Advanced Software Engineering
DESIGN REPORT

Team number:	0601

Team member 1	
Name:	David Reichert
Student ID:	00809424
E-mail address:	a00809424@unet.univie.ac.at

Team member 2	
Name:	Fritz Ziernhöld
Student ID:	11709105
E-mail address:	a11709105@unet.univie.ac.at

Team member 3	
Name:	Matthias Fritz
Student ID:	11910902
E-mail address:	a11910902@unet.univie.ac.at

Team member 4	
Name:	Jin Yu
Student ID:	11717460
E-mail address:	a11717460@unet.univie.ac.at

Remark: Unfortunately Philipp Saborosch left on very short notice a few days before without delivering contributions to his SRs (SR4, SR5). As such, these were only included in the global view of the system, and any in depth documentation concerning his SRs is missing.

1. Design Draft

1.1. Design Approach and Overview

For the initial design approach it was most important for us to create a consistent understanding of the system. Main discussion points here were how to store the data across the system and the communication between services. (for this see also design decisions). While this took quite some time and discussions during meetings, we were able to arrive at a point where our views and considerations regarding the system were consistent.

To then start designing the system, we started at a very high and abstract level and then continuously added more levels of detail. These were then continuously refined based on new realizations. We then separated the responsibilities, where each detailed view of their selected service is being modeled and realized (see section 3). Even though we are designing a microservice architecture where each service is independent from each other, we always presented our solutions and design decisions to the entire team so that everybody can get an equal understanding of the system and find inaccuracies and contradictions within the system.

1.1.1. Assumptions

SR 3:

- No other types of roles will be added in the future
- Registration possibilities are also given by this service
- Account deletion is also given by this service

SR6:

- It is assumed that currently there are only 3 Tags (SPORT, FOOD and EDUCATION) that a User could put on an event and could only be seen by the user themselves in the system. E.g. if I add a SPORT tag on an Event, my friend will not see that Tag on that Event in his interface.

SR7:

- It is assumed that all Attendees have a valid e-mail address in the system to which newly newly added event suggestions can be delivered.
- It is assumed that location information of the user is available (country, city).

SR8:

- It is assumed that only users who have attended an event that has already taken place can leave feedback
- It is assumed that when an attendee wishes to provide feedback for an event, they also need to rate various criteria on a five-point scale
- It is assumed that it is possible to rate different criteria on an event without leaving a written feedback description
- It is assumed that submitted feedback can not be updated by the user

SR9:

- It is assumed that the standard Calendar export will be in .ics file format.
- It is assumed that only the most important information of an Event will be exported, which includes the *event name*, *event location, event tags and event date.*

SR10:

- It is assumed that a user can view analytics that are based on the ratings submitted by other users for an event
- It is assumed that feedback analytics show an average rating per feedback/rating criteria
- It is assumed that organizers can only create reports of their own events

SR11:

- It is assumed that the payment of booking attendance to events is not handled within the system.
- It is assumed that organizers can only send text-messages to attendees, and no attachments can be added.
- It is assumed that an Attendee can freely attend events, and no restrictions, such as minimum age, are present.

SR12:

- It is assumed that the users will only get notified via email

1.1.2. System Design Decisions

For the overall system, we decided to design it as a microservice architecture right away, utilizing the database-per-service pattern. This way, each service maintains its own view of the data - storing only what is relevant to achieve its use-case. The model is then updated through messages (events) based on which topics the microservices are subscribed to (publish-subscribe pattern). This communication is then built around a standardized data format, in our case JSON. The number of messages sent for each event was kept minimal by using DTOs (Data transfer object).

While this approach can lead to inconsistent views of the data at times (only achieving eventual consistency), it makes the microservices a lot more loosely coupled, as each service can work independent of others. In turn, this also improves the fault-tolerance of the system, as other services can continue to operate in case a single service experiences an outage.

1.1.3. Design Decisions of Individual Services

SR 1:

Right now we are still discussing two designs for our api-gateway. One is a lightweight design with Spring Cloud Gateway which can specify its routes in a configuration class and act as a very simple api-gateway. The second approach we discussed is adding a discovery server (Netflix Eureka) in addition to Spring Cloud Gateway. A discovery server implementation would allow us to have a better overview of the system and endpoint health, but would add an additional layer of complexity. In the next phase we will implement both possibilities and compare the pros/cons and document our decision in the report.

SR 2:

For SR2, we are using Spring Cloud Stream as it eliminates a lot of boilerplate code and allows to change the underlying message broker implementation. Currently, we are using Apache Kafka (+ ZooKeeper) because of previous experience of some team members working with it.

SR3:

The login service went through many design iterations. It began with trying to setup a keycloak server that would handle our role-based access control automatically. Having had a lot of issues with the setup, we ultimately decided to use keycloak simply as a JWT token provider and implement all the role-based access logic in the login service. Therefore the login-service acts as an abstraction of the keycloak api for creating access tokens and creating users, but also implements role-based access logic using a configuration file. In addition it provides facilities to create / delete system users (this is different from keycloak users) that represent our standard users.
The api-gateway, with every request that wants to access a protected resource, asks the login-service if this request with the given access token has proper rights. The login-service confirms with keycloak and confirms with its own role-based access logic and responds to the api-gateway with the proper information.

SR6:

The current design of storing tags is being done with a list of enum class. From past Java experience, it is considered to be safer, more readable and easier to refactor. However, during implementation, the readability in database is less readable and more complex to implement than expected.

It is concerned that is harder to extend further. E.g. when the requirements from stakeholder expanded/changes in the future, and a new tag needs to be added. The system will not be able to add a new Tag during runtime. Alternatively, storing tags as

a list of strings or as a string of JSON could be other options, it can improve the readability in the database, and also easier but slower to implement.

SR7:

The functional requirements of the recommender service were extended to also allow attendees to retrieve personal event recommendations (“Personal Picks”) based on the currently available events, instead of only notifying them when a newly added event could be interesting to them. In this regard, one of the main considerations was how to determine which events should be recommended to users, taking into account the information available of the user (previously attended events, etc.).

To realize this behavior, the strategy pattern is utilized which either injects a very personalized recommendation strategy, or if no event-interests of the user are known, a generalized recommendation strategy which considers, e.g. the location or trending events. This approach also makes it easy to add additional strategies in the future, which can then simply be chosen at runtime.

Moreover, because incoming messages of other microservices were used in different parts of the service, the observer pattern is utilized to reduce the coupling between components and make the communication more flexible as well as extensible for future use-cases.

Additionally, Data Transfer Objects (DTO) are utilized for any outgoing/incoming communication to only expose the minimum and necessary data as well as to provide a clear separation of the domain objects within the system. Furthermore, all available endpoints were designed according to the RESTful architectural style and best practices to be self-descriptive.

Lastly, the whole service was designed with the single-responsibility principle in mind in addition to using a layered-architecture and not allowing classes to bypass layers, i.e. a RestController is not concerned with the business logic at all, and delegates to a dedicated service class. (see also UML-class diagram).

SR8:

I first tried to identify the business logic behind my microservice. After that I designed the system according to the domain-driven design approach by structuring the parts into the DDD building blocks in a layered architecture. Since we decided on a database per service design there will be an own database container inside this microservice to make the service independent from others.

SR9:

When designing the ways of handling user’s attending events and bookmarked events of export service, two different designs came into my mind. My first design thought is to get two different list of events from other services. The user’s bookmarked events could be received from bookmark and tag service (SR6) and the

user’s attending events could be received from attendance service. In this way, the export service would not need to store any data on its own, but instead have a high coupling with other services. Also it is considered to be inflexible if one service that I am communicating with is down, then the data will not be available, thus affect the export service too.

The chosen design decision currently is to have an own Postgres database, and stores the incoming bookmark and attending events changes from the user. In that way, it will achieve a higher decoupling, thus a down time in attendance service or bookmark and tag service will not have any effects here.

I was concerned about the performance of the second design, since it will need more frequent communication with the event broker, thus might be slower than the first design choice. However, after testing with Apache Kafka, it seems that even millions of events could be handled in seconds, so the performance should not be an issue at the current stage.

SR10:

During the initial design phase of the microservice, it was planned to store report entities and analytic report entities in the database and have a ReportRepository as well as an Analytics Repository. However, during the implementation process, it was realized that this would lead to unnecessary resource consumption and data inconsistency, as every time an attendee would bookmark or attend/unattend an event, the saved report would become inaccurate. Therefore, the decision was made not to store reports and feedback analytic reports in the database.

Additionally, the current design of the microservice has combined the feedback analytics use case with the report use case, which means that feedbacks also have to be saved in the SR10-database. This design decision has the advantage that attendees can still access the feedback analytics even if the feedback service is unavailable. However, it doesn't feel like a proper design to separate the feedback service from the feedback analytics. Therefore, further consideration while the implementation process is required and it’s possible that this design decision will be changed in the actual implementation.

SR11:

SR11 was designed with the same design principles in mind as SR7, as such DTOs are used to facilitate the transfer of data to the outside and the observer pattern is used for incoming messages of other services to improve maintainability and simplify future extensions by making observer/subject agnostic of each other. Moreover, a layered architecture is also used as well as repositories to facilitate the data-access.

For SR11, additional requirements were added to provide functionality that is common in other systems. Namely, Attendees can cancel Event-Bookings up to 24 h

before the event starts. Furthermore, digital entrance tickets (QR-Codes) are created for each event they attend. For this, a factory is utilized due to the rather complex construction of ticket objects which includes the ZXing library to encode the most important information in a QR-code.

One design decision was to keep the mapping class of Event – Attendee (M:N - EventBooking) separate from the entry ticket. While these two are related, they still are quite different concepts and thus were kept separated. However, to avoid situations where an EventBooking is canceled and the ticket remains valid (or an eventbooking is created without ticket), a cascade constraint was added to maintain consistency.

SR12:

The notification service was designed from the beginning to be very lightweight since it has little responsibilities. It uses a simple spring boot library to send emails and receive event change DTOs from the message bus. As soon as it receives event change data, it gets all the corresponding user emails from its database, creates the proper email template and uses an SMTP server to send all emails to the users.

1.1.4. Design Overview

For the current state of our solution, refer to the in-depth class diagrams of each SR which is documented as an additional zoom-level in the logical view as well as the mock-up / skeleton code of our individual services.

1.2. Development Stack and Technology Stack

For the used technologies, our first decision went towards using Java 17 (LTS) and Spring Boot. Most of us already had some experience working with the framework and moreover, it was also recommended in the course.

Additionally, it allowed us to utilize many Spring-native solutions, such as Spring Cloud Gateway or Spring Cloud Stream, which allows us to deploy event-driven communication independent of underlying message broker while also eliminating a lot of boilerplate code. While the message broker can be freely exchanged, we decided to use Apache Kafka as it is a very well established and highly performant platform which fulfilled all of our needs.

For the database, we decided on a relational database, namely PostgreSQL, in combination with Hibernate ORM to simplify the development as well as making the code less dependent on the underlying database implementation. Lastly, because the databases can vary between microservices, we also explored other options individually, like using NoSQL databases such as MongoDB, but ultimately decided to use PostgreSQL for each microservice.

For generating the OAuth2 JWT access tokens, we decided to use a Keycloak server. There we can set up realm roles which are then appended to the access token which will be verified in the login service.

1.2.1. Development Stack

Build Management	Maven 3.9.1
Version Control	GitLab
Integrated Development Environment	IntelliJ
CI/CD Platform	GitLab CI/CD
API-Client (Development and Testing)	Postman

1.2.2. Technology Stack

Programming Language	Java 17
Application Framework	Spring Boot 3.0.5
API Gateway Framework	Spring Cloud Gateway 4.0.3

Messaging Framework	Spring Cloud Stream 4.0.2
Testing Framework	JUnit 5.9.2
Database	PostgreSQL 15.2
Message Broker	Apache Kafka 3.2.3
Distributed Coordination Service	Apache ZooKeeper 3.7.0
Containerization	Docker Engine 20.10.23
OAuth2 Provider	Keycloak 15.0.2
Netflix Eureka Discovery Service (currently considering using with spring cloud gateway)	Eureka Server 4.0.1

1.2.3. External Libraries

Library	Usage (SR)
com.google.zxing	Creation of QR-Codes for digital entrance tickets (SR11)
org.mnode.ical4j	Exporting as Calendar format (SR9) (https://www.ical4j.org/)

2. System Requirements

2.1 Main Functional Requirements

Identifier	FR03-01
Priority	High
Description	The system should support 3 different roles: the Attendee, the Organizer and the Administrator.
Affected use-cases	All
User roles	Attendee, Organizer, Administrator
Verification method	Manual inspection
Associated SR	SR3 (mainly)

Identifier	FR03-02
Priority	High
Description	Role based access control should be granted using an access token
Affected use-cases	UC03-02
User roles	Attendee, Organizer, Administrator
Verification method	Manual inspection, Spring integration tests
Associated SR	SR3

Identifier	FR03-03
Priority	Medium
Description	An administrator can see the maintenance of the whole system.
Affected use-cases	UC03-01
User roles	Administrator
Verification method	Manual inspection
Associated SR	SR3

Identifier	FR04-1
Priority	High
Description	As an Attendee, the system should allow the user to see events that an organizer posted.
Affected use-cases	-
User roles	Attendee, Organizer
Verification method	Manual inspection
Associated SR	SR4 (mainly)

Identifier	FR06-1
Priority	Medium
Description	As an Attendee, I should be able to bookmark my favorite events or the one that interests me.
Affected use-cases	UC06-01, UC06-02
User roles	Attendee
Verification method	Manual inspection, Spring integration Test
Associated SR	SR6

Identifier	FR06-2
Priority	Medium
Description	As an Attendee, I should be able to bookmark or unbookmark every events that are available (displayed).
Affected use-cases	UC06-01
User roles	Attendee
Verification method	Manual inspection, Spring integration Test
Associated SR	SR6

Identifier	FR06-3
Priority	Medium
Description	As an Attendee, to find events quicker, but also provides an overview without looking into the details, I should be able to add and remove the following tag/s on event/s once (no duplications): #sports, #food, #education. (which also should be only visible to me and not other Attendees)
Affected use-cases	UC06-03, UC06-04
User roles	Attendee
Verification method	Manual inspection, Spring integration Test
Associated SR	SR6

Identifier	FR07-01
Priority	Medium
Description	As an Attendee I should be notified via E-Mail if a newly added Event matches my interest profile, taking into account my previously attended events or information such as hometown.
Affected use-cases	UC07-01
User roles	Attendee
Verification method	Manual inspection, Unit Tests, Integration Test
Associated SR	SR7

Identifier	FR07-02
Priority	Low
Description	As an Attendee I should be able to opt out of receiving promotional E-Mails that suggest new events to me.
Affected use-cases	UC07-01
User roles	Attendee
Verification method	Manual inspection, Unit Tests
Associated SR	SR7

Identifier	FR07-03
Priority	Medium
Description	As an Attendee I should be able to view “Personal Event Picks”, which shows me events that are recommended to me.
Affected use-cases	UC07-02
User roles	Attendee
Verification method	Manual inspection, Unit Tests
Associated SR	SR7

Identifier	FR07-04
Priority	Medium
Description	As an Attendee I should be notified via E-Mail if an event which I bookmarked only has 10% of vacancies left.
Affected use-cases	UC07-01
User roles	Attendee
Verification method	Manual inspection, Integration Test
Associated SR	SR7, SR6

Identifier	FR08-01
Priority	Medium
Description	As an Attendee I should be able to leave written feedback for an event I attended.
Affected use-cases	UC08-01
User roles	Attendee
Verification method	Manual inspection, Unit Test
Associated SR	SR8

Identifier	FR08-02
Priority	Medium
Description	As an Attendee I should be able to give ratings on a scale for different criteria on an event I attended.
Affected use-cases	SR08-02
User roles	Attendee
Verification method	Manual inspection, Unit Test
Associated SR	SR8

Identifier	FR09-01
Priority	Low
Description	As an Attendee, to make user's attending or bookmarked events more accessible, the system should allow user to be able download (export) them into an external file include the following 3 formats: .ics .json .xml
Affected use-cases	UC09-01, UC09-02
User roles	Attendee
Verification method	Manual inspection
Associated SR	SR9

Identifier	FR10-01
Priority	Medium
Description	As an Organizer I should be able to generate a report for events I'm organizing
Affected use-cases	UC10-01
User roles	Organizer
Verification method	Manual inspection, Unit Test
Associated SR	SR10

Identifier	FR10-02
Priority	Medium
Description	As an Attendee I should be able to see analytics about given feedback of events. This analysis should include averages of different criteria.
Affected use-cases	UC10-02, UC08-01
User roles	Attendee
Verification method	Manual inspection, Unit Test
Associated SR	SR10, SR8

Identifier	FR11-01
Priority	High
Description	As an Attendee I should be able select an event that I want to attend, which then creates a digital entrance ticket for me.
Affected use-cases	UC11-01
User roles	Attendee
Verification method	Manual inspection, Unit Tests
Associated SR	SR11, SR4

Identifier	FR11-02
Priority	Medium
Description	As an Attendee I should be able to directly view all of my event-bookings and the corresponding entrance tickets under my profile section.
Affected use-cases	UC11-01, UC11-02
User roles	Attendee
Verification method	Manual inspection
Associated SR	SR11

Identifier	FR11-03
Priority	Medium
Description	As an Attendee I should be able to cancel a previously made Event-Booking up to 24 h hours before the event begins.
Affected use-cases	UC11-02
User roles	Attendee
Verification method	Manual inspection, Unit Tests
Associated SR	SR11

Identifier	FR11-04
Priority	Medium
Description	As an Organizer, I should be able to directly message all or a subset of the attendees of an event that I am organizing, which will then be forwarded to the E-Mail addresses of the Attendees.
Affected use-cases	UC11-03
User roles	Organizer, Attendee
Verification method	Manual inspection
Associated SR	SR11

Identifier	FR12-01
Priority	Medium
Description	As a user who attends or bookmarks an event, whenever the event data changes I get notified via email about the changes.
Affected use-cases	UC12-01
User roles	Attendee
Verification method	Manual inspection
Associated SR	SR12

2.2 Non-Functional Requirements

Identifier	NFR1
Priority	High
Description	Users are only allowed to access protected resources that their role has access rights to.
Category	Security

Identifier	NFR2
Priority	High
Description	The system must only return the minimum amount of information and not expose any sensitive data to the outside.
Category	Security & Privacy

Identifier	NFR3
Priority	High
Description	The system’s architecture must allow to accommodate for future changes without major modifications of the existing source code.
Category	Extensibility

Identifier	NFR4
Priority	Medium
Description	The system should be intuitive to use and its functionality should be self-describing.
Category	Usability

Identifier	NFR5
Priority	High
Description	The system should be able to stay performant and responsive even under high traffic loads.
Category	Scalability

Identifier	NFR6
Priority	Low
Description	The system should be able to work with regardless of the underlying platform or used browser.
Category	Compatibility

Identifier	NFR7
Priority	Low
Description	The data shown and presented to the user throughout the system must be consistent
Category	Consistency

Identifier	NFR8
Priority	Low
Description	The calendar export should start downloading within 5 seconds as soon as a user clicks the export button.
Category	Performance

3. 4+1 Views Model

For the Logical View, we decided to utilize zoom factors, as such there is a more abstract view on the system-level that shows the core domain entities and in which contexts they reside in or are shared. For the bounded contexts, we avoided shared kernels in order to reduce the coupling between the services on one hand. On the other hand, it made the most sense for each service to have its own view of the data, keeping only what is relevant to fulfill its use-case. The communication is then facilitated using DTOs. This is also the reason why we decided to have each service in its own bounded context, with the exception of the event-management and search being combined into one.

Consequently, more detailed views were created on the service-level which shows the structure of our microservices in more detail.

IMPORTANT REMARK:
For higher quality diagrams, please refer to the model folder of our git repo.

3.1. Scenarios / Use Case View

3.2.1 Use Case Diagram

Figure 3.1: Use Case Diagram for Event Management System

3.1.1. Use Case Descriptions

SR 3

Use case:	View system maintenance
Actor(s):	Administrator
Use case ID:	UC03-01
Brief description:	An administrator wants to view the system maintenance by accessing the maintenance view.
Pre-conditions:	- A user of type administrator has to be logged into the system - User has to access the maintenance view
Post-conditions:	- User gets redirected to the maintenance view - User can see health of system
Main Success Scenario:	1. An administrator is using the system 2. Administrator accesses maintenance view 3. Administrator can view the health of the entire system
Priority:	Medium

Use case:	Authentication
Actor(s):	User
Use case ID:	UC03-02
Brief description:	A user logs into the system using their credentials and receives a role-based access token with which they have limited access to services depending on their role.
Pre-conditions:	The user must be registered to the system.
Post-conditions:	The user receives the access token and will be redirected to its role specific welcome page.
Main Success Scenario:	1. User inserts credentials into the login form 2. User clicks login 3. User receives access token and gets rerouted to the role specific welcome page

Priority:	High

Use case:	Register new user
Actor(s):	User
Use case ID:	UC03-03
Brief description:	As a user I can choose to create an account on the login page to have access to the system
Pre-conditions:	- User wants to create a new account - Newly created account does not exist in the system yet
Post-conditions:	The new account will be added persistently to the system
Main Success Scenario:	1. User chooses to create a new account 2. User fills in all necessary information to the account form 3. User presses the “create account” button 4. New account is created successfully
Priority:	Medium

Use case:	Delete existing user
Actor(s):	User
Use case ID:	UC03-04
Brief description:	As a user I can choose to delete my account any time I want.
Pre-conditions:	• User is already logged into the system • User wants to delete his or her account

Post-conditions:	• User gets logged out of its current session • User account does not exist in the system anymore
Main Success Scenario:	1. User presses the “delete account” button 2. User confirms the dialog 3. User gets logged out of its current session and brought back to the login page 4. User account is deleted from the system
Priority:	Medium

SR 6

Use case:	Bookmark an Event
Use case ID:	UC06-01
Actor(s):	Attendee
Brief description:	An attendee marks an event via the User Interface then the Event will be updated as “bookmarked” on the UI.
Pre-conditions:	- The Attendee must have an account in the system. - There must be Event(s) that exist already (posted by organizers). - The Event should not have been “bookmarked” before.
Post-conditions:	- The Event will be updated as “marked” on the UI. - The Event should not be able to be “marked” again. - The “Mark” text/button will turn into “Unmark”
Main Success Scenario:	1. The Attendee selects an Event that interests him. 2. The Attendee clicks “Bookmark”. 3. “Mark” will turn into “Unmark” 4. The selected Event will be marked. 5. That Event will be updated as “marked” on the UI.
Extensions:	The Event could be “unmarked” afterward by clicking the same button.
Priority:	Medium
Performance Target:	All Events should be able to be marked
Issues:	Different ways of disabling “Mark” after the Event has been marked. (E.g. hide the button, disable the button, toggle, still allowing to click but it will not do anything.)

