# Security & Production Readiness

## Cel zajec

Dodanie warstwy bezpieczenstwa i monitoringu do API ToDo:

- **Helmet.js** - security headers
- **Rate Limiting** - ochrona przed brute-force
- **Sentry** - monitoring i error tracking

---

## Czesc 1: Helmet.js (15 min)

### Co to jest?

Helmet.js dodaje nagłówki HTTP ktore chronia przed popularnymi atakami:

- XSS (Cross-Site Scripting)
- Clickjacking
- MIME sniffing

### Krok 1: Instalacja

```
npm install helmet
```

### Krok 2: Dodanie do index.ts

Otwoorz `src/index.ts` i dodaj:

```
import helmet from 'helmet';

// Dodaj PRZED innymi middleware (zaraz po utworzeniu app)
app.use(helmet());
```

### Krok 3: Weryfikacja

1. Uruchom serwer: `npm run dev`
2. Otworz przegladarke -> DevTools (F12)
3. Network -> kliknij na dowolny request
4. Sprawdz Response Headers

**Powinny pojawic sie nowe naglowki:**

- `X-Content-Type-Options: nosniff`
- `X-Frame-Options: SAMEORIGIN`

- `X-XSS-Protection: 0`
- `Strict-Transport-Security`

---

## Czesc 2: Rate Limiting (30 min)

### Co to jest?

Rate limiting ogranicza liczbe requestow z jednego IP w danym czasie. Chroni przed:

- Atakami brute-force na logowanie
- DDoS
- Naduzywaniem API

### Krok 1: Instalacja

```
npm install express-rate-limit
```

### Krok 2: Utworz plik middleware

Utworz nowy plik `src/infrastructure/middleware/rateLimit.ts`:

```
import rateLimit from 'express-rate-limit';

/**
 * Globalny limiter - 100 requestow na 15 minut
 */
export const globalLimiter = rateLimit({
  windowMs: 15 * 60 * 1000, // 15 minut
  max: 100, // maksymalnie 100 requestow
  message: {
    error: 'Zbyt wiele requestow. Sprobuj ponownie za 15 minut.'
  },
  standardHeaders: true, // Zwraca info o limicie w headerach
  legacyHeaders: false,
});

/**
 * Limiter dla auth - 5 prob na minute (ochrona przed brute-force)
 */
export const authLimiter = rateLimit({
  windowMs: 60 * 1000, // 1 minuta
  max: 5, // maksymalnie 5 prob
  message: {
    error: 'Zbyt wiele prob logowania. Poczekaj minute.'
  },
```

```
  standardHeaders: true,
  legacyHeaders: false,
});
```

### Krok 3: Dodaj do index.ts

```
import { globalLimiter, authLimiter } from './infrastructure/middleware/rat

// Dodaj PO helmet, PRZED routes
app.use(globalLimiter);

// Osobny, ostrzejszy limit dla /auth
app.use('/auth', authLimiter);
```

### Krok 4: Weryfikacja

1. Uruchom serwer
2. Sprobuj zalogowac sie 6 razy pod rzad (szybko!)
3. Przy 6. probie powinienes dostac blad 429:

```
{
  "error": "Zbyt wiele prob logowania. Poczekaj minute."
}
```

**Sprawdz tez headery odpowiedzi:**

- `RateLimit-Limit: 5`
- `RateLimit-Remaining: 0`
- `RateLimit-Reset: <timestamp>`

---

## Czesc 3: Sentry - Monitoring (30 min)

### Co to jest?

Sentry to narzedzie do:

- Sledzenia bledow w produkcji
- Monitorowania wydajnosci
- Alertow gdy cos sie zepsuje

### Krok 1: Zaloz konto Sentry

1. Wejdz na https://sentry.io/signup/
2. Zaloz konto (najszybciej przez GitHub)
3. Utworz nowy projekt:

- Wybierz platform: **Node.js**
- Wybierz framework: **Express**
- Nadaj nazwe projektu (np. `todo-api`)

4. **Skopiuj DSN** - bedzie wygladal tak:

```
https://abc123@o123456.ingest.sentry.io/1234567
```

### Krok 2: Instalacja

```
npm install @sentry/node
```

### Krok 3: Utworz plik konfiguracji

Utworz nowy plik `src/infrastructure/monitoring/sentry.ts`:

```
import * as Sentry from '@sentry/node';

export function initSentry(): void {
  // Inicjalizuj tylko jesli mamy DSN
  if (!process.env.SENTRY_DSN) {
    console.log('Sentry: brak DSN, monitoring wylaczony');
    return;
  }

  Sentry.init({
    dsn: process.env.SENTRY_DSN,
    environment: process.env.NODE_ENV || 'development',
    tracesSampleRate: 1.0, // 100% requestow w development
  });

  console.log('Sentry: monitoring wlaczony');
}

export { Sentry };
```

