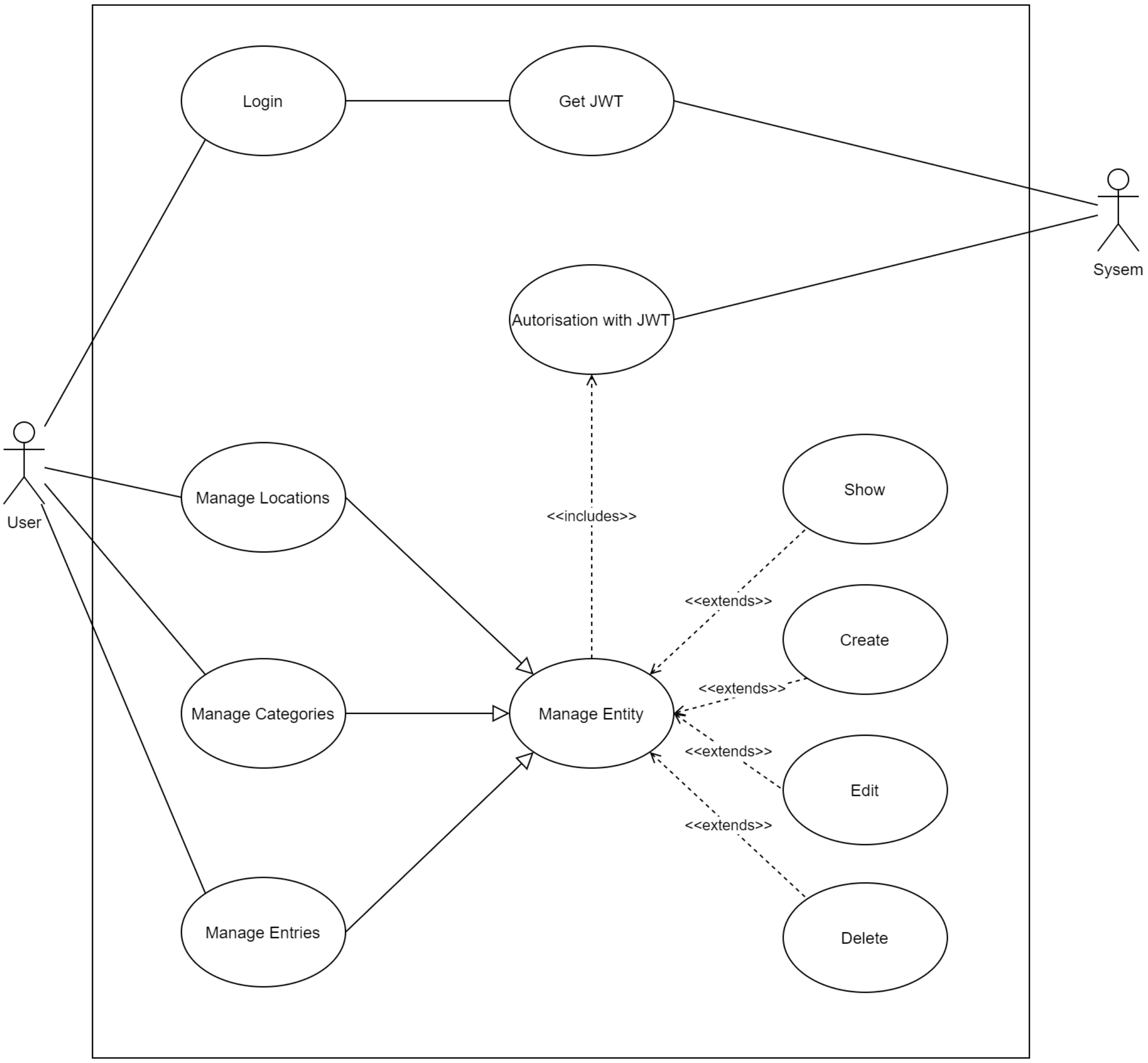
M223 - Planing

This document is the main plan for upcoming changes and contains all necesarry information and diagrams stated in the assignment.

Use-Case Diagram



Login

When the user logs in, the application recieves a `JSON Web Token` (JWT) which it then uses for autorisation on each `REST` request.

