

Webtechnológia és webalkalmazás-fejlesztés

CORS, Dependency Injection, Email Sending

Magda Donát

Széchenyi István Egyetem, Győr

https://git.mdnd-it.cc/Donat/GKNB_MSTM071

2026. március 3.

Mi az a CORS?

Cross-Origin Resource Sharing

- Biztonsági mechanizmus, amely szabályozza a kereszt-eredet HTTP kéréseket
- A böngészők alapértelmezés szerint blokkolják a különböző eredetről (origin) érkező kéréseket
- Az **origin** = protokoll + domain + port
- CORS policy lehetővé teszi biztonságos erőforrás-megosztást különböző domainek között

Példák különböző originekre

- `https://example.com:443` \neq `http://example.com:80` (protokoll)
- `https://example.com` \neq `https://api.example.com` (subdomain)
- `https://example.com:3000` \neq `https://example.com:4000` (port)

Miért van szükség CORS-ra?

Same-Origin Policy (SOP)

- Böngészők biztonsági mechanizmusa
- Megakadályozza, hogy rosszindulatú scriptek más domainek adataihoz férjenek hozzá
- Alapértelmezés szerint csak azonos origin-ről tölthet le erőforrásokat

Probléma

Modern alkalmazásoknál gyakori:

- Frontend: `http://localhost:3000`
- Backend API: `http://localhost:5000`
- Különböző origin → CORS hiba!

CORS működése - Simple Request

Egyszerű kérések

Olyan kérések, amelyek nem váltanak ki preflight kérést:

- Metódus: GET, HEAD, vagy POST
- Content-Type: application/x-www-form-urlencoded, multipart/form-data, text/plain
- Csak standard headerek

Kliens oldali kérés

```
fetch('http://api.example.com/data')  
  .then(response => response.json())  
  .then(data => console.log(data));
```

CORS működése - Preflight Request

Preflight kérés

- Komplex kérések előtt a böngésző automatikusan küld egy OPTIONS kérést
- Ellenőrzi, hogy a szerver engedélyezi-e a tényleges kérést
- PUT, DELETE, PATCH metódusok vagy custom headerek esetén

1. Preflight (OPTIONS)

- Access-Control-Request-Method
- Access-Control-Request-Headers

2. Actual Request

- A tényleges HTTP kérés
- Csak akkor megy el, ha a preflight engedélyező választ adott

CORS Headers - Szerver válaszok

Legfontosabb CORS headerek

`Access-Control-Allow-Origin` Megadja, mely origin-ek férhetnek hozzá

`Access-Control-Allow-Methods` Engedélyezett HTTP metódusok

`Access-Control-Allow-Headers` Engedélyezett HTTP headerek

`Access-Control-Allow-Credentials` Cookie-k küldése engedélyezett-e

`Access-Control-Max-Age` Preflight válasz cache ideje (másodperc)

CORS engedélyezése Express.js-ben - Manuális (1/2)

Egyszerű megoldás - minden origin engedélyezése

```
app.use((req, res, next) => {  
  res.header('Access-Control-Allow-Origin', '*');  
  res.header('Access-Control-Allow-Methods',  
    'GET, POST, PUT, DELETE, OPTIONS');  
  res.header('Access-Control-Allow-Headers',  
    'Content-Type, Authorization');  
  
  next();  
});
```

CORS engedélyezése Express.js-ben - Manuális (2/2)

Preflight request kezelése

```
app.use((req, res, next) => {  
  // Preflight request kezelése  
  if (req.method === 'OPTIONS') {  
    return res.sendStatus(200);  
  }  
  next();  
});
```

Figyelem!

A * wildcard nem biztonságos production környezetben!

CORS engedélyezése Express.js-ben - CORS csomag

Telepítés

```
npm install cors
```

CORS csomag - Alapértelmezett használat

Minden origin engedélyezése

```
const express = require('express');
const cors = require('cors');
const app = express();

// Minden origin engedélyezése
app.use(cors());

app.get('/api/data', (req, res) => {
  res.json({ message: 'CORS-enabled!' });
});
```

CORS konfiguráció - Specifikus origin

Csak egy origin engedélyezése

```
const corsOptions = {  
  origin: 'https://example.com'  
};  
  
app.use(cors(corsOptions));
```

CORS konfiguráció - Több origin (1/2)

Engedélyezett origin-ek listája

```
const allowedOrigins = [  
  'https://example.com',  
  'https://app.example.com',  
  'http://localhost:3000',  
];
```

CORS konfiguráció - Több origin (2/3)

Dinamikus origin validáció

```
const corsOptions = {
  origin: (origin, callback) => {
    if (!origin || allowedOrigins.includes(origin)) {
      callback(null, true);
    } else {
      callback(new Error('Not allowed by CORS'));
    }
  }
};
```

CORS konfiguráció - Több origin (3/3)

CORS middleware beállítása

```
// Korábban definiált corsOptions használata  
app.use(cors(corsOptions));
```

CORS konfiguráció - Credentials

Mi az a credentials?

- Cookie-k, HTTP authentication, client-side SSL certificates
- Alapértelmezés szerint a böngésző nem küldi el ezeket cross-origin kéréseknél

Szerver oldal - Express

```
const corsOptions = {  
  origin: 'https://example.com',  
  credentials: true // Cookie-k engedélyezése  
};  
  
app.use(cors(corsOptions));
```

CORS Credentials - Kliens oldal

Kliens oldal - Fetch API

```
fetch('https://api.example.com/data', {  
  credentials: 'include' // Cookie-k küldése  
})  
  .then(response => response.json())  
  .then(data => console.log(data));
```

CORS konfiguráció - További opciók (1/2)

Részletes konfiguráció

```
const corsOptions = {  
  origin: 'https://example.com',  
  methods: ['GET', 'POST', 'PUT', 'DELETE'],  
  allowedHeaders: ['Content-Type', 'Authorization'],  
  exposedHeaders: ['X-Total-Count'],  
  credentials: true  
};
```

CORS konfiguráció - További opciók (2/2)

Cache és preflight beállítások

```
const corsOptions = {  
  // ...előző beállítások  
  maxAge: 3600, // Preflight cache 1 órára  
  preflightContinue: false,  
  optionsSuccessStatus: 204  
};  
  
app.use(cors(corsOptions));
```

CORS-specifikus route-okhoz (1/3)

Public endpoint CORS-szal

```
const express = require('express');
const cors = require('cors');
const app = express();

// Csak public API-hoz engedélyezve
app.get('/api/public', cors(), (req, res) => {
  res.json({ message: 'Public_data' });
});
```

CORS-specifikus route-okhoz (2/3)

Protected endpoint CORS nélkül

```
// Protected endpoint CORS nélkül
app.get('/api/private', (req, res) => {
  res.json({ message: 'Private_data' });
});
```

CORS-specifikus route-okhoz (3/3)

Specifikus CORS konfiguráció egy route-hoz

```
const corsOptionsRestricted = {
  origin: 'https://trusted.example.com'
};

app.post('/api/sensitive',
  cors(corsOptionsRestricted),
  (req, res) => {
    res.json({ message: 'Sensitive operation' });
  }
);
```

CORS hibakezelés (1/2)

Engedélyezett origin-ek

```
const allowedOrigins = [  
  'https://example.com',  
  'http://localhost:3000',  
];
```

CORS hibakezelés (2/3)

Dinamikus origin ellenőrzés hibakezeléssel

```
const corsOptions = {
  origin: (origin, callback) => {
    if (!origin) return callback(null, true);
    if (allowedOrigins.includes(origin)) {
      callback(null, true);
    } else {
      const msg = `Origin ${origin} not allowed`;
      callback(new Error(msg), false);
    }
  },
}
```

CORS hibakezelés (3/3)

CORS konfiguráció befejezése

```
// ...folytatás
credentials: true
};
app.use(cors(corsOptions));
```

Gyakori CORS hibák és megoldások (1/2)

1. "No 'Access-Control-Allow-Origin' header"

Ok: A szerver nem küldi vissza a CORS headereket

Megoldás: CORS middleware beállítása

2. "Credentials + wildcard origin"

Ok: `credentials: true` és `origin: '*'` együttes használata

Megoldás: Specifikus origin megadása `credentials` használatakor

Gyakori CORS hibák és megoldások (2/2)

3. "Preflight request failed"

Ok: OPTIONS kérés nem megfelelő válasza

Megoldás: OPTIONS metódus explicit kezelése, megfelelő headerek

4. "Custom headers blokkolva"

Ok: Custom header nincs az allowedHeaders listában

Megoldás: Header hozzáadása az allowedHeaders-hoz

Best Practices

Biztonsági ajánlások

- 1 **Soha ne használj** `origin: '*'` production környezetben, főleg credentials esetén
- 2 **Whitelist megközelítés:** csak konkrét, megbízható origin-ek engedélyezése
- 3 **Minimális jogosultságok:** csak a szükséges metódusok és headerek engedélyezése
- 4 **Credentials óvatos kezelése:** csak akkor engedélyezd, ha feltétlenül szükséges
- 5 **maxAge beállítás:** csökkenti a preflight kérések számát

Production konfiguráció

Ajánlott production konfiguráció

```
const corsOptions = {  
  origin: process.env.ALLOWED_ORIGINS?.split(',')  
    || [],  
  methods: ['GET', 'POST', 'PUT', 'DELETE'],  
  allowedHeaders: ['Content-Type', 'Authorization'],  
  credentials: true,  
  maxAge: 86400 // 24 óra  
};
```

CORS proxy fejlesztési környezetben (1/2)

Development server proxy

Frontend fejlesztési szerverekben (pl. Vite, Create React App) konfigurálható proxy

Vite proxy alapkonzfiguráció (vite.config.js)

```
export default {  
  server: {  
    proxy: {  
      '/api': {  
        target: 'http://localhost:5000',  
        changeOrigin: true  
      }  
    }  
  }  
};
```



CORS proxy fejlesztési környezetben (2/2)

Rewrite funkció hozzáadása

```
// Proxy konfigurációban:  
proxy: {  
  '/api': {  
    rewrite: (path) =>  
      path.replace(/^\/api/, '')  
  }  
}
```

Proxy működése

- Kliens `http://localhost:3000 → /api/users`
- Proxy továbbítja `http://localhost:5000 → /users`
- Nincs CORS probléma, mert a böngésző szempontjából azonos origin
- Csak development használatra, production esetén backend CORS konfiguráció szükséges

Összefoglalás

CORS lényege

- Biztonsági mechanizmus a cross-origin kérések szabályozására
- Same-Origin Policy kiegészítése, nem helyettesítése
- Preflight requests komplex kérések előtt
- Explicit server-oldali engedélyezés szükséges

Implementáció

- Express.js-ben: cors middleware csomag
- Konfiguráció: origin, methods, headers, credentials
- Fejlesztés: proxy használata, production: explicit whitelist
- Biztonsági szempontok elsődlegesek!

