TERMINKALENDER

Aron Rama, Resul Skenderi

Abschlussarbeit

Modul 183 - Applikationssicherheit implementieren

Sicherheitsanalyse nach OWASP für das Projekt *Terminkalender (Raum-Reservationssystem)*

1. Broken Access Control (OWASP A01:2021)

Beschreibung:
Ohne konsequente Prüfung des privaten Schlüssels im Backend könnten unberechtigte Nutzer auf geschützte Reservierungen zugreifen.

Lösung:

- Im Backend bei jedem Bearbeiten/Löschen den privaten Schlüssel prüfen
- Sicherheitslogik nie ins Frontend auslagern

2. Injection (A03:2021)

Beschreibung:
Ungefilterte Eingaben wie Bemerkungen oder Teilnehmer können zu Angriffen wie XSS oder SQL-Injection führen.

Lösung:

- Eingaben validieren (z. B. mit @Size)
- SQL nur über JPA/Repository
- Keine HTML-Ausgabe ungefilterter Eingaben

3. Security Misconfiguration (A05:2021)

Beschreibung:
Fehlkonfigurationen (z. B. aktivierte H2-Konsole, keine HTTPS-Weiterleitung) können Angriffe erleichtern.

Lösung:

- Dev-Tools im Livebetrieb deaktivieren
- HTTPS erzwingen
- Sicherheits-Header aktivieren

4. Cryptographic Failures (A02:2021)

Beschreibung:
Schlüssel dürfen nicht vorhersagbar oder unsicher übertragen werden (z. B. per GET-Parameter oder ohne HTTPS).

Lösung:

- Sicher generierte UUIDs (UUID.randomUUID())
- Schlüssel nur über POST oder sichere Kanäle senden
- HTTPS aktivieren

5. Insecure Design (A04:2021)

Beschreibung:
Ohne Benutzerkonten fehlen Authentifizierung, Logging und Schutz vor Missbrauch.

Lösung:

- Rate Limiting und IP-Logging einführen
- Reservierungen mit Ablaufdatum
- Optional: Authentifizierung für Bearbeitungen

Modul 223 - Multi-User-Applikationen objektorientiert realisieren

1. Projektübersicht

Projektname: Terminkalender (Reservationssystem für Räume)

Modul: Multiuser-Applikationen objektorientiert realisieren (Modul 223)

Projektziel:
Entwicklung einer Webanwendung zur Verwaltung von Raumreservierungen in einem Unternehmensgebäude. Die Anwendung erlaubt die Erfassung, Verwaltung, Anzeige und Bearbeitung von Reservierungen ohne Benutzerkonto.

2. Ausgangslage und Anforderungen

- **Räume:** Mehrere Räume (z.B. 101, 102, 103, 104, 105) stehen zur Verfügung und können für Sitzungen und Veranstaltungen reserviert werden.
- **Reservierungen:** Ein Benutzer kann Reservierungen für einen Raum mit folgenden Daten anlegen:
    - Datum (muss in der Zukunft liegen)
    - Startzeit und Endzeit (sinnvolle Reihenfolge, Start vor Ende)
    - Zimmernummer (int)
    - Bemerkung (10-200 Zeichen)
    - Liste von Teilnehmern (Vor- und Nachname, getrennt durch Komma)
- **Verfügbarkeitsprüfung:**
    - Vor Erstellen einer Reservierung wird geprüft, ob der Raum zu dem gewünschten Zeitpunkt noch frei ist.
- **Schlüssel:**
    - Nach der Reservierung erhält der Benutzer zwei Schlüssel:
        - **Privater Schlüssel:** Ermöglicht Bearbeiten und Löschen der Reservierung.
        - **Öffentlicher Schlüssel:** Dient zur Einsicht (z.B. Einladung an andere Teilnehmer).
- **Startseite:**
    - Möglichkeit, Schlüssel einzugeben, um eine Reservierung zu bearbeiten oder einzusehen.
- **Liste:**
    - Übersicht aller gespeicherten Reservierungen.
- **Datenhaltung:**
    - Alle Reservierungen werden permanent in einer MySQL-Datenbank gespeichert.