Manage Locations, Categories, Entries

These cases all inherit the Manage Entity case. The case is listed below and all `extending` / `including` cases will also be inherited.

Manage Entity

The Manage Entity contains the main cases for inheriting cases. It's extended by the following cases:

- **Show**: Returns all avaliable entities
- **Create**: Creates a new entity
- **Edit**: Edits/Updates the entity
- **Delete**: Deletes the entity

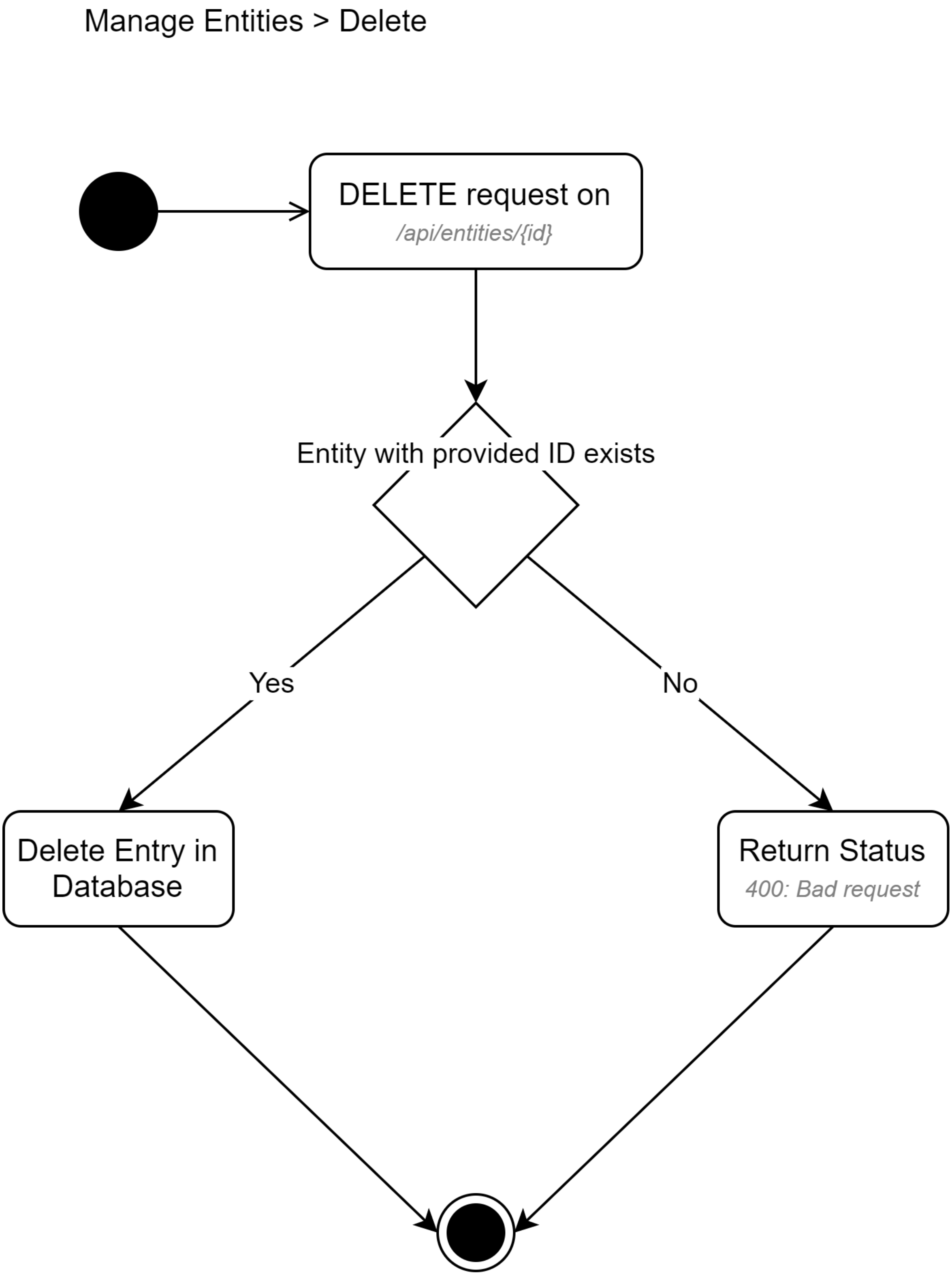
Non-functional requirements

- **Security**: Entities should only be able to be created with a `JWT` token that has the correct autorisation.

