PRODYNA
information architects

Architektur
Aircraft Reservation System (ARS)

Inhaltsverzeichnis

1 Technologien und Laufzeitumgebung

1.1 Frontend

- Android
- Apache HttpClient
- GSON

1.2 Backend

- JavaEE 7
    - EJB
    - JPA (Hibernate)
    - JAX-RS (RestEasy)
    - CDI
    - BeanValidation
    - JMX
- Wildfly 8
- MySQL 5

- JUnit
- Arquillian

2 Tools

2.1 Entwicklungsumgebung

2.1.1 Frontend
Android Developer Tool

2.1.2 Backend
Eclipse

2.1.3 Preferences
Einhalten folgender Vorgaben:

- Zeichensatz: UTF-8
- Tab-Size: 4 Zeichen
- Zeilenbreite: 120 Zeichen

2.2 Build- und Dependency-Management

2.2.1 Frontend
Ant

2.2.2 Backend
Maven

2.3 Code-Repository

Git

Git-Client: Git für Windows

2.3.1 Frontend
Url: https://github.com/cnicolaus/ars-frontend

2.3.2 Backend:
Url: https://github.com/cnicolaus/ars-backend

2.4 Build-Server

Jenkins

Url …

2.5 Artifact-Repository

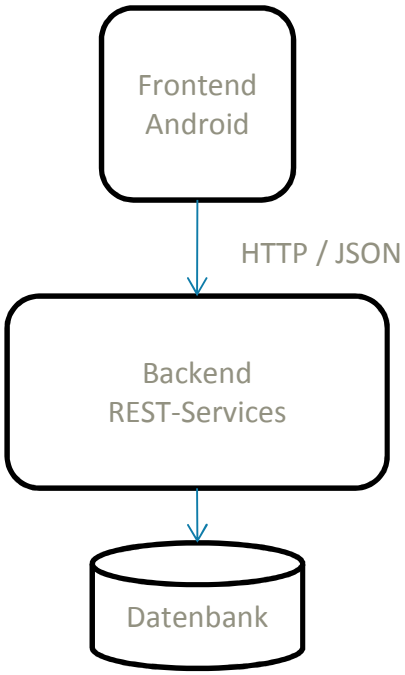
Nexus

Url …

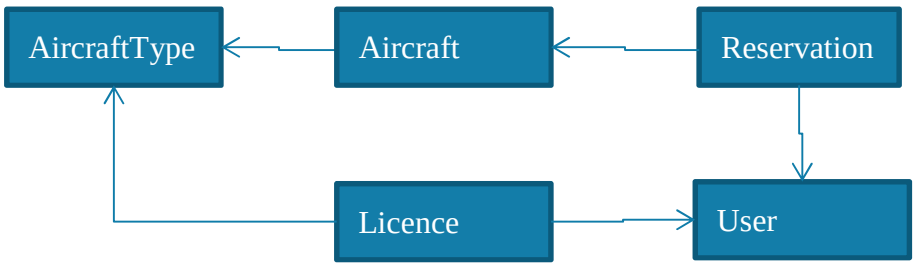
Tools

3 Grundlegende Architektur

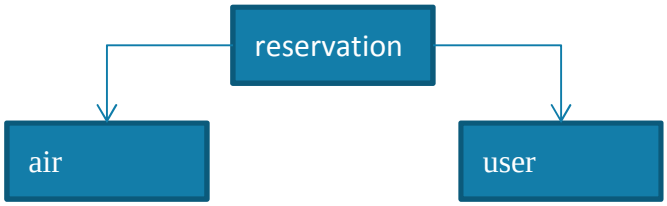
3.1 Übersicht



3.2 Domain-Model



3.3 Komponenten



3.4 Security

Alle REST-Services sind geschützt. Für die Authentifizierung wird Basic Authentication verwendet. Die User mit ihrer Rollenzugehörigkeit werden von der Anwendung (Komponente user) verwaltet. Das Frontend muss bei jedem Aufruf eines Services sowohl den Usernamen als auch das Passwort im HTTP-Header senden.
Die Authentifizierung erfolgt durch den ApplicationServer.

4 Projektstruktur Backend

Das Backend besteht aus folgenden Projekten:

- Hauptprojekt ars-backend

- Projekt ars-backend-common
  Beinhaltet zentrale Klassen die ausschließlich im Backend verwendet werden
  z.B.
  - Allgemeine Exceptions
  - Annotations
  - Producer für EntityManager, Logger

- Projekt ars-backend-common-client
  Zentrale Klassen die auch in den öffentlichen Schnittstellen sichtbar sein müssen
  z.B.
  - Allgemeine Exceptions
  - Konstanten

