✓ Wykonano

Ponieważ na najbliższych zajęciach nie będzie już zadania ćwiczeniowego, dlatego do samodzielnego wykonania jest pięć zadań przygotowawczych (ikona 🏠).

Materiały pomocnicze

Deno

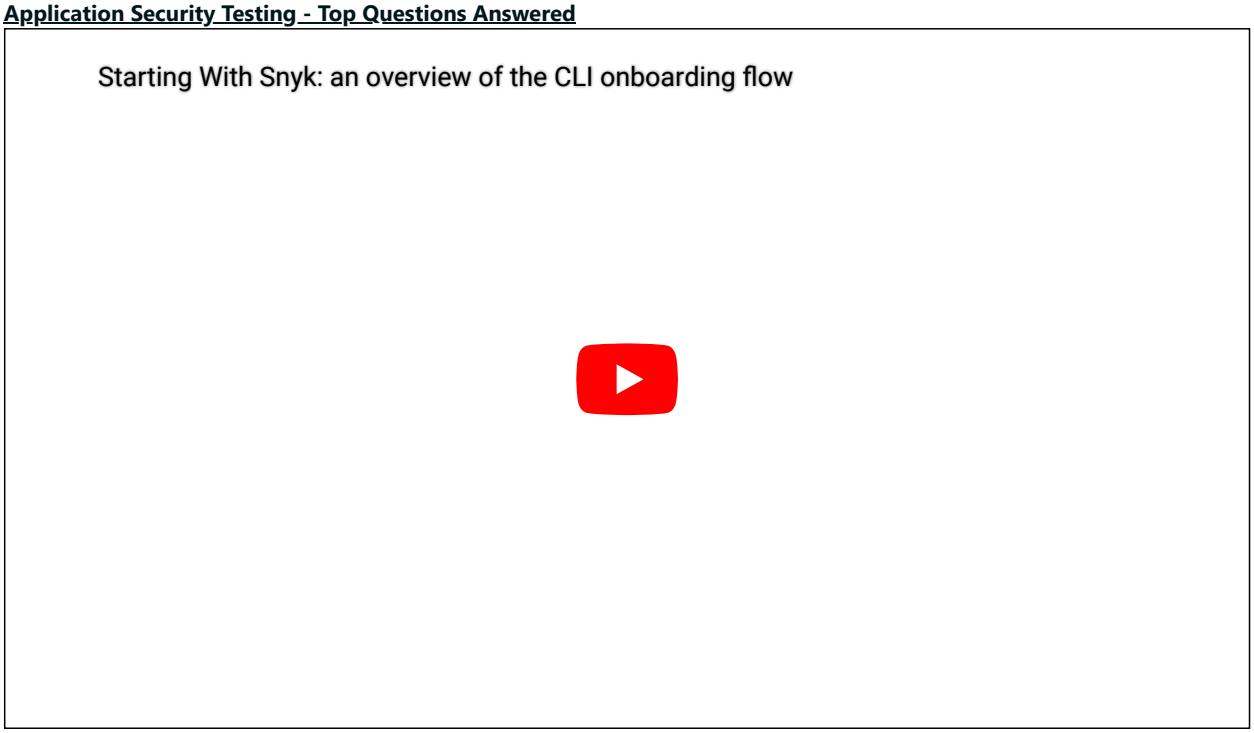
1. Deno - następca Node.js? Backend dla JavaScript developerów

2. Getting Started with Deno & MongoDB

Bezpieczeństwo aplikacji Node.js

1. **Production Best Practices: Security**
2. **Node.js Security Checklist**

Narzędzia do badania bezpieczeństwa aplikacji WWW

1. **Różnice między SAST, DAST, IAST i RASP**
2. **Application Security Testing - Top Questions Answered**
3. 

4. **Wprowadzenie do narzędzia Zed Attack Proxy (ZAP)**
5. **Samouczek OWASP ZAP: Kompleksowy przegląd narzędzia OWASP ZAP**
6. 

🏠 1. Deno

1. Zainstaluj Deno
2. Zainstaluj wtyczkę *Deno*.
3. Wykonaj następujące komendy:

```
deno init cw7
cd cw7
mkdir views
```

4. Obejrzyj zawartość wszystkich wygenerowanych plików — `more * | cat`
5. Sprawdź, w praktyce, działanie poniższych komend:

```
$ deno run main.ts
$ deno task dev
$ deno test
```

6. Utwórz plik 'app1.ts' o następującej zawartości:

```
/**
 * @author Stanisław Polak <polak@agh.edu.pl>
 */

// @deno-types="npm:@types/express@^4"
import express, { Express, Request, Response } from "npm:express@^4";
import morgan from "npm:morgan@^1";
import "npm:pug@^3";

const app: Express = express();
deno_logo =
'https://upload.wikimedia.org/wikipedia/commons/thumb/e/e8/Deno_2021.svg/120px-
Deno_2021.svg.png';

app.set('view engine', 'pug');
app.locals.pretty = app.get('env') === 'development';

app.use(morgan('dev'));
app.use(express.urlencoded({ extended: false }));

app.get('/', function (_req: Request, res: Request) {
    res.render('index', {deno_logo});
});

app.post('/', function (req: Response, res: Response) {
    res.send(`Hello '${req.body.name}'`);
});

app.listen(8000, function () {
    console.log('The application is available on port 8000');
});
```