3. Architektur und Technologien

- **Backend:**
    - Spring Boot (Version 3.3.4) mit Maven
    - Java 17
    - Spring MVC für Web-Controller
    - Spring Data JPA zur Datenbankanbindung
    - MySQL-Datenbank
- **Frontend:**
    - Thymeleaf als Template-Engine für HTML-Seiten
    - CSS für grundlegendes Styling
- **Build-Tool:**
    - Maven
- **IDE:**
    - IntelliJ IDEA Ultimate (oder Spring Initializr für Projektsetup)

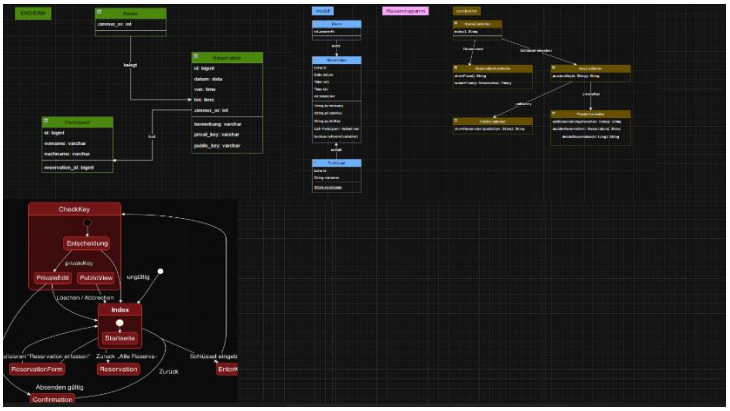
---

4. Projektstruktur

- src/main/java/ch/example/terminkalender/model
  Enthält die Datenmodelle (Reservation, Participant).
- src/main/java/ch/example/terminkalender/repository
  Repository-Schnittstellen für Datenzugriff (ReservationRepository).
- src/main/java/ch/example/terminkalender/controller
  Controller-Klassen zur Steuerung der Webanwendung (MainController, ReservationController).
- src/main/resources/templates
  Thymeleaf-Templates (index.html, create.html, list.html, edit.html, view.html, confirmation.html).
- src/main/resources/static/css
  CSS-Dateien für das Frontend.
- src/main/resources/application.properties
  Konfiguration der Datenbank und Spring Boot.
- src/main/Doku
  Projektdokumentation, UML-Diagramme, ERM, SQL-Skripte etc.

5. UML-Diagramme und ERM (Kurzfassung)

5.1 Klassendiagramm



6. Wichtige Implementierungspunkte

6.1 Validierung im Backend

- Datum muss in der Zukunft liegen.
- Startzeit vor Endzeit.
- Bemerkung zwischen 10 und 200 Zeichen.
- Raum darf in der angegebenen Zeit nicht schon reserviert sein.

6.2 Schlüsselverwaltung

- Bei Anlage einer Reservierung werden zwei UUIDs generiert (privater und öffentlicher Schlüssel).
- Schlüssel erlauben Bearbeitung oder Einsicht in die Reservierung.

6.3 Fehlerbehandlung

- Im Fehlerfall (z.B. ungültige Daten, Raum belegt) wird die create-Seite mit Fehlermeldung neu angezeigt.
- Fehler werden in Thymeleaf-Templates mit ${error} dargestellt.

---

7. Bedienung der Webanwendung

Starten des Projekts

bashKopierencd C:\Users\aronr\Desktop\terminkalender\terminkalender
mvn spring-boot:run

Zugriff im Browser

- Hauptseite: http://localhost:8080/
- Reservierung erfassen: http://localhost:8080/create
- Reservierungsliste: http://localhost:8080/list

Funktionen

- **Reservation erfassen:** Formular ausfüllen und absenden.
- **Reservation bearbeiten:** Privater Schlüssel auf Startseite eingeben.
- **Reservation einsehen:** Öffentlicher Schlüssel auf Startseite eingeben.
- **Reservation löschen:** Über Bearbeitungsseite möglich.

8. Zusammenfassung

Das Projekt "Terminkalender" ist eine einfache, aber funktionale Webapplikation zur Verwaltung von Raumreservierungen. Es verbindet ein modernes Spring Boot Backend mit Thymeleaf-basiertem Frontend. Wesentliche Funktionalitäten wie Validierung, Verfügbarkeit, Schlüsselverwaltung und CRUD (Create, Read, Update, Delete) sind umgesetzt. Die Verwendung von UUID-Schlüsseln für privaten und öffentlichen Zugriff gewährleistet Sicherheit und Nutzerkomfort ohne Benutzerkonten.

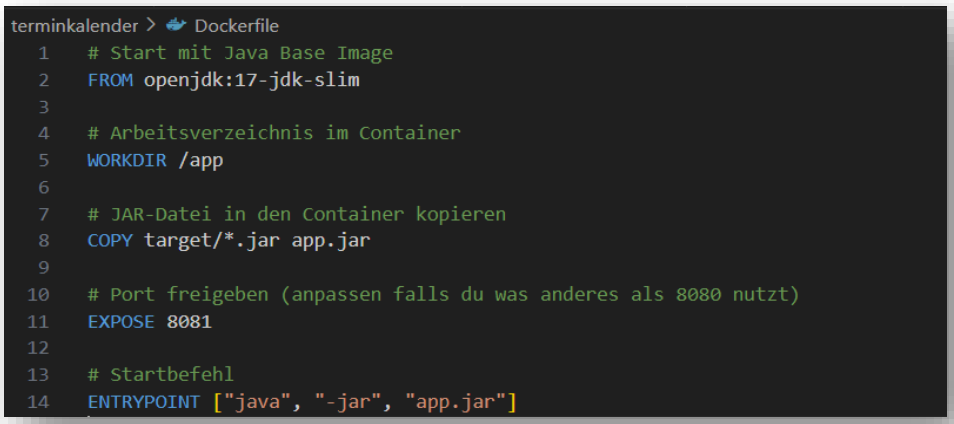
Modul 347 - Dienst mit Container anwenden

