 'UserController');`

<i>HTTP metóda</i>	<i>URL</i>	<i>Akcia</i>
• GET	/users	index
• GET	/users/create	create
• POST	/users	store
• GET	/users/{user}	show
• GET	/users/{user}/edit	edit
• PUT/PATCH	/users/{user}	update
• DELETE	/users/{user}	destroy

HTTP Context



- Špecifický objekt, ktorý agreaguje viaceré informácie v súvislosti s požiadavkou (angl. request-specific object), obsahuje napr.:
 - Hlavica požiadavky
 - Telo požiadavky
 - Cookies
 - Aktuálne prihláseného používateľa
- Je dostupný/prístupný kdekoľvek v aplikácii

HTTP Context /2



```
// request, auth, response su vlastnosti
// HTTP Context objektu

router.get('/', ({ request, auth, response, view }) => {
    console.log(request.url())
    console.log(auth.user)
    response.send('hello world')
    // return view.render('welcome')
})
```

- Ďalšie vlastnosti napr.: session, params

HTTP Context /3



```
import type { HttpContext } from '@adonisjs/core/http'

// HTTP context ako parameter
export default class PostsController {
  public async index(ctx: HttpContext) {
    }
}

// pristup k HTTP Contextu odkialkolvek
class SomeService {
  public async someOperation() {
    const ctx = HttpContext.get()
    }
}
```

CTX: request



```
router.get('/posts/:id/:slug', async ({ request }) => {  
  /*  
   * URL: /posts/1/hello-world  
   * Params: { id: '1', slug: 'hello-world' }  
  */  
  request.params()  
}  
  
// hodnota konkretného parameteru z URL adresy  
request.param('id')  
// hodnota konkretného vstupného pola /form input  
request.input('title')
```

CTX: response



```
router.get('/', () => {
    return 'This is the homepage'
    // response.send('hello world')
})  
  
response.header('Content-type', 'text/html')
response.status(401)
response.redirect().toPath('/some/url')
```

Controller



- Obsluha HTTP požiadaviek
- Biznis logika
- Triedy obsahujúce metódy
 - Konkrétnie požiadavky (requests) obsluhujú konkrétnie metódy
- Vytvárame v priečinku: app/Controllers/Http
- Cez príkazový riadok:
 - node ace make:controller **Post**

```
// routu /posts obsluzi metoda index
// controllera PostsController:
• router.get('posts',
  '#controllers/posts_controller.index')
```

PostsController



```
import type { HttpContext } from '@adonisjs/core/http'

export default class PostsController {
  public async index(ctx: HttpContext) {
    ctx.response.send('posts')
  }
}
```

Middleware



- Logika (funkcia), ktorá je vykonaná nad požiadavkou (requestom) predtým, ako sa ňou začne zaoberať obsluha (napr. metóda controllera)
- Môžu byť reťazené
 - požiadavku spracuje funkciaA, posunie ju funkciuB, ..., metoda controllera
- V pričinku: app/middleware
- Vytvorenie middleware:
- node ace make:middleware MyFirstMiddleware

Middleware /2



```
import type { HttpContext } from '@adonisjs/core/http'

export default class MyFirstMiddleware {
    public async handle({ }: HttpContext, next: () =>
Promise<void>) {
        // code for middleware goes here. ABOVE THE NEXT CALL
console.log('posts request')
        await next()
    }
}
```

**MIDDLEWARE MUSÍME ZAREGISTROVAŤ
v `start/kernel.ts`**

Middleware kernel.ts



- Globálny middleware
 - prejde ním každá HTTP požiadavka, napr. sessions
- Pomenovaný (named) middleware:

```
export const middleware = router.named({  
  myFirst: () => import('#middleware/my_first_middleware'),  
})
```

- Použijeme middleware v routes:

```
import { middleware } from '#start/kernel'  
router.get('posts',  
  '#controllers/posts_controller.index').use(middleware.my  
First())
```

Database



- Podpora SQL relačných databáz
 - PostgreSQL, MySQL, MariaDB, MSSQL, SQLite
- Je potrebné doinštalovať podporu – balíček **lucid**:
 - npm i @adonisjs/lucid

```
// vytvori konfiguracny subor a priecinok database
```

- node ace configure @adonisjs/lucid
- Konfigurácia v súbore config/database.ts

Database - SQLite



- Zjednodušený relačný DBMS napísaný v C
- Databáza uložená v jednom súbore
- Syntax SQL, neobsahuje pokročilé vlastnosti
- Použitie, napr. ako relačná databáza v mobilných telefónoch

“[SQLite supports](#) an unlimited number of simultaneous readers, but it will only allow one writer at any instant in time. For many situations, this is not a problem. Writers queue up. Each application does its database work quickly and moves on, and no lock lasts for more than a few dozen milliseconds. But there are some applications that require more concurrency, and those applications may need to seek a different solution.”

- <https://www.sqlite.org/index.html>

Database - SQLite



- Nainštalujeme sqlite:
 - npm i @vscode/sqlite3
- Vytvoríme súbor tmp/db.sqlite3

Auth



- Nainštalujme balíček na autentifikáciu:
 - npm i @adonisjs/auth
- Spustime konfiguráciu pre auth:
 - node ace configure @adonisjs/auth
 - // lucid
 - // guard: web
 - // model: User
 - // migracia: y
- Nainštalujeme balíček na hashovanie (hesiel):
 - npm i phc-argon2
- auth vytvori migráciu, spustíme ju:
 - node ace migration:run

Auth - register/login



- Vytvorme UsersController:

- node ace make:controller User
- import User from '#models/user'
- import hash from '@adonisjs/core/services/hash'

- Vytvorme metódu getRegister:

```
public async getRegister(ctx: HttpContext) {  
    return ctx.view.render('user/register')  
}
```

Adonis má podporu na vytváranie šablón (Edge):

- Vytvorme šablónu register:
- node ace make:view user/register

Auth - register šablóna



```
@layout.app({ title: "Register title" })

@slot('meta')

  <meta name="description" content="A welcome page made with EdgeJS" />

@endslot

<h1>
  Register User
</h1>

<form action="/register" method="POST">
  {{ csrfField() }}

  <div class="form-group">
    <label for="">Email</label>
    <input type="text" name="email" class="form-control" />
  </div>

  <div class="form-group">
    <label for="">Password</label>
    <input type="password" name="password" class="form-control" />
  </div>

  <input type="submit" value="Submit" class="btn btn-primary" />
</form>

@end
```

Auth - Layout



- `node ace make:view components/layout/app`

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8" />
    <title>
      {{ title || "Your default title" }}
    </title>
    @if ($slots.meta)
      {{ await $slots.meta() }}
    @endif
  </head>
  <body>
    {{ await $slots.main() }}
  </body>
</html>
```

Auth - Routing



```
router.get('register',
  '#controllers/users_controller.getRegister')

router.get('login',
  '#controllers/users_controller.getLogin')

router.post('register',
  '#controllers/users_controller.postRegister')

router.post('login',
  '#controllers/users_controller.postLogin')
```

Auth - postRegister



- Do UsersController-a pridáme metódu na zaregistrovanie používateľa:

```
import User from '#models/user'

public async postRegister(ctx: HttpContext) {
    const data = ctx.request.only(['email', 'password'])
    const user = await User.create(data)
    return ctx.response.redirect('/')
}
```

Auth – login šablóna



- **node ace make:view user/login**

```
@layout.app({ title: "Login title" })

@slot('meta')

<meta name="description" content="A welcome page made with EdgeJS" />

@endslot

<h1>

    Login User

</h1>

<form action="/login" method="POST">

    {{ csrfField() }}

    <div class="form-group">

        <label for="">Email</label>

        <input type="text" name="email" class="form-control" />

    </div>

    <div class="form-group">

        <label for="">Password</label>

        <input type="password" name="password" class="form-control" />

    </div>

    <input type="submit" value="Submit" class="btn btn-primary" />

</form>

@end
```

Auth - getLogin



- Do UsersController-a pridáme metódu na obsluhu požiadavky pre prihlásenie:

```
public async getLogin(ctx: HttpContext) {  
    try {  
        await ctx.auth.use('web').authenticate()  
        ctx.response.send('You are already logged in')  
    } catch {  
        return ctx.view.render('user/login')  
    }  
}
```

Auth - postLogin



- Do UsersController-a pridáme metódu na obsluhu požiadavky pre prihlásenie ako také:

```
public async postLogin(ctx: HttpContext) {  
    const { email, password } = ctx.request.only(['email', 'password'])  
    const user = await User.findBy('email', email)  
  
    if (!user) {  
        return ctx.response.abort('Invalid credentials')  
    }  
    const isPasswordValid = await hash.verify(user.password, password)  
  
    if (!isPasswordValid) {  
        return ctx.response.abort('Invalid credentials')  
    }  
    ctx.response.send('OK, you are in!')  
}
```

Auth - logout



- Do routes pridáme:

```
router.get('/logout', async ({ auth, response }) => {
  await auth.use('web').logout()
  response.send('logged out')
})
```

Database – Query builder



- Lucid Query Builder

```
import Database from '@ioc:Adonis/Lucid/Database'

// select
const users = await Database
    .from('users')
    .select('*')

// insert
await Database
    .table('users')
    .insert({ username: 'ed', email: 'ed@ed.ed' })
```

Database – Raw queries



- Použitím `rawQuery` môžeme napísať priamo SQL dopyt:

```
const user = await Database  
  .rawQuery('select * from users where id = ?', [1])
```

Database – Transakcie



```
const trx = await Database.transaction()

try {
    await trx
        .insertQuery()
        .table('users')
        .insert({ username: 'virk' })
    await trx.commit()
} catch (error) {
    await trx.rollback()
}
```

Database – Migrácie



- Vytvorenie migračného súboru:
 - node ace make:migration users
- Spustenie migrácie:
 - node ace migration:run

```
public async up() { // create or alter
    this.schema.createTable(this.tableName, (table) => {
        table.increments('id')
        table.timestamp('created_at', { useTz: true })
        table.timestamp('updated_at', { useTz: true })
    })
}

public async down() { // rollback pre up
    this.schema.dropTable(this.tableName)
}
```

Database – ORM



- Objektovo-relačný mapovač
- Model ~= záznam/riadok v tabuľke
- Ak chceme pristupovať k dátam tabuľky, musíme vytvoriť model
- Vzťahy:
 - hasOne, hasMany, inverzne: belongsTo
 - manyToMany

ORM - belongsTo



```
// model Message
export default class Message extends BaseModel {
  ...
  @belongsTo(() => User, {
    foreignKey: 'createdBy',
  })
  public author: BelongsTo<typeof User>

  @belongsTo(() => Channel, {
    foreignKey: 'channelId',
  })
  public channel: BelongsTo<typeof Channel>
}
```

ORM - hasMany



```
// model Channel

export default class Channel extends BaseModel {
  ...

  @hasMany(() => Message, {
    foreignKey: 'channelId',
  })

  public messages: HasMany<typeof Message>
}
```

ORM - manyToMany



```
// model User
export default class User extends BaseModel {
  ...
  @hasMany(() => Message, {
    foreignKey: 'createdBy',
  })
  public sentMessages: HasMany<typeof Message>

  @manyToMany(() => Channel, {
    pivotTable: 'user_channels',
    pivotForeignKey: 'user_id',
    pivotRelatedForeignKey: 'channel_id',
    pivotTimestamps: true,
  })
  public channels: ManyToMany<typeof Channel>
}
```

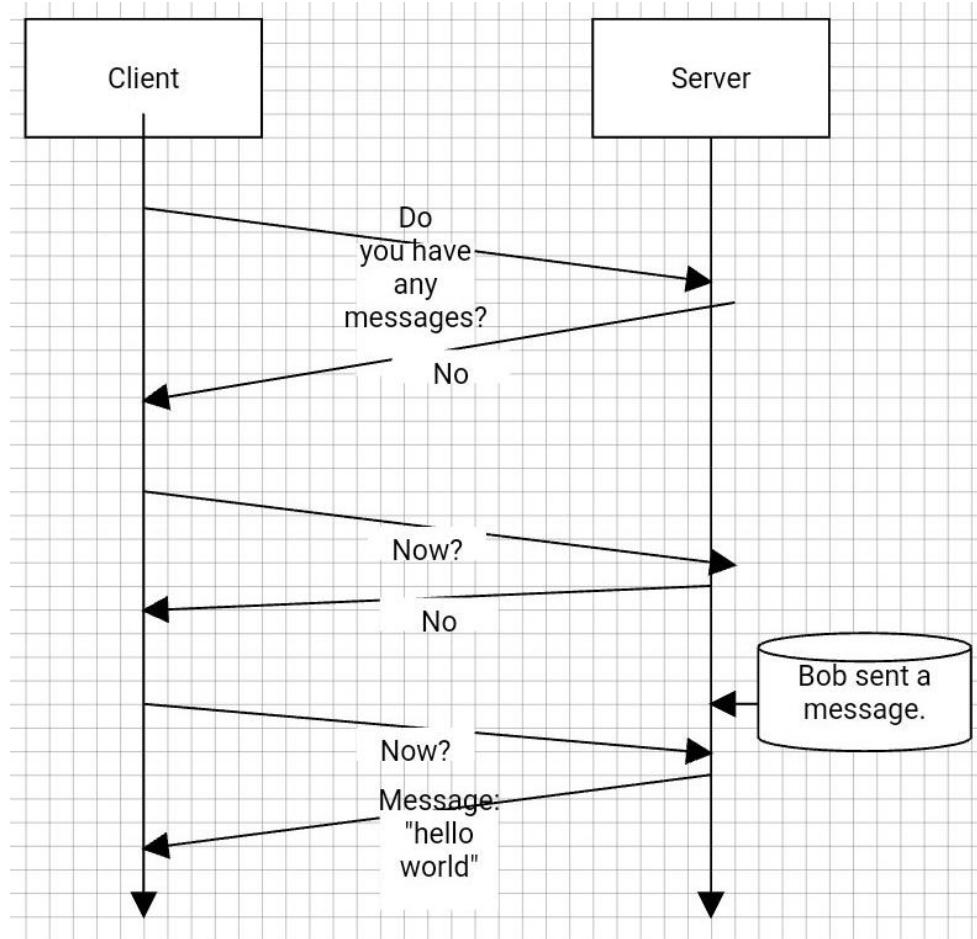
ORM - query



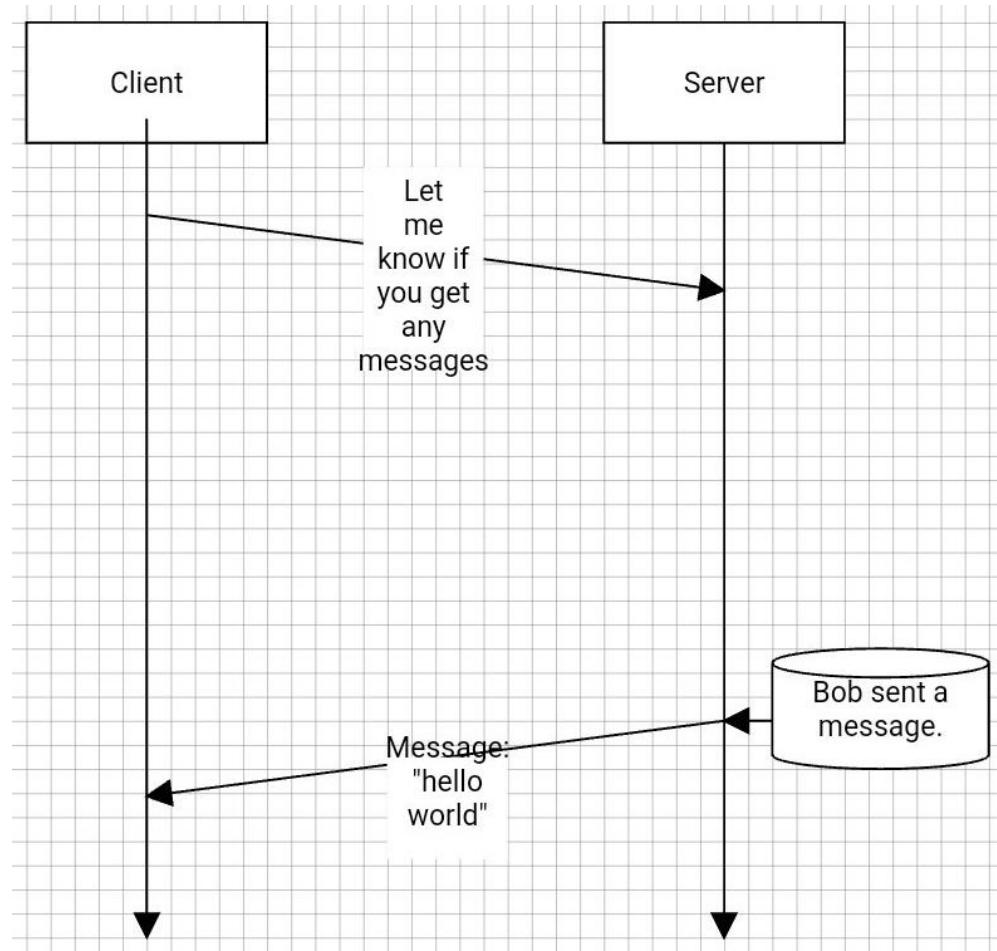
```
// ku kanalu vrati zoznam vsetkych sprav,  
// pricom kazda sprava bude mat priamo meno autora
```

```
const channel = await Channel.query()  
    .where('name', channelName)  
    .preload('messages', (messagesQuery) =>  
        messagesQuery.preload('author'))  
    .firstOrFail()
```

HTTP – bezstavový, req/resp



Websocket – full duplex



Websocket vs. AJAX

- AJAX (`XMLHttpRequest`), Fetch API
- socket – vytvorené spojenie so serverom pretrváva, (vytvorí sa kanál) – udržiava sa stav
 - AJAX/HTTP požiadavky sú bezstavové
 - server o klientovi nevie
(iba tak, že si cez cookie prenášajú session id)
- socket - server môže odoslať údaje klientovi (socketu) kedykoľvek
 - v prípade AJAXu - server môže odpovedať klientovi, iba prostredníctvom požiadavky