WebSec-Dokumentation

Unser WebSec-Projekt ist eine Webanwendung, die es Benutzern ermöglicht, Tickets zu erstellen, anzuzeigen und zu verwalten. Die Anwendung ermöglicht sowohl die Registrierung und die Authentifizierung von Benutzern als auch die Interaktion mit Tickets.

OWASP TOP 10 Sicherheitslücken

Insecure Design

1. **Cross-Origin-Schwachstelle (Cross-Origin)**: Die Anwendung zeigt eine Schwäche im Umgang mit Cross-Origin-Anfragen, was zu einem erhöhten Risiko von Angriffen durch böswillige Skripte aus anderen Quellen führen kann.
2. **Schwache Token-Generierung (Benutzername + Passwort)**: Die Art und Weise, wie die Anwendung Tokens generiert, ist unsicher, da sensible Informationen wie Benutzernamen und Passwörter kombiniert werden, was ein Sicherheitsrisiko darstellt.
3. **Fehlende Passwortrichtlinie**: Die Anwendung verfügt nicht über angemessene Passwortrichtlinien, wie beispielsweise Mindestzeichenanforderungen oder Komplexitätsanforderungen, was sie anfällig für Brute-Force-Angriffe und andere Passwort-basierte Angriffe macht.

Security logging and Monitory failures

Die Sicherheitsmechanismen für das Protokollieren und Überwachen in der WebSec-Anwendung weisen Mängel auf, wodurch potenzielle Sicherheitsverletzungen oder ungewöhnliche Aktivitäten möglicherweise nicht rechtzeitig erkannt und protokolliert werden. Diese Schwachstelle könnte zu einer verminderten Fähigkeit führen, auf Sicherheitsbedrohungen angemessen zu reagieren und diese zu beheben, was letztendlich die Sicherheit und Integrität der Anwendung gefährden könnte.

Broken Access Control

Die Anwendung weist Schwächen in der Zugriffskontrolle auf, was potenziell unzureichende Mechanismen zur Steuerung und Beschränkung des Zugriffs auf bestimmte Ressourcen und Funktionen der Anwendung bedeutet. Dies könnte dazu führen, dass nicht autorisierte Benutzer unangemessenen Zugriff auf vertrauliche Daten oder Funktionalitäten erlangen und unerwünschte Änderungen vornehmen können, was die Integrität und Sicherheit der Anwendung gefährdet. Es ist wichtig, angemessene Zugriffskontrollmechanismen zu implementieren, um die Vertraulichkeit und Integrität der Anwendung zu gewährleisten.

Verwendete Technologien

In diesem faszinierenden Projekt haben wir bewusst auf Technologien gesetzt, die uns bisher weniger vertraut waren. Daher war es eine aufregende Gelegenheit, erstens TypeScript in unsere Entwicklungspraxis zu integrieren. Die Entscheidung fiel auf TypeScript, um die Robustheit und Wartbarkeit unserer Anwendung zu verbessern. Zweitens haben wir uns für den Einsatz einer NoSQL-Datenbank entschieden, um die Flexibilität und Skalierbarkeit unserer Datenspeicherung zu erweitern. Diese Wahl ermöglicht es uns, komplexe Datenstrukturen effizient zu handhaben und zu verwalten. Schließlich haben wir einen Reverse-Proxy in unsere Architektur integriert, um den Datenverkehr zu steuern und eine effiziente Verteilung zwischen verschiedenen Anwendungskomponenten zu gewährleisten.

Im Frontend, kommen die folgenden Technologien zum Einsatz:

- **Lit Element**: Ein modernes, leichtgewichtiges JavaScript-Framework zur Erstellung von Webkomponenten, das auf Webstandards wie Custom Elements, Shadow DOM und ES Modules basiert.

- **Node.js**: Eine JavaScript-Laufzeitumgebung, die es ermöglicht, JavaScript-Code außerhalb eines Browsers auszuführen.
- **TypeScript**: Eine typisierte Erweiterung von JavaScript, die die Entwicklung robuster und wartbarer Anwendungen ermöglicht.

