# Designdokument:

* **Modellierung der Kernstrukturen der Applikation:**

Modellieren Sie wichtige wiederkehrende Strukturen der Applikation in Form eines Klassendiagramms.

Wenn Sie ein Framework einsetzen, dokumentieren Sie das Zusammenspiel und die Integration des Frameworks in die Applikation.

TUTOR FRAGEN

* **Eingesetzte Patterns:**

Welche Patterns werden wie innerhalb der Applikation eingesetzt?

Dependency Injection durch Spring Boot. Es ist aufgebaut als 3 Schichten Architektur.

* **Domänenmodell:**

Das Domänenmodell umfasst die Klassen und Objekte der Domäne, jedoch keine technischen Klassen wie DAOs oder UI-Klassen. Es kann in Form eines Klassen- oder **EER**-Diagramms modelliert werden.

* **Exception-Handling:**

Beschreiben Sie das Exception-Handling-Konzept der Applikation: Welche Arten von Exceptions werden eingesetzt (Checked oder Runtime Exceptions)? Welche applikationsspezifische Exceptions werden verwendet?

Daten werden überprüft im Client und auch Server. Exceptions im Server werden geworfen zum Client, der fängt die ab und zeigt dem User eine Meldung.

Wir verwenden Runtime Exceptions für alle Exceptions mit http Status.

Alle anderen Exceptions sind Checked Exceptions.

Welche applikationsspezifische Exceptions werden verwendet? – noch zu machen

* **REST Spezifikation:**

Um eine schön aufbereitete Dokumentation der REST Schnittstelle mittels Swagger zu bekommen, muss zuerst der Server gestartet werden und danach findet man diese unter: <http://localhost:8080/swagger-ui.html>

* + authentication-endpoint : **Authentication Endpoint**
    - *GET /authentication*: Get some valid authentication tokens

Methode die aufgerufen wird:

1. **public** AuthenticationToken authenticate(
2. @ApiIgnore @RequestHeader(value = HttpHeaders.AUTHORIZATION) String authorizationHeader)
   * + *POST /authentication*: Get an authentication token with your username and password

Methode die aufgerufen wird:

1. **public** AuthenticationToken authenticate(
2. @RequestBody **final** AuthenticationRequest authenticationRequest)
   * + *GET /authentication/info*: Get information about the current users authentication token

Methode die aufgerufen wird:

1. **public** AuthenticationTokenInfo tokenInfoCurrent(
2. @ApiIgnore @RequestHeader(value = HttpHeaders.AUTHORIZATION) String authorizationHeader)
   * + *GET /authentication/info/{token}* : Get information about a specific authentication token

Methode die aufgerufen wird:

**public** AuthenticationTokenInfo tokenInfoAny(@PathVariable String token)

* + customer-endpoint : **Customer Endpoint**
    - *GET /customer*: Get list of customer

Methode die aufgerufen wird:

**public** List<CustomerDTO> findAll(Pageable pageable)

* + - *POST /customer*: Save a customer

Methode die aufgerufen wird:

**public** CustomerDTO saveCustomer(@RequestBody CustomerDTO customerDTO)

* + - *GET /customer/search*: Get searched customers

Methode die aufgerufen wird:

**public** List<CustomerDTO> search(@RequestParam(value = "query") String query, Pageable pageable)

* + - *GET /customer/{id} :* Get one Customer by id

Methode die aufgerufen wird:

**public** CustomerDTO find(@PathVariable UUID id)

* + event-endpoint : **Event Endpoint**
    - *GET /event*: Gets a list of all Events

Methode die aufgerufen wird:

**public** List<EventDTO> findAll(Pageable pageable)

* + news-endpoint : **News Endpoint**
    - *GET /news*: Get list of simple news entries

Methode die aufgerufen wird:

**public** List<SimpleNewsDTO> findAll()

* + - *POST /news*: Publish a new news entry

Methode die aufgerufen wird:

**public** DetailedNewsDTO publishNews(@RequestBody DetailedNewsDTO detailedNewsDTO)

* + - *GET /news/{id}* : Get detailed information about a specific news entry

Methode die aufgerufen wird:

**public** DetailedNewsDTO find(@PathVariable UUID id)

* + performance-endpoint : **Performance Endpoint**
    - *GET /performance/{id}* : Get detailed information about a specific performance

Methode die aufgerufen wird:

**public** PerformanceDTO findOne(@PathVariable UUID id)

* + ticket-transaction-endpoint : **Ticket Transaction Endpoint**
    - *GET /tickettransaction*: Gets a list of bought and reserved Ticket Reservations

Methode die aufgerufen wird:

1. **public** List<DetailedTicketTransactionDTO> getAllReservedAndBoughtTransactions(
2. Pageable pageable)
   * + *PATCH /tickettransaction*: Updates a single Ticket Transaction

Methode die aufgerufen wird:

1. **public** DetailedTicketTransactionDTO patchTicketTransaction(
2. @RequestBody DetailedTicketTransactionDTO dto)
   * + *GET /tickettransaction/filter*: Gets a list of Ticket Reservations for the customer and the performance name

Methode die aufgerufen wird:

1. **public** List<DetailedTicketTransactionDTO> findTicketTransaction(
2. @RequestParam(value = "firstname") String customerFirstName,
3. @RequestParam(value = "lastname") String customerLastName,
4. @RequestParam(value = "performance") String performance,
5. Pageable pageable
6. )
   * + *GET /tickettransaction/find*: Get a list of transactions with the id

Methode die aufgerufen wird:

1. **public** List<DetailedTicketTransactionDTO> findTicketTransactionsByID(
2. @RequestParam(value = "id") String id, Pageable pageable)
   * + *GET /tickettransaction/{id}* : Get one Ticket Transaction by ID

Methode die aufgerufen wird:

**public** DetailedTicketTransactionDTO findTicketTransactionByID(@PathVariable UUID id)

* **Logging:**

Error Logging im *error channel*

Information Logging im i*nfo channel* (Meldungen wie z.B: User A kauft Ticket XYZ)

Debug Logging im *debug channel* (nur zu Debugzwecken für größere Features im Zusammenspiel oder als temporäre Ausgabe gedacht)

* **Security-Aspekte:**
  + SQL-Injection

Hibernate übernimmt diese Aufgabe.

* + Berechtigungen

JPA übernimmt die Prüfung nach ausreichenden Berechtigungen aufgrund der gesetzten Annotationen.

* + Kryptographie

Passwörter werden gehasht in der Datenbank gespeichert. **Zahlungsinformationen werden verschlüsselt. – oder gar nicht gespeichert?**

“Weder Passwörter noch **Zahlungsinformationen** dürfen im Klartext gespeichert werden.”