Mi az a Dependency Injection (DI)?

Definíció

A **Dependency Injection** egy tervezési minta, ahol az objektumok nem maguk hozzák létre a függőségeiket, hanem kívülről kapják meg azokat.

DI előnyei és hátrányai

Nélküle (rossz)

- Szoros kapcsolás (tight coupling)
- Nehéz tesztelni
- Nehéz karbantartani
- Függőségek rejtettek

Vele (jó)

- Laza kapcsolás (loose coupling)
- Könnyű tesztelni
- Könnyen karbantartható
- Függőségek explicit módon láthatók

Probléma DI nélkül (1/3)

Szoros kapcsolat példa - Osztály

```
class UserService {  
    constructor() {  
        // A UserService közvetlenül létrehozza  
        // a függőségeit  
        this.database = new MySQLDatabase();  
        this.emailService = new EmailService();  
        this.logger = new Logger();  
    }  
}
```

Probléma DI nélkül (2/3)

Használat

```
const userService = new UserService();

async createUser(userData) {
  this.logger.log('Creating user...');
  const user = await this.database.save(userData);
  await this.emailService.sendWelcomeEmail(
    user.email
  );
  return user;
}
```

Problémák

- Nem lehet könnyen PostgreSQL-re váltani
- Teszteléskor valódi email-t küldene
- Nem lehet mock objektumokat használni

Megoldás Dependency Injection-nel (1/3)

Laza kapcsolat DI-val - Konstruktor

```
class UserService {  
    constructor(database, emailService, logger) {  
        // Függőségek kívülről érkeznek  
        this.database = database;  
        this.emailService = emailService;  
        this.logger = logger;  
    }  
}
```

Megoldás Dependency Injection-nel (2/3)

createUser metódus

```
async createUser(userData) {  
  this.logger.log('Creating user...');  
  const user = await this.database.save(userData);  
  await this.emailService.sendWelcomeEmail(  
    user.email);  
  return user;  
}
```

Megoldás Dependency Injection-nel (3/3)

Használat - függőségek kívülről

```
const database = new MySQLDatabase();  
const emailService = new EmailService();  
const logger = new Logger();  
  
const userService = new UserService(  
    database ,  
    emailService ,  
    logger  
);
```

DI előnyei

Miért jó a Dependency Injection?

1 Tesztelhetőség

- Mock és stub objektumok használata
- Izolált unit tesztek

2 Rugalmasság

- Könnyen cserélhető implementációk
- Különböző konfigurációk más környezetekben

3 Karbantarthatóság

- Egyértelmű függőségek
- Single Responsibility Principle

4 Újrafelhasználhatóság

- Komponensek függetlenek
- Könnyű más projektekben használni

DI típusok - Constructor Injection (1/2)

Constructor Injection (Ajánlott)

Függőségek átadása a konstruktorban

Osztály konstruktor

```
class OrderService {
    constructor(paymentGateway, inventoryService) {
        this.paymentGateway = paymentGateway;
        this.inventoryService = inventoryService;
    }
}
```

DI típusok - Constructor Injection (2/2)

processOrder metódus

```
async processOrder(order) {  
    await this.inventoryService.reserveItems(  
        order.items);  
    await this.paymentGateway.charge(order.total);  
    return { success: true, orderId: order.id };  
}
```

Constructor Injection - Használat

Dependency injection

```
const paymentGateway = new StripeGateway();  
const inventoryService = new InventoryService();  
  
const orderService = new OrderService(  
    paymentGateway ,  
    inventoryService  
);
```

DI típusok - Setter Injection (1/3)

Setter Injection

Függőségek beállítása setter metódusokon keresztül

Setter metódusok

```
class ReportService {  
    setDatabase(database) {  
        this.database = database;  
    }  
  
    setEmailService(emailService) {  
        this.emailService = emailService;  
    }  
}
```

DI típusok - Setter Injection (2/3)

generateAndSendReport metódus

```
async generateAndSendReport () {  
    const data = await this.database.fetchData();  
    const report = this.createReport(data);  
    await this.emailService.send(report);  
}
```

DI típusok - Setter Injection (3/3)

Használat

```
const reportService = new ReportService();  
reportService.setDatabase(new MySQLDatabase());  
reportService.setEmailService(new EmailService());
```

Setter Injection - Hátrányok

Hátrány

Opcionális függőségek → könnyebb hibázni

DI típusok - Property Injection (1/2)

Property Injection

Közvetlen property beállítás (ritkábban használt)

Osztály definíció

```
class NotificationService {  
    async sendNotification(message) {  
        if (this.emailSender) {  
            await this.emailSender.send(message);  
        }  
        if (this.smsSender) {  
            await this.smsSender.send(message);  
        }  
    }  
}
```

DI típusok - Property Injection (2/2)

Használat

```
const notificationService =  
    new NotificationService();  
  
notificationService.emailSender =  
    new EmailSender();  
notificationService.smsSender =  
    new SMSSender();
```

Property Injection - Figyelmeztetés

Figyelem

Kevésbé explicit, nehezebb követni a függőségeket

Dependency Injection TypeScript-tel (1/3)

Interfészek definiálása

```
interface IDatabase {  
  save(data: any): Promise<any>;  
  find(id: string): Promise<any>;  
}  
  
interface ILogger {  
  log(message: string): void;  
  error(message: string): void;  
}
```

Dependency Injection TypeScript-tel (2/4)

Service osztály konstruktor

```
class UserService {  
  constructor(  
    private database: IDatabase,  
    private logger: ILogger  
  ) {}  
}
```

Dependency Injection TypeScript-tel (3/4)

createUser metódus

```
async createUser(userData: any) {  
  this.logger.log('Creating user');  
  return await this.database.save(userData);  
}
```

Dependency Injection TypeScript-tel (4/4)

Implementációk és használat

```
const db: IDatabase = new MongoDBDatabase();  
const logger: ILogger = new ConsoleLogger();  
const service = new UserService(db, logger);
```

DI gyakorlatban - Tesztelés (1/4)

Eredeti service konstruktor

```
class AuthService {  
    constructor(database, jwtService) {  
        this.database = database;  
        this.jwtService = jwtService;  
    }  
}
```

DI gyakorlatban - Tesztelés (2/4)

login metódus

```
async login(email, password) {  
  const user = await this.database.findByEmail(  
    email);  
  if (user && user.password === password) {  
    return this.jwtService.generateToken(user);  
  }  
  throw new Error('Invalid credentials');  
}
```

DI gyakorlatban - Tesztelés (3/4)

Unit teszt - Mock objektumok (1/2)

```
test('login with valid credentials', async () => {
  const mockDatabase = {
    findByEmail: jest.fn().mockResolvedValue({
      email: 'test@test.com',
      password: 'pass123'
    })
  };
});
```

DI gyakorlatban - Tesztelés (4/4)

Unit teszt - Mock objektumok (2/2)

```
const mockJwtService = {  
  generateToken: jest.fn()  
    .mockReturnValue('token123')  
};
```

DI gyakorlatban - Tesztelés (5/5)

Unit teszt - Érvényesítés

```
const authService = new AuthService(  
  mockDatabase, mockJwtService);  
const token = await authService.login(  
  'test@test.com', 'pass123');  
  
expect(token).toBe('token123');  
expect(mockDatabase.findByEmail)  
  .toHaveBeenCalledWith('test@test.com');  
});
```

DI gyakorlatban - Környezeti különbségek (1/3)

Development környezet

```
if (process.env.NODE_ENV === 'development') {  
  const database = new MockDatabase();  
  const emailService = new ConsoleEmailService();  
  const logger = new VerboseLogger();  
  
  app.locals.userService = new UserService(  
    database, emailService, logger  
  );  
}
```

DI gyakorlatban - Környezeti különbségek (2/3)

Production környezet

```
if (process.env.NODE_ENV === 'production') {  
  const database = new PostgreSQLDatabase();  
  const emailService = new SendGridEmailService();  
  const logger = new CloudLogger();  
  
  app.locals.userService = new UserService(  
    database, emailService, logger  
  );  
}
```

DI gyakorlatban - Környezeti különbségek (3/3)

Használat route-okban

```
app.post('/users', async (req, res) => {  
  const user = await app.locals.userService  
    .createUser(req.body);  
  res.json(user);  
});
```

Manuális DI - Factory Pattern (1/4)

Factory függvény - Alapok (1/2)

```
function createServices(config) {  
  // Alapvető szolgáltatások  
  const logger = new Logger(config.logLevel);  
  const database = new Database(config.dbUrl);
```

Manuális DI - Factory Pattern (2/4)

Factory függvény - Alapok (2/2)

```
// Összetett szolgáltatások
const userRepository =
  new UserRepository(database);
const emailService =
  new EmailService(config.emailConfig);
```

Manuális DI - Factory Pattern (3/5)

Service-ek létrehozása (1/2)

```
const userService = new UserService(  
    userRepository , emailService , logger  
);  
  
const authService = new AuthService(  
    userRepository , config.jwtSecret , logger  
);
```

Manuális DI - Factory Pattern (4/5)

Service-ek létrehozása (2/2)

```
return {  
    userService, authService,  
    logger, database  
};  
}
```

Manuális DI - Factory Pattern (5/5)

Használat

```
const config = loadConfig();  
const services = createServices(config);  
  
export default services;
```

Service Locator Pattern (1/4)

Service container konstruktor

```
class ServiceContainer {  
    constructor() {  
        this.services = new Map();  
    }  
}
```