Use case:	Unbookmark an Event
Use case ID:	UC06-02
Actor(s):	Attendee
Brief description:	An attendee unbookmarks an event via the User Interface then the Event will be updated as “unmarked” on the UI.
Pre-conditions:	- The Attendee must have an account in the system. - There must be Event(s) that exist already (posted by organizers). - The Event should have been “bookmarked” before.
Post-conditions:	- The Event will be updated as “unmarked” on the UI. - The Event should not be able to be “unbookmarked” again. - The “Unmark” text/button will turn into “Bookmark”
Main Success Scenario:	1. The Attendee selects an Event that he is no longer interested in. 2. The Attendee clicks “Unmark” from the marked Event. 3. “Unmark” will turn into “Bookmark” 4. The selected Event will be unmarked. 5. That Event will be updated as “unmarked”.
Extensions:	The Event can be “bookmarked” afterwards by clicking the same button.
Priority:	Medium
Performance Target:	All marked Events should be able to be unmarked.
Issues:	Different ways of disabling “Unmark” after the Event has been unmarked. (E.g. hide the button, disable the button, toggle, still allowing to click but it will not do anything.)

Use case:	Add a Tag to an Event
Use case ID:	UC06-03
Actor(s):	Attendee
Brief description:	An attendee adds one of the Tag options (sports, food, education) on an event via the User Interface then the Tag will be displayed on that Event.
Pre-conditions:	- The Attendee must have an account in the system. - There must be Event(s) that exist already (posted by organizers). - The Event should not have the Tag that the attendee is going to add. (the same Tag should not be added twice.)
Post-conditions:	- The Tag will be added and displayed to the selected Event.
Main Success Scenario:	1. The Attendee selects an Event that he wants to add a Tag on. 2. The Attendee selects one of the 3 Tags (sports, food, education). 3. The Attendee clicks “Add Tag”. 4. The Tag is added to that Event.
Extensions:	The Tag could be removed (“untagged”) afterwards.
Priority:	Medium
Performance Target:	All Events should be able to add up to 3 Tags
Issues:	

Use case:	Remove a Tag from an Event
Use case ID:	UC06-04
Actor(s):	Attendee
Brief description:	An attendee removes a Tag on an Event via the User Interface then the Tag will not be displayed anymore on that Event.
Pre-conditions:	- The Attendee must have an account in the system. - There must be Event(s) that exist already (posted by organizers). - The Event should have at least a Tag within.
Post-conditions:	- The Tag will be removed and will not be displayed to the selected Event.
Main Success Scenario:	1. The Attendee selects an Event. 2. The Attendee selects a Tag that he wants to remove. 3. The Attendee clicks “Remove Tag”. 4. The Tag is removed on that Event.
Extensions:	The removed Tag could be added (“tagged”) again afterwards.
Priority:	Medium
Performance Target:	All Tags on an Event could be removed
Issues:	

SR 7

Use case:	Suggest new Events to Attendees
Use case ID:	UC07-01
Actor(s):	Attendee
Brief description:	The attendee is notified if a newly added event is recommended to them, i.e., matches their interests
Pre-conditions:	The attendee must have a registered account in the system.
Post-conditions:	The user receives an e-mail informing them that an event is recommended to them.
Main Success Scenario:	1. A new event is added to the system. 2. The system determines whether the event matches the interest-profile of a user. 3. An event-recommendation notification is sent to all attendees that are likely to be interested in the event.
Extensions:	- Additionally, a recommendation notification is displayed if a bookmarked event is almost at capacity (>90%) - The Attendee can opt-out of receiving these promotional E-Mails
Priority:	Medium
Issues:	Fallback recommendation criteria are necessary if no interest profile has been established for a user.

Use case:	Create personalized Recommendations for Attendee
Use case ID:	UC07-02
Actor(s):	Attendee
Brief description:	The attendee can request personalized recommendations, which is a list of events that match their interests and location.
Pre-conditions:	The user must have a registered account in the system.
Post-conditions:	-
Main Success Scenario	1. The user selects “View personal recommendations”. 2. The system generates event-recommendations based on the recorded interests of the user. 3. The list of recommended events is displayed to the attendee.
Extensions:	Behavior of other attendees with similar interests can be taken into account (“similar users also visit X”).
Priority:	Medium
Issues:	Fallback recommendation criteria are necessary if no interest profile has been established for a user, such as generally trending events.

SR 8

Use case:	Leave Event Feedback
Use case ID:	SR8-01
Actor(s):	Attendee
Brief description:	Users are able to give feedback on events. A feedback includes an optional comment and some five-point-scales for different criteria and an overall rating of the event.
Pre-conditions:	- User must be authenticated - Event must have taken place - The attendee must have attended to the event
Post-conditions:	- Feedback is stored in the database
Main Success Scenario:	1. User chooses the event they want to leave a feedback 2. System displays feedback comment field and rating options 3. User enters text and uses five-point scale for different criteria 4. System displays confirmation message to the user
Extensions:	- Error Message in case of missing authentication / user account / authorization / not attended event - Information about missing feedback options (E.g. Event has not taken place (yet))
Priority:	- Medium
Issues:	- None

SR 9

Use case:	Export all bookmarked Events
Use case ID:	UC09-01
Actor(s):	Attendee
Brief description:	An Attendee exports all Events that he bookmarked into a Calendar, JSON or XML format based on Attendee's choice. If Attendee does not select any specific format, the Event will be exported as Calendar format.
Pre-conditions:	- The Attendee must have an account in the system. - There must be Event(s) that he bookmarked already.
Post-conditions:	- The bookmarked Event(s) will be exported as a Calendar, JSON or XML file based on attendee's choice, if there is no specific format is selected, the export will be in Calendar format.
Main Success Scenario:	1. The Attendee clicks "Export bookmarked Event". 2. The Attendee ticks "XML" or "JSON" (optional) 3. The Attendee clicks "Export" 4. All bookmarked events will be exported as an external file based on Attendee's choice.
Extensions:	The exported file could be opened externally.
Priority:	Low
Performance Target:	The file export should start within 5 seconds after user clicked "Export".
Issues:	If there are too many Events, the export might be slow?

Use case:	Export all attending Events
Use case ID:	UC09-02
Actor(s):	Attendee
Brief description:	An Attendee exports all Events that he is attending into a Calendar, JSON or XML format based on Attendee's choice. If Attendee does not select any specific format, the Event will be exported as Calendar format.
Pre-conditions:	- The Attendee must have an account in the system. - There must be Event(s) that he is attending already.
Post-conditions:	- The bookmarked Event(s) will be exported as a Calendar, JSON or XML file based on attendee's choice, if there is no specific format is selected, the export will be in Calendar format.
Main Success Scenario:	1. The Attendee clicks "Export Attending Event". 2. The Attendee ticks "XML" or "JSON" (optional) 3. The Attendee clicks "Export" 4. All events that he is attending will be exported as an external file based on Attendee's choice.
Extensions:	The exported file could be opened externally.
Priority:	Low
Performance Target:	The file export should start within 5 seconds after the user clicked "Export".
Issues:	If there are too many Events, the export might be slow?

SR 10

Use case:	Generate Event Report
Use case ID:	SR10-01
Actor(s):	Organizer
Brief description:	Users are able to generate reports about events they are authorized for. The report content is customizable e.g. number of registered or attended people, number of bookmarks.
Pre-conditions:	- User must be logged in - User has to have proper access rights - Event must be created beforehand
Post-conditions:	- Report is removed
Main Success Scenario:	- User chooses the event they are interested in - System displays available report options - User selects report options of interest - System generates and displays report
Extensions / Additional Remarks:	- Error Message in case of missing user account / authentication / authorization - Information about missing report options (e.g. The event has not yet taken place.)
Priority:	- Medium
Issues:	- None

Use case:	View Feedback Analytics
Use case ID:	SR10-02
Actor(s):	Attendee
Brief description:	Users have the ability to view analytics on events. These analytics are divided into categories such as average rating, number of ratings, comments, and so on.
Pre-conditions:	- User must be registered and have proper access rights
Post-conditions:	- User sees feedback analytics
Main Success Scenario:	1. User chooses the event they are interested in and selects view analytics 2. The System displays all available feedback analytics on the selected event
Extensions:	- Error Message in case of missing user account / authentication / authorization - Information about missing analytics (e.g. The event has not yet taken place.)
Priority:	- Medium
Issues:	- None

SR 11

Use case:	Attend an Event
Use case ID:	UC11-01
Actor(s):	Attendee
Brief description:	Attendees can browse a catalog of events and select one they want to attend.
Pre-conditions:	The attendee must be logged in to the system.
Post-conditions:	The attendee is registered for the event and the entrance ticket is stored under his user-account.
Main Success Scenario:	1. The user browses the event catalog and chooses an event by selecting “Attend Event”. 2. The system verifies that vacancies are still available. 3. The system marks the spot as taken and creates an entrance ticket (QR-Code). 4. The booking confirmation is displayed to the user.
Extensions:	The booking can be optionally canceled by the attendee. (see UC11-02)
Priority:	High
Issues:	It must be ensured that events are not overbooked due to concurrent purchases.

Use case:	Cancel an Event-Booking
Use case ID:	UC11-02
Actor(s):	Attendee
Brief description:	Attendees can cancel previously booked events up to 24h before the event starts.
Pre-conditions:	The user must be logged in to the system and is registered to an event.
Post-conditions:	The Event-Booking and entrance ticket are no longer available to the attendee. The vacancies of the event are adjusted accordingly.
Main Success Scenario:	1. The attendee browses their booked events and selects “Request Cancellation”. 2. The system verifies whether the request is still within the cancellation window (24h before event start). 3. The Event-Booking and corresponding entrance ticket are deleted. 4. The cancellation confirmation is displayed to the user.
Extensions:	The booking can be optionally reverted by the user within the first five minutes.
Priority:	Medium
Issues:	Avoid edge-cases by utilizing UTC instead of local time.

Use case:	Send message to Attendees
Use case ID:	UC11-03
Actor(s):	Organizer, Attendees
Brief description:	Organizers can send messages to the attendees of their event (e.g. to inform them of information/schedule changes).
Pre-conditions:	The organizer is logged in to the system and has an event registered with the system.
Post-conditions:	The created message is sent to all participants of the event via their registered e-mail.
Main Success Scenario:	1. The organizer chooses one of their events and selects “Notify Attendees”. 2. The organizer selects all or a subset of attendees as recipients 3. The organizer enters the message and selects “Submit”. 4. The system retrieves the e-mail address of the selected attendees and forwards the message to their stored e-mail address. 5. The organizer is notified that the messages have been successfully forwarded.
Extensions:	-
Priority:	Medium
Issues:	Verification that only the organizer of the specific event can send messages.

SR 12

Use case:	Notification of attending or bookmarked event changes
Use case ID:	UC12-01
Actor(s):	Attendee
Brief description:	As an attendee, when I mark an event as attending or bookmarked, I will receive a notification when any event information gets updated.
Pre-conditions:	• Attendee marks event as attending or bookmarked • Event data gets updated
Post-conditions:	• Attendee receives email notification for related change
Main Success Scenario:	• Attendee marks event as bookmarked or attending • Event data changes • Attendee get notified about the change via email
Priority:	Medium
Performance Target:	The notification should be sent ideally within one minute up to five minutes.

3.2. Logical View

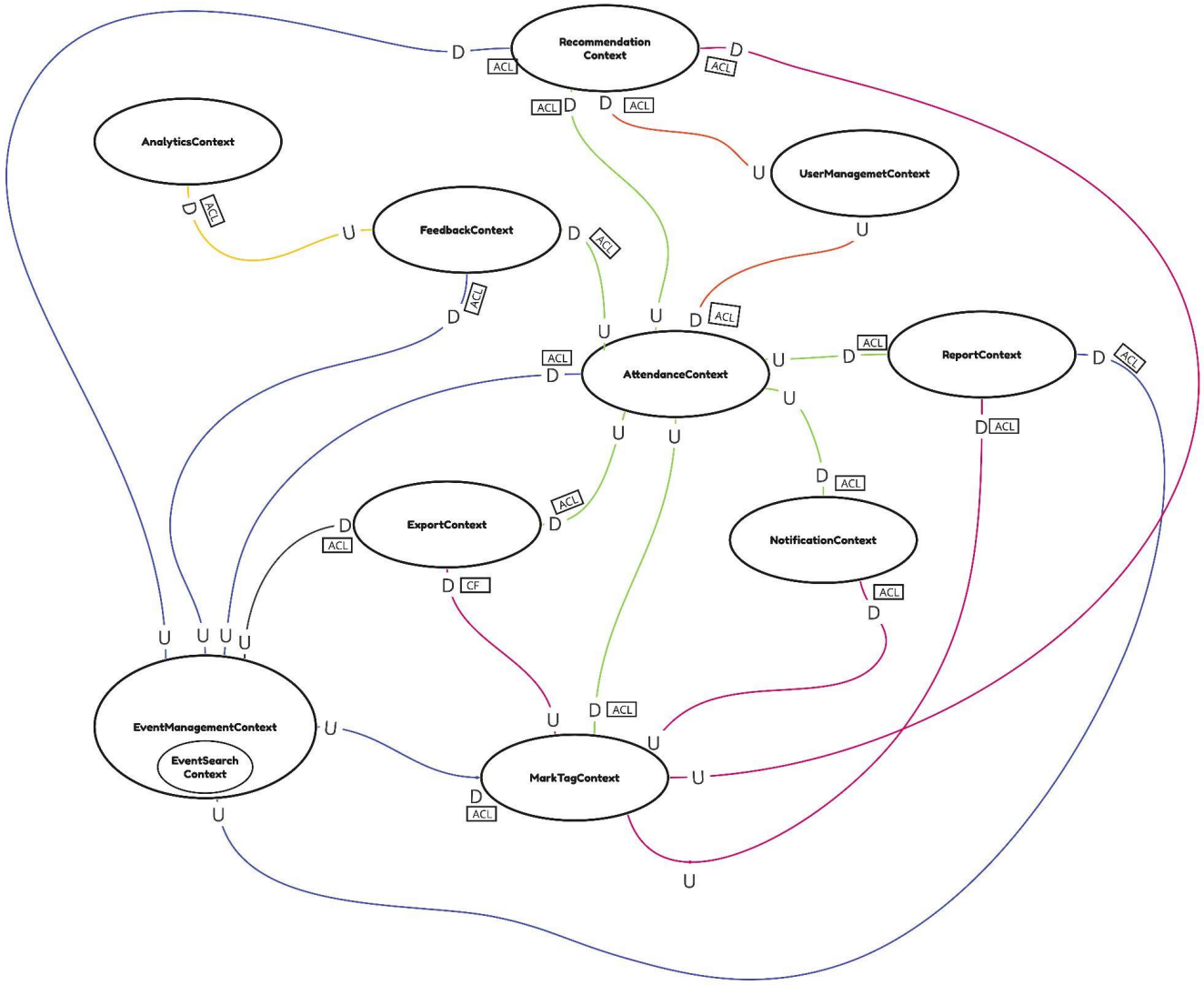


Figure 3.2.1: Context map for the whole system

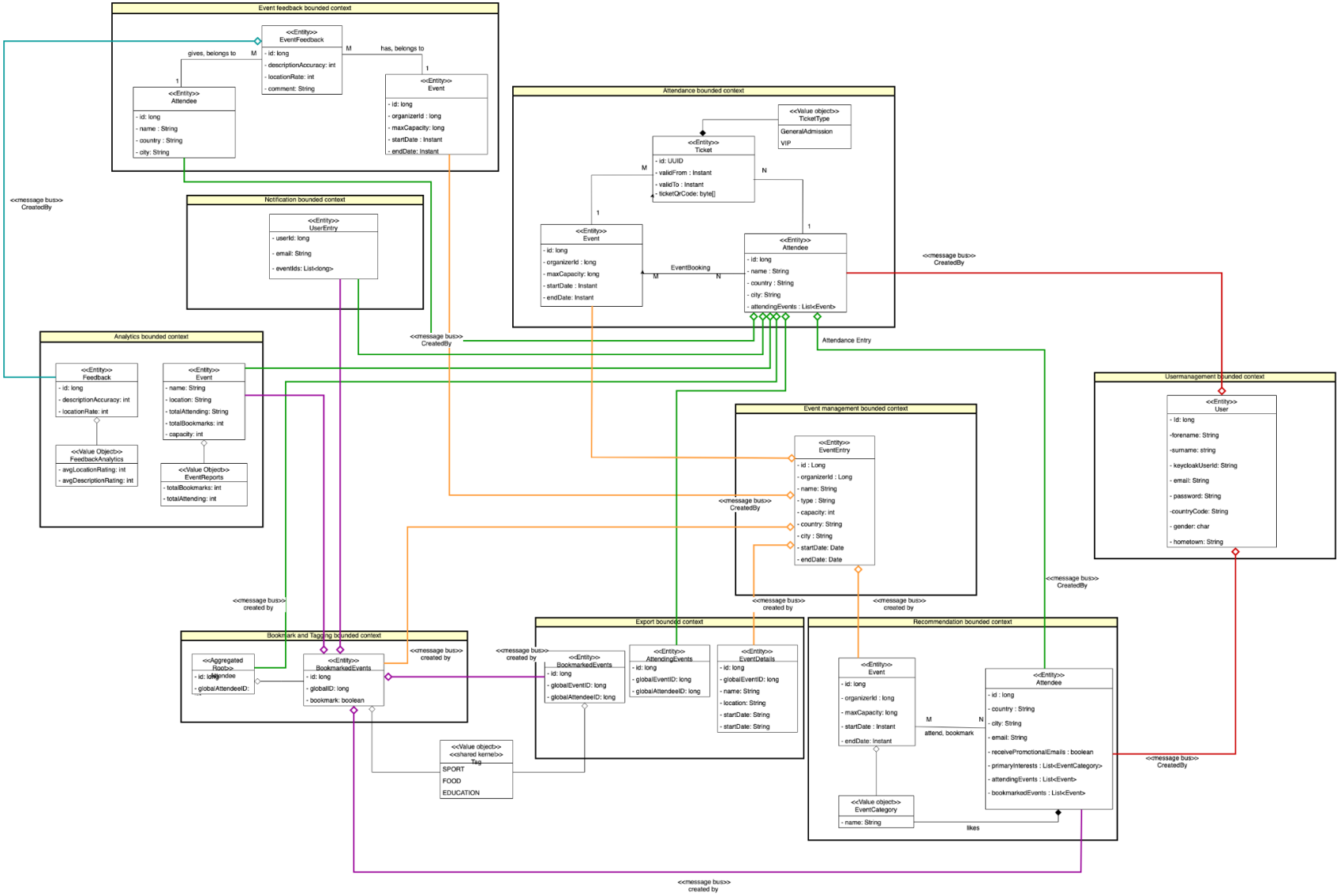


Figure 3.2.2: A more detailed Context map ("the new bigger picture" suggested by MS[1])