### Krok 4: Dodaj do index.ts

```
import { initSentry, Sentry } from './infrastructure/monitoring/sentry';

// NA SAMYM POCZATKU (przed innymi importami aplikacji)
initSentry();

// ... tworzenie app ...
```

```
// ZARAZ PO UTWORZENIU APP (przed innymi middleware)
app.use(Sentry.Handlers.requestHandler());

// ... reszta middleware i routes ...

// NA KONCU - PRZED twoim error handlerem
app.use(Sentry.Handlers.errorHandler());

// Twoj error handler MUSI byc na samym koncu
app.use(errorHandler);
```

### Krok 5: Dodaj DSN do .env

```
SENTRY_DSN=https://twoj-dsn@sentry.io/123456
```

### Krok 6: Testowanie

Dodaj tymczasowy endpoint do testowania (mozesz go pozniej usunac):

```
// Endpoint testowy - USUN PO TESTACH
app.get('/debug-sentry', () => {
  throw new Error('Testowy blad Sentry!');
});
```

1. Uruchom serwer
2. Wejdz na `http://localhost:3000/debug-sentry`
3. Powinien pojawic sie blad 500
4. Wejdz do Sentry dashboard -> Issues
5. Powinienes zobaczyc "Testowy blad Sentry!"

---

## Pelny przyklad index.ts

Po wszystkich zmianach twoj `index.ts` powinien wygladac mniej wiecej tak:

```
import express from 'express';
import cors from 'cors';
import helmet from 'helmet';

import { initSentry, Sentry } from './infrastructure/monitoring/sentry';
import { globalLimiter, authLimiter } from './infrastructure/middleware/rate
import { errorHandler } from './infrastructure/middleware/errorHandler';
import routes from './presentation/routes';
```

```
// 1. Inicjalizacja Sentry (na samym poczatku)
initSentry();

const app = express();

// 2. Sentry request handler (pierwszy middleware)
app.use(Sentry.Handlers.requestHandler());

// 3. Security
app.use(helmet());
app.use(cors());

// 4. Rate limiting
app.use(globalLimiter);
app.use('/auth', authLimiter);

// 5. Body parsing
app.use(express.json());

// 6. Routes
app.use(routes);

// 7. Sentry error handler (przed twoim error handlerem)
app.use(Sentry.Handlers.errorHandler());

// 8. Twoj error handler (na samym koncu)
app.use(errorHandler);

const PORT = process.env.PORT || 3000;
app.listen(PORT, () => {
  console.log(`Server running on port ${PORT}`);
});
```

---

## Weryfikacja API Testerem

Po zakonczeniu wszystkich krokow, przetestuj API:

```
./api-tester --url http://localhost:3000 --setup
```

Wszystkie testy powinny przechodzic.

---

## Pytania kontrolne

1. Jakie naglowki dodaje Helmet i przed czym chronia?
2. Dlaczego limit dla `/auth` jest ostrzejszy niz globalny?
3. Co sie stanie gdy przekroczysz rate limit?
4. Jak Sentry pomaga w produkcji?
5. Dlaczego kolejnosc middleware jest wazna?

---

## Zadanie dodatkowe (dla chetnych)

1. Zmien limit authLimiter na 3 proby na 30 sekund
2. Dodaj osobny limit dla `/tasks` (50 requestow na minute)
3. Sprawdz w Sentry Performance jak dlugo trwaja requesty

---

## Rozwiazywanie problemow

### Helmet nie dziala

- Upewnij sie ze `app.use(helmet())` jest PRZED routes

### Rate limit nie dziala

- Sprawdz czy middleware jest przed routes
- Pamietaj ze limit jest per IP - zrestartuj serwer zeby zresetowac

### Sentry nie pokazuje bledow

- Sprawdz czy DSN jest poprawny w .env
- Sprawdz logi serwera - powinno byc "Sentry: monitoring wlaczony"
- Poczekaj 1-2 minuty, Sentry czasem ma opoznienie

### Blad importu Sentry.Handlers

- Upewnij sie ze uzywasz `@sentry/node` w wersji 7.x lub 8.x
- Sprawdz czy masz `import * as Sentry from '@sentry/node'`