Service Locator Pattern (2/4)

register és get metódusok

```
register(name, instance) {  
    this.services.set(name, instance);  
}  
  
get(name) {  
    if (!this.services.has(name)) {  
        throw new Error('Service ${name} not found');  
    }  
    return this.services.get(name);  
}
```

Service Locator Pattern (3/4)

Service-ek regisztrálása (1/2)

```
const container = new ServiceContainer();  
  
container.register('database',  
    new MySQLDatabase());  
container.register('logger', new Logger());
```

Service Locator Pattern (4/4)

Service-ek regisztrálása (2/2)

```
container.register('userService',  
    new UserService(  
        container.get('database'),  
        container.get('logger')  
    )  
);
```

Service Locator Pattern (5/5)

Használat

```
const userService = container.get('userService');
```

DI az Express middleware-ekben (1/4)

Service-ek létrehozása

```
const database = new Database();  
const userService = new UserService(database);  
const authService = new AuthService(database);
```

DI az Express middleware-ekben (2/6)

Middleware factory - Setup

```
function createAuthMiddleware(authService) {  
  return async (req, res, next) => {  
    const token = req.headers.authorization  
      ?.split(' ')[1];  
    if (!token) {  
      return res.status(401)  
        .json({ error: 'No token' });  
    }  
  }  
}
```

DI az Express middleware-ekben (3/6)

Middleware factory - Verifikáció

```
try {  
  req.user = await authService.verifyToken(  
    token);  
  next();  
} catch (error) {  
  res.status(401)  
    .json({ error: 'Invalid_token' });  
}  
};  
}
```

DI az Express middleware-ekben (4/6)

Route handler factory

```
function createUserController(userService) {  
  return {  
    async createUser(req, res) {  
      const user = await userService.createUser(  
        req.body);  
      res.json(user);  
    }  
  };  
}
```

DI az Express middleware-ekben (5/6)

Setup (1/2)

```
const authMiddleware =  
  createAuthMiddleware(authService);  
const userController =  
  createUserController(userService);
```

DI az Express middleware-ekben (6/6)

Setup (2/2)

```
app.post('/users',  
  authMiddleware,  
  userController.createUser  
);
```

SOLID elvek és DI

DI kapcsolata a SOLID elvekkel

S - Single Responsibility Osztályok egy dolgot csinálnak, nem hozzák létre saját függőségeiket

O - Open/Closed Új implementációk hozzáadhatók a meglévő kód módosítása nélkül

L - Liskov Substitution Interfészek lehetővé teszik a helyettesítést

I - Interface Segregation Kis, specifikus interfészek használata

D - Dependency Inversion **Függőség absztrakciókra, nem konkrét implementációkra**

Dependency Inversion Principle

- High-level modulok nem függnék low-level moduloktól
- Mindkettő absztrakcióktól (interfészeketől) függ
- DI ezt a célt szolgálja

Best Practices

Ajánlások

1 Constructor Injection előnyben részesítése

- Kötelező függőségek explicit módon láthatók
- Immutábilis objektumok

2 Interfészekre programozás

- TypeScript interfészek használata
- Laza kapcsolat

3 Ne használj Service Locator az üzleti logikában

- Rejtett függőségek
- Csak konfigurációs szinten

4 Kerüld a túl sok függőséget

- Ha sok függőség van, valószínűleg túl sok felelőssége van az osztálynak
- Refaktorálás szükséges

Antipattern - Service Locator visszaélés (1/3)

Rossz gyakorlat - Konstruktor

```
class UserService {  
    constructor(container) {  
        // Teljes container injektálása  
        this.container = container;  
    }  
}
```

Antipattern - Service Locator visszaélés (2/3)

Rossz gyakorlat - createUser metódus

```
async createUser(userData) {  
  // Függőségek rejtettek!  
  const db = this.container.get('database');  
  const email = this.container.get('emailService');  
  const logger = this.container.get('logger');  
  logger.log('Creating user');  
  const user = await db.save(userData);  
  await email.sendWelcome(user.email);  
  return user;  
}
```

Antipattern - Service Locator visszaélés (3/3)

Jó gyakorlat

```
class UserService {  
    constructor(database, emailService, logger) {  
        this.database = database;  
        this.emailService = emailService;  
        this.logger = logger;  
    }  
}
```

Összefoglalás (1/2)

Dependency Injection lényege

- Függőségek kívülről történő átadása
- Laza kapcsolat objektumok között
- Tesztelhetőség növelése
- SOLID elvek támogatása

Implementációs módszerek

- Constructor Injection (preferált)
- Setter Injection (opcionális függőségekhez)
- Manuális DI factory pattern-nel
- Service Container (konfigurációs szinten)

Összefoglalás (2/2)

Előnyök

- Könnyű tesztelés mock objektumokkal
- Rugalmas, cserélhető implementációk
- Tisztább, karbantarthatóbb kód
- Explicit függőségek

Mi az a Dependency Injection Container?

DI Container (DIC) / IoC Container

- Automatizálja a dependency injection-t
- Központi hely a függőségek kezelésére
- Automatikus dependency resolution
- Lifecycle management (singleton, transient, scoped)
- Inversion of Control (IoC) megvalósítása

Miért használjunk DI Container-t?

- **Manuális DI problémái:** sok függőség esetén bonyolult az instanciálás
- **Automatizálás:** a container automatikusan feloldja a függőségeket
- **Konfiguráció:** egy helyen konfiguráljuk az összes service-t
- **Tesztelhetőség:** könnyű mock service-eket injektálni



Probléma manuális DI-val (1/3)

Dependency hell - Alapvető szolgáltatások

```
const database = new Database(config.db);  
const logger = new Logger(config.log);  
const cache = new RedisCache(config.redis);  
  
const userRepository =  
    new UserRepository(database, logger);  
const productRepository =  
    new ProductRepository(database, logger);  
const orderRepository =  
    new OrderRepository(database, logger);
```

Probléma manuális DI-val (2/3)

Dependency hell - Service-ek

```
const emailService =  
  new EmailService(config.email , logger);  
const paymentService =  
  new PaymentService(config.payment , logger);  
  
const userService = new UserService(  
  userRepository , emailService , logger);  
const productService = new ProductService(  
  productRepository , cache , logger);
```

Probléma manuális DI-val (3/3)

Dependency hell - Komplex függőségek

```
const orderService = new OrderService(  
  orderRepository ,  
  productService ,  
  userService ,  
  paymentService ,  
  emailService ,  
  logger  
);  
// És még tovább...
```

Problémák

Bonyolult, hibára hajlamos, nehéz karbantartani, függőségi sorrend fontos

DI Container életciklusok

Service Lifetimes

Transient Minden kérésnél új instance

- Lightweight, stateless service-ek
- Példa: validators, utilities

Singleton Egy instance az egész alkalmazásra

- Shared state, resource-intenzív
- Példa: database connection pool, logger, config

Scoped Egy instance per request/scope

- HTTP request cycle alatt ugyanaz
- Példa: database transaction, user context

Népszerű DI Container-ek Node.js-hez

JavaScript/TypeScript DI library-k

- InversifyJS** TypeScript-friendly, decorator-alapú
- Awilix** Lightweight, flexible, Node.js-specifikus
- TSyringe** Microsoft TypeScript DI container
- TypeDI** Decorator-alapú, TypeScript support
- BottleJS** Egyszerű DI container pure JS-hez

Választás

TypeScript projekt: **InversifyJS** vagy **TSyringe**

JavaScript projekt: **Awilix**

Egyszerű projektek: manuális DI vagy **BottleJS**

Awilix - Lightweight DI Container

Telepítés

```
npm install awilix
```

Awilix - Alapvető használat (1/2)

Container létrehozása és singleton-ok

```
const { createContainer, asClass,  
  asFunction, asValue } = require('awilix');  
  
const container = createContainer();  
  
container.register({  
  // Singleton - egy instance  
  database: asClass(Database).singleton(),  
  logger: asClass(Logger).singleton()  
});
```

Awilix - Alapvető használat (2/3)

Transient és Scoped

```
container.register({  
    // Transient - minden resolve-nál új instance  
    userService: asClass(UserService).transient(),  
  
    // Scoped - scope-on belül ugyanaz  
    orderService: asClass(OrderService).scoped()  
});
```

Awilix - Alapvető használat (3/3)

Value injection

```
container.register({  
  // Value injection - konstans értékek  
  config: asValue({  
    dbUrl: process.env.DATABASE_URL,  
    port: process.env.PORT || 3000  
  })  
});
```

Awilix - Service-ek használata (1/4)

Osztály definíciók - Repository rész 1

```
class UserRepository {  
    constructor({ database, logger }) {  
        this.database = database;  
        this.logger = logger;  
    }  
}
```

Awilix - Service-ek használata (2/4)

Osztály definíciók - Repository rész 2

```
async findById(id) {  
  this.logger.log('Finding user ${id}');  
  return await this.database.query(  
    'SELECT * FROM users WHERE id = ?', [id]  
  );  
}
```

Awilix - Service-ek használata (3/5)

Osztály definíciók - Service rész 1

```
class UserService {  
    constructor({ userRepository , emailService ,  
        logger }) {  
        this.userRepository = userRepository ;  
        this.emailService = emailService ;  
        this.logger = logger ;  
    }  
}
```

Awilix - Service-ek használata (4/5)

Osztály definíciók - Service rész 2

```
async createUser(userData) {  
  this.logger.log('Creating user');  
  const user = await this.userRepository  
    .create(userData);  
  await this.emailService.sendWelcomeEmail(  
    user);  
  return user;  
}
```

Awilix - Service-ek használata (5/5)

Dependency resolution

```
// Awilix automatikusan feloldja  
// a függőségeket  
const userService =  
  container.resolve('userService');  
// userRepository, emailService, logger  
// automatikusan injektálva
```

Awilix - Auto-loading (1/3)

Setup

```
const { createContainer , asClass } =  
  require('awilix');  
  
const container = createContainer();
```

Awilix - Auto-loading (2/4)

Automatikus service regisztráció

```
container.loadModules([  
  'services/**/*.js',  
  'repositories/**/*.js',  
  'controllers/**/*.js'  
], {  
  formatName: 'camelCase',  
})
```