[1] https://www.mirkosertic.de/blog/2013/04/domain-driven-design-example/

SR3

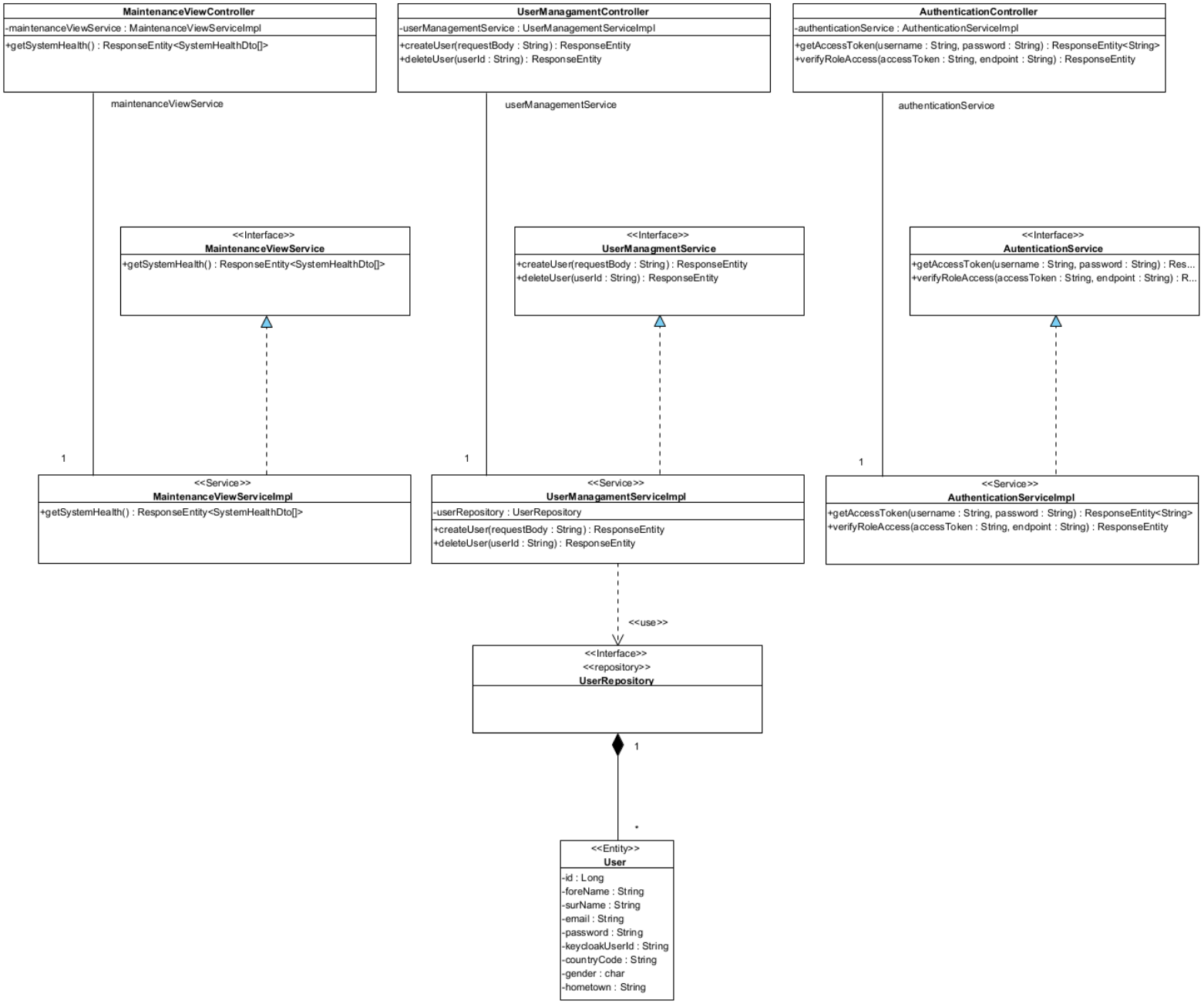


Figure 3.2.3: Class diagram for login service

SR6

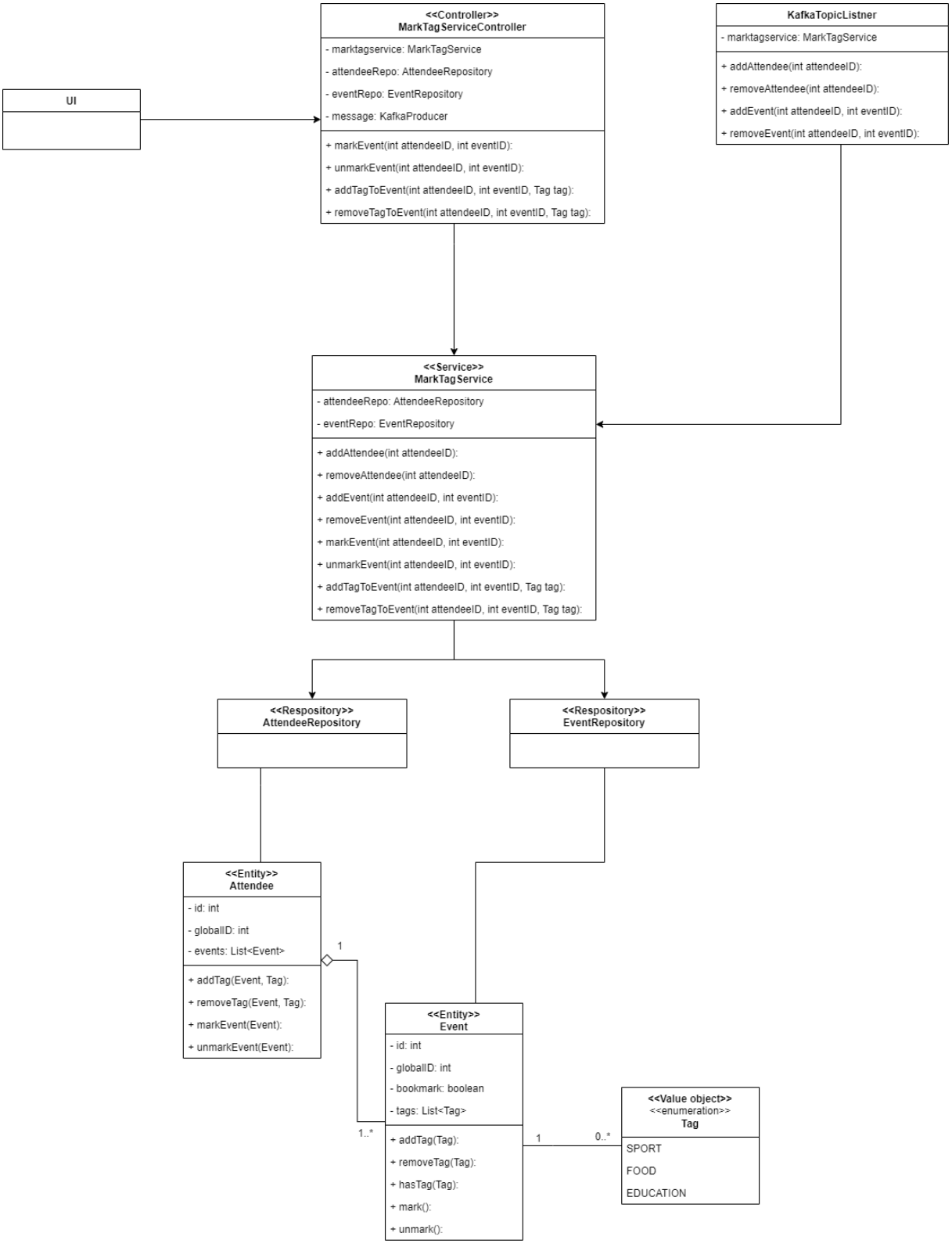


Figure 3.2.6: Class diagram for Bookmarking and Tagging Service

SR7

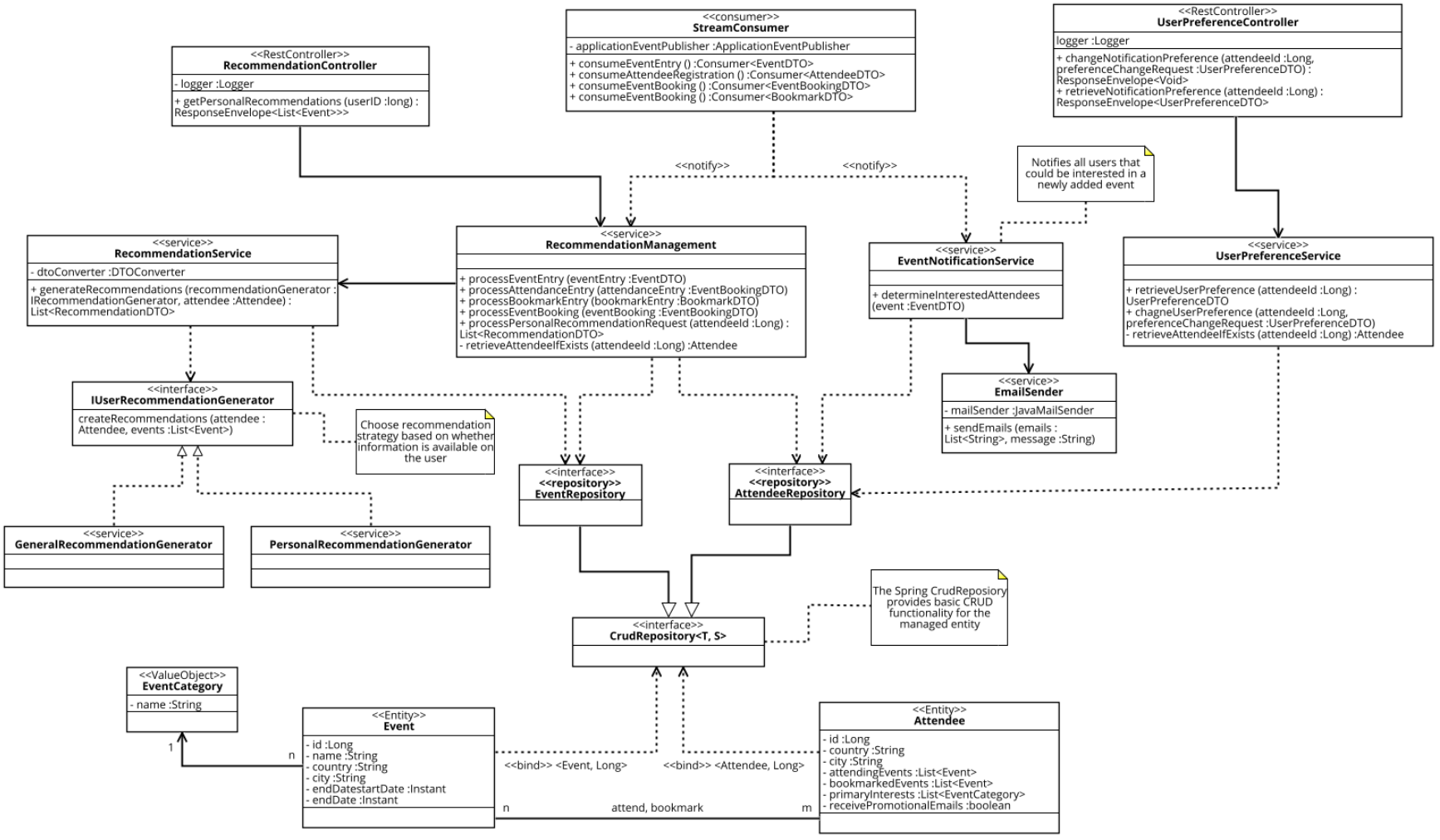


*Figure 3.2.7: Class diagram for Recommender Service (***Remark**: The diagram was kept to the core concepts, and some details, such as DTOs / Exceptions / DB-Association classes (M:N) were left out.)

SR8

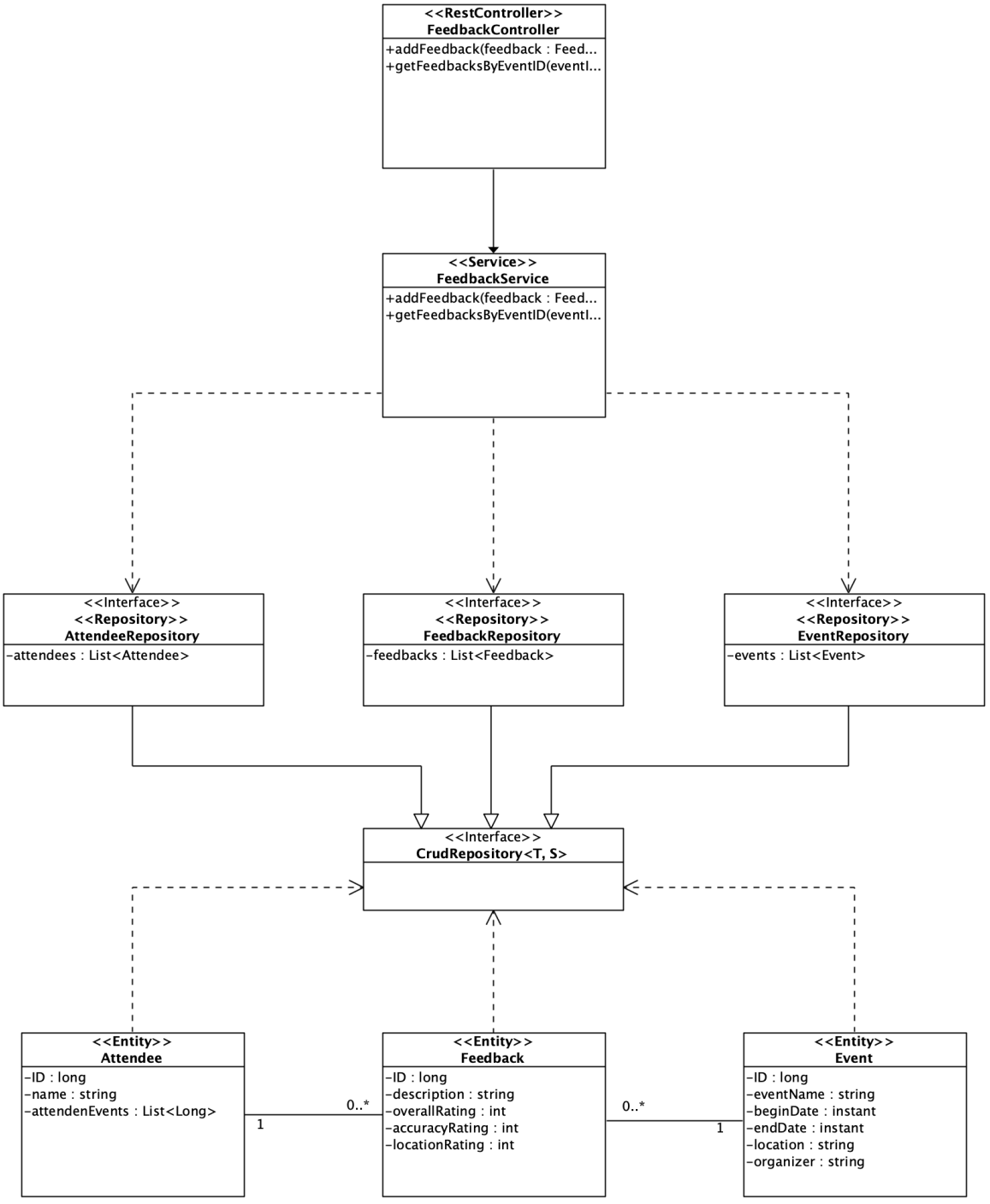


Figure 3.2.8: Class diagram for Feedback Service

SR9

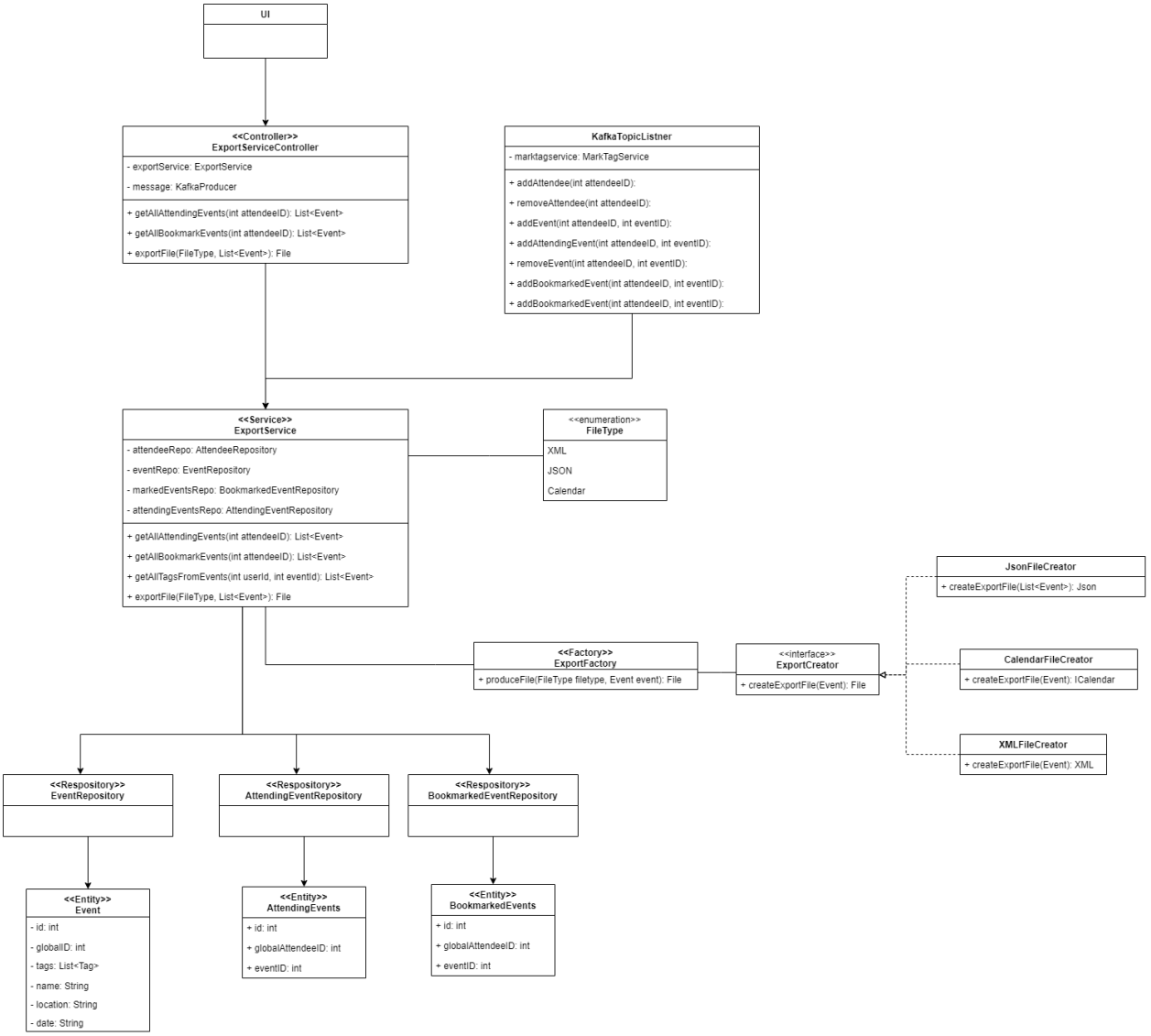


Figure 3.2.9: Class diagram for (Calendar) Export Service

SR10

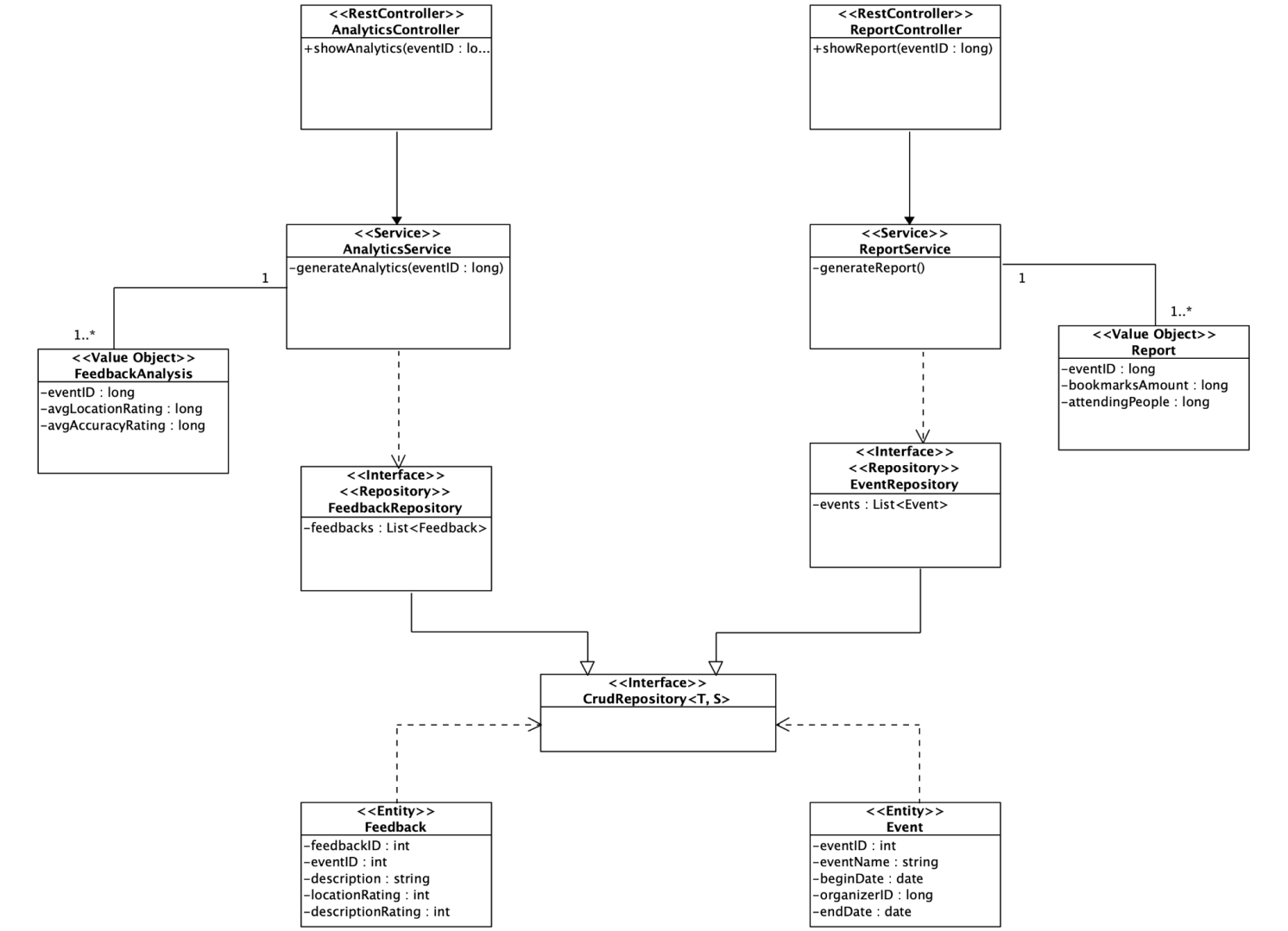


Figure 3.2.10: Class diagram for Feedback Analytics and Event Report Service

SR11

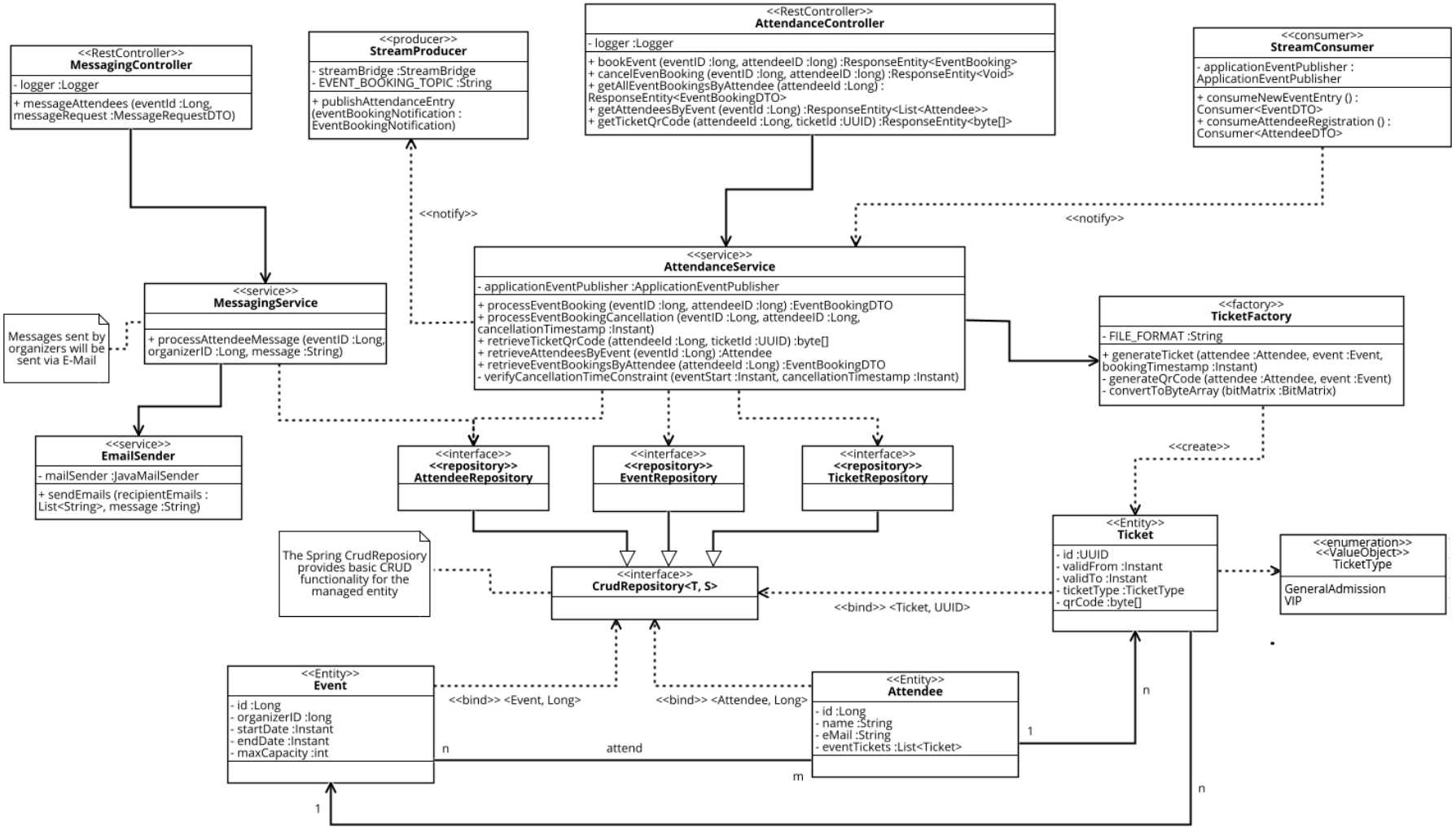


*Figure 3.2.11: Class diagram for Attendance service (***Remark**: The diagram was kept to the core concepts, and some details, such as DTOs / Exceptions / DB-Association classes (M:N) were left out.)