7. Utwórz plik 'views/index.pug' o następującej zawartości:

```
doctype html
html(lang='en')
    head
        meta(charset='utf-8')
        meta(name='viewport' content='width=device-width, initial-scale=1')
        title First Express application in Deno
    body
        main
            h1 First Express application in Deno
            img(src=deno_logo alt='[Deno logo]')
            form(method='POST' action='/')
                label(for='name') Give your name
                input(name='name')
                br
                input(type='submit')
                input(type='reset')
```

8. Uruchom aplikację — `deno run app1.ts` i sprawdź, czy *Deno*, domyślnie:
    - Używa trybu ścisłego?
    - Uwzględnia typowanie?
9. Popraw błędną linię i ponownie uruchom aplikację.
10. Wpisz w przeglądarce adres http://localhost:8000/ i obejrzyj stronę wynikową.
11. Zatrzymaj serwer.
12. Sprawdź poprawność typowania za pomocą komendy `deno check app1.ts`, a następnie popraw znalezione błędy typowania.
13. Przeczytaj następujące sekcje manuala: Command Line Interface, Configuration File oraz Permissions
14. Uruchom aplikację w taki sposób (z takim opcjami linii komend), aby *Deno*:
    - Wykonywał sprawdzanie typowania przed jej uruchomieniem.
    - Restartował ją, gdy treść kodu źródłowego uległa zmianie.
    - Nie pytał o uprawnienia do wykonywania operacji wrażliwych (dostęp do sieci, itp.), tzn. należy zezwolić na wykonywanie tych operacji za pomocą opcji linii komend.
15. Spowoduj, aby wykonanie komendy `deno task express` uruchomiało aplikację z wyżej opisanymi opcjami linii komend.

Dla ciekawskich

- **Zarządzanie zależnościami**
- **Odpowiedniki komend NodeJS**

🏠 2. Język TypeScript

- Poniższe zadanie proszę wykonać w oparciu o **slajdy z wykładu** lub **podręcznik języka TypeScript**
- Możesz skorzystać z pakietu *Express* — patrz 'app1.ts' lub użyć **wbudowanego serwera WWW** — patrz przykład poniżej.

```ts
/**
 * @author Stanisław Polak <polak@agh.edu.pl>
 */

Deno.serve(async (req) => {
  const url = new URL(req.url);

  switch ([req.method, url.pathname].join(" ")) {
    case "GET /":
      return new Response(
`<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <title>Vanilla Deno application</title>
  </head>
  <body>
    <main>
      <h1>Vanilla Deno application</h1>
      <form method="POST" action="/">
        <label for="name">Give your name</label>
        <input name="name">
        <br>
        <input type="submit">
        <input type="reset">
      </form>
    </main>
  </body>
</html>`,
        {
          headers: {
            "content-type": "text/html; charset=utf-8",
          },
        }
      );
    case "POST /": {
      const data = await req.formData();
      return new Response(`Hello '${data.get("name")}'`);
    }
    default:
      return new Response("Error 501: Not implemented", { status: 501 });
  }
});
```

Zmodyfikuj aplikację "Księga gości", którą utworzyłeś/aś na zajęciach "Środowisko uruchomieniowe NodeJS":

1. Implementacja ma bazować na języku *TypeScript*, a nie *JavaScript*:
   - **Obowiązkowo**, wszystkie zmienne oraz argumenty funkcji/metod mają posiadać deklarację typu. W przypadku ostatnich z wymienionych należy również określić typ zwracanej wartości.
     O ile to możliwe, proszę unikać następujących typów: `any` oraz `unknown`
   - **Dla ambitnych**, użyj również innych składników TypeScript, na przykład: klasy, moduły / przestrzenie nazewnicze, ...
2. Wpisy mają być przechowywane w bazie danych MongoDB, a nie w pliku
3. Trasy:
   - `GET(/)` — wyświetlenie aktualnych wpisów oraz formularza dodawania nowego wpisu
   - `POST(/)` — przetwarzanie zawartości formularza — dodawanie nowego wpisu do bazy danych

🏠 3. Oak

1. Przeczytaj Web Frameworks
2. Utwórz plik 'app2.ts' o następującej zawartości:

```
// Original source: https://medium.com/recoding/rendering-html-css-in-deno-using-view-engine-e07469613598
// Modifications: Stanisław Polak <polak@agh.edu.pl>

// Requiring modules
import {
  Application,
  Router,
  Context,
  // send
} from "https://deno.land/x/oak/mod.ts";
import {
  dejsEngine,
  oakAdapter,
  viewEngine,
} from "https://deno.land/x/view_engine/mod.ts";
import logger from "https://deno.land/x/oak_logger/mod.ts";

// Initiate app
const app: Application = new Application();
const router: Router = new Router({
  // prefix: "/admin",
});

// Allowing Static file to fetch from server
/*
app.use(async (ctx, next) => {
  await send(ctx, ctx.request.url.pathname, {
    root: `${Deno.cwd()}/public`,
  });
  next();
});
*/

app.use(logger.logger);
app.use(logger.responseTime);

// Passing view-engine as middleware
app.use(viewEngine(oakAdapter, dejsEngine, { viewRoot: "./views" }));

// Creating Routes
router.get("/", async (ctx: Context) => {
  await ctx.render("index.ejs", {
    data: { title: "First Oak application in Deno" },
  });
});

router.post("/", async (ctx: Context) => {
  const reqBodyForm = await ctx.request.body.form();
  // ctx.response.type = 'text/html'
  ctx.response.body = `Hello '${reqBodyForm.get("name")}'`;
});

// Adding middleware to require our router
app.use(router.routes());
app.use(router.allowedMethods());

// Making app to listen to port
console.log("App is listening to port: 8000");
await app.listen({ port: 8000 });
```

3. Utwórz plik 'views/index.ejs' o następującej zawartości:

```
<html lang="en">

<head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <title><%= data.title %></title>
</head>

<body>
    <main>
        <h1><%= data.title %></h1>
        <img
src="https://upload.wikimedia.org/wikipedia/commons/thumb/e/e8/Deno_2021.svg/120px-Deno_2021.svg.png"
            alt="[Deno logo]">
        <form method="POST" action="/">
            <label for="name">Give your name</label>
            <input name="name"><br>
            <input type="submit">
            <input type="reset">
        </form>
    </main>
</body>

</html>
```

4. Uruchom aplikację, a następnie wpisz w przeglądarce adres http://localhost:8000/ i obejrzyj stronę wynikową.
5. Bazując na powyższym przykładzie, utwórz alternatywną wersję aplikacji "Księga gości":
    - Zamiast z *Express* ma korzystać z Oak
    - Zamiast *Pug* ma używać systemu szablonów wspieranego przez View Engine
    - Obsługa bazy danych *MongoDB* — deno_mongo, npm:mongodb lub dowolny inny sterownik

Uwagi

1. Program *Deno* należy uruchamiać bez opcji `--check`

🏠 4. Narzędzie Snyk

1. Zainstaluj wtyczkę Snyk Security - Code, Open Source Dependencies, IaC Configurations

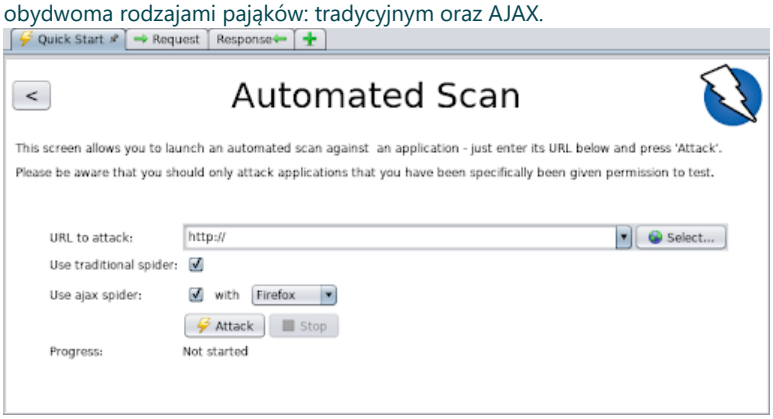
2. Korzystając z tej wtyczki, znajdź luki w kodzie źródłowym aplikacji, którą tworzymy przez cały cykl ćwiczeń z JavaScript. Usuń znalezione luki.

- W przypadku osób, które nie dysponują własną implementacją aplikacji z ćwiczeń, proszę użyć Snyk-a do znalezienia luk w kodzie źródłowym aplikacji "**Księga gości**".
- Nie musisz instalować programu *Snyk*, gdyż jest on automatycznie instalowany wraz ze wtyczką. Jeżeli chcesz go używać z linii komend, to skorzystaj z komendy 'npx', np. `npx snyk --help` .

🏠 5. Narzędzie ZAP

1. Zainstaluj narzędzie ZAP
2. Uruchom aplikację z ćwiczeń
3. Za pomocą programu *ZAP* — opcja "Automated Scan" — przetestuj bezpieczeństwo swojej aplikacji. Popraw ją — usuń, znalezione w niej, luki **poziomu ryzyka średniego oraz wysokiego**

- Ponieważ aplikacja z ćwiczeń używa również JavaScript po stronie klienta, zachęcam do przetestowania jej bezpieczeństwa obydwoma rodzajami pająków: tradycyjnym oraz AJAX.



- W przypadku osób, które nie dysponują własną implementacją aplikacji z ćwiczeń, proszę przetestować aplikację "**Księga gości**".

Edytuj zadanie | Usuń zadanie