- **Data Consistency**: The API should only accept correct and possible data in the correct formats. If the provided data is faulty or incomplete a `401 Bad Request` error should be sent back as response.
- **Performance**: The system should be able to handle multiple requests at a time, so more than one user can use the software simultaneously.
- **Easy-to-use**: The `REST` API should not be overcomplicated and only contain the neccessary interfaces. Difference requests for similar things should be accessible from the same URL. The request URL's should all follow the same style (*naming-convention*).

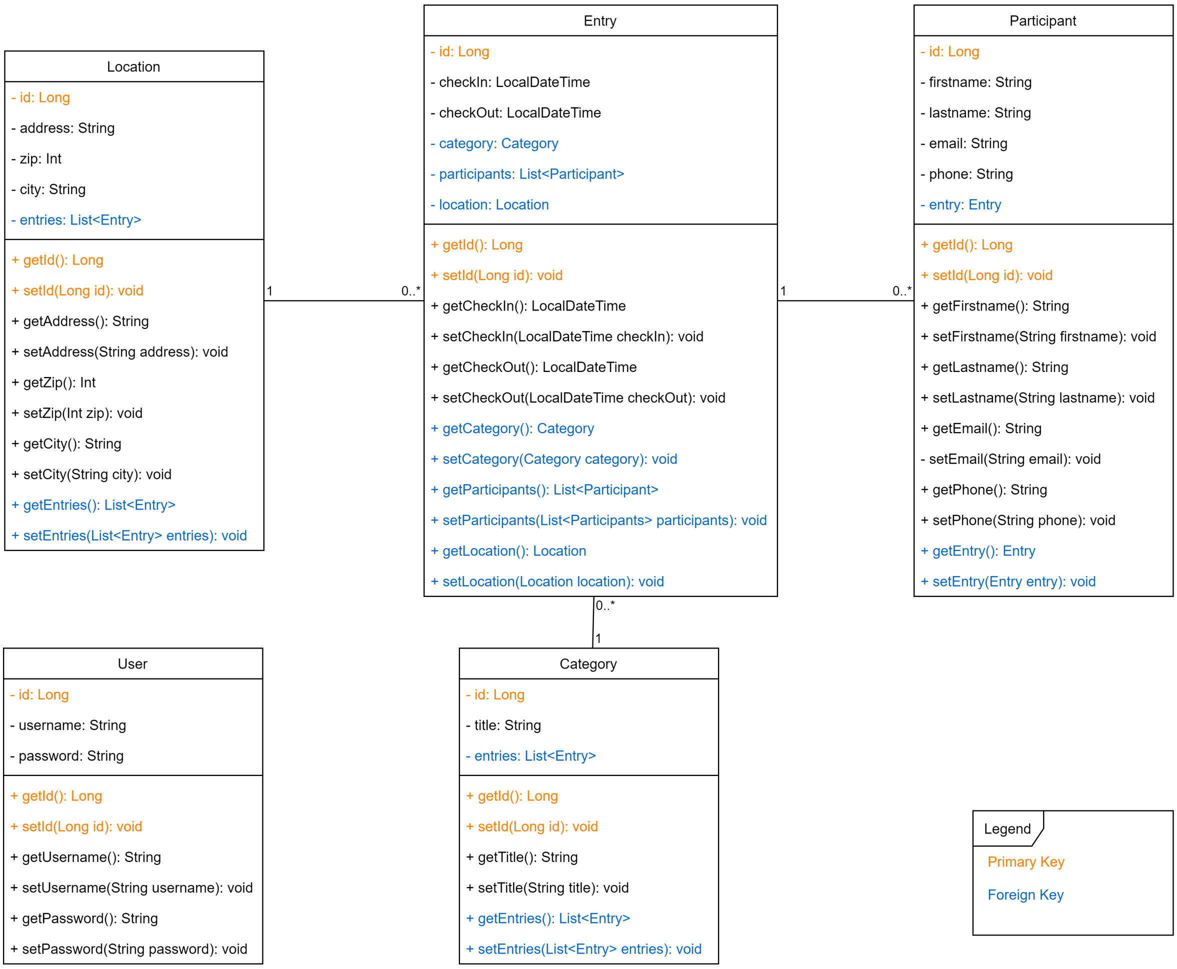
Activity Diagram

Entity Deletion Request

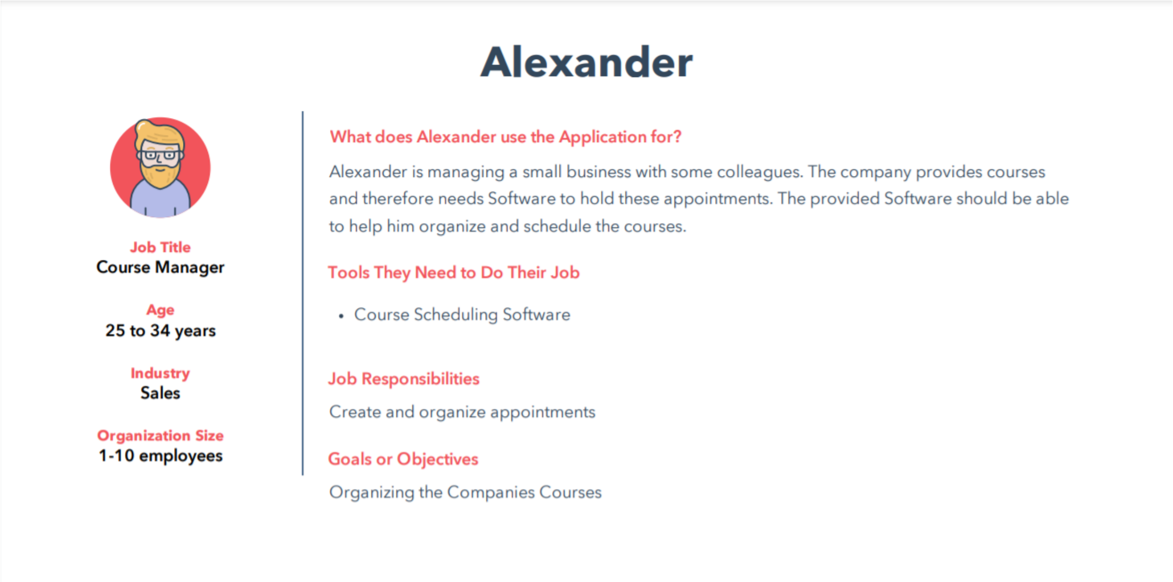


Subject Class Diagram

'Subject class diagram' is a literal translation of the german name 'Fachklassendiagramm'. The difference to a normal class diagram (to my knowledge) is that only model classes are illustrated.


Location
- id: Long
- address: String
- zip: Int
- city: String
- entries: List<Entry>
+ getId(): Long
+ setId(Long id): void
+ getAddress(): String
+ setAddress(String address): void
+ getZip(): Int
+ setZip(Int zip): void
+ getCity(): String
+ setCity(String city): void
+ getEntries(): List<Entry>
+ setEntries(List<Entry> entries): void
1
0..*
Entry
- id: Long
- checkIn: LocalDateTime
- checkOut: LocalDateTime
- category: Category