SR12

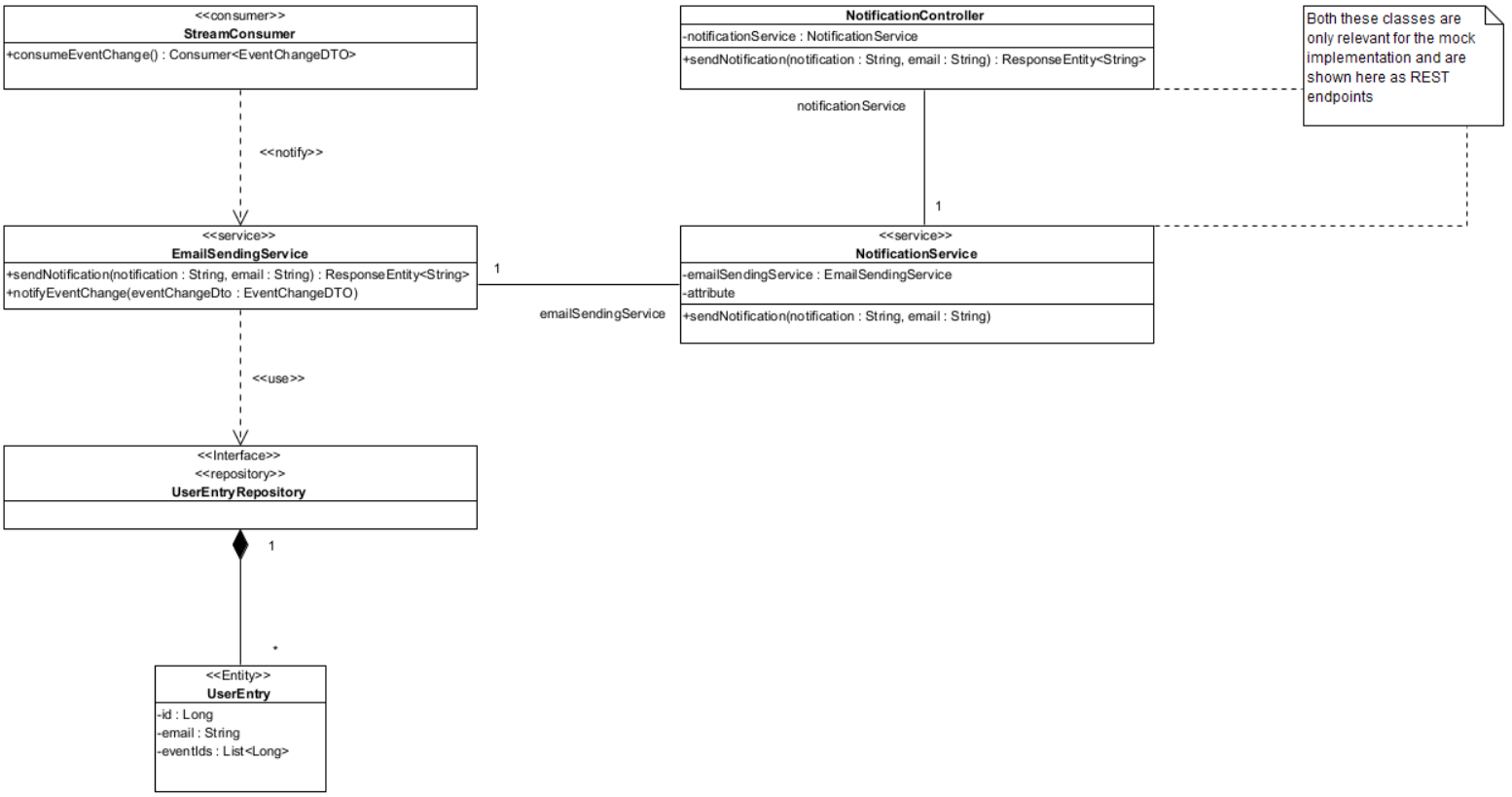


Figure 3.2.12: Class diagram of Notification Service

*(***Remarks**: As stated in the note, the NotificationController and NotificationService are both representing a REST endpoint and its service implementation are both used for showing the mock behavior of the notification service. *)*

3.3. Process View

To highlight the system's behavior during the use case, we decided to use a Sequence Diagram as it provides a simple representation of the interactions between actors and objects.

SR3:

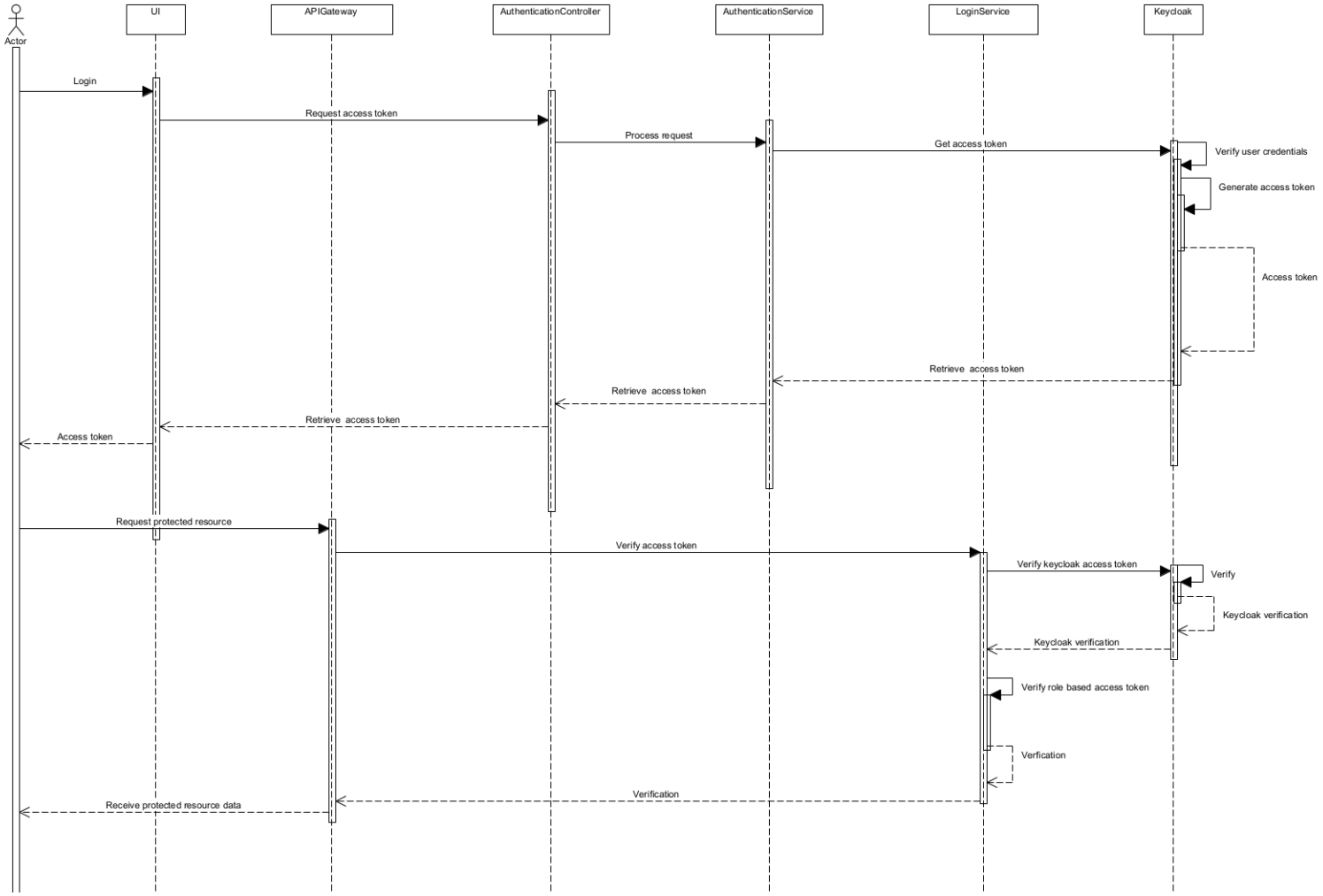


Figure 3.3.3.1: Sequence Diagram for authentication

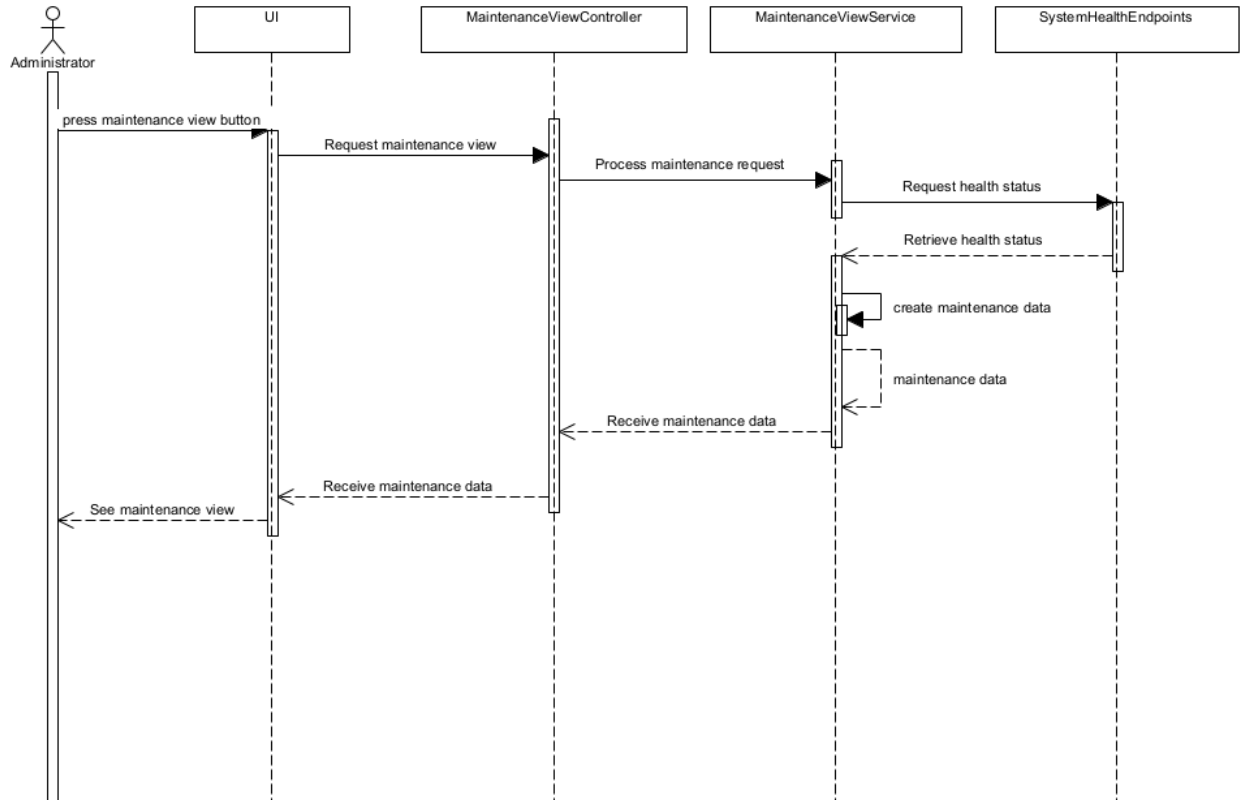


Figure 3.3.3.2: Sequence Diagram for viewing system maintenance

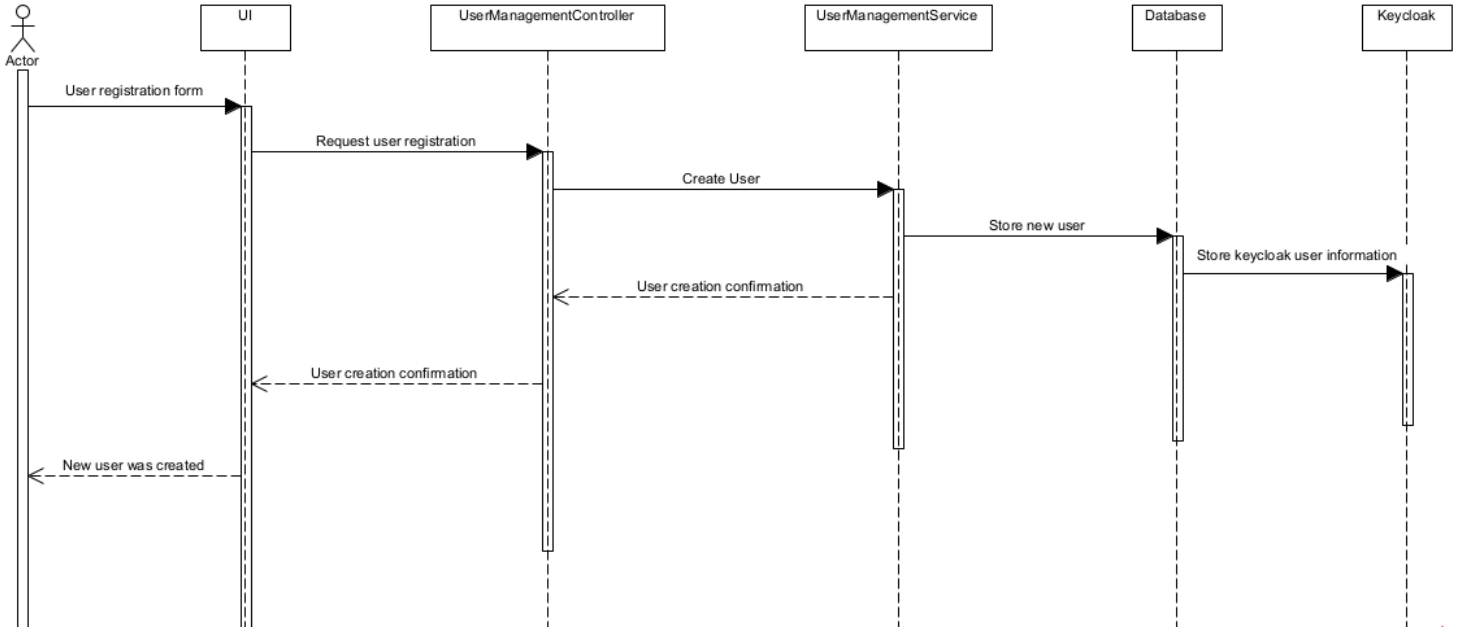


Figure 3.3.3.3: Sequence Diagram for registering new user

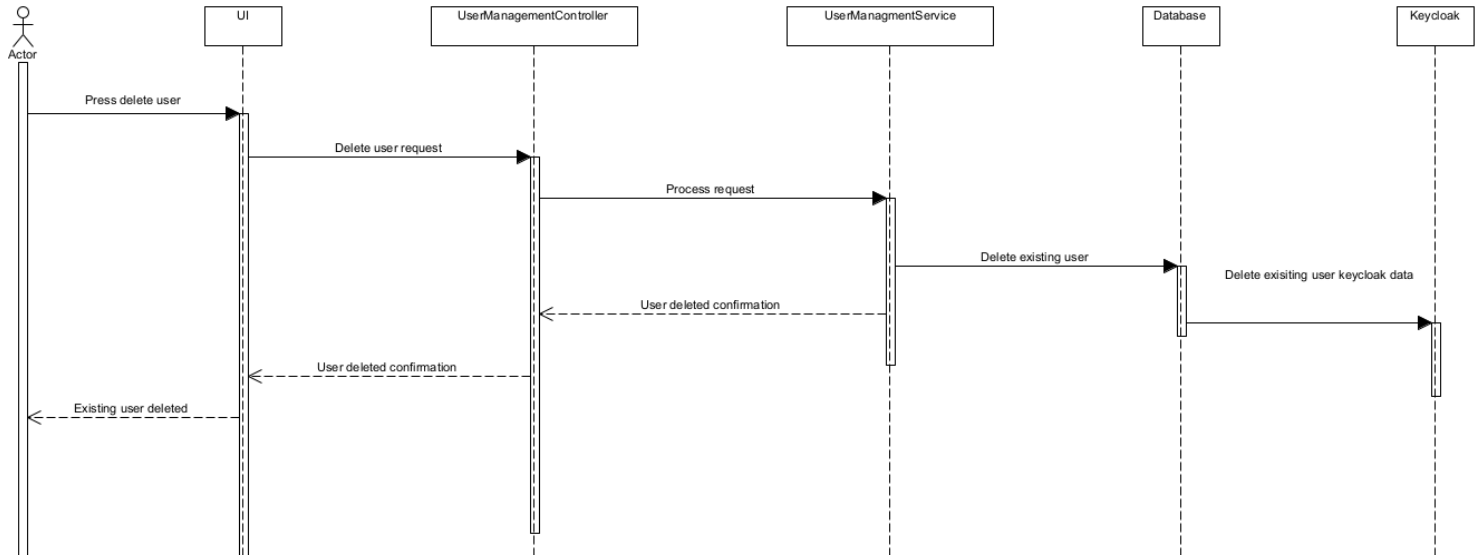


Figure 3.3.3.4: Sequence Diagram for deleting an existing user

SR6

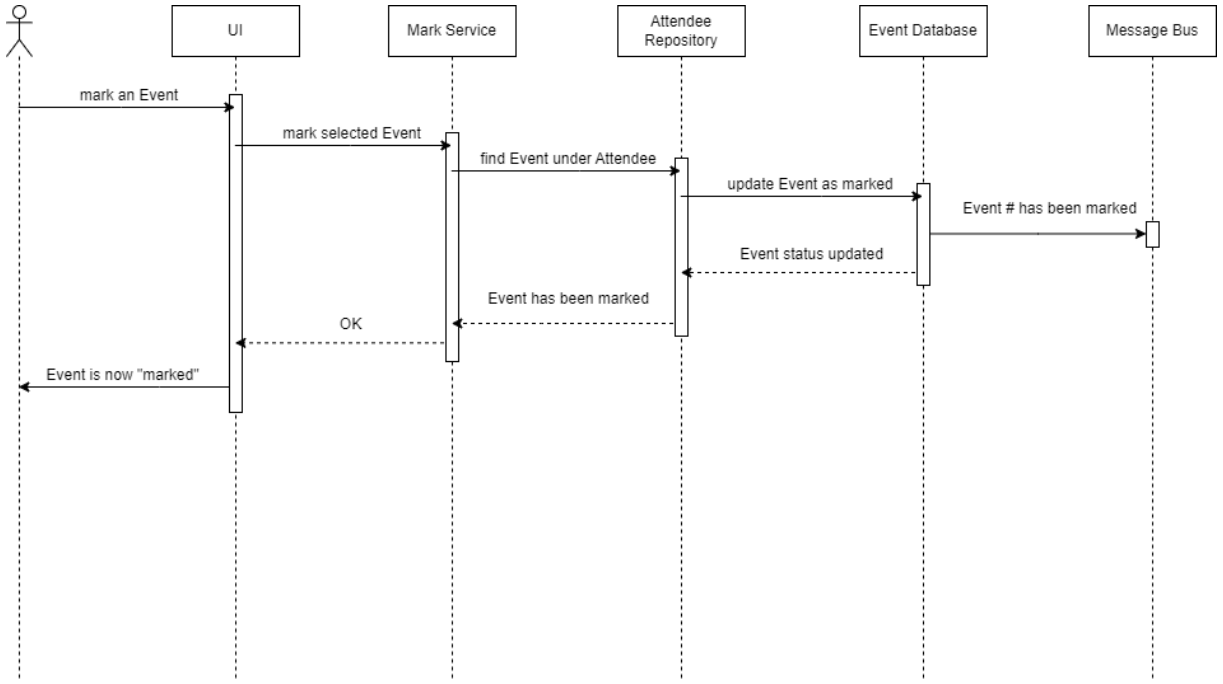


Figure 3.3.6.1: Sequence Diagram for bookmarking an Event

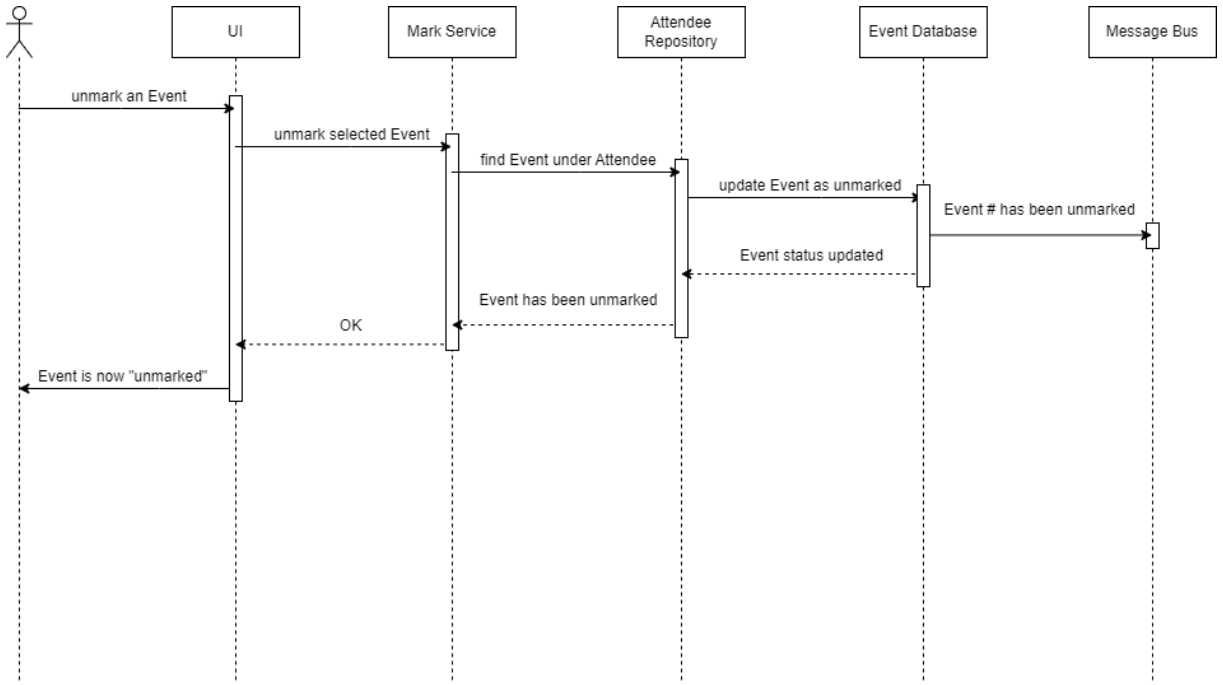


Figure 3.3.6.2: Sequence Diagram for unbookmark an Event

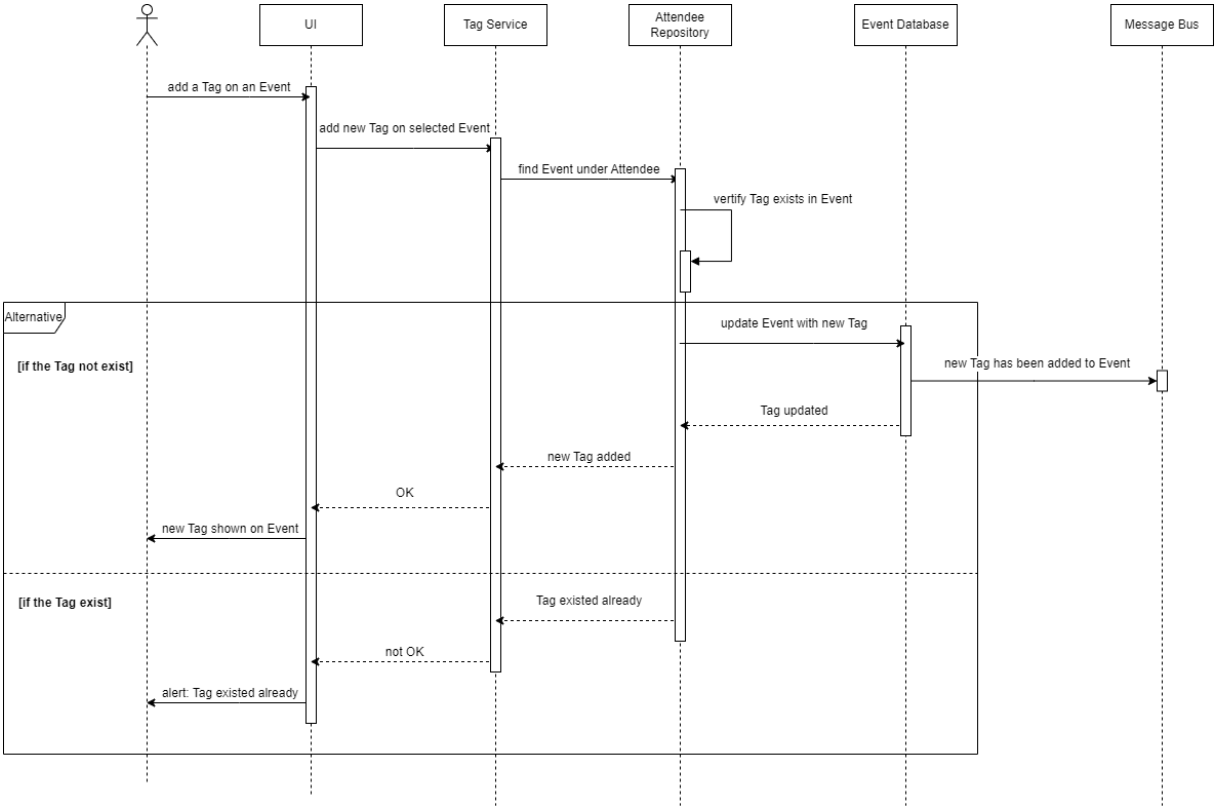


Figure 3.3.6.3: Sequence Diagram for adding a Tag to an Event

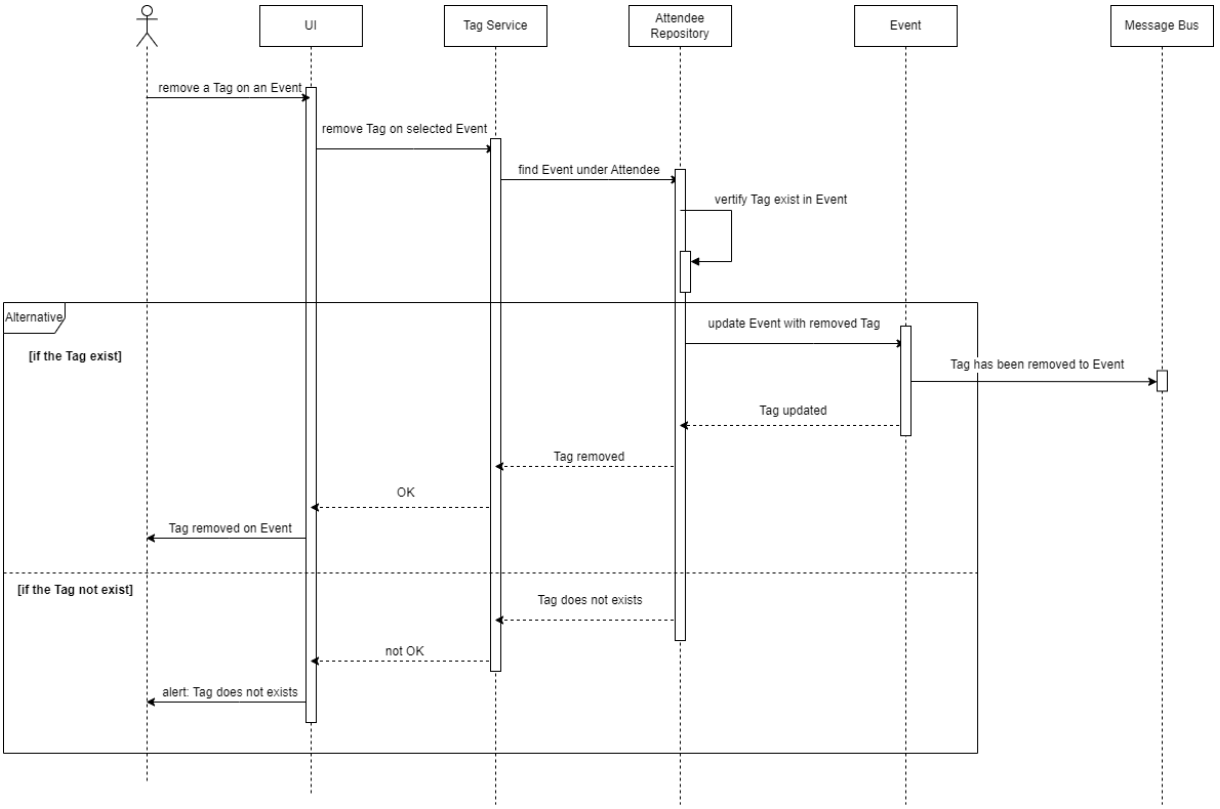


Figure 3.3.6.2: Sequence Diagram for removing a Tag to an Event

SR7

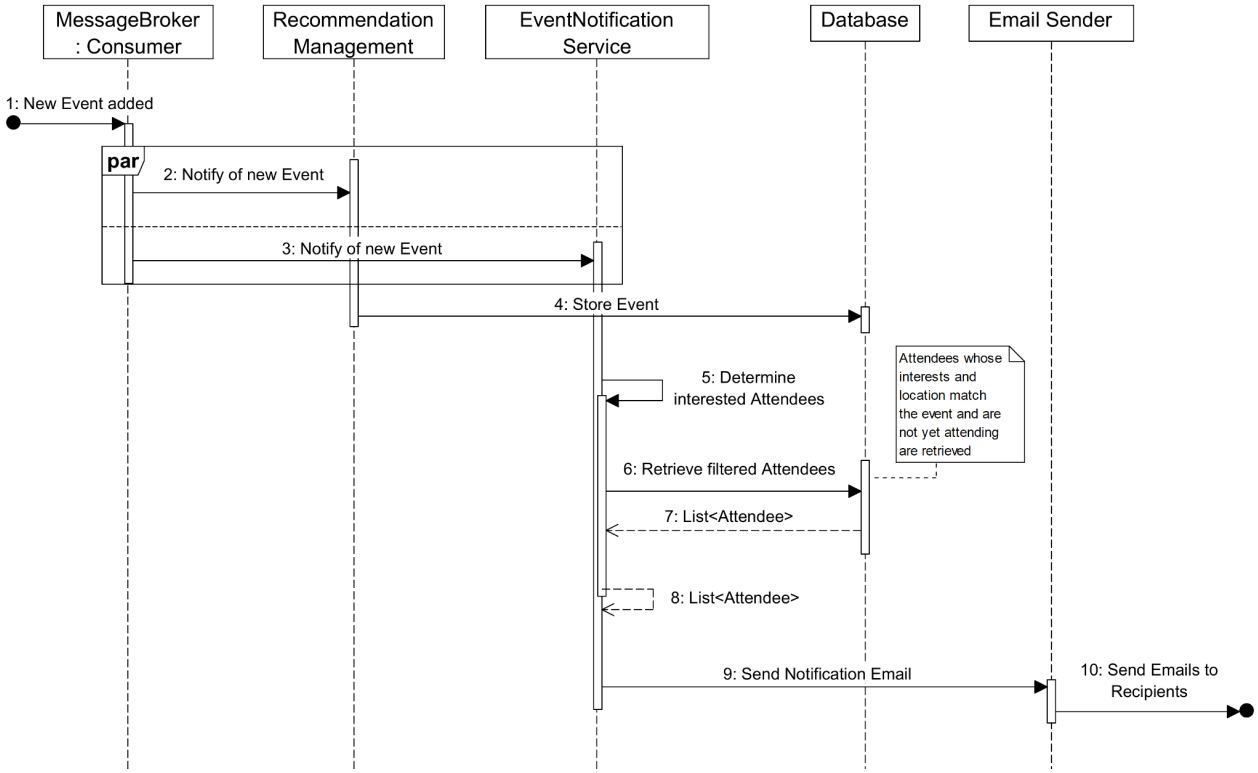


Figure 3.3.7.1: Sequence Diagram for suggesting newly added Events to Attendees (see UC07-01)

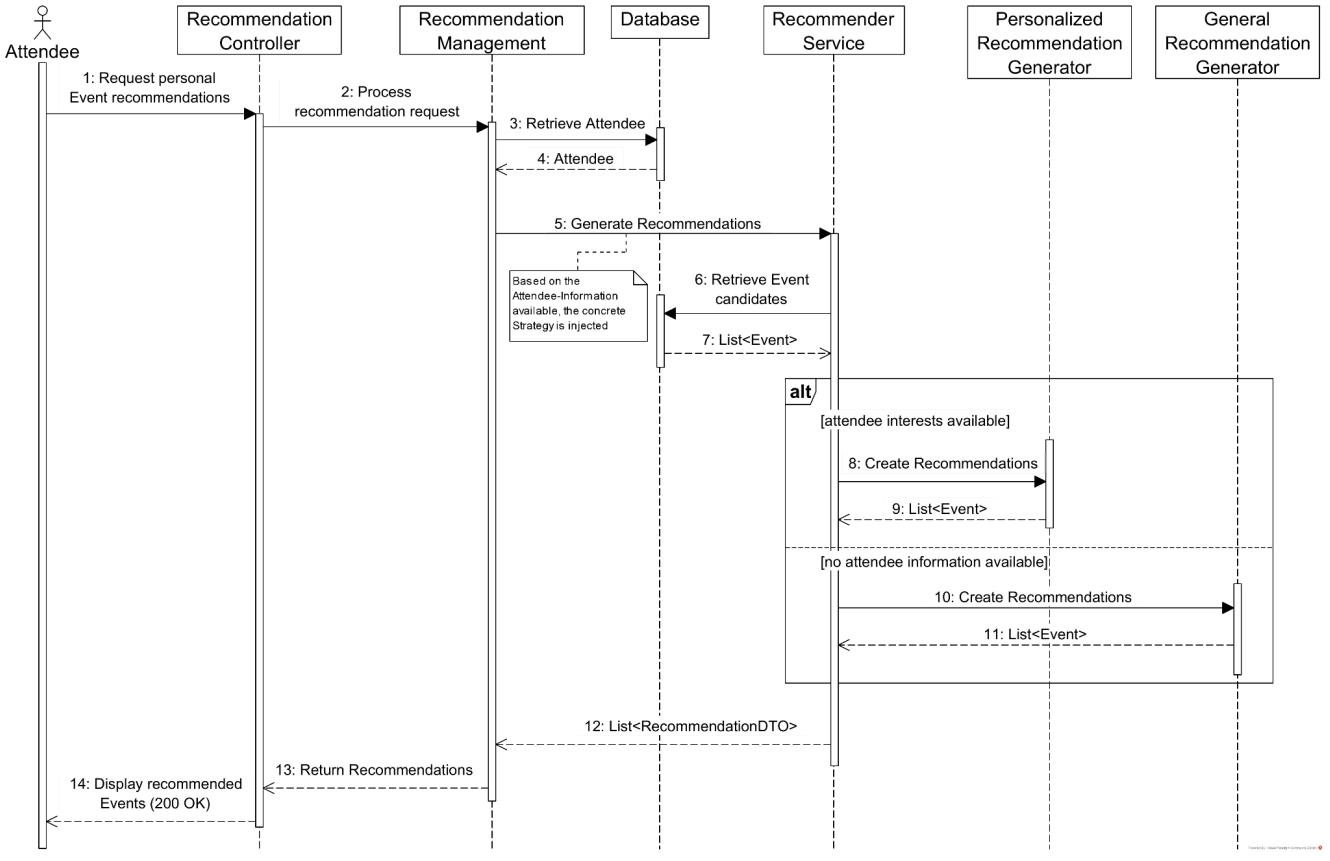


Figure 3.3.7.2: Sequence Diagram for request personal Event Recommendations (see UC07-02)

SR8

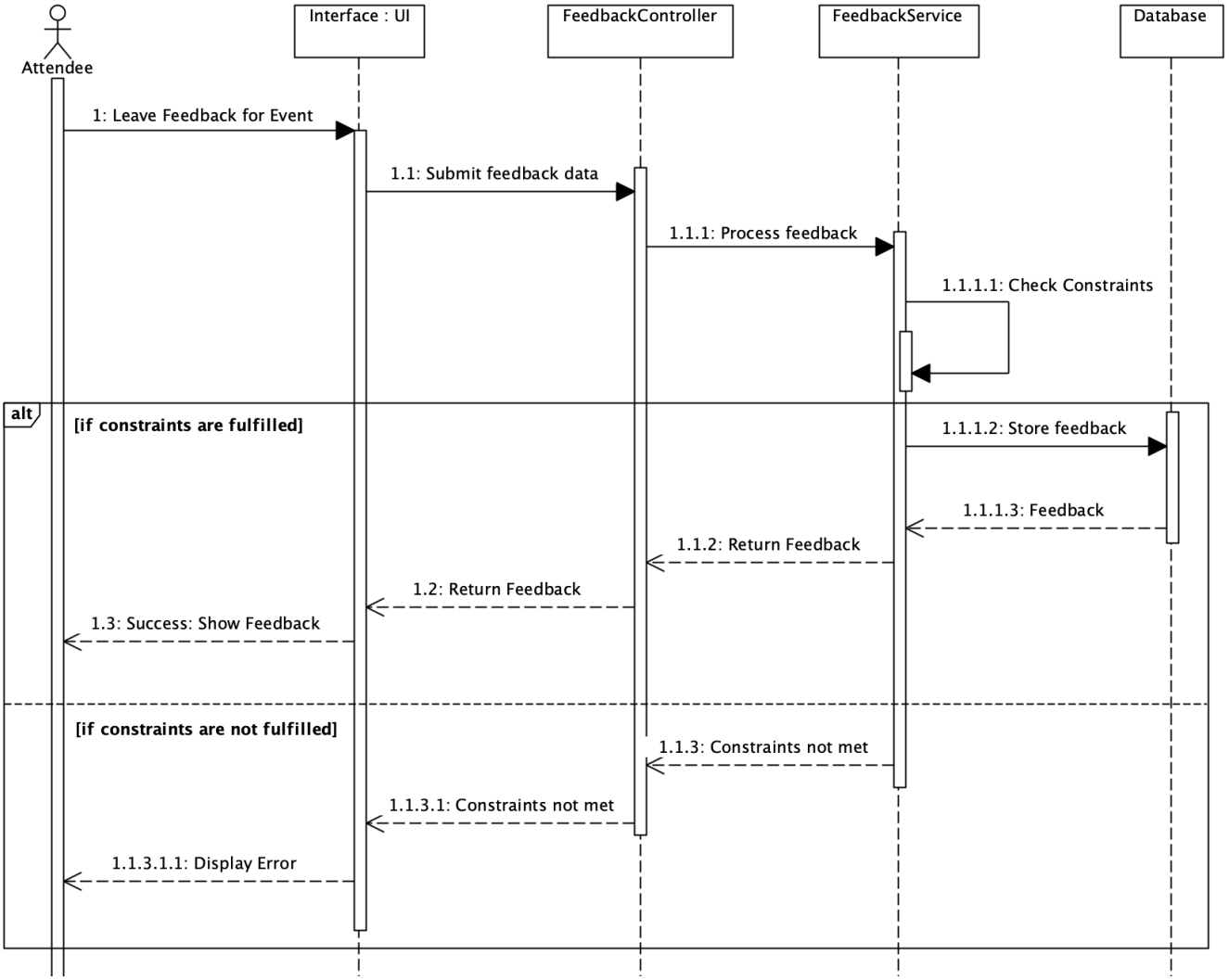


Figure 3.3.8.1: Sequence Diagram for Attende leaves feedback for event Event Report (see UC08-01)