Awilix - Auto-loading (3/4)

Resolver options

```
resolverOptions: {  
  lifetime: 'SINGLETON',  
  register: asClass  
}  
});
```

Awilix - Auto-loading (3/4)

Egyedi konfiguráció

```
container.loadModules([
  [ 'services/**/*.js',
    { register: asClass, lifetime: 'SINGLETON' } ],
  [ 'repositories/**/*.js',
    { register: asClass, lifetime: 'SCOPED' } ],
  [ 'utils/**/*.js',
    { register: asFunction, lifetime: 'SINGLETON' } ]
]);
```

Awilix - Auto-loading (4/4)

Fájl struktúra

```
services/userService.js → userService  
repositories/userRepository.js → userRepository
```

Awilix Express integráció (1/2)

Setup és container

```
const express = require('express');
const { createContainer, asClass } =
  require('awilix');
const { loadControllers, scopePerRequest } =
  require('awilix-express');

const app = express();
const container = createContainer();
```

Awilix Express integráció (2/2)

Service regisztráció

```
container.register({  
  userService: asClass(UserService).singleton(),  
  orderService: asClass(OrderService)  
    .singleton()  
});
```

Awilix Express - Middleware

Scope és controller betöltés

```
// Scope per request middleware
app.use(scopePerRequest(container));

// Controller-ek automatikus betöltése
app.use(loadControllers('controllers/*.js',
  { cwd: __dirname }));

app.listen(3000);
```

Awilix Controller példa (1/4)

controllers/userController.js - Import

```
const { createController } =  
  require('awilix-express');
```

Awilix Controller példa (2/4)

Osztály definíció

```
class UserController {  
    constructor({ userService , logger }) {  
        // Awilix automatikusan injektálja  
        this.userService = userService;  
        this.logger = logger;  
    }  
}
```

Awilix Controller példa (3/7)

GetUser metódus - try block

```
async getUser(req, res) {  
  try {  
    const user = await this.userService  
      .getUserById(req.params.id);  
    res.json(user);  
  } catch (error) {  
    this.logger.error(error);  
  }  
}
```

Awilix Controller példa (4/7)

GetUser metódus - error handling

```
res.status(500).json({  
    error: 'Internal_server_error' });  
}  
}
```

Awilix Controller példa (5/7)

CreateUser metódus - rész 1

```
async createUser(req, res) {  
  try {  
    const user = await this.userService  
      .createUser(req.body);  
    res.status(201).json(user);  
  } catch (error) {  
    this.logger.error(error);  
  }  
}
```

Awilix Controller példa (6/7)

CreateUser metódus - rész 2

```
        res.status(500).json({  
            error: 'Internal_server_error' });  
    }  
}
```

Awilix Controller példa (7/7)

Route mapping

```
module.exports = createController(UserController)
  .prefix('/users')
  .get('/:id', 'getUser')
  .post('/', 'createUser');
```

InversifyJS - TypeScript DI Container

Telepítés

```
npm install inversify reflect-metadata
```

tsconfig.json beállítások

```
{
  "compilerOptions": {
    "experimentalDecorators": true,
    "emitDecoratorMetadata": true
  }
}
```

InversifyJS - Alapvető setup

Setup

```
import 'reflect-metadata';  
import { Container } from 'inversify';  
  
const container = new Container();  
  
// Az app belépési pontján  
// import 'reflect-metadata' kell lennie legelső sorban
```

InversifyJS - Decorator-alapú DI (1/4)

Symbol-ok definíciója

```
import { injectable, inject } from 'inversify';
import 'reflect-metadata';

const TYPES = {
  Database: Symbol.for('Database'),
  Logger: Symbol.for('Logger'),
  UserRepository: Symbol.for('UserRepository'),
  UserService: Symbol.for('UserService')
};
```

InversifyJS - Decorator-alapú DI (2/4)

Database osztály

```
@injectable()  
class Database {  
    query(sql: string) { /* ... */ }  
}
```

InversifyJS - Decorator-alapú DI (2/3)

Logger osztály

```
@injectable()
class Logger {
  log(message: string) {
    console.log(message);
  }
}
```

InversifyJS - Decorator-alapú DI (3/4)

UserRepository constructor

```
@injectable()  
class UserRepository {  
  constructor(  
    @inject(TYPES.Database)  
    private database: Database,  
    @inject(TYPES.Logger)  
    private logger: Logger  
  ) {}  
}
```

InversifyJS - Decorator-alapú DI (4/4)

UserRepository metódusok

```
async findById(id: string) {  
  this.logger.log('Finding user ${id}');  
  return await this.database.query(  
    'SELECT * FROM users WHERE id = ${id}');  
}
```

InversifyJS - Container konfiguráció (1/3)

Setup

```
import { Container } from 'inversify';  
  
const container = new Container();
```

InversifyJS - Container konfiguráció (2/3)

Singleton és Transient binding

```
// Singleton binding
container.bind<Database>(TYPES.Database)
  .to(Database).inSingletonScope();
container.bind<Logger>(TYPES.Logger)
  .to(Logger).inSingletonScope();

// Transient binding (default)
container.bind<UserRepository>(
  TYPES.UserRepository).to(UserRepository);
```

InversifyJS - Container konfiguráció (3/4)

Scoped és Constant binding

```
// Scoped binding (request scope)
container.bind<UserService>(TYPES.UserService)
    .to(UserService).inRequestScope();

// Constant value
container.bind<Config>(TYPES.Config)
    .toConstantValue({
        dbUrl: process.env.DATABASE_URL,
        port: 3000
    });
```

InversifyJS - Container konfiguráció (4/4)

Factory binding

```
// Factory binding
container.bind<EmailService>(TYPES.EmailService)
  .toFactory((context) => {
    return () => new EmailService(
      context.container.get(TYPES.Config));
  });

export { container, TYPES };
```

InversifyJS - Service használat (1/3)

Service lekérése

```
import { container, TYPES }  
  from './container';  
  
// Service lekérése  
const userService =  
  container.get<UserService>(TYPES.UserService);  
await userService.createUser({  
  name: 'John', email: 'john@example.com'  
});
```

InversifyJS - Service használat (2/3)

Express route-okban

```
app.get('/users/:id', async (req, res) => {  
  const userService =  
    container.get<UserService>(TYPES.UserService);  
  const user = await userService.getUserById(  
    req.params.id);  
  res.json(user);  
});
```

InversifyJS - Service használat (3/4)

Middleware factory

```
function injectService(type: symbol) {  
  return (req, res, next) => {  
    req.service = container.get(type);  
    next();  
  };  
}
```

InversifyJS - Service használat (4/4)

Middleware használat

```
app.get('/users/:id',  
  injectService(TYPES.UserService),  
  async (req, res) => {  
    const user = await req.service.getUserById(  
      req.params.id);  
    res.json(user);  
  })
```

InversifyJS - Interface binding (1/3)

Interface definíció

```
interface ILogger {  
    log(message: string): void;  
    error(message: string): void;  
}
```

InversifyJS - Interface binding (2/4)

ConsoleLogger implementáció

```
@injectable()
class ConsoleLogger implements ILogger {
  log(message: string) {
    console.log(message);
  }
  error(message: string) {
    console.error(message);
  }
}
```

InversifyJS - Interface binding (3/4)

FileLogger implementáció

```
@injectable()
class FileLogger implements ILogger {
  log(message: string) {
    /* write to file */
  }
  error(message: string) {
    /* write to file */
  }
}
```

InversifyJS - Interface binding (4/5)

Container konfiguráció

```
const TYPES = { ILogger: Symbol.for('ILogger') };

// Development környezet
if (process.env.NODE_ENV === 'development') {
  container.bind<ILogger>(TYPES.ILogger)
    .to(ConsoleLogger).inSingletonScope();
} else {
  container.bind<ILogger>(TYPES.ILogger)
    .to(FileLogger).inSingletonScope();
}
```

InversifyJS - Interface binding (5/5)

Használat

```
@injectable()  
class UserService {  
    constructor(@inject(TYPES.ILogger)  
        private logger: ILogger) {}  
}
```

Container scope és lifecycle (1/2)

Container setup

```
const { createContainer , asClass } =  
  require('awilix');  
const { scopePerRequest } =  
  require('awilix-express');  
  
const container = createContainer();
```

Container scope és lifecycle (2/2)

Lifetime regisztráció

```
container.register({  
    // Singleton - egy instance az egész app-ra  
    database: asClass(Database).singleton(),  
    // Scoped - egy instance per HTTP request  
    userContext: asClass(UserContext).scoped(),  
    // Transient - mindig új instance  
    validator: asClass(Validator).transient()  
});
```

Request scope - Használat (1/2)

Middleware setup

```
app.use(scopePerRequest(container));

// Minden request új scope-ot kap
app.get('/users', (req, res) => {
  // req.container - scoped container
  const userContext =
    req.container.resolve('userContext');
```

Request scope - Használat (2/2)

Scope életciklus

```
// Ugyanaz a userContext instance  
// a request során  
});
```

Testing DI Container-rel (1/5)

Import és describe

```
const { createContainer, asClass, asValue } =  
  require('awilix');  
  
describe('UserService', () => {  
  let container;  
  let userService;
```

Testing DI Container-rel (2/5)

BeforeEach setup

```
beforeEach(() => {  
    container = createContainer();  
});
```

Testing DI Container-rel (3/5)

Mock repository

```
const mockRepository = {  
  findById: jest.fn().mockResolvedValue(  
    { id: 1, name: 'John' }  
  ),  
  create: jest.fn().mockResolvedValue(  
    { id: 2, name: 'Jane' }  
  )  
};
```

Testing DI Container-rel (4/5)

Mock email és logger

```
const mockEmailService = {  
  sendWelcomeEmail: jest.fn()  
    .mockResolvedValue(true)  
};  
const mockLogger = {  
  log: jest.fn(), error: jest.fn()  
};
```

Testing DI Container-rel (5/6)

Container regisztráció

```
container.register({  
    userRepository: asValue(mockRepository),  
    emailService: asValue(mockEmailService),  
    logger: asValue(mockLogger),  
    userService: asClass(UserService)  
});  
userService = container.resolve('userService');  
});
```