- participants: List<Participant>
- location: Location
+ getId(): Long
+ setId(Long id): void
+ getCheckIn(): LocalDateTime
+ setCheckIn(LocalDateTime checkIn): void
+ getCheckOut(): LocalDateTime
+ setCheckOut(LocalDateTime checkOut): void
+ getCategory(): Category
+ setCategory(Category category): void
+ getParticipants(): List<Participant>
+ setParticipants(List<Participants> participants): void
+ getLocation(): Location
+ setLocation(Location location): void
1
0..*
Participant
- id: Long
- firstname: String
- lastname: String
- email: String
- phone: String
- entry: Entry
+ getId(): Long
+ setId(Long id): void
+ getFirstname(): String
+ setFirstname(String firstname): void
+ getLastname(): String
+ setLastname(String lastname): void
+ getEmail(): String
- setEmail(String email): void
+ getPhone(): String
+ setPhone(String phone): void
+ getEntry(): Entry
+ setEntry(Entry entry): void
0..*
1
User
- id: Long
- username: String
- password: String
+ getId(): Long
+ setId(Long id): void
+ getUsername(): String
+ setUsername(String username): void
+ getPassword(): String
+ setPassword(String password): void
Category
- id: Long
- title: String
- entries: List<Entry>
+ getId(): Long
+ setId(Long id): void
+ getTitle(): String
+ setTitle(String title): void
+ getEntries(): List<Entry>
+ setEntries(List<Entry> entries): void
Legend
Primary Key
Foreign Key