SR9

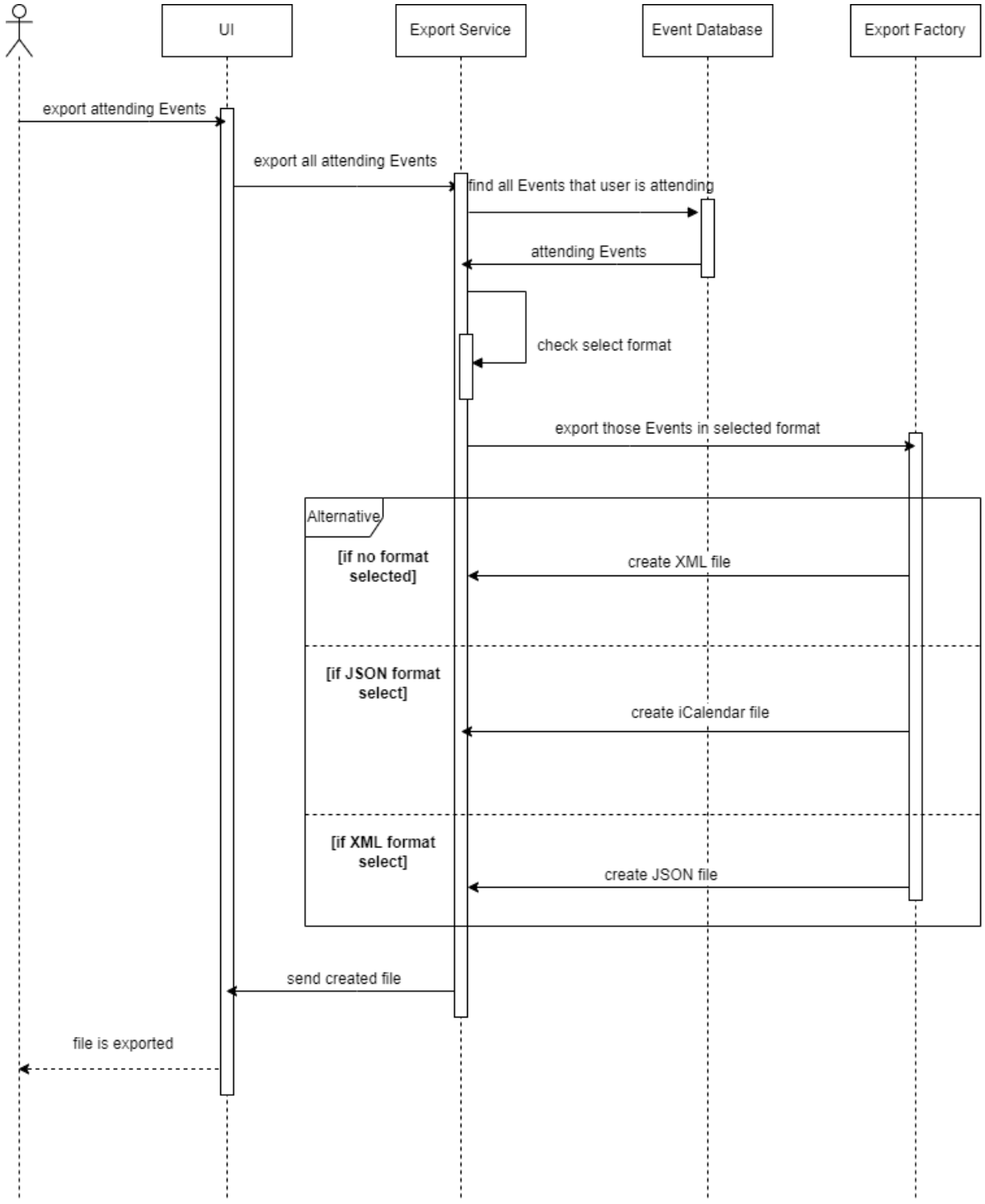


Figure 3.3.9.1: Sequence Diagram for exporting all attending Events

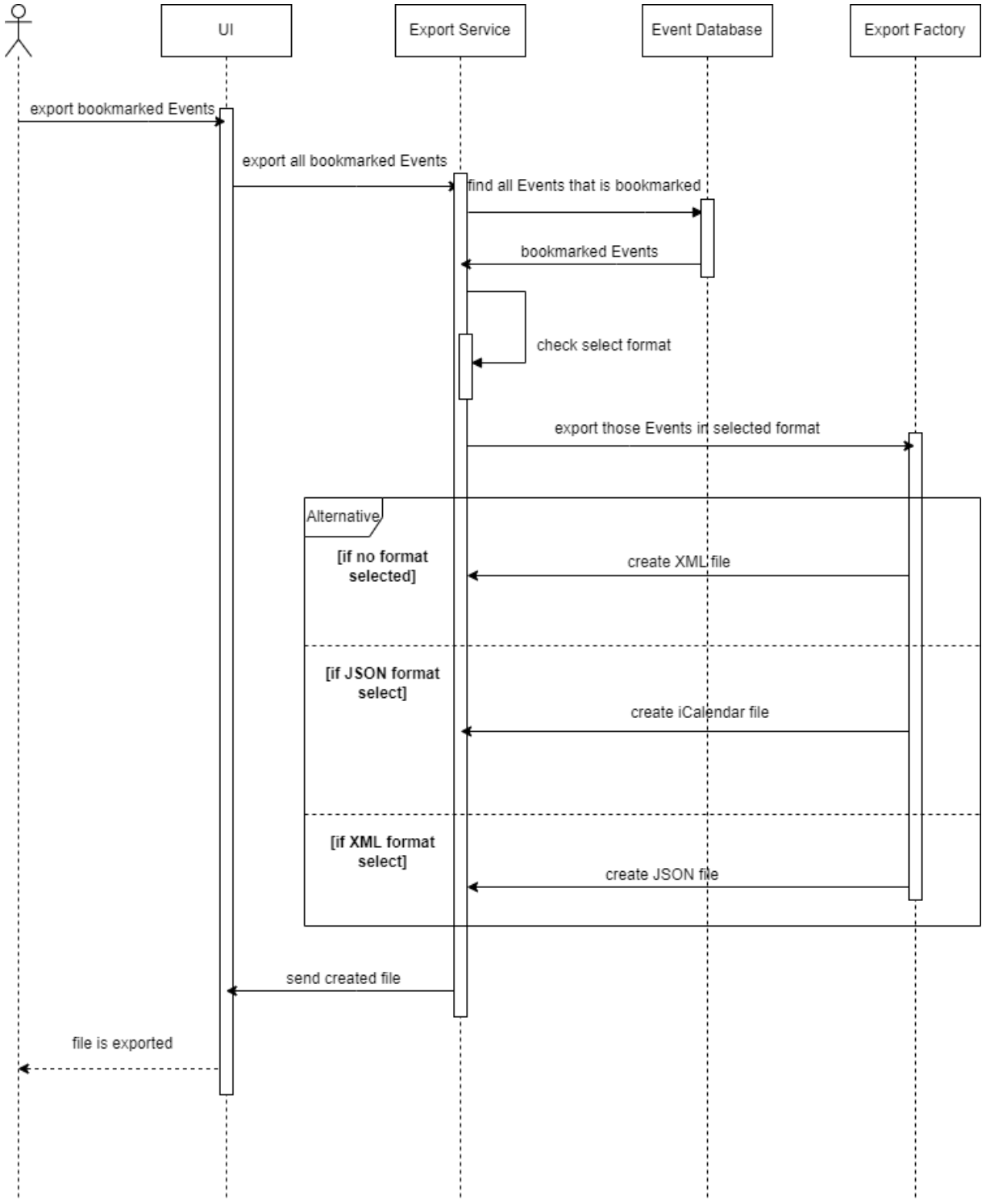


Figure 3.3.9.2: Sequence Diagram for exporting all bookmarked Events

SR10

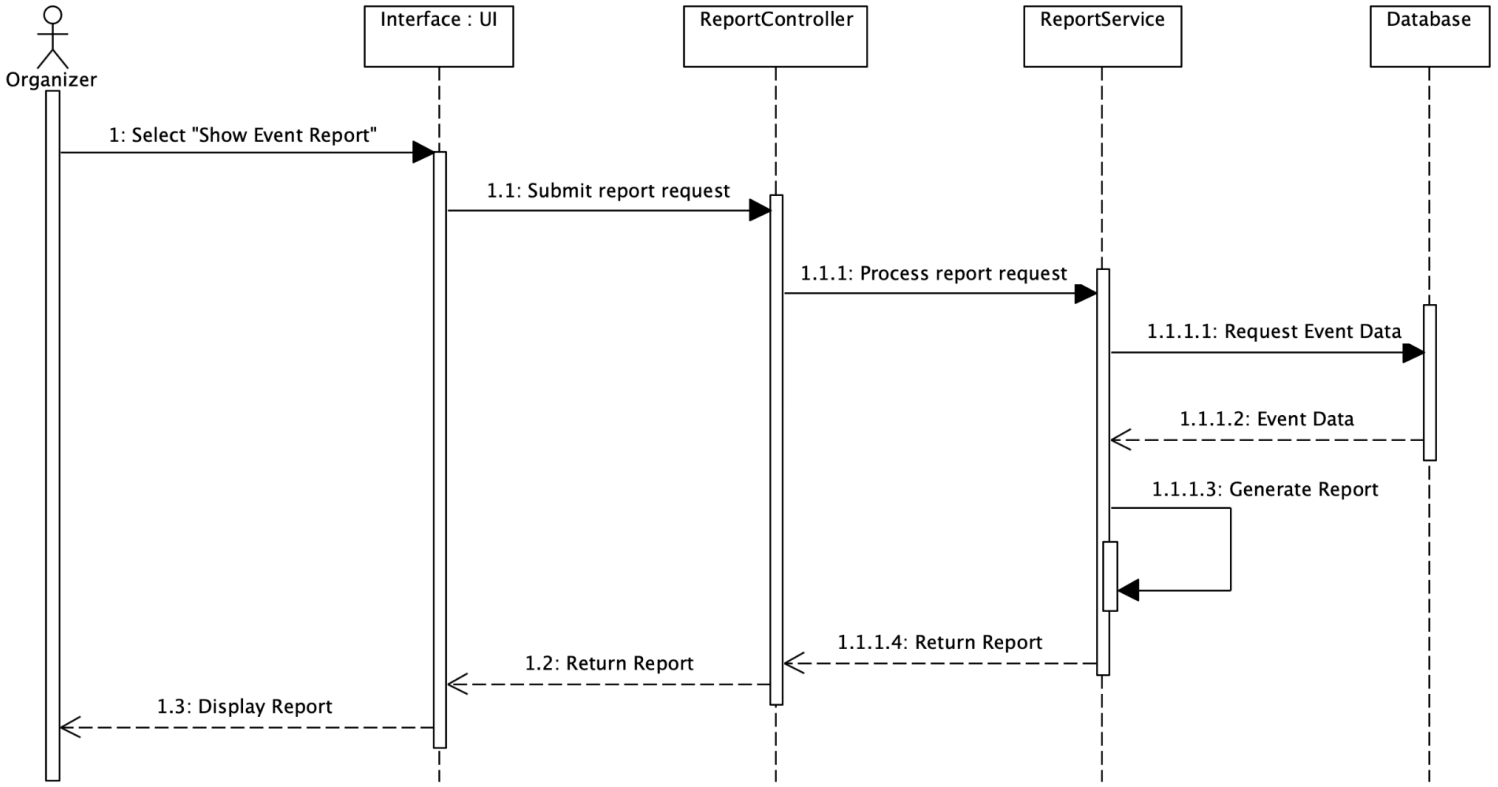


Figure 3.3.10.1: Sequence Diagram for Organizer requests Event Report (see UC10-01)

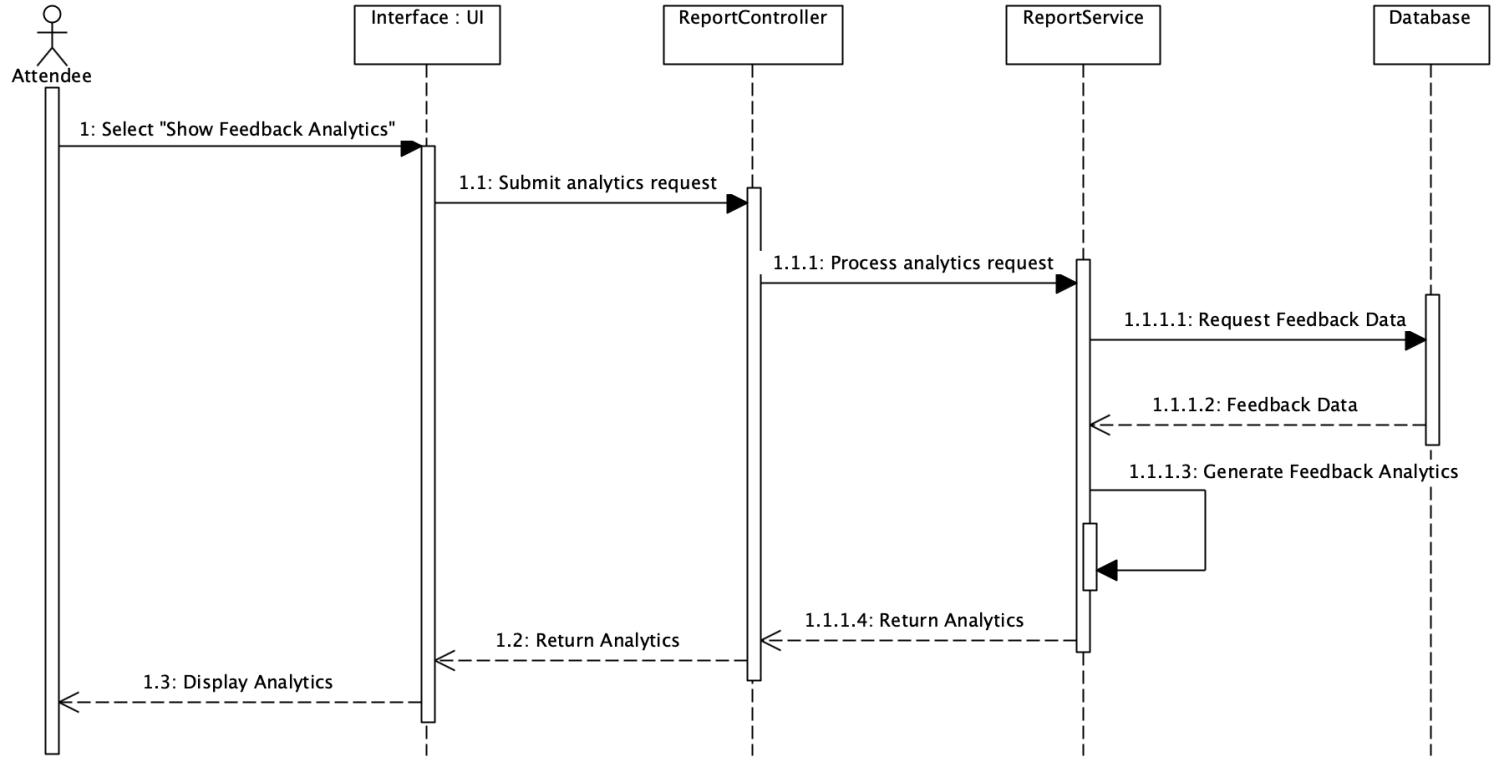


Figure 3.3.10.2: Sequence Diagram for Attendee requests Feedback Analytics (see UC10-02)

SR11

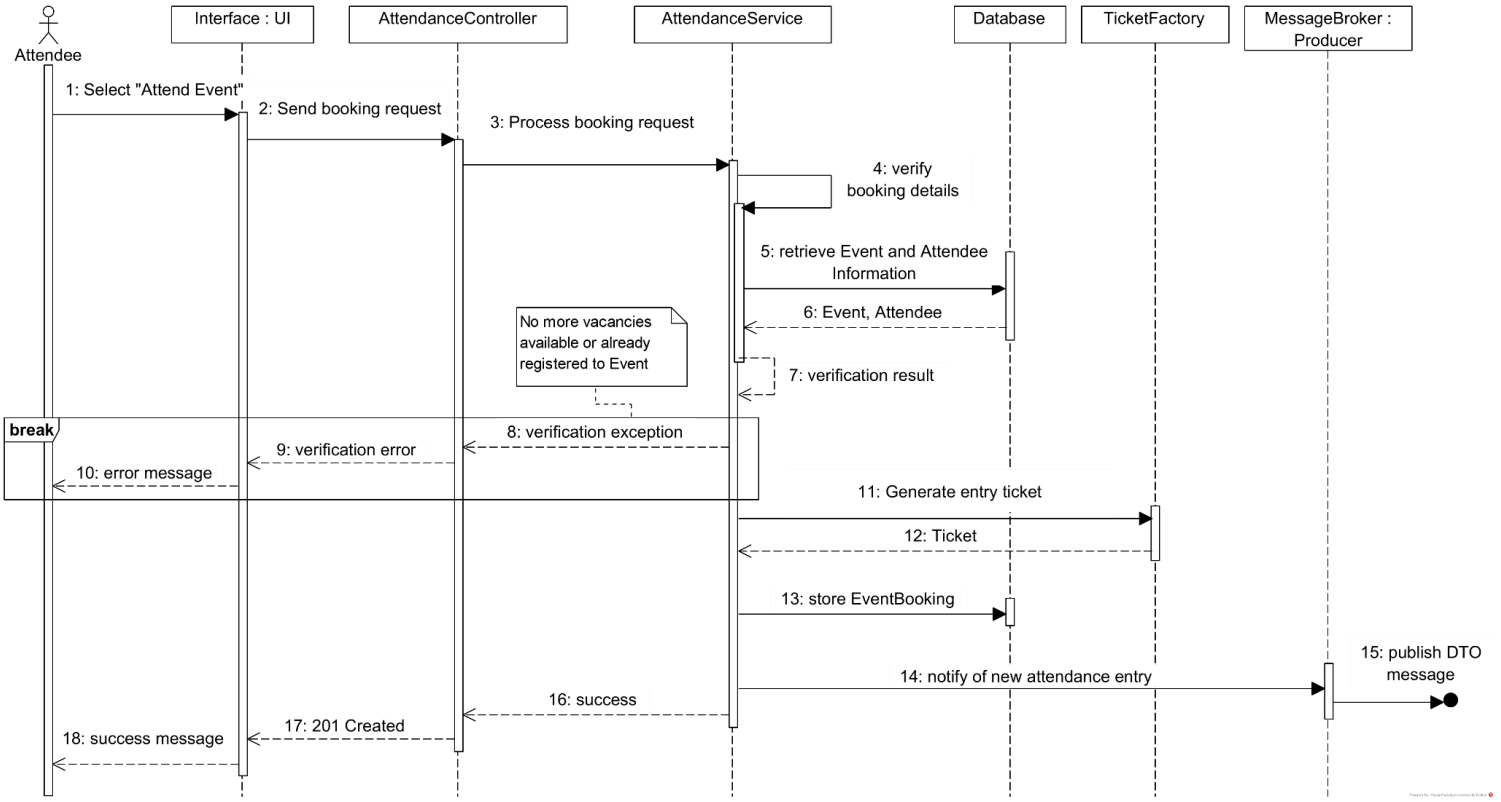


Figure 3.3.11.1: Sequence Diagram for Attendees attending (booking) Events (see UC11-01)

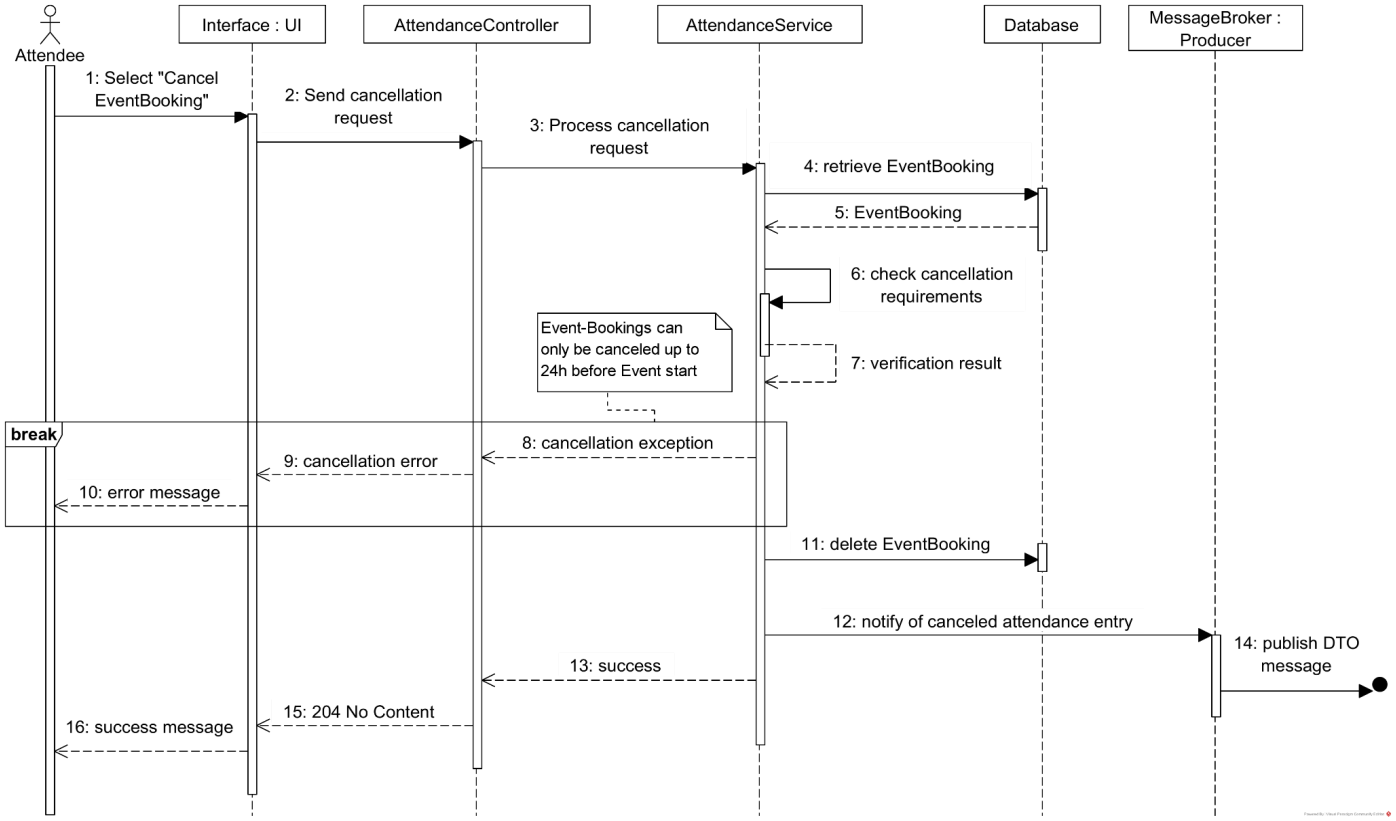


Figure 3.3.11.2: Sequence Diagram for Attendees canceling an EventBooking (see UC11-02)

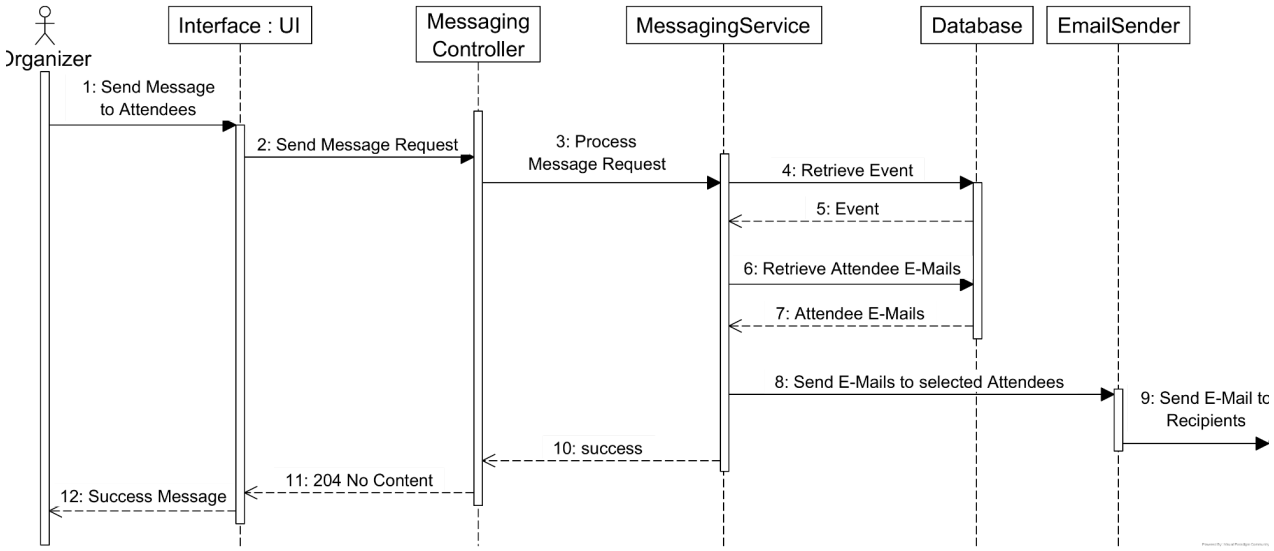


Figure 3.3.11.3: Sequence Diagram for Organizer sends message to Attendees (see UC11-03)

SR12

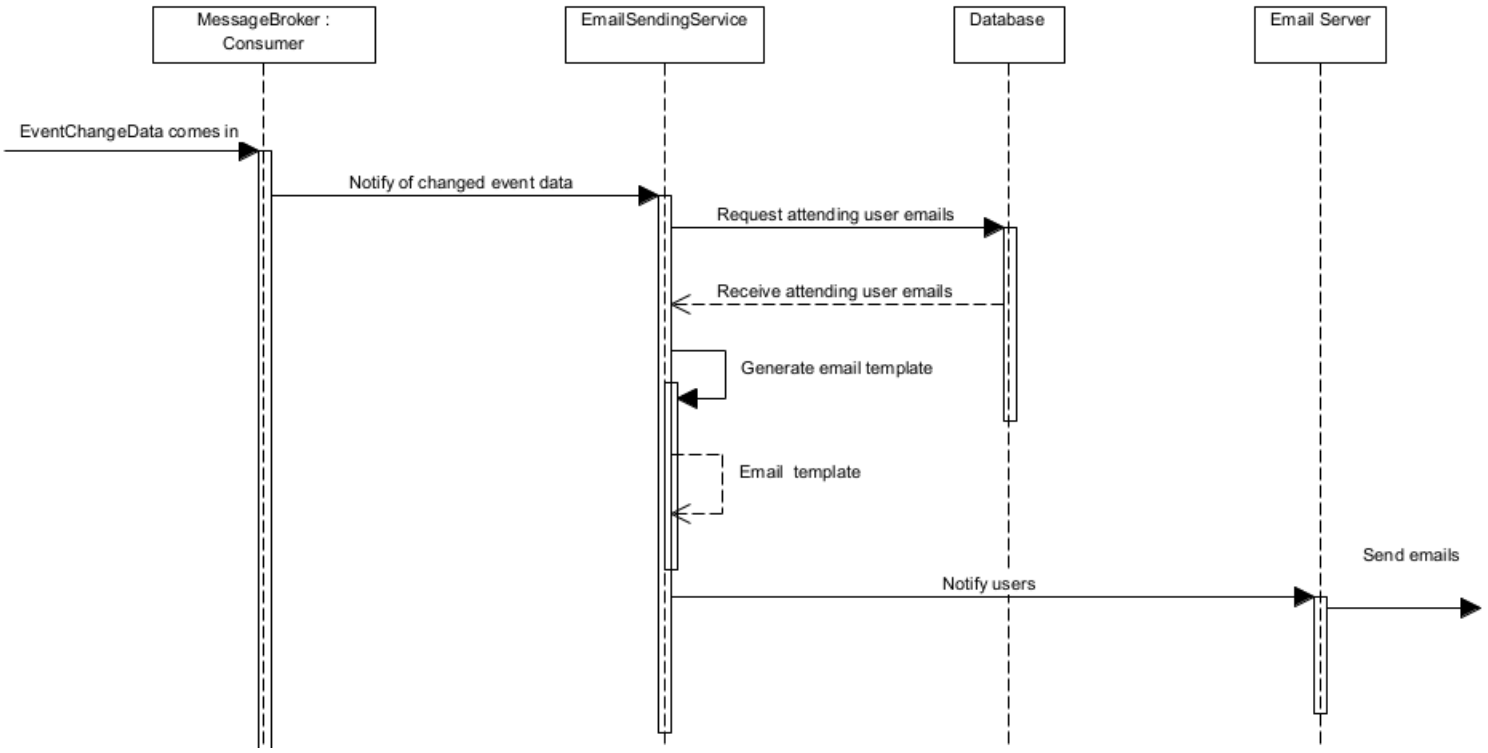


Figure 3.3.12.1: Sequence Diagram for Notification of attending or bookmarked event changes (see UC12-01)