- Projekt ars-backend-assembly
  Projekt für die Erzeugung des Assembly

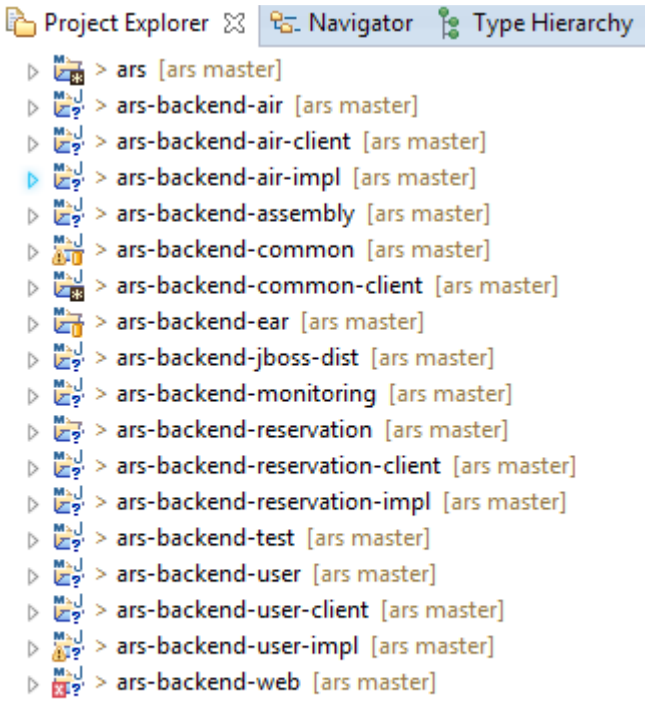
- WEB-Projekt ars-backend-web (Konfiguration JAX-RS)

- EAR-Projekt ars-backend-ear
  Projekt für die Erzeugung der EAR-Datei

- Projekt ars-backend-monitoring
  Beinhaltet alle Klassen für das Monitoring

- Projekt ars-backend-jboss-dist
  Beinhaltet das Datenbank-Schema

- Projekte fachlicher Komponenten (Schnittstellen und Implementierungen)



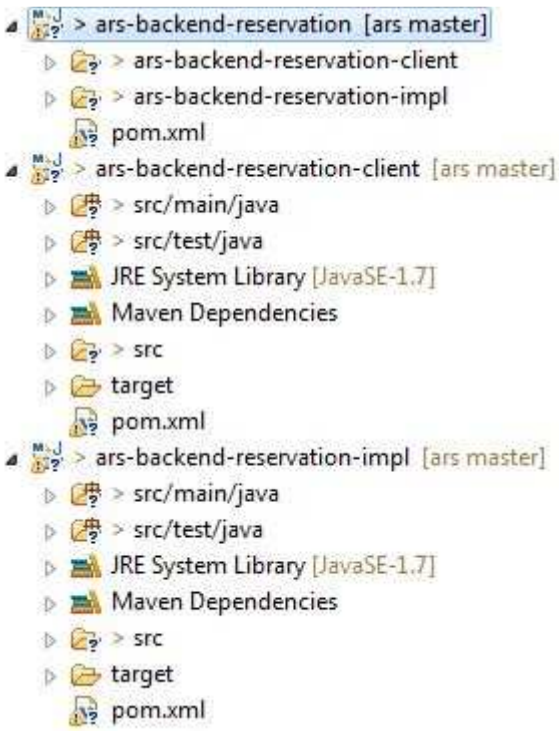
Projektstruktur Backend

5 Aufbau einer fachlichen Komponente

5.1 Projektstruktur

Eine fachliche Komponente besteht aus folgenden Projekten:

- Hauptprojekt ars-backend-<Komponentenname>
- Client-Projekt ars-backend--<Komponentenname>-client
- Implementierung-Projekt ars-backend--<Komponentenname>-impl



5.1.1 Client-Projekt

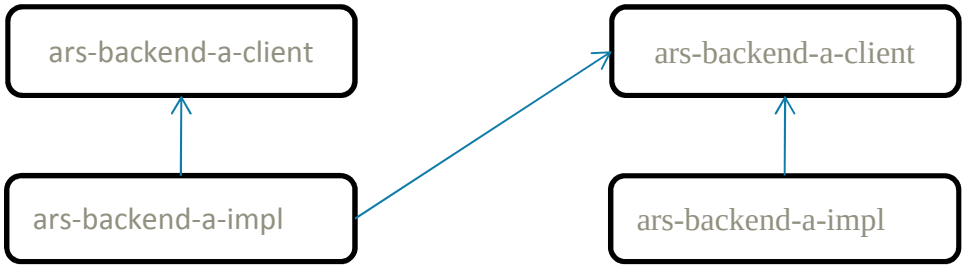
Enthält sämtliche Java-Klassen und Java-Interfaces der Komponenten-Schnittstelle

- Interfaces
- Fachliche Entitäten
- DTOs
- Exceptions

5.1.2 Implementierung-Projekt

Enthält sämtliche Klassen für die Implementierung der Komponenten-Schnittstelle sowie die entsprechenden Testklassen.