Persona(s)



Alexander

Job Title
Course Manager

Age
25 to 34 years

Industry
Sales

Organization Size
1-10 employees

What does Alexander use the Application for?

Alexander is managing a small business with some colleagues. The company provides courses and therefore needs Software to hold these appointments. The provided Software should be able to help him organize and schedule the courses.

Tools They Need to Do Their Job

- Course Scheduling Software

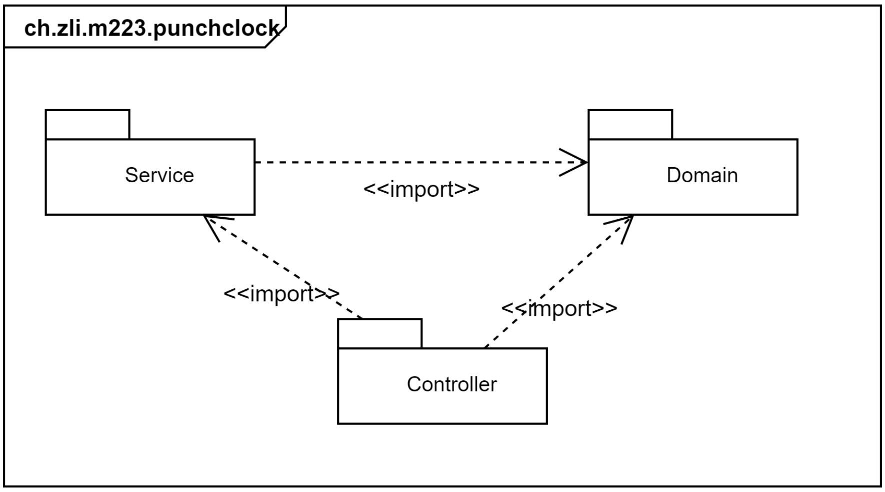
Job Responsibilities

Create and organize appointments

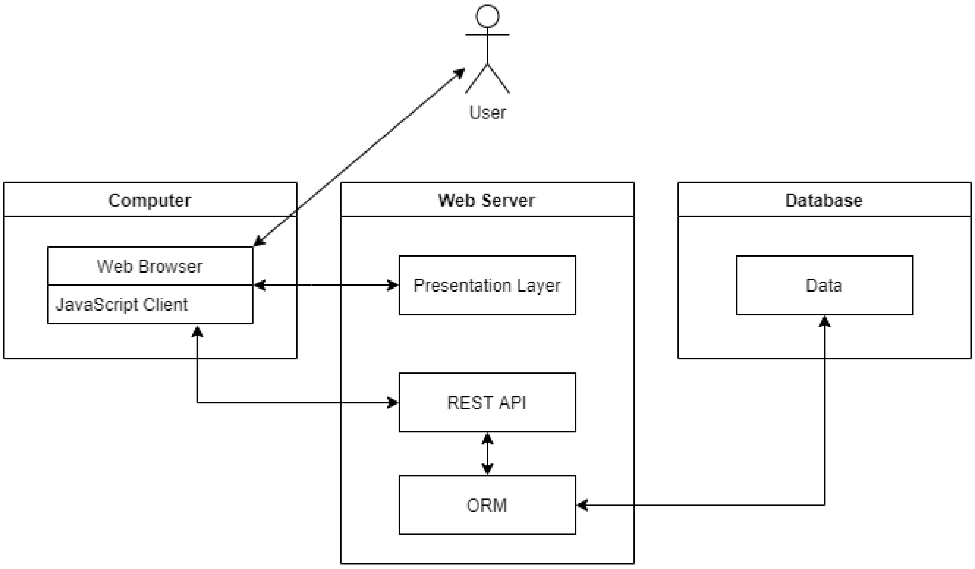
Goals or Objectives

Organizing the Companies Courses

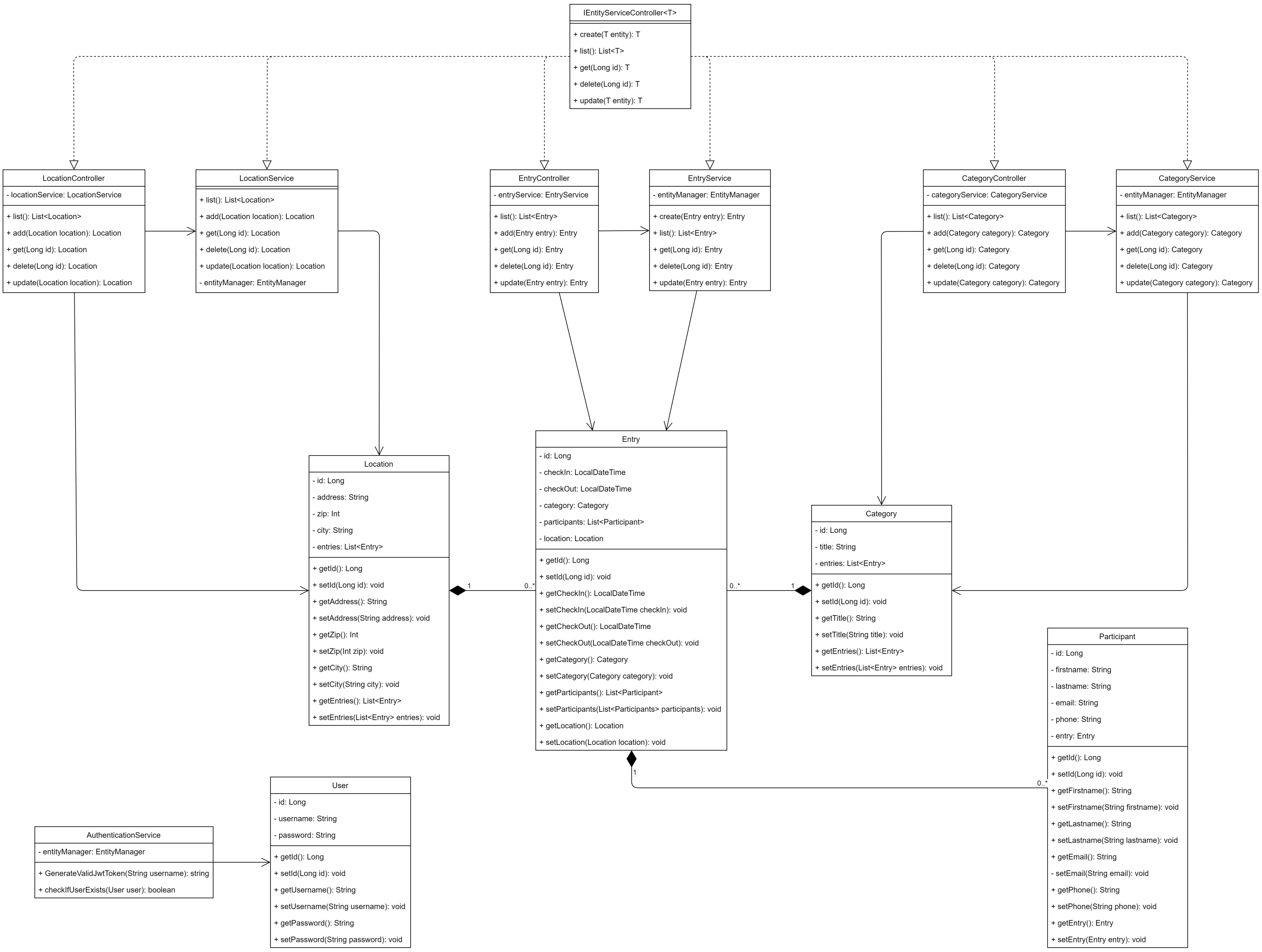
Package Diagram



Deployment Diagram



Class Diagram


IEntityServiceController<T>
+ create(T entity): T
+ list(): List<T>
+ get(Long id): T
+ delete(Long id): T
+ update(T entity): T
LocationController
- locationService: LocationService
+ list(): List<Location>
+ add(Location location): Location
+ get(Long id): Location
+ delete(Long id): Location
+ update(Location location): Location
LocationService
+ list(): List<Location>
+ add(Location location): Location
+ get(Long id): Location
+ delete(Long id): Location
+ update(Location location): Location
- entityManager: EntityManager
EntryController
- entryService: EntryService
+ list(): List<Entry>
+ add(Entry entry): Entry
+ get(Long id): Entry
+ delete(Long id): Entry
+ update(Entry entry): Entry
EntryService