3.4. Development View

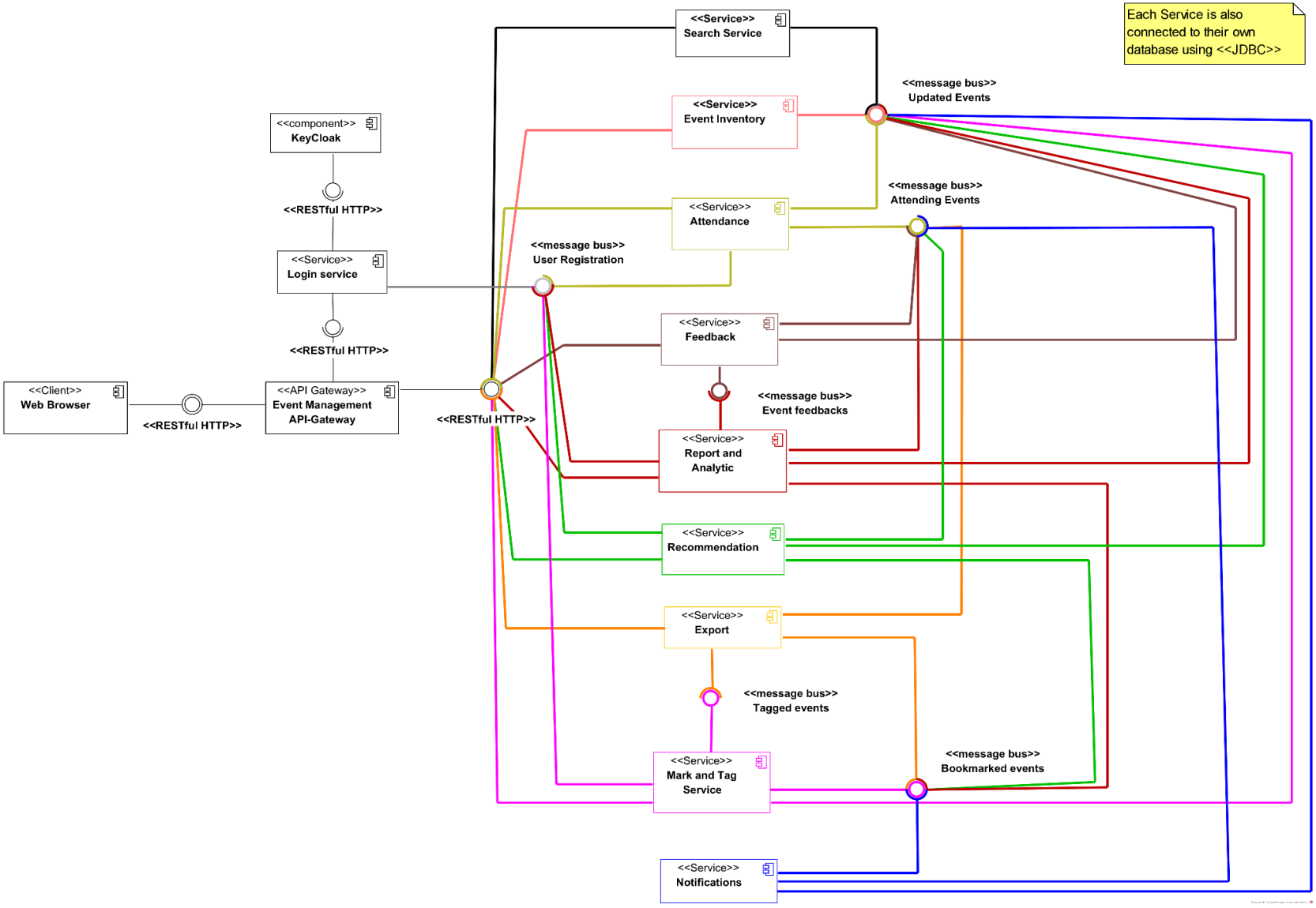


Figure 3.4.1: Component diagram for event management system

3.5. Physical View

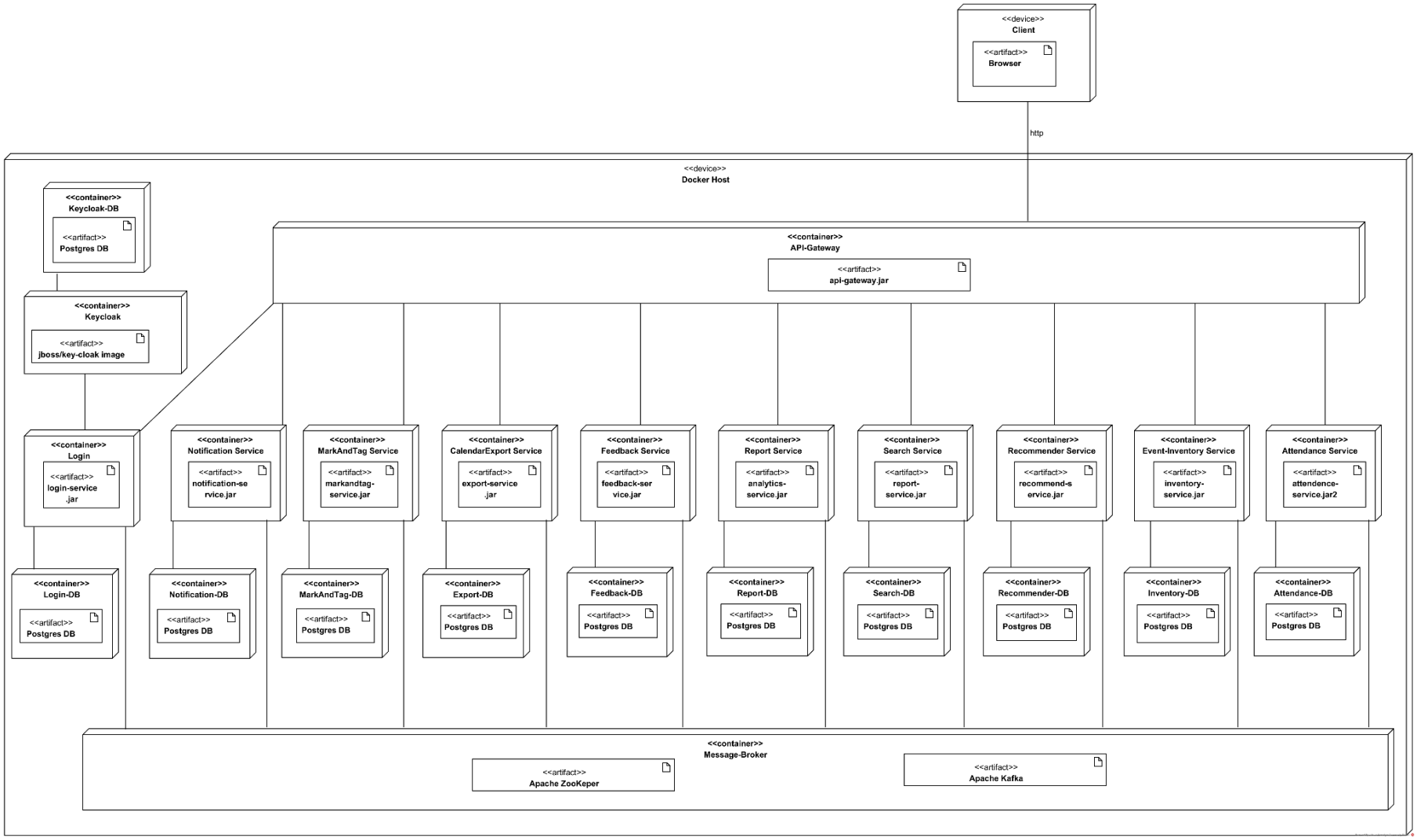
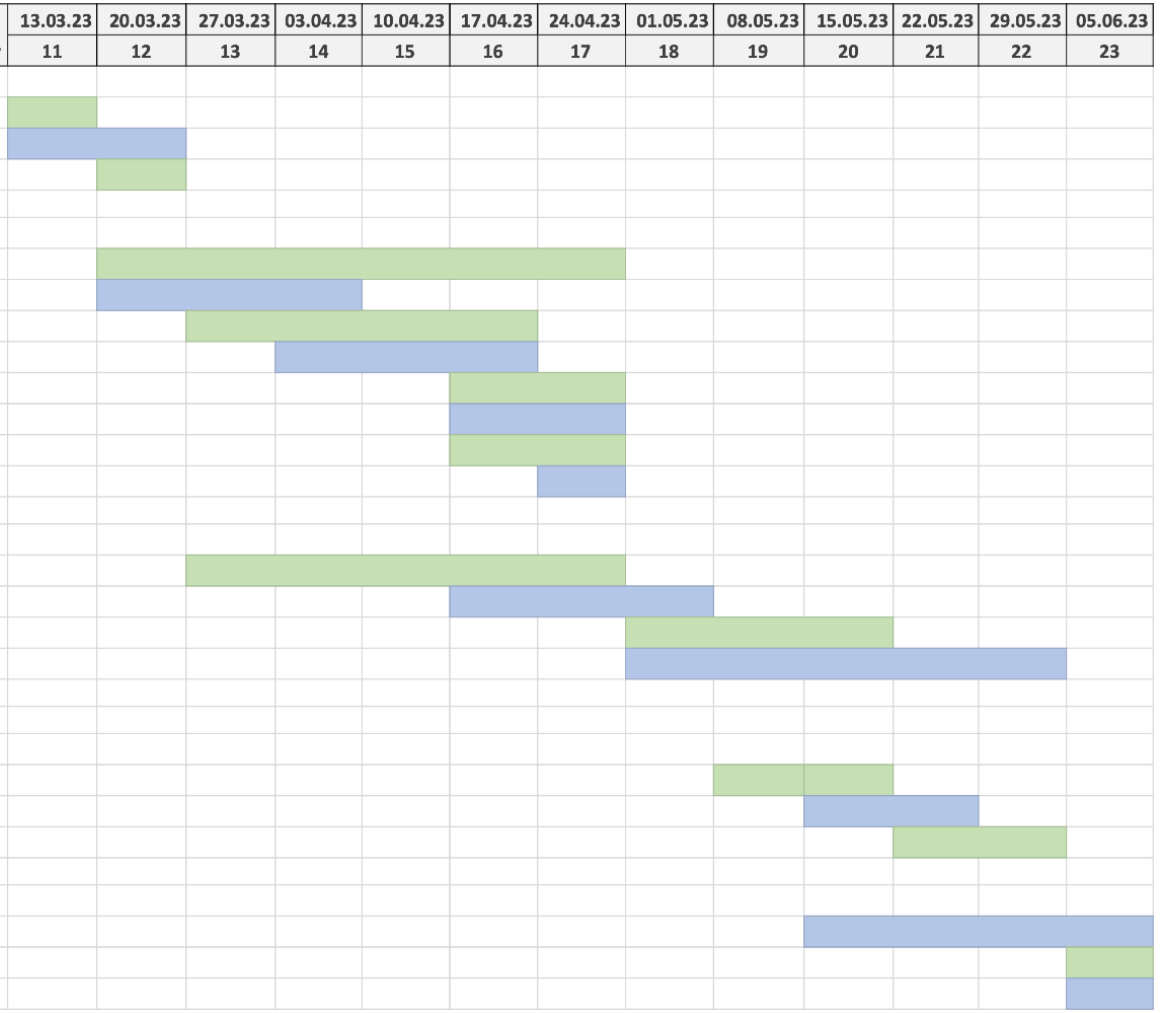


Figure 3.5.1: Deployment diagram for event management system

4. Team Contribution and Continuous development method

4.1. Project Tasks and Schedule

Task	Start	Finish	Duration	Priority	Responsibility
Analysis					
Requirements Analysis	10.03.23	13.03.23	3	1	ALL
Techstack Determination	13.03.23	20.03.23	7	1	ALL
Work Distribution	20.03.23	20.03.23	1	1	ALL
Design					
Indiviual Approach, Assumptions, Decisions	20.03.23	24.04.23	35		ALL
Use Case Definition and Description	20.03.23	03.04.23	14		ALL
Logical View	27.03.23	17.04.23	21		ALL
Process View	10.04.23	17.04.23	7		ALL
Development View	17.04.23	24.04.23	7		ALL
Deployment View	17.04.23	24.04.23	7		ALL
Documentation	17.04.23	24.04.23	7		ALL
DESIGN Submission	24.04.23	24.04.23	1		ALL
Development					
Mock-Up of each service	27.03.23	24.04.23	28		ALL
Containerization	17.04.23	05.05.23	18		ALL
Implementation CI/CD Pipeline	03.05.23	12.05.23	9		ALL
Implementation of each services	01.05.23	29.05.23	28		ALL
Testing					
Unit-Tests	08.05.23	15.05.23	7		ALL
Integration-Tests	15.05.23	22.05.23	7		ALL
End-to-end-Tests	22.05.23	29.05.23	7		ALL
Final Delivery					
Report	15.05.23	05.06.23	21		ALL
Final Project Submission	05.06.23	05.06.23	1		ALL
Project Presentation			1		ALL



4.2. Continuous Integration, Delivery and Deployment Plan

Explain how you continuously build, test, and deploy the iterative changes of your code, i.e., how you automate integration, delivery, deployment activities and describe – with the help of diagram(s) – your CI/CD workflow. Define the review/approval procedure among you.

CI/CD Pipeline. Source: https://www.redhat.com/en/topics/devops/what-cicd-pipeline

To optimally take advantage of CI/CD during the development, we plan to prioritize and set up the CI/CD pipeline early on. Not only does it help to speed up and enable a more efficient development process by automating building, testing and deployment of software, it also helps to create more robust and higher quality code and make it a core focus throughout the development process. Moreover, it perfectly complements the iterative and fast-paced development process of microservice by creating faster feedback loops and decreasing the time to market. [2]

Our proposed pipeline would include these steps: [3]

1. **Source Stage (CI):** A team member makes some changes to the source code and pushes the changes to GitLab
2. **Build Stage (CI)**: The CI/CD pipeline is triggered and automatically builds the artifact. In our case this would include building the JAR and then packaging the application code into a docker image.
3. **Test Stage (CI):** Available Tests are automatically executed for the built artifact. In our case this would include Unit-Tests as well as Integration Tests. The exact extent of the testing done in this stage still needs to be further refined.
4. **Deployment Stage (CD)**: If the previous stages were successful, the code can be merged into the repository. Additionally, this could also include pushing the container image to a registry, such as Docker Hub. Whether these changes are then automatically deployed to production depends on the choice of Continuous Delivery and Continuous Deployment:

Continuous Delivery: The changes need to be manually reviewed and approved to put into production.

Continuous Deployment: All changes are automatically put into production.

[2] Source: https://docs.gitlab.com/ee/ci/introduction/index.html#continuous-integration
[3] Source: https://www.guru99.com/ci-cd-pipeline.html

We decided to use Continuous Deployment for our project to increase the speed in which changes can be pushed to production.

4.3. Distribution of Work and Efforts

Responsibility	Team Member
SR1 + SR2	ALL
SR3 + SR12	Fritz Ziernhöld
SR4 + SR5	LEFT THE TEAM
SR6 + SR9	Jin Yu
SR7 + SR11	Matthias Fritz
SR8 + SR10	David Reichert

Additional note:

As stated in the GitLab issue, one of our team members left abruptly prior to the submission of the design phase. Therefore, his responsibilities SR4 and SR5 were left out as long as the system's design allows it.

Team Member: David Reichert	
DESIGN	**Time spent**
Digital Meetings	15h
Research	3h
Design Assumptions, Design Decisions, System Requirements	4h
Use Case Descriptions	1h
4+1 Model: Diagrams	13h
Mock-Up Implementation + Documentation	12h
Others	3h
Overall	51h
FINAL Estimation	**Estimation**
Meetings (20h) , Implementation (25h), Testing (6h), Documentation (4h)	47h

Team Member: Fritz Ziernhöld	
DESIGN	**Time spent**
Digital Meetings	15h
Research	10h
Design Assumptions, Design Decisions, System Requirements	10h
Use Case Descriptions	2h
4+1 Model: Diagrams	6h
Mock-Up Implementation + Documentation	3h
Others	2h
Overall	43h
FINAL Estimation	**Estimation**
Meetings (20h) , Implementation (30h), Testing (15h), Documentation (5h)	70h

Team Member: Matthias Fritz	
DESIGN	**Time spent**
Digital Meetings	15h
Research	7h
Design Assumptions, Design Decisions, System Requirements	10h
Use Case Description	3h
4+1 Model: Diagrams	16h
Mock-Up Implementation + Documentation	25h
Others	4h
Overall	80h
FINAL Estimation	**Estimation**
Meetings (20h) , Implementation (12h), Testing (14h),	56h

CI/CD-Pipeline (6h), Documentation (4h)	

Team Member: Yu Jin	
DESIGN	**Time spent**
Digital Meetings	15h
Research	6h
Design Assumptions, Design Decisions, System Requirements	10h
Use Case Description	5h
4+1 Model: Diagrams	10h
Mock-Up Implementation + Documentation	30h
Others (mostly docker setups)	12h
Overall	88h
FINAL Estimation	**Estimation**
Meetings (20h), implementation (10h~12h), Testing (6h), Documentation (4h)	41h

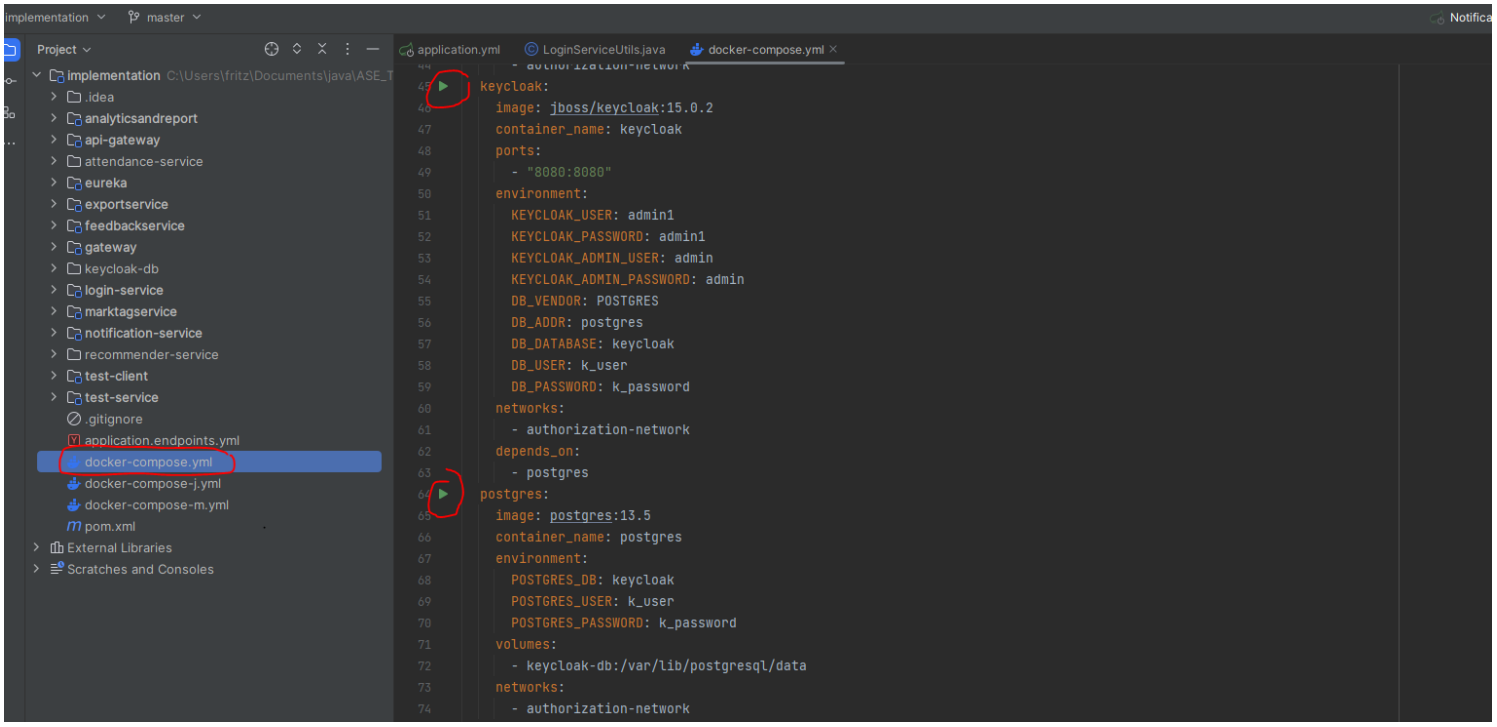
5. How-to / mock-up documentation

In the following, we document the mock-up implementation for the respective microservices. The individual documentation and how-tos require an initial 'git clone' of the project:

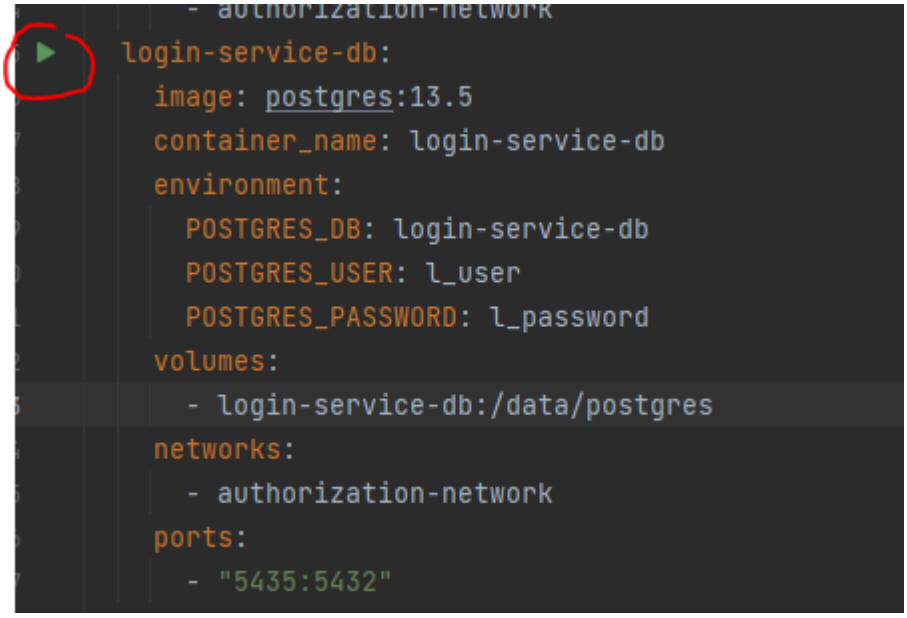
```
git clone
https://git01lab.cs.univie.ac.at/vu-advanced-software-engineering/students/2023s/ASE_Team_0601.git
```

How-to SR3

To test the mockup of the login service, you first need to start keycloak. For that, navigate to the docker-compose.yml in the parent folder, run the postgres database first then run keycloak. Make sure that the postgres database is running before running keycloak because the keycloak is dependent on that database since it contains configuration files as well. All relevant files and buttons you need to press are circled in red!