5.2 Abhängigkeiten zwischen Komponenten



- Eine Komponente A kennt nur die Schnittstelle einer anderen Komponente B.
- Externe Aufrufe erfolgen über REST.
- Der Aufruf zwischen zwei Komponenten erfolgt direkt.

5.3 Coding

5.3.1 Namenskonventionen

- Komponente: ars-backend-<Komponentenname>
- Package: com.prodyna.pac.ars.backend.<Komponentenname>
- Service-Interface: XXXService
- Service-Klasse <InterfaceName> + Bean
- Methoden
  - create, read, update, delete
  - find, findBy

5.3.2 Service-Interface

```
@Path("aircraft")
@Produces(MediaType.APPLICATION_JSON)
@Consumes(MediaType.APPLICATION_JSON)
public interface AircraftService {

    @PUT
    @Path("/")
    @RolesAllowed({ UserRoleNames.ADMIN })
    public void create(@NotNull @Valid Aircraft aircraft) throws ArsDuplicateEntityException;

    @GET
    @Path("/{id}")
    @RolesAllowed({ UserRoleNames.ADMIN, UserRoleNames.CHARTERER, UserRoleNames.GUEST })
    public Aircraft read(@PathParam("id") @NotNull String id);
```

5.3.2.1 REST-Annotationen

- Verwendung von @Path, @Produces, @Consumes

- Bei CRUD-Services:
  - create: @PUT
  - read: @GET
  - update: @POST
  - delete: @DELETE

5.3.2.2 Validierung

Verwendung von Annotationen des BeanValidation-Frameworks zur Validierung der übergebenen Parameter.

5.3.2.3 Security

Alle Service-Methoden werden mit @RolesAllowed annotiert unter Angabe des Namens der berechtigten Rolle.

5.3.2.4 Exceptions

Exceptions, die der Client gezielt behandeln können muss, werden von der Klasse ArsBusinessException abgeleitet und im Service-Interface bei der Methodendefinition deklariert. ArsBusinessException ist eine checked-Exception.
Für jede definierte Exception-Klasse, die von ArsBusinessException abgeleitet ist, muss ein ExceptionMapper im zugehörigen Implementierungs-Projekt der Fachkomponente implementiert werden, der die Exception auf einen HTTP-Status-Code mapped.
Alle übrigen Exceptions werden von ArsRuntimeException abgeleitet. Sie sind keine checked-Exceptions und müssen nicht auf einen http-Status-Code gemapped werden. Sie werden automatisch auf den Status-Code 500 gemapped.

5.3.3 Service-Klasse

```
@Stateless
@Logged
@MonitoredMethodCall
public class ReservationServiceBean implements ReservationService {

    @Inject
    private Logger logger;

    @Inject
    private EntityManager entityManager;
```

Ein Service wird als Stateless-SessionBean implementiert.

5.3.3.1 Logging

Wird ein Logger benötigt, so wird dieser injected. In Projekt ars-backend-common steht ein entsprechender Producer zur Verfügung.
Damit alle Methodenaufrufe automatisch geloggt werden, wird die Service-Klasse mit @Logged annotiert. Das Logging erfolgt durch einen bereitgestellten Interceptor.

5.3.3.2 EntityManager

Wird ein EntityManager benötigt, so wird dieser injected. In Projekt ars-backend-common steht ein entsprechender Producer zur Verfügung.

5.3.3.3 Monitoring

Zum Erfassen der Statistiken wird die Service-Klasse mit @MonitoredMethodCall annotiert. Das Erfassen der Daten erfolgt durch einen bereitgestellten Interceptor.

5.3.3.4 Aufruf eines anderen Service

```
@Stateless
@MonitoredMethodCall
@Logged
public class ReservationBusinessServiceBean implements ReservationBusinessService {

    @Inject
    private UserService userService;
```

Aufzurufende Services werden injected. Die Implementierung erfolgt ausschließlich gegen das Service-Interface.

6 Test

- Entwicklertests
  - JUnit-Tests
  - JUnit + Arquillian
    - Zu jeder Service-Methode ein Test
    - Hohe Test-Abdeckung
- Push nur erlaubt, wenn alle Test durchlaufen
- Ausführen aller Tests auf dem Build-Server
- Fachliche Test in Integrationsumgebung

7 Release-Management

Derzeit existieren zwei Release-Units

- ars-client
- ars-backend

- Pro Release-Unit ein Repository
- Versionierung einer Release-Unit: MajorRelease.MinorRelease.Bugfix (z.B. 1.1.12)
- Im Normalfall werden Features im Master entwickelt
    - Mit jedem Release wird das MajorRelease und/oder MinorRelease erhöht
    - Zwischen zwei Releases werden beim Build SNAPSHOTS erzeugt
- Bugfixes werden in einem Branch entwickelt
    - Branch-Name: <ReleaseUnit-Name>_<MajorRelease>.<MinorRelease>
    - Mit jedem Bugfix-Release wird <Bugfix> erhöht
    - Bugfixes werden anschließend in den Master gemergt