Testing DI Container-rel (6/6)

Teszt

```
test('createUser_sends_welcome_email',  
  async () => {  
    await userService.createUser(  
      { name: 'Jane', email: 'jane@ex.com' });  
    expect(container.resolve('emailService')  
      .sendWelcomeEmail).toHaveBeenCalled();  
  });  
});
```

DI Container projekt struktúra (1/3)

Config és services

```
src /  
    config/  
        database.js  
        email.js  
    services/  
        userService.js  
        emailService.js  
        authService.js
```

DI Container projekt struktúra (2/3)

Repositories és controllers

```
repositories /  
    userRepository.js  
    orderRepository.js  
controllers /  
    userController.js  
    orderController.js  
container.js  
app.js
```

DI Container projekt struktúra (3/4)

container.js - Setup

```
const { createContainer, asClass, asValue } =  
  require('awilix');  
const config = require('./config');  
  
function setupContainer() {  
  const container = createContainer();
```

DI Container projekt struktúra (4/5)

container.js - LoadModules

```
container.loadModules([
  'services/**/*.js',
  'repositories/**/*.js'
], {
  formatName: 'camelCase',
  resolverOptions: {
    lifetime: 'SINGLETON',
    register: asClass
  }
});
```

DI Container projekt struktúra (5/5)

container.js - Export

```
    container.register({ config: asValue(config) });  
    return container;  
}  
module.exports = setupContainer;
```

Advanced: Multi-tenancy DI (1/3)

Setup

```
const { createContainer, asClass } =  
  require('awilix');  
  
function tenantMiddleware(req, res, next) {  
  const tenantId = req.headers['x-tenant-id'];  
  
  // Tenant-specifikus scope  
  req.tenantScope = req.container.createScope();  
}
```

Advanced: Multi-tenancy DI (2/3)

Tenant regisztráció

```
req.tenantScope.register({  
    tenantId: asValue(tenantId),  
    tenantDatabase: asClass(TenantDatabase)  
        .scoped(),  
    tenantConfig: asClass(TenantConfig).scoped()  
});  
next();  
}
```

Advanced: Multi-tenancy DI (3/4)

Middleware setup

```
app.use(scopePerRequest(container));  
app.use(tenantMiddleware);  
  
app.get('/data', async (req, res) => {  
  // Tenant-specifikus database  
  const db = req.tenantScope.resolve(  
    'tenantDatabase');  
});
```

Advanced: Multi-tenancy DI (4/4)

Query használat

```
const data = await db.query(  
  'SELECT * FROM tenant_data' );  
res.json(data);  
});
```

DI Container Best Practices

Ajánlások

- 1 **Egy container per alkalmazás** - singleton pattern
- 2 **Centralizált konfiguráció** - egy helyen regisztráld a service-eket
- 3 **Helyes lifetime választás**
 - Singleton: stateless, expensive resources
 - Scoped: request-specific state
 - Transient: lightweight, stateless
- 4 **Ne add tovább a container-t** - ne injektáld service-ekbe (service locator antipattern)
- 5 **Explicit függőségek** - konstruktor injection
- 6 **Interface-alapú programozás** - TypeScript interface-ek
- 7 **Testing** - mock service-ek könnyű injektálása

Antipattern - Service Locator (1/3)

Rossz: Container injection - Constructor

```
class UserService {  
    constructor({ container }) {  
        this.container = container; // NE!  
    }  
}
```

Antipattern - Service Locator (2/3)

Rossz: Rejtett függőségek

```
async createUser(userData) {  
  // Rejtett függőségek  
  const repo = this.container.resolve(  
    'userRepository');  
  const email = this.container.resolve(  
    'emailService');  
  // ...  
}
```

Antipattern - Service Locator (3/4)

Jó: Explicit constructor

```
class UserService {  
    constructor({ userRepository , emailService ,  
        logger }) {  
        this.userRepository = userRepository ;  
        this.emailService = emailService ;  
        this.logger = logger ;  
    }  
}
```

Antipattern - Service Locator (4/4)

Jó: Explicit függőségek

```
async createUser(userData) {  
  // Explicit függőségek  
  const user = await this.userRepository  
    .create(userData);  
  await this.emailService.sendWelcome(user);  
  return user;  
}
```

Mikor használj DI Container-t?

Használd, ha:

- Közepes vagy nagy projekt sok függőséggel
- TypeScript projektek (InversifyJS, TSyringe)
- Komplex service lifecycle management szükséges
- Enterprise alkalmazások
- Microservices architektúra

Ne használd, ha:

- Kis projektek, kevés függőséggel
- Egyszerű CRUD alkalmazások
- Learning project - manuális DI jobb a megértéshez
- Overhead túl nagy a project méretéhez képest

DI Container - Alternatívák

Alternatíva

Kis projektekhez: manuális DI factory pattern-nel vagy egyszerű service registry

Összefoglalás (1/2)

DI Container lényege

- Automatizált dependency injection
- Központi konfiguráció
- Lifecycle management (singleton, scoped, transient)
- Inversion of Control (IoC)

Népszerű library-k

- **Awilix** - Lightweight, JavaScript-friendly
- **InversifyJS** - TypeScript, decorator-alapú
- **TSyringe** - Microsoft TypeScript DI

Összefoglalás (2/2)

Előnyök

- Automatikus dependency resolution
- Könnyű tesztelhetőség
- Tiszta architektúra
- Skálázható kódbázis

Miért kell email küldés a backend-ből?

Gyakori felhasználási esetek

- **Regisztráció megerősítés** - Verification email
- **Jelszó visszaállítás** - Password reset link
- **Értesítések** - Rendelés visszaigazolás, státusz változás
- **Marketing emailek** - Newsletter, promóciók
- **Rendszer értesítések/ Jelentések** - Hibák, figyelmeztetések - Napi/heti összesítők

Miért backend-ből?

- Biztonság: SMTP hitelesítési adatok nem kerülnek a kliens oldalra
- Megbízhatóság: szerver oldali retry logika
- Komplexitás: template renderelés, csatolmányok
- Skálázhatóság: külön email queue kezelés

Email protokollok

SMTP - Simple Mail Transfer Protocol

- Email **küldésére** használt standard protokoll
- Port: 25 (alap), 587 (TLS), 465 (SSL)
- Autentikáció: username + password vagy API key

Egyéb protokollok

IMAP Email fogadás és szinkronizálás (Port 143/993)

POP3 Email letöltés (Port 110/995)

HTTP API Modern email szolgáltatók (SendGrid, Mailgun)

Email protokollok - Backend használat

Backend használat

Backend fejlesztésben jellemzően csak **SMTP**-t vagy **HTTP API**-t használunk email küldésre

Nodemailer - A legnépszerűbb Node.js email könyvtár

Telepítés

```
npm install nodemailer
```

Jellemzők

- SMTP támogatás
- HTML és plain text emailek
- Csatolmányok (attachments)
- Beágyazott képek (inline images)
- Unicode támogatás
- Stream támogatás nagyméretű csatolmányokhoz
- OAuth2 autentikáció

Nodemailer alapok - Transporter létrehozása (1/3)

Import és konfiguráció Object

```
const nodemailer = require('nodemailer');

const config = {
  host: 'smtp.example.com',
  port: 587,
  secure: false,
  auth: {
```

Nodemailer alapok - Transporter létrehozása (2/4)

Auth és zárás

```
    user: 'your-email@example.com',  
    pass: 'your-password'  
  }  
};
```

Nodemailer alapok - Transporter létrehozása (3/4)

Transporter létrehozása

```
// Transporter létrehozása SMTP-vel  
const transporter = nodemailer.createTransport(config);
```

Nodemailer alapok - Transporter létrehozása (4/4)

Transporter ellenőrzés

```
// Ellenőrzés
transporter.verify((error, success) => {
  if (error) {
    console.log('SMTP_connection_error:', error);
  } else {
    console.log('Server_is_ready_to_send_emails');
  }
});
```

Egyszerű email küldés (1/3)

Mail options megadása

```
const mailOptions = {  
  from: '"Sender_Name"<sender@example.com>',  
  to: 'recipient@example.com',  
  subject: 'Test_Email',  
  text: 'This_is_a_plain_text_email_message.'  
};
```

Egyszerű email küldés (2/3)

Email küldés callback-kel

```
transporter.sendMail(mailOptions, (error, info) => {  
  if (error) {  
    console.log('Error_sending_email:', error);  
  } else {  
    console.log('Email_sent:', info.messageId);  
  }  
});
```

Egyszerű email küldés (3/3)

Async/await használat

```
async function sendEmail() {  
  try {  
    const info = await transporter.sendMail(mailOptions);  
    console.log('Email_sent:', info.messageId);  
    return info;  
  } catch (error) {  
    console.error('Error_sending_email:', error);  
    throw error;  
  }  
}
```

HTML email küldés (1/3)

HTML mail options - Alapadatok

```
const mailOptions = {  
  from: '"Company_Name"<noreply@company.com>',  
  to: 'user@example.com',  
  subject: 'Welcome to Our Service!',  
  html: '  
    <h1>Welcome!</h1>  
    <p>Thank you for registering.</p>  
    <p>Click below to verify your email:</p>
```

HTML email küldés (2/4)

HTML mail options - Gomb stílusok

```
<a href="https://example.com/verify?token=abc123"
  style="background-color: #4CAF50;
        color: white; padding: 10px 20px;
        text-decoration: none;
        display: inline-block;
        border-radius: 4px;">
  Verify Email
</a>
```

HTML email küldés (3/4)

HTML mail options - Lezárás

```
<p>Best regards ,<br>The Team</p>
```

HTML email küldés (4/5)

Plain text alternatíva

```
text: 'Welcome!_Verify_your_email:_\n\n\n\nhttps://example.com/verify?token=abc123'\n};
```

HTML email küldés (5/5)

Email küldés

```
await transporter.sendMail(mailOptions);
```

Fontos

Mindig adj meg text alternatívát a html mellett!