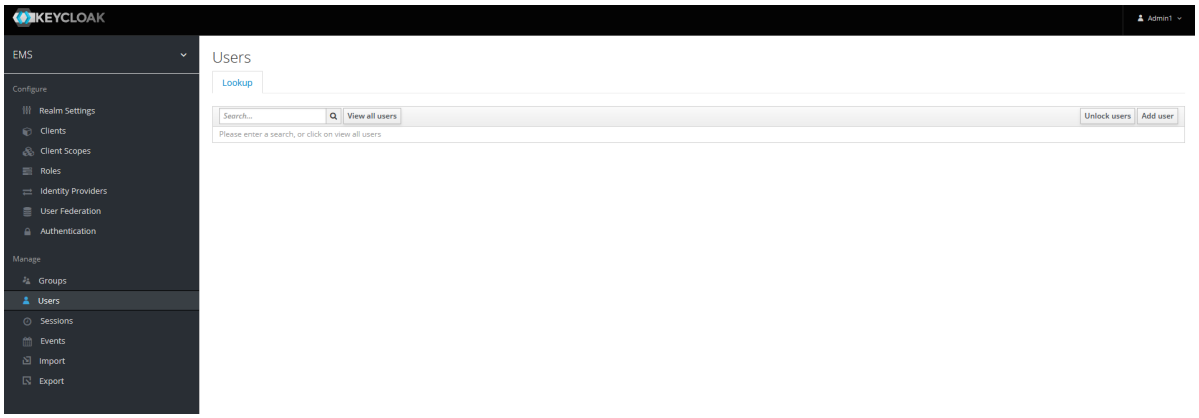
In the same docker-compose.yml, start the login-service db:



When all 3 docker containers are running you can access the Keycloak admin console on **localhost:8080/auth/** open the admin console and you can access keycloak with the following credentials:

Username: admin1
Password: admin1

For demonstrations, navigate to the “Users” tab:

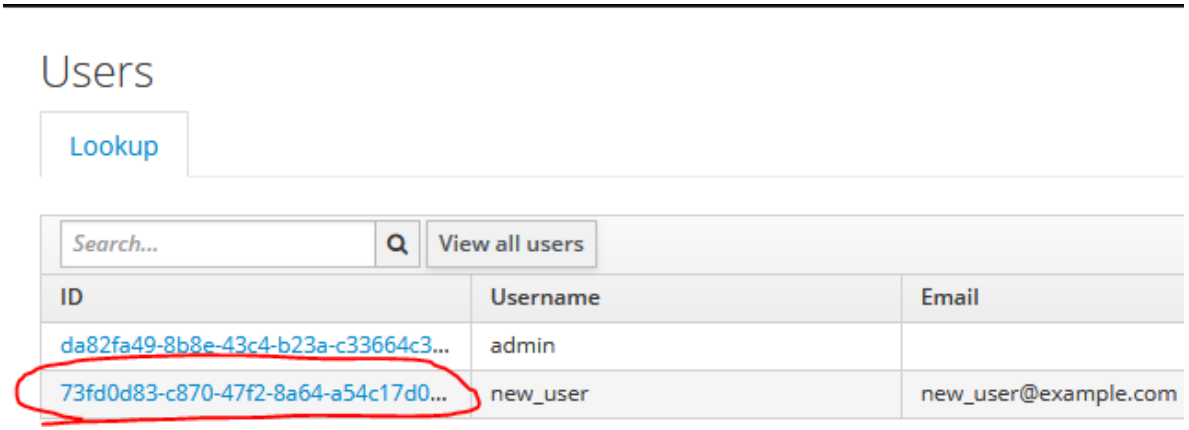


Navigate to the ‘login-service’ in the parent project and run the application.

To create a user, send the following request to the service endpoint ‘/management/users’:

```
curl --location 'localhost:8083/management/users' \
--header 'Content-Type: application/json' \
--data-raw '{
  "username": "new_user",
  "email": "new_user@example.com",
  "firstName": "New",
  "lastName": "User",
  "enabled": true,
  "emailVerified": true,
  "attributes": {
    "key": "value"
  },
  "credentials": [
    {
      "type": "password",
      "value": "new_user_password",
      "temporary": false
    }
  ]
}'
```

Now you should see the newly created user in the keycloak “Users”” tab. To see the new user, press “View all users” and the new user should appear. To delete the user, copy its keycloakId and send a request to /management/users/<keycloakId>:




```
curl --location --request DELETE
'localhost:8083/management/users/ebc674de-5a90-498e-a4d1-6e0ea27b77
b9'
```

Lastly, to get an access token for the a user, do the following:

Create a new user (if you have deleted the newly created one) and call the following endpoint ‘/login/oauth/token’ with your username and password:

```
curl --location
'localhost:8083/login/oauth/token?username=new_user&password=new_us
er_password'
```

How-to SR7 and SR11:

Both services and all their dependencies (message broker, databases) can be started by just executing: **`docker compose -f docker-compose-m.yml up`**

Use-Case demonstrations SR7

0. **Prerequisite**: HTTP POST to http://localhost:8087/mock

- curl --location --request POST 'http://localhost:8087/mock'
- This fills the database with some test data (5 Events & 5 Attendees with IDs = {1,2,3,4,5}) to execute the below use-cases

1. Get personalized recommendations for Attendee (UC07-02)

- HTTP GET: http://localhost:8087/attendees/{1}/recommendations
- Body (JSON): {“eventId” : {id}}

Example Call:

```
curl --location 'http://localhost:8087/attendees/4/recommendations'
```

Example Result (abridged):


```
[
    {
        "eventId": 0,
        "name": "Event0",
        "type": "Football",
        "city": "Vienna",
        "country": "Austria",
        "startDate": "2023-04-24T19:46:18.894612Z",
        "endDate": "2023-04-26T19:46:18.894612Z"
    },
    {
        "eventId": 1,
        "name": "Event1",
        "type": "Football",
        "city": "Vienna",
        "country": "Austria",
        "startDate": "2023-04-24T19:46:18.903619Z",
        "endDate": "2023-04-26T19:46:18.903619Z"
    },
    {
```


2. Get notified when new event is suggested to Attendee (UC07-01)

→ Not testable via an endpoint, as it is even-based and relies on messages over other services (namely SR4).

3. Change EMail suggestion preference

- HTTP PATCH to http://localhost:8087/attendees/{id}/preferences
- Body (application/json): { “receivePromtionalEmails” : true | false}

Example Call

```
curl --location --request PATCH 'http://localhost:8087/attendees/1/preferences' \
--header 'Content-Type: application/json' \
--data '{ "receivePromotionalEmails": false}'
```

Example Result: 204 No Content

Use-Case demonstrations SR11

0. **Prerequisite**: HTTP POST to http://localhost:8091/mock

- curl --location --request POST 'http://localhost:8091/mock'
- This fills the database with some test data (5 Events & 5 Attendees with IDs = {1,2,3,4,5}) to execute the below use-cases

1. Attend an Event (UC11-01):

- HTTP POST: http://localhost:8091/attendees/{1}/event-bookings
- Body (application/json): {“eventId” : {id}}

Example Call:

```
curl --location 'http://localhost:8091/attendees/1/event-bookings' \
--header 'Content-Type: application/json' \
--data '{ "eventId": 1}'
```

Example Result:


```
{
    "id": {
        "attendeeId": 1,
        "eventId": 1
    },
    "entryTicket": {
        "id": "75d260c3-23b6-4662-a6c6-3219d97f88d0",
        "ticketType": "GeneralAdmission",
        "validFrom": "2023-04-23T22:37:37.463298Z",
        "validTo": "2023-04-25T22:37:37.463305Z"
    },
    "bookingTimestamp": "2023-04-21T22:42:58.200859407Z"
}
```

2. Retrieve all EventBookings of an Attendee

- HTTP GET: http://localhost:8091/attendees/{id}/event-bookings

Example Call:

```
curl --location 'http://localhost:8091/attendees/1/event-bookings'
```

Example Result:


```
[
    {
        "attendeeId": 1,
        "attendeeName": "Attendee Test1",
        "eventId": 1,
        "eventName": "EventName1",
        "ticketSerialNr": "ad669857-25a1-4dba-ae1b-8b654cc9c977",
        "ticketType": "GeneralAdmission",
        "validFrom": "2023-04-23T22:20:00.852627Z",
        "validTo": "2023-04-25T22:20:00.852627Z",
        "links": [
            {
                "rel": "QR-Code",
                "href": "http://localhost:8091/attendee/1/tickets/ad669857-25a1-4dba-ae1b-8b654cc9c977"
            }
        ]
    }
]
```


By following the link provided in the response, the QR-Code of the entrance ticket can be retrieved.

3. Delete the (previously created) EventBooking (UC11-02)

- HTTP DELETE:
  http://localhost:8091/attendees/{attendeeId}/event-bookings?eventId={eventId}

Example Call: (returns 204 No Content)

```
curl --location --request DELETE
'http://localhost:8091/attendees/1/event-bookings?eventId=1'
```

4. Organizer sends messages to Attendees (UC11-03)

- HTTP POST: http://localhost:8091/events/{id}/messages

Example Call:

```
curl --location 'http://localhost:8091/events/1/messages' \
--header 'Content-Type: application/json' \
--data '{
      "organizerId" : 1,
      "message" : "hello world",
      "recipientIds":[1, 2, 3]}'
```

Example Result: 204 No Content

How-to SR6 and SR9:

0. Prerequisite setup step by step:

1. Open (Install) Docker Desktop to get Docker running
2. Executing in shell cmd in the directory of implementation: **docker compose -f docker-compose-j.yml up**

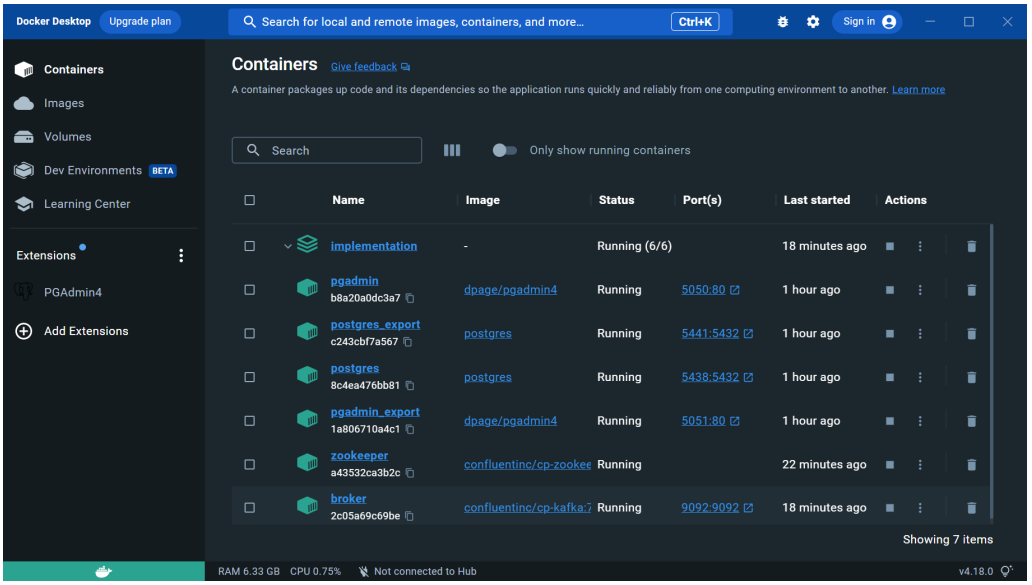


Figure 5.6.1-1: all docker containers running

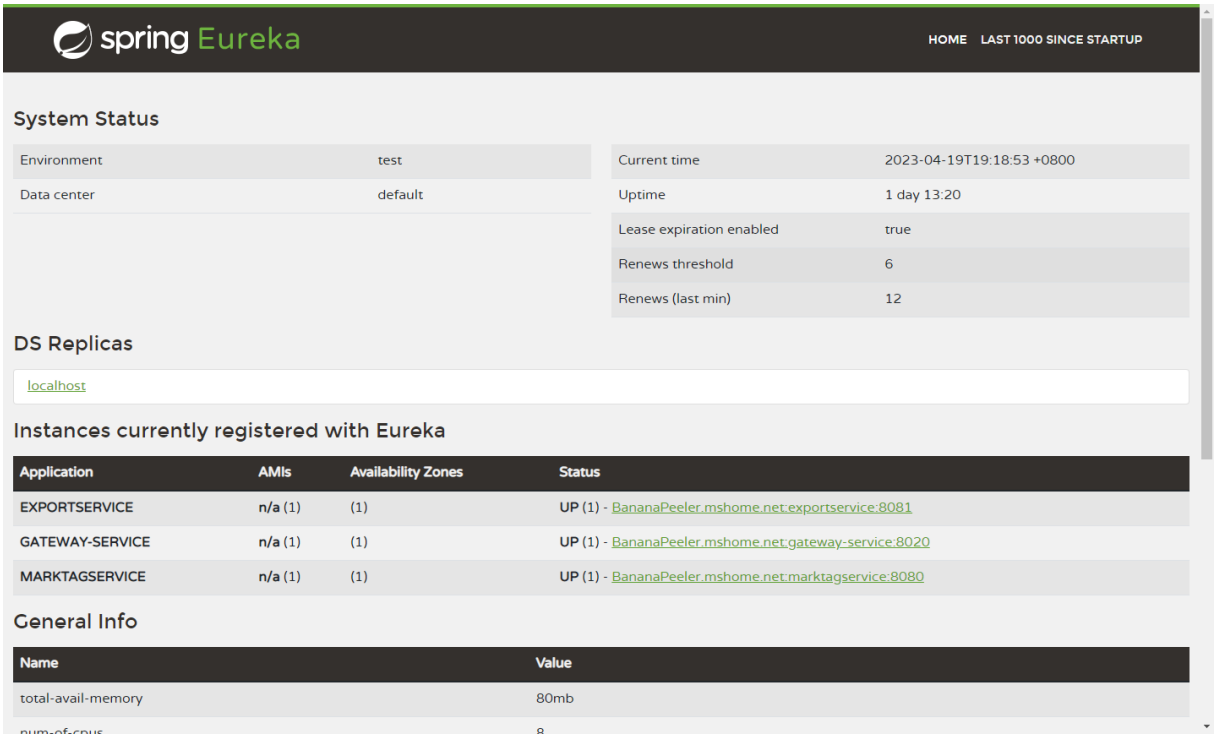


Figure 5.6.2: Eureka Server with all services (SR6, SR9 and gateway) running

3. Start the **EurekaApplication** to start the discovery server.
4. Start the **GatewayApplication** to enable the gateway.
5. Start both **ExportserviceApplication** and **MarktagserviceApplication**
6. To test if everything is working correctly before mock up test:

- Endpoint: http://localhost:8761/

  Expected result: All registered clients should be displayed in the Eureka discovery server (as shown in *Figure 5.2*).

- Endpoint: http://localhost:8020/marktag-service/marktag-service

  Expected result: message “hello im mark and tag service” should be displayed.

- Endpoint: http://localhost:8020/export-service/export-service

  Expected result: message “hello im export service” should be displayed.

- For extra database check with pgAdmin (There has been some data generated in the main application files in each services as mock ups, details see *Figure 5.6.3 - 5.6.7*):
  - Endpoint: http://localhost:5051/browser/ could check the data persisted in export service.
  - Endpoint: http://localhost:5050/browser/ could check the data persisted in mark and tag service.

	event_id [PK] integer	bookmark boolean	global_id integer	attendee_id integer
1	1	false	0	1
2	2	false	10	1
3	3	false	20	1
4	4	false	30	1
5	5	false	40	1
6	6	false	50	1
7	7	false	60	1
8	8	false	70	1
9	9	false	80	1
10	10	false	90	1

*Figure 5.6.3: Expected data entries in table “events” in **mark and tag service**.*

	attendee_id [PK] integer	global_id integer
1	1	0
2	2	1
3	3	2
4	4	3
5	5	4
6	6	5
7	7	6
8	8	7
9	9	8
10	10	9

*Figure 5.6.4: Expected data entries in table “attendees” in **mark and tag service**.*

	event_id [PK] integer	tags character varying[] (255)	date character varying (255)	global_event_id integer	location character varying (255)	name character varying (255)
1	1	{FOOD,SPORT,EDUCATION}	2023/3/6	1	Vienna	name of event1
2	2	{FOOD,SPORT,EDUCATION}	2023/3/9	2	Graz	name of event2
3	3	{FOOD,SPORT,EDUCATION}	2023/3/12	3	Linz	name of event3

*Figure 5.6.5: Expected data entries in table "event" in **export service**.*

	attending_event_id [PK] integer	global_attendee_id integer	global_event_id integer
1	1	1	1
2	2	1	2

*Figure 5.6.6: Expected data entries in table "attending" in **export service**.*

	bookmark_event_id [PK] integer	global_attendee_id integer	global_event_id integer
1	1	1	2

*Figure 5.6.7: Expected data entries in table "bookmark" **export service**.*

Use-Case demonstrations SR6:

1. Bookmark an Event (UC06-01)

- HTTP PUT: http://localhost:8020/marktag-service/user/0/event/70/bookmark

Expected outcome:

- HTTP GET: http://localhost:8020/marktag-service/user/0/bookmarked-events

  [{"id":8,"bookmark":true,"globalId":70,"tags":["SPORT"]}]

- HTTP GET: http://localhost:8020/marktag-service/user/0/event/70/isbookmarked

  true

- SELECT * FROM events;

	event_id [PK] integer	bookmark boolean	global_id integer	attendee_id integer
1	1	false	0	1
2	2	false	10	1
3	3	false	20	1
4	4	false	30	1
5	5	false	40	1
6	6	false	50	1
7	7	false	60	1
8	9	false	80	1
9	10	false	90	1
10	8	true	70	1

2. Unbookmark an Event (UC06-02)

- HTTP PUT: http://localhost:8020/marktag-service/user/0/event/70/unbookmark

Expected outcome:

- HTTP GET: http://localhost:8020/marktag-service/user/0/bookmarked-events

  []

- HTTP GET: http://localhost:8020/marktag-service/user/0/event/70/isbookmarked

  false

- SELECT * FROM events;

	event_id [PK] integer	bookmark boolean	global_id integer	attendee_id integer
1	1	false	0	1
2	2	false	10	1
3	3	false	20	1
4	4	false	30	1
5	5	false	40	1
6	6	false	50	1
7	7	false	60	1
8	9	false	80	1
9	10	false	90	1
10	8	false	70	1

3. Add a SPORT Tag to an Event (UC06-03)

- HTTP PUT: http://localhost:8020/marktag-service/user/0/event/70/add/SPORT

Expected outcome:

- HTTP GET: http://localhost:8020/marktag-service/user/0/event/70/tags

  ["SPORT"]

- SELECT * FROM event_tags;

	event_event_id integer	tags smallint
1	8	0

4. Remove the SPORT Tag to an Event (UC06-04)

- HTTP PUT: http://localhost:8020/marktag-service/user/0/event/70/remove/SPORT

Expected outcome:

- HTTP GET: http://localhost:8020/marktag-service/user/0/event/70/tags

- SELECT * FROM event_tags;

	event_event_id integer	tags smallint

Use-Case demonstrations SR9:

1. Export all attending Events by a User as a Calendar file (UC09-02)

- HTTP GET: http://localhost:8081/export-service/user/1/export/attending-events

Expected outcome:

An **events.ics** file should be downloaded through browser

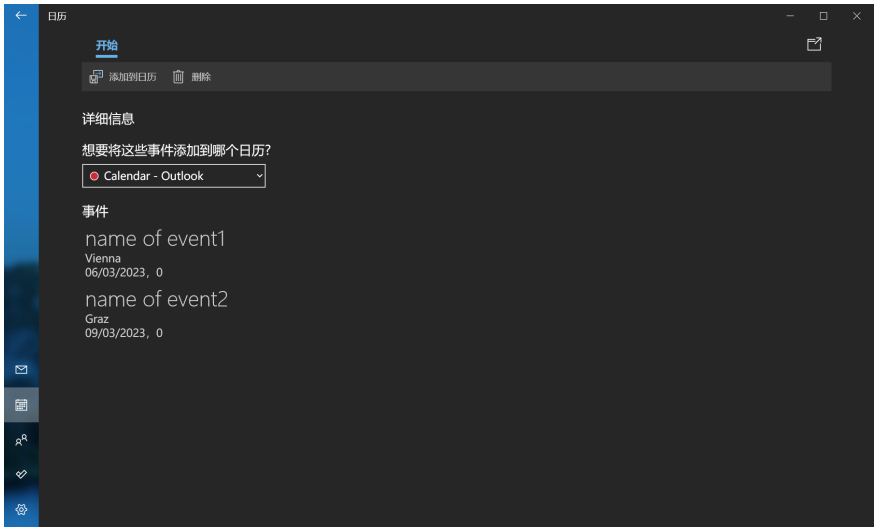


Figure 5.6.7: events.ics file when opened

2. Export all attending Events by a User as a JSON file (UC09-02)

- HTTP GET: http://localhost:8081/export-service/user/1/export/attending-events/JSON

Expected outcome:

An **events.json** file should be downloaded through browser and should contain the following information:


```
[{"id":1,"globalId":1,"name":"name of
event1","location":"Vienna","date":"2023/3/6","tags":["FOOD","SPORT","EDUCATION"]},{"id":
2,"globalId":2,"name":"name of
event2","location":"Graz","date":"2023/3/9","tags":["FOOD","SPORT","EDUCATION"]}]
```


3. Export all attending Events by a User as a XML file (UC09-02)

- HTTP GET: http://localhost:8081/export-service/user/1/export/attending-events/JSON

Expected outcome:

An **events.xml** file should be downloaded through browser and should contain the following information:

```xml
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<events>
    <events>
        <date>2023/3/6</date>
        <globalId>1</globalId>
        <id>1</id>
        <location>Vienna</location>
        <name>name of event1</name>
        <tags>FOOD</tags>
        <tags>SPORT</tags>
        <tags>EDUCATION</tags>
    </events>
    <events>
        <date>2023/3/9</date>
        <globalId>2</globalId>
        <id>2</id>
        <location>Graz</location>
        <name>name of event2</name>
        <tags>FOOD</tags>
        <tags>SPORT</tags>
        <tags>EDUCATION</tags>
    </