- entityManager: EntityManager
+ create(Entry entry): Entry
+ list(): List<Entry>
+ get(Long id): Entry
+ delete(Long id): Entry
+ update(Entry entry): Entry
CategoryController
- categoryService: CategoryService
+ list(): List<Category>
+ add(Category category): Category
+ get(Long id): Category
+ delete(Long id): Category
+ update(Category category): Category
CategoryService
- entityManager: EntityManager
+ list(): List<Category>
+ add(Category category): Category
+ get(Long id): Category
+ delete(Long id): Category
+ update(Category category): Category
Location
- id: Long
- address: String
- zip: Int
- city: String
- entries: List<Entry>
+ getId(): Long
+ setId(Long id): void
+ getAddress(): String
+ setAddress(String address): void
+ getZip(): Int
+ setZip(Int zip): void
+ getCity(): String
+ setCity(String city): void
+ getEntries(): List<Entry>
+ setEntries(List<Entry> entries): void
1
0..*
Entry
- id: Long
- checkIn: LocalDateTime
- checkOut: LocalDateTime
- category: Category
- participants: List<Participant>
- location: Location
+ getId(): Long
+ setId(Long id): void
+ getCheckIn(): LocalDateTime
+ setCheckIn(LocalDateTime checkIn): void
+ getCheckOut(): LocalDateTime
+ setCheckOut(LocalDateTime checkOut): void
+ getCategory(): Category
+ setCategory(Category category): void
+ getParticipants(): List<Participant>
+ setParticipants(List<Participants> participants): void
+ getLocation(): Location
+ setLocation(Location location): void