Im Backend, das als Server fungiert, werden die folgenden Technologien verwendet:

- **Bun**: Als Paketmanager und JavaScript-Laufzeitumgebung, eine Alternative zu Node.js.
- **Express**: Ein beliebtes Webanwendungs-Framework für Node.js, das zur Erstellung von robusten und skalierbaren Serveranwendungen verwendet wird.
- **TypeScript**: Auch hier wird TypeScript eingesetzt, um die Serverlogik zu schreiben und die Vorteile von Typsicherheit zu nutzen.

Als Datenbank wird **MongoDB** verwendet, eine dokumentenorientierte NoSQL-Datenbank, die Flexibilität und Skalierbarkeit für die Speicherung und Verwaltung von Daten bietet.

Ein **Nginx**-Proxy wird eingesetzt, um den Datenverkehr zu steuern. Er routet den Datenverkehr so, dass Anfragen an die API auf den Backend-Container und Anfragen an die Hauptanwendung auf den Frontend-Container weitergeleitet werden. Dies ermöglicht eine effiziente Verteilung des Datenverkehrs zwischen den verschiedenen Teilen der Anwendung.

Server

Dateibeschreibung

Hier nochmal die Dateistruktur von /server

```
/server
├── bun.lockb
├── index.ts
├── package.json
├── package-lock.json
├── src
│   ├── auth-controller.ts
│   ├── data-access.ts
│   ├── interfaces
│   │   └── ticket-interface.ts
│   └── ticket-controller.ts
└── tsconfig.json
```

Der Server ist in Node.js geschrieben und verwendet Express.js als Webframework. Die Anwendung kommuniziert mit einer MongoDB-Datenbank, um Tickets und Benutzerinformationen zu speichern. Hier ist eine Übersicht über die wichtigsten Dateien im Server:

1. **ticket-controller.ts:** Diese Datei enthält den Controller für die Verwaltung von Ticketoperationen, wie das Abrufen einzelner Tickets, das Abrufen einer Liste von Tickets und das Erstellen neuer Tickets.
2. **auth-controller.ts:** Diese Datei enthält den Controller für die Authentifizierung von Benutzern. Sie ermöglicht die Registrierung und Anmeldung von Benutzern und die Generierung von Authentifizierungstoken.
3. **data-access.ts:** Dieses Modul stellt den Zugriff auf die MongoDB-Datenbank her. Es initialisiert die Verbindung zur Datenbank und ermöglicht den Zugriff auf verschiedene Sammlungen.
4. **interfaces/ticket-interface.ts:** Definiert das Format eines Tickets in der Anwendung.
5. **index.ts:** Der Hauptserverdatei, die die Routen definiert, auf Anfragen reagiert und die entsprechenden Controller verwendet, um die Anfragen zu verarbeiten.

Client

Lit-Element

Lit Element ist ein modernes, leichtes JavaScript-Framework, das von den Entwicklern des Polymer-Projekts entwickelt wurde. Es basiert auf den Webstandards Custom Elements v1, Shadow DOM v1 und ES Modules und bietet eine effiziente Möglichkeit, Webkomponenten zu erstellen. Lit Element bietet eine einfachere Syntax im Vergleich zu den traditionellen Webkomponenten-Methoden und ermöglicht Entwicklern, wiederverwendbare benutzerdefinierte Elemente mit minimaler Boilerplate-Code zu erstellen.

Es erleichtert die Erstellung von Komponenten, indem es eine intuitive Kombination aus JavaScript (bzw. bei uns Typescript) und HTML-Code ermöglicht. Mit Lit Element können Entwickler dynamische, reaktive Benutzeroberflächen erstellen und gleichzeitig von den Vorteilen des Shadow-DOMs profitieren, um die Komponentenisolierung zu gewährleisten.