Ziel

Das Ziel der Containerisierung war es, die Anwendung unabhängig von der lokalen Entwicklungsumgebung lauffähig zu machen. Dadurch kann der Terminkalender auf jedem System (mit Docker) einheitlich gestartet werden, egal ob lokal, auf einem Server oder in der Cloud.

Verwendete Technologien

Technologie	Zweck
Docker	Zum Erstellen eines Containers für die Java-Anwendung
Dockerfile	Beschreibt, wie das Image gebaut wird
docker-compose	Startet Container ggf. gemeinsam mit Datenbank etc.



Das Dockerfile sorgt dafür, dass unsere Java Spring Boot Anwendung (Terminkalender) in einem isolierten Container läuft. Es verwendet ein leichtgewichtiges Java-Image, kopiert die .jar-Datei ins Container-Verzeichnis und startet sie automatisch. Durch das Freigeben des Ports 8081 wird die Anwendung von aussen erreichbar gemacht. So ist unsere Anwendung unabhängig von der lokalen Umgebung und überall einsetzbar ideal für Entwicklung, Test und Produktion.

Modul 450 - Applikationen testen

Testfall ID	Testbeschreibung	Testmethode	Erwartetes Ergebnis
T01	Reservierung mit gültigen Daten erfassen	Systemtest / Manueller Test	Reservierung wird erfolgreich erstellt, Bestätigungsseite mit Schlüsseln wird angezeigt
T02	Reservierung mit Datum in der Vergangenheit	Validierungstest / Unit-Test	Fehlermeldung: „Das Datum muss in der Zukunft liegen“
T03	Reservierung mit Startzeit nach Endzeit	Validierungstest / Unit-Test	Fehlermeldung: „Startzeit muss vor Endzeit liegen“
T04	Reservierung mit leerer oder zu kurzer Bemerkung	Validierungstest / Unit-Test	Fehlermeldung: „Bemerkung muss zwischen 10 und 200 Zeichen lang sein“
T05	Reservierung mit bereits belegtem Raum	Integrationstest	Fehlermeldung: „Raum ist zu diesem Zeitpunkt bereits reserviert“
T06	Bearbeitung einer Reservierung mit gültigem privatem Schlüssel	Systemtest / Manueller Test	Änderungen werden gespeichert und bestätigt
T07	Bearbeitung mit ungültigem privatem Schlüssel	Systemtest / Manueller Test	Fehlermeldung: „Reservierung nicht gefunden oder Schlüssel ungültig“
T08	Einsicht mit gültigem öffentlichem Schlüssel	Systemtest / Manueller Test	Reservierung wird im Lesemodus korrekt angezeigt
T09	Einsicht mit ungültigem öffentlichem Schlüssel	Systemtest / Manueller Test	Fehlermeldung: „Reservierung nicht gefunden oder Schlüssel ungültig“
T10	Löschen einer Reservierung über Bearbeitungsseite	Systemtest / Manueller Test	Reservierung wird gelöscht, Bestätigungsseite wird angezeigt
T11	Aufruf der Reservierungsliste (/list)	Integrationstest	Alle bestehenden Reservierungen werden korrekt aufgelistet
T12	Überprüfung der Datenpersistenz nach Serverneustart	Systemtest	Reservierungen bleiben gespeichert und werden korrekt angezeigt
T14	Eingabe einer korrekten Teilnehmerliste	Systemtest	Teilnehmer werden korrekt gespeichert und dargestellt

Testfall ID	Testbeschreibung	Testmethode	Erwartetes Ergebnis
T15	Eingabe einer leeren Teilnehmerliste	Systemtest	Reservierung wird gespeichert – Teilnehmerfeld ist optional

Legende:

- **Unit-Test**: Testet einzelne Methoden/Funktionen (z. B. Validierung von Eingabedaten).
- **Integrationstest**: Testet Zusammenspiel mehrerer Komponenten (z. B. Datenbank + Backend).
- **Systemtest**: Testet das Gesamtsystem über alle Schichten hinweg.
- **Manueller Test**: Durchführung über die Benutzeroberfläche (Browser, GUI).
- **Sicherheitstest**: Test auf Sicherheitslücken wie HTML-Injection oder unbefugten Zugriff.