0..*
1
Category
- id: Long
- title: String
- entries: List<Entry>
+ getId(): Long
+ setId(Long id): void
+ getTitle(): String
+ setTitle(String title): void
+ getEntries(): List<Entry>
+ setEntries(List<Entry> entries): void
1
0..*
Participant
- id: Long
- firstname: String
- lastname: String
- email: String
- phone: String
- entry: Entry
+ getId(): Long
+ setId(Long id): void
+ getFirstname(): String
+ setFirstname(String firstname): void
+ getLastname(): String
+ setLastname(String lastname): void
+ getEmail(): String
- setEmail(String email): void
+ getPhone(): String
+ setPhone(String phone): void
+ getEntry(): Entry
+ setEntry(Entry entry): void
AuthenticationService
- entityManager: EntityManager
+ GenerateValidJwtToken(String username): string
+ checkIfUserExists(User user): boolean
User
- id: Long
- username: String
- password: String
+ getId(): Long
+ setId(Long id): void
+ getUsername(): String
+ setUsername(String username): void
+ getPassword(): String
+ setPassword(String password): void

Test Cases

Functional

- Entites can be created
- Entites can be deleted
- Categories can be listed
- Categories can be updated
- Locations can be created
- Locations can be updated

Non-Functional

- All API interfaces are following the same naming convention
- Data can only be viewed and edited when logged in
- THe API does not accept wrong/inconsistent data

Source

The repository for this file is on github: github.com/speyck