Dateibeschreibung

Hier nochmal die Dateistruktur von /app

```
/app
├── index.html
├── package.json
├── package-lock.json
├── src
│   ├── components
│   │   ├── loginform-component.ts
│   │   ├── navbar-component.ts
│   │   ├── ticket-detail-component.ts
│   │   ├── ticket-form-component.ts
│   │   └── ticket-list-component.ts
│   ├── contexts
│   │   └── auth-context.ts
│   ├── interfaces
│   │   ├── session-interface.ts
│   │   └── ticket-interface.ts
│   └── pages
│       ├── index.ts
│       ├── login-page.ts
│       ├── profile-page.ts
│       ├── signup-page.ts
│       └── tickets-page.ts
└── tsconfig.json
```

Der Client-Code enthält verschiedene Seiten und Komponenten, die für die Benutzeroberfläche der Anwendung verantwortlich sind. Hier sind die wichtigsten Dateien im Client:

1. **navbar-component.ts:** Stellt die Navigationsleiste der Anwendung dar und ermöglicht die Navigation zwischen verschiedenen Seiten.
2. **ticket-detail-component.ts:** Zeigt die Details eines ausgewählten Tickets an.
3. **ticket-form-component.ts:** Bietet ein Formular für die Erstellung neuer Tickets.
4. **ticket-list-component.ts:** Listet alle verfügbaren Tickets auf und ermöglicht die Auswahl eines bestimmten Tickets.
5. **loginform-component.ts:** Stellt das Anmeldeformular der Anwendung dar.
6. **index.ts:** Die Hauptdatei, die verschiedene Seiten zusammenführt und die Navigation und Authentifizierung verwaltet.
7. **signup-page.ts:** Enthält ein Formular für die Registrierung neuer Benutzer.
8. **login-page.ts:** Enthält ein Formular für die Anmeldung von Benutzern.

9. **profile-page.ts:** Eine einfache Profilseite.
10. **tickets-page.ts:** Eine Seite, die Tickets auflistet und die Funktionen zum Erstellen und Anzeigen von Ticketdetails bereitstellt.

Installation und Ausführung mit Docker Compose

Die WebSec-Anwendung kann mithilfe von Docker Compose gestartet werden. Stellen Sie sicher, dass Docker und Docker Compose auf Ihrem System installiert sind. Führen Sie die folgenden Schritte aus, um die Anwendung zu starten:

1. Stellen Sie sicher, dass Sie Zugriff auf das Docker-Compose-Konfigurationsdatei haben, in der Regel `docker-compose.yml`.
2. Navigieren Sie mit der Befehlszeile in das Verzeichnis, das die `docker-compose.yml`-Datei enthält.
3. Führen Sie den folgenden Befehl aus, um die Anwendung zu starten:

```
$ docker-compose up -d
```

Dieser Befehl wird den Build-Prozess für die Anwendung starten und die Container entsprechend der Konfiguration ausführen. Die Anwendung wird dann auf den angegebenen Ports zugänglich sein.

Datenbank

MongoDB ist eine dokumentenorientierte NoSQL-Datenbank, die auf Flexibilität und Skalierbarkeit ausgelegt ist. Sie verwendet das Konzept von Sammlungen und Dokumenten anstelle von Tabellen und Zeilen, wie es bei relationalen Datenbanken der Fall ist. In diesem WebSec-Projekt wird MongoDB als Backend-Datenbank verwendet, um Benutzer-, Ticket- und Sitzungsdaten zu speichern und zu verwalten.

Die MongoDB-Datenbank für das WebSec-Projekt enthält drei verschiedene Sammlungen:

1. **Benutzer (Users)**: Diese Sammlung speichert Informationen über die registrierten Benutzer, einschließlich ihrer Benutzernamen, Passwörter und anderer relevanter Informationen.
2. **Tickets**: In dieser Sammlung werden Daten zu den erstellten Tickets gespeichert. Dies umfasst Informationen wie den Tickettitel, den Verfasser, das Erstellungsdatum, den Inhalt des Tickets und den aktuellen Status des Tickets.
3. **Sitzungen (Sessions)**: Diese Sammlung dient zur Verwaltung von Benutzersitzungen und authentifizierten Sitzungsdaten. Hier werden Sitzungsinformationen wie Sitzungs-UUIDs, Authentifizierungsstatus und zugehörige Benutzerinformationen gespeichert.

Durch die Verwendung von MongoDB können Daten in einem flexiblen und skalierbaren Format gespeichert werden, was eine einfache Handhabung und Speicherung von komplexen Datenstrukturen ermöglicht. MongoDB ermöglicht zudem eine schnelle und effiziente Verarbeitung großer Datenmengen, was besonders wichtig ist, wenn eine Anwendung eine Vielzahl von Benutzern und Daten verwalten muss.