Több címzett és CC/BCC (1/3)

Több címzett kezelése

```
const mailOptions = {  
  from: 'sender@example.com',  
  // Több címzett vesszővel  
  to: 'user1@example.com, user2@example.com',  
  // Vagy tömbként  
  to: [ 'user1@example.com', 'user2@example.com' ],  
}
```

Több címzett és CC/BCC (2/3)

CC és BCC

```
// CC - Carbon Copy (látható másolat)
cc: 'manager@example.com',
// BCC - Blind Carbon Copy (rejtett másolat)
bcc: [ 'admin@example.com', 'audit@example.com' ],
```

Több címzett és CC/BCC (3/3)

Email küldés

```
subject: 'Team_Notification',  
text: 'This_message_is_sent_to_multiple_\n\nrecipients'  
};  
  
await transporter.sendMail(mailOptions);
```

Tömeges email küldés

Sok címzethez NE egy emailben küld! Használj email service provider-t vagy queue-t!

Csatolmányok (Attachments) (1/3)

Fájl csatolása - Mail options

```
const mailOptions = {  
  from: 'sender@example.com',  
  to: 'recipient@example.com',  
  subject: 'Invoice',  
  text: 'Please find your invoice attached.',  
  attachments: [  

```

Csatolmányok (Attachments) (2/3)

Fájl csatolása - Fájlok

```
{  
  filename: 'invoice.pdf',  
  path: '/path/to/invoice.pdf',  
},  
{  
  filename: 'report.xlsx',
```

Csatolmányok (Attachments) (3/4)

Buffer és String csatolmányok

```
    path: '/path/to/report.xlsx',  
  },  
  {  
    // Buffer-ből  
    filename: 'data.txt',  
    content: Buffer.from('File_content_here')  
  },  
}
```

Csatolmányok (Attachments) (4/4)

String tartalom és küldés

```
{  
    // String-ből  
    filename: 'info.txt',  
    content: 'Text_content'  
}  
];  
};  
  
await transporter.sendMail(mailOptions);
```

Beágyazott képek (Inline Images) (1/3)

HTML struktúra CID-vel

```
const mailOptions = {  
  from: 'sender@example.com',  
  to: 'recipient@example.com',  
  subject: 'Email_with_Image',  
  html: '  
    <h1>Company Logo</h1>  
      
    <p>Email content here...</p>  
'
```

Beágyazott képek (Inline Images) (2/3)

HTML és attachments

```

```

Attachments list

```
attachments: [  
  {  
    filename: 'logo.png',  
    path: '/path/to/logo.png',  
    cid: 'logo' // Content-ID referencia  
  },  
]
```

Beágyazott képek (Inline Images) (3/3)

Banner és küldés

```
{  
  filename: 'banner.jpg',  
  path: '/path/to/banner.jpg',  
  cid: 'banner'  
}  
];  
};  
  
await transporter.sendMail(mailOptions);
```

Gmail használata SMTP-vel

Gmail konfiguráció

```
const transporter = nodemailer.createTransport({  
  service: 'gmail', // Előre konfigurált service  
  auth: {  
    user: 'your-email@gmail.com',  
    pass: 'your-app-password' // NEM a normál jelszó!  
  });
```

App Password szükséges!

- 1 Gmail Account Settings → Security
- 2 2-Step Verification bekapcsolása
- 3 App passwords létrehozása
- 4 Az generált app password használata



Gmail korlátozások

Korlátozások

- Gmail: max 500 email/nap
- Production környezetben NE használj Gmail-t!

Email Service Provider - SendGrid példa

Miért használj ESP-t?

- Magas kézbesítési arány (deliverability)
- Spam szűrők elkerülése
- Analytics és tracking
- Skálázhatóság
- Email validáció és bounce kezelés

SendGrid telepítés

```
npm install @sendgrid/mail
```

SendGrid használat (1/2)

SendGrid inicializálás

```
const sgMail = require('@sendgrid/mail');
sgMail.setApiKey(process.env.SENDGRID_API_KEY);

const msg = {
  to: 'recipient@example.com',
  from: 'sender@verified-domain.com',
  subject: 'Hello_from_SendGrid',
```

SendGrid használat (2/2)

SendGrid email küldés

```
text: 'Plain_text_content',  
html: '<strong>HTML_content</strong>'  
};  
  
await sgMail.send(msg);
```

Nodemailer SendGrid-del (1/2)

SendGrid SMTP-n keresztül Nodemailer-rel

```
const nodemailer = require('nodemailer');

const transporter = nodemailer.createTransport({
  host: 'smtp.sendgrid.net',
  port: 587,
  auth: {
    user: 'apikey',
```

Nodemailer SendGrid-del (2/4)

Auth konfiguráció

```
    pass: process.env.SENDGRID_API_KEY  
  }  
});
```

Nodemailer SendGrid-del (3/4)

Mail options

```
const mailOptions = {  
  from: 'verified-sender@yourdomain.com',  
  to: 'recipient@example.com',  
  subject: 'Email via SendGrid SMTP',  
  html: '<h1>Hello from SendGrid!</h1>',  
};
```

Nodemailer SendGrid-del (4/4)

Email küldés

```
try {  
  const info = await transporter.sendMail(mailOptions);  
  console.log('Email sent:', info.messageId);  
} catch (error) {  
  console.error('Error:', error);  
}
```

Email Service Providers összehasonlítás

Szolgáltató	Ingyenes limit	Jellemzők	Ár
SendGrid	100/nap	Kiváló docs, sablonok	\$19.95/hó
Mailgun	5000/hó	Fejlesztő-barát API	\$35/hó
Amazon SES	62000/hó	AWS integráció	\$0.10/1000
Postmark	100/hó	Transactional spec.	\$15/hó
Resend	3000/hó	Modern, egyszerű	\$20/hó

Választási szempontok

- **Mennyiség:** hány email/hó szükséges?
- **Típus:** transactional vs marketing?
- **Funkciók:** analytics, webhooks, sablonok?
- **Ár:** költségvetés
- **Kézbesíthetőség:** deliverability rate

Email küldés Express route-ban (1/4)

Express setup

```
const express = require('express');  
const nodemailer = require('nodemailer');  
const app = express();  
  
app.use(express.json());
```

Email küldés Express route-ban (2/4)

Transporter konfiguráció

```
const transporter = nodemailer.createTransport({  
  host: process.env.SMTP_HOST,  
  port: process.env.SMTP_PORT,  
  auth: {  
    user: process.env.SMTP_USER,
```

Email küldés Express route-ban (3/4)

Auth és route

```
    pass: process.env.SMTP_PASS
  }
});

app.post('/api/register', async (req, res) => {
  const { email, username } = req.body;

  try {
```

Email küldés Express route-ban (4/5)

User mentés és token

```
// User mentése adatbázisba (példa)
// const user = await saveUser({ email, username });
const verificationToken = generateToken();
```

Email küldés Express route-ban (5/6)

Email küldés - sendMail

```
// Email küldés
await transporter.sendMail({
  from: '"MyApp"<noreply@myapp.com>',
  to: email,
  subject: 'Verify your email',
  html: '<h1>Welcome ${username}</h1>
        <p>Click to verify:
```

Email küldés Express route-ban (6/6)

Email küldés - zárás és válasz

```
        <a href="https://myapp.com/verify?token=${verificationToken}"  
          Verify Email</a></p>'  
    });  
    res.json({ message: 'Registration_successful, check_email' });  
  } catch (error) {  
    res.status(500).json({ error: 'Registration_failed' });  
  }  
});
```

Email Service osztály - Clean Architecture (1/4)

EmailService class

```
class EmailService {  
  constructor(transporter) {  
    this.transporter = transporter;  
  }  
  
  async sendWelcomeEmail(user) {  
    const mailOptions = {  
      from: '"MyApp" <noreply@myapp.com>',  
      to: user.email,  
    };  
  }  
}
```

Email Service osztály - Clean Architecture (2/4)

EmailService sendWelcomeEmail

```
        subject: 'Welcome to MyApp!',  
        html: '<h1>Welcome ${user.username}!</h1>'  
    };  
    return await this.transporter.sendMail(mailOptions);  
}
```

Email Service osztály - Clean Architecture (3/4)

EmailService sendPasswordReset

```
async sendPasswordReset(email, resetToken) {  
  const resetUrl = `${process.env.APP_URL}/reset?token=${resetToken}`  
  return await this.transporter.sendMail({  
    from: '"MyApp" <noreply@myapp.com>',  
    to: email,
```

Email Service osztály - Clean Architecture (4/5)

sendPasswordReset (folytatás)

```
    subject: 'Password Reset Request',  
    html: '<h2>Password Reset</h2>  
    <p>Click here to reset your password:</p>  
    <a href="${resetUrl}">Reset Password</a>  
    <p>This link expires in 1 hour.</p>'  
  });  
}
```

Email Service osztály - Clean Architecture (5/5)

EmailService sendOrderConfirmation

```

async sendOrderConfirmation(order) {
  return await this.transporter.sendMail({
    from: '"Shop" <orders@shop.com>',
    to: order.customerEmail,
    subject: 'Order Confirmation #${order.id}',
    html: this.renderOrderTemplate(order) });
}
}
module.exports = EmailService;

```

EmailService használata DI-val (1/4)

Transporter setup

```
// app.js - Setup
const nodemailer = require('nodemailer');
const EmailService = require('./services/EmailService');

const transporter = nodemailer.createTransport({
  host: process.env.SMTP_HOST,
  port: process.env.SMTP_PORT,
```

EmailService használata DI-val (2/5)

Auth

```
auth: {  
  user: process.env.SMTP_USER,  
  pass: process.env.SMTP_PASS  
}  
});
```

EmailService használata DI-val (3/5)

EmailService inicializálás

```
const emailService = new EmailService(transporter);
```

EmailService használata DI-val (4/5)

Route használat

```
// Route-okban használat
app.post('/api/register', async (req, res) => {
  try {
    const user = await userService.createUser(req.body);
    await emailService.sendWelcomeEmail(user);
  }
```

EmailService használata DI-val (5/5)

Register route - válasz

```
    res.json({ message: 'Registration_successful' });  
  } catch (error) {  
    res.status(500).json({ error: error.message });  
  }  
});
```

EmailService használata DI-val (6/6)

Forgot password route

```
app.post('/api/forgot-password', async (req, res) => {  
  try {  
    const resetToken = await userService.generateResetToken(req.body.email);  
    await emailService.sendPasswordReset(req.body.email, resetToken);  
    res.json({ message: 'Password reset email sent' });  
  } catch (error) {  
    res.status(500).json({ error: error.message });  
  }  
});
```

Hibakezelés és retry logika (1/4)

Retry mechanizmus - Setup

```
class EmailService {  
  async sendEmailWithRetry(mailOptions, maxRetries = 3) {  
    for (let attempt = 1; attempt <= maxRetries; attempt++) {  
      try {  
        const info = await this.transporter.sendMail(mailOptions);  
        console.log('Email sent on attempt ${attempt}:', info.messageId);  
      }  
    }  
  }  
}
```

Hibakezelés és retry logika (2/4)

Retry mechanizmus - Return és error

```
return info;  
} catch (error) {
```

Hibakezelés és retry logika (3/4)

Error handling és backoff

```
console.error('Attempt ${attempt} failed:', error.message);
if (attempt === maxRetries) {
    // Utolsó próbálkozás sikertelen
    throw new Error('Failed to send email after ${maxRetries} atte

}

// Exponential backoff: 1s, 2s, 4s...
const delay = Math.pow(2, attempt - 1) * 1000;
```

Hibakezelés és retry logika (4/5)

Exponential backoff

```
    await new Promise(resolve => setTimeout(resolve, delay));  
  }  
}  
}
```

Hibakezelés és retry logika (5/6)

sendWelcomeEmail setup

```
async sendWelcomeEmail(user) {  
  const mailOptions = {  
    from: '"MyApp"<noreply@myapp.com>',  
    to: user.email,  
    subject: 'Welcome!',  
    html: '<h1>Welcome ${user.username}!</h1>'  
  };
```

Hibakezelés és retry logika (6/6)

sendEmailWithRetry call

```
        return await this.sendEmailWithRetry( mailOptions );  
    }  
}
```

Email queue - Bull használatával

Miért kell queue?

- Email küldés lassú (network I/O)
- User ne várjon a válaszra
- Megbízhatóság: retry failed jobs
- Rate limiting kezelése

Bull telepítés

```
npm install bull redis
```

Email queue - Queue létrehozás

Queue létrehozás

```
const Queue = require('bull');

const emailQueue = new Queue('email', {
  redis: {
    host: process.env.REDIS_HOST || 'localhost',
    port: process.env.REDIS_PORT || 6379
  }
});

module.exports = emailQueue;
```

Email queue - Job feldolgozás (1/3)

Worker létrehozása

```
const emailQueue = require('./emailQueue');
const EmailService = require('./services/EmailService');

const emailService = new EmailService(transporter);

// Job processor
emailQueue.process(async (job) => {
  const { type, data } = job.data;
  console.log('Processing email job: ${type}');
```

Email queue - Job feldolgozás (2/3)

Switch case processing

```
switch (type) {  
  case 'welcome':  
    return await emailService.sendWelcomeEmail(data.user);  
  case 'order-confirmation':  
    return await emailService.sendOrderConfirmation(data.order);  
  default:  
    throw new Error('Unknown email type: ${type}');  
}  
});
```

Email queue - Job feldolgozás (3/3)

Event handlers

```
// Event handlers
emailQueue.on('completed', (job, result) => {
  console.log('Job ${job.id} completed:', result.messageId);
});
emailQueue.on('failed', (job, err) => {
  console.error('Job ${job.id} failed:', err.message);
});
```

Email queue - Job hozzáadása (1/4)

Register route setup

```
const emailQueue = require('./emailQueue');

app.post('/api/register', async (req, res) => {
  try {
    const user = await userService.createUser(req.body);
    // Email job hozzáadása a queue-hoz
    await emailQueue.add('welcome', {
```

Email queue - Job hozzáadása (2/4)

Job type és data

```
    type: 'welcome',  
    data: { user }  
  }, {  
    removeOnComplete: true,  
    removeOnFail: false  
  });
```

Email queue - Job hozzáadása (3/4)

Job configuration és response

```
// Azonnal válaszolunk, email küldés háttérben történik
res.json({
  message: 'Registration_successful',
  userId: user.id
});
} catch (error) {
  res.status(500).json({ error: error.message });
}
});
```

Email queue - Job hozzáadása (4/4)

Forgot password route

```
app.post('/api/forgot-password', async (req, res) => {  
  const resetToken = await userService.generateResetToken(req.body.email);  
  await emailQueue.add('password-reset', {  
    type: 'password-reset',  
    data: { email: req.body.email, token: resetToken } });  
  res.json({ message: 'If email exists, reset link was sent' });  
});
```

Környezeti változók kezelése (1/3)

.env fájl - SMTP

```
# SMTP Configuration
SMTP_HOST=smtp.example.com
SMTP_PORT=587
SMTP_USER=your-email@example.com
SMTP_PASS=your-password
```

Környezeti változók kezelése (2/3)

.env fájl - Email settings

```
# Email settings  
EMAIL_FROM_NAME=MyApp  
EMAIL_FROM_ADDRESS=noreply@myapp.com
```

Környezeti változók kezelése (3/4)

.env fájl (folytatás)

```
# SendGrid (alternatíva)  
SENDGRID_API_KEY=SG.xxxxxxx  
  
# Application  
APP_URL=https://myapp.com
```

Környezeti változók kezelése (4/5)

Config betöltése

```
require('dotenv').config();

const emailConfig = {
  host: process.env.SMTP_HOST,
  port: parseInt(process.env.SMTP_PORT),
```

Környezeti változók kezelése (5/5)

Auth és transporter

```
auth: {  
  user: process.env.SMTP_USER,
```

Környezeti változók kezelése (6/6)

Transporter létrehozás

```
    pass: process.env.SMTP_PASS
  }
};

const transporter = nodemailer.createTransport(emailConfig);
```

Best Practices

Biztonsági és fejlesztési ajánlások

- 1 **Soha ne hard-code-old credentials-t** - használj környezeti változókat
- 2 **Használj queue-t** production környezetben - ne blokkolj HTTP requesteket
- 3 **Rate limiting** - véd meg magad spam-től
- 4 **Email validáció** - ellenőrizd az email formátumot
- 5 **Unsubscribe link** - marketing emailekhez kötelező
- 6 **SPF, DKIM, DMARC** - domain autentikáció
- 7 **Teszt környezet** - használj Mailtrap/Ethereal development-hez
- 8 **Template-ek** - külön fájlokban tartsd az email template-eket
- 9 **Error logging** - monitorozd a sikertelen küldéseket
- 10 **Bounce handling** - kezeld a visszapattanó email-eket

Development email teszt - Ethereum Email (1/3)

Ethereum Email

Fake SMTP service teszteléshez - <https://ethereal.email>

Teszt account és config

```
const nodemailer = require('nodemailer');  
  
// Teszt account létrehozása  
const testAccount = await nodemailer.createTestAccount();
```

Development email teszt - Ethereum Email (2/4)

Transporter létrehozása

```
const transporter = nodemailer.createTransport({  
  host: 'smtp.ethereal.email',  
  port: 587,
```

Development email teszt - Ethereum Email (3/4)

Transporter konfiguráció

```
secure: false ,  
auth: {  
    user: testAccount.user ,  
    pass: testAccount.pass  
}  
});
```

Development email teszt - Ethereum Email (4/4)

Email küldés és preview

```
const info = await transporter.sendMail({
  from: '"Test"<test@example.com>',
  to: 'recipient@example.com',
  subject: 'Test_Email',
  html: '<h1>This_is_a_test</h1>',
});
// Preview URL - böngészőben megnézhető
console.log('Preview_URL:', nodemailer.getTestMessageUrl(info));
```

Összefoglalás (1/2)

Email küldés backend-ből

- SMTP protokoll használata
- Nodemailer - de facto standard Node.js-hez
- HTML, plain text, attachments támogatása
- Email Service Providers (SendGrid, Mailgun, SES)

Production best practices

- Queue használata (Bull + Redis)
- Retry logika hibák kezelésére
- EmailService osztály clean architecture-höz
- Környezeti változók biztonságos kezelése
- Error logging és monitoring

Összefoglalás (2/2)

Development

- Ethereum Email vagy Mailtrap teszteléshez
- Ne használj production SMTP-t development-ben
- Template-ek külön kezelése (következő téma)

Miért használjunk email template-eket? (1/2)

Problémák template nélkül

- HTML string-ek a kódban → nehéz karbantartani
- Duplikált kód különböző email típusokhoz
- Nehéz változtatni a design-t
- Frontend és backend keveredik
- Nincs újrafelhasználhatóság

Miért használjunk email template-eket? (2/2)

Template-ekkel

- Szeparált template fájlok
- Dinamikus adatok injektálása
- Újrafelhasználható komponensek (header, footer)
- Könnyű módosítás és tesztelés
- Designer-ek is módosíthatják
- Verziókövetés

Template Engine-ek

Népszerű template engine-ek Node.js-hez

[Handlebars](#) Logika-mentes, egyszerű szintaxis, blokk helper-ek

[EJS](#) Embedded JavaScript, ismerős szintaxis

[Pug \(Jade\)](#) Tömör, whitespace-based szintaxis

[Nunjucks](#) Jinja2-szerű, gazdag funkciók

[Mustache](#) Minimális, logika-mentes

Email-specifikus eszközök

[MJML](#) Responsive email framework - JSON/XML-szerű szintaxis

[Foundation for Emails](#) Responsive email framework

[Email-templates](#) Nodemailer integráció többféle engine-hez

Handlebars alapok

Telepítés

```
npm install handlebars
```

Template szintaxis (1/2)

```
<!DOCTYPE html>
<html>
<head>
  <title>{{subject}}</title>
</head>
<body>
  <h1>Hello {{username}}!</h1>
  <p>Your email is: {{email}}</p>
```

Handlebars alapok (2/3)

Template szintaxis - Conditionals

```
{{#if isPremium}}  
  <p>Thanks for being a premium member!</p>  
{{else}}  
  <p>Upgrade to premium for more features.</p>  
{{/if}}
```

Handlebars alapok (3/3)

Template szintaxis - Loops

```
<ul>
  {{#each items}}
    <li>{{this.name}} - ${{this.price}}</li>
  {{/each}}
</ul>
</body>
</html>
```

Handlebars használata Node.js-ben (1/3)

Setup és template beolvasás

```
const handlebars = require('handlebars');
const fs = require('fs').promises;

async function renderTemplate(templateName, data) {
  // Template fájl beolvasása
  const templatePath = `./templates/${templateName}.hbs`;
  const templateSource = await fs.readFile(templatePath, 'utf-8');

  // Template compile
  const template = handlebars.compile(templateSource);
```

Handlebars használata Node.js-ben (2/3)

Render adatokkal

```
async function renderTemplate(templateName, data) {  
  // ... template beolvasás és compile  
  
  // Render adatokkal  
  const html = template(data);  
  
  return html;  
}
```

Handlebars használata Node.js-ben (3/4)

Használat példa - Data

```
const data = {  
  username: 'John_Doe',  
  email: 'john@example.com',  
  isPremium: true,  
  items: [  
    { name: 'Product_1', price: 29.99 },  
    { name: 'Product_2', price: 49.99 }  
  ]  
};
```

Handlebars használata Node.js-ben (4/4)

Használat példa - Render

```
const html = await renderTemplate('welcome', data);
```

Handlebars custom helper-ek (1/3)

Dátum helper

```
const handlebars = require('handlebars');  
// Egyszerű helper - dátum formázás  
handlebars.registerHelper('formatDate', (date) => {  
  return new Date(date).toLocaleDateString('hu-HU');});
```

Currency helper

```
// Helper paraméterekkel  
handlebars.registerHelper('formatCurrency',  
  (amount, currency = 'USD') => {  
    return new Intl.NumberFormat('en-US', {  
      style: 'currency', currency: currency}).format(amount);});
```

Handlebars custom helper-ek (2/3)

Block helper - feltételes logika

```
// Block helper - feltételes logika
handlebars.registerHelper('ifEquals',
function(arg1, arg2, options) {
return (arg1 === arg2)
? options.fn(this)
: options.inverse(this);
});
```

Handlebars custom helper-ek (3/3)

Template használat

```
<p>Order date: {{formatDate orderDate}}</p>
<p>Total: {{formatCurrency total 'EUR'}}</p>

{{#ifEquals status 'completed'}}
  <p>Order is completed!</p>
{{else}}
  <p>Order is pending.</p>
{{/ifEquals}}
```

Handlebars partials - újrafelhasználható részek (1/3)

Setup

```
const handlebars = require('handlebars');
const fs = require('fs').promises;

// Partial-ok betöltése
async function registerPartials() {
  const header = await fs.readFile(
    './templates/partials/header.hbs', 'utf-8');
}
```

Handlebars partials - újrafelhasználható részek (2/3)

Partial fájlok beolvasása

```
const footer = await fs.readFile(  
  './templates/partials/footer.hbs', 'utf-8');  
const button = await fs.readFile(  
  './templates/partials/button.hbs', 'utf-8');
```

Handlebars partials - újrafelhasználható részek (3/3)

Partial regisztráció

```
// Partials regisztrálása
handlebars.registerPartial('header', header);
handlebars.registerPartial('footer', footer);
handlebars.registerPartial('button', button);
}

await registerPartials();
```

Partial példák (1/2)

header.hbs

```
<div style="background: #333;
          color: white;
          padding: 20px;">
  
  <h1>{{ companyName }}</h1>
</div>
```

footer.hbs

```
<div style="text-align: center;">
  <p>    {{year}}</p>
</div>
```

Partial példák (2/2)

main.hbs - Partials használata

```
<!DOCTYPE html>
<html>
<body>
  {{> header}}
  <div class="content">
    {{body}}
  </div>
  {{> footer}}
</body>
</html>
```

Email-templates csomag - All-in-one megoldás

Telepítés

```
npm install email-templates
```

Jellemzők

- Automatikus HTML és text verzió generálás
- CSS inlining (email client compatibility)
- Template engine választék (Pug, Handlebars, EJS, etc.)
- Preview server fejlesztéshez
- Nodemailer integráció

Email-templates használata (1/3)

Transporter setup

```
const Email = require('email-templates');
const nodemailer = require('nodemailer');
const transporter = nodemailer.createTransport({
  host: process.env.SMTP_HOST,
  port: process.env.SMTP_PORT,
  auth: {
    user: process.env.SMTP_USER,
    pass: process.env.SMTP_PASS
  }
});
```

Email-templates használata (2/3)

Email konfiguráció

```
const email = new Email({  
  message: { from: 'noreply@example.com' },  
  transport: transporter,  
  views: {  
    root: './emails', // Template könyvtár  
    options: {  
      extension: 'hbs' // Handlebars  
    }  
  },  
});
```

Email-templates használata (3/3)

CSS inlining config

```
// ... folytatás
juice: true, // CSS inlining
juiceResources: {
  preserveImportant: true,
  webResources: {
    relativeTo: './emails'
  }
}
});
```

Email-templates - Email küldés (1/2)

Welcome email példa

```
// Email küldés template-tel
await email.send({
  template: 'welcome', // ./emails/welcome könyvtár
  message: { to: 'user@example.com' },
  locals: {
    username: 'John_Doe',
    verifyUrl: 'https://example.com/verify?token=abc123',
    year: new Date().getFullYear()
  }
});
```

Email-templates - Email küldés (2/2)

Order confirmation példa

```
// Több template eset
await email.send({
  template: 'order-confirmation',
  message: { to: order.customerEmail },
  locals: {
    orderNumber: order.id, items: order.items,
    total: order.total, shippingAddress: order.address
  }
});
```

Template könyvtár struktúra (1/2)

Fájl struktúra - Layouts és Partials

```
emails /  
  _layouts /  
    default.hbs      # Alap layout  
  _partials /  
    header.hbs  
    footer.hbs  
    button.hbs
```

Template könyvtár struktúra (2/3)

Fájl struktúra - Welcome template

```
welcome/  
    html.hbs      # HTML verzió  
    text.hbs      # Plain text  
    subject.hbs   # Subject  
    style.css     # CSS (inline-olva lesz)
```

Template könyvtár struktúra (3/3)

Fájl struktúra - További templates

```
password-reset /  
    html.hbs  
    text.hbs  
    subject.hbs  
order-confirmation /  
    html.hbs  
    text.hbs  
    subject.hbs
```

Fontos

Minden template-hez készíts text.hbs verziót is (accessibility)!

Welcome template példa - HTML (1/3)

emails/welcome/html.hbs - Head

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <link rel="stylesheet" href="style.css">
</head>
<body>
  {{> header companyName="MyApp" logoUrl="/logo.png"}}
```

Welcome template példa - HTML (2/3)

emails/welcome/html.hbs - Content

```
<div class="container">
  <h1>Welcome, {{username}}!</h1>
  <p>Thank you for joining MyApp.
    We're excited to have you on board!</p>

  <p>To get started , please verify your email address
    by clicking the button below:</p>

  {{> button text="Verify Email" url=verifyUrl color="#4CAF50"}}
```

Welcome template példa - HTML (3/3)

emails/welcome/html.hbs - Footer

```
<p>If you didn't create an account ,  
    please ignore this email.</p>  
</div>  
  
{{> footer year=year}}  
</body>  
</html>
```

Welcome template példa - Text (1/2)

emails/welcome/text.hbs - Content

```
Welcome, {{username}}!  
Thank you for joining MyApp.  
We're excited to have you on board!
```

```
To get started , please verify your email address  
by visiting :  
{{verifyUrl}}
```

```
If you didn't create an account ,  
please ignore this email.
```

Welcome template példa - Text (2/2)

emails/welcome/text.hbs - Signature

Best regards ,
The MyApp Team

{{year}} MyApp. All rights reserved .

emails/welcome/subject.hbs

Welcome to MyApp, {{username}}!

Template subject - Megjegyzés

Megjegyzés

A `subject.hbs` is lehet dinamikus template!

Reusable button partial (1/2)

emails/_partials/button.hbs - Table structure

```
<table role="presentation" cellpadding="0" cellspacing="0"
      border="0" align="center" style="margin: 20px 0;">
  <tr>
    <td style="border-radius: 4px; background: {{color}};">
      <a href="{{url}}"
        style="border: none;
        color: white;
        padding: 12px 24px;
        text-decoration: none;
        display: inline-block;
```

Reusable button partial (2/2)

emails/_partials/button.hbs - Link styling

```
        style=" ...  
        font-size: 16px;  
        font-weight: bold;">  
        {{text}}  
    </a>  
  </td>  
</tr>  
</table>
```

Miért table layout?

Email client-ek (Outlook, Gmail) rosszul támogatják a modern CSS-t. Table-based layout a legkompatibilisebb megoldás.

CSS inlining

Miért kell CSS inlining?

- Gmail és sok email client eltávolítja a `<style>` tag-et
- Biztonságos megoldás: inline style attribute-ok
- Automatikus: juice library végzi

Előtte (style.css)

```
.button {  
  background: #4CAF50;  
  color: white;  
  padding: 12px 24px;  
}
```

Utána (inline)

```
<a style="background: #4CAF50;  
        color: white;  
        padding: 12px 24px;">  
  Button  
</a>
```

CSS inlining - Fontos beállítások

Email-templates automatikusan végzi

```
const email = new Email({  
  // ...  
  juice: true, // CSS inlining engedélyezése  
  juiceResources: {  
    preserveImportant: true  
  }  
});
```

Összefoglalás (1/2)

Email templating lényege

- Szeparált, karbantartható email template-ek
- Dinamikus adatok injektálása
- Újrafelhasználható komponensek (partials)
- HTML és plain text verziók

Eszközök

- Template engine-ek: Handlebars, EJS, Pug
- Email-specific: MJML, Foundation for Emails
- Node.js library: email-templates csomag
- CSS inlining: juice library

Összefoglalás (2/2)

Kulcsfontosságú szempontok

- Email client kompatibilitás (table layout, inline CSS)
- Responsive design mobil eszközökre
- Tesztelés és preview development-ben
- Clean architecture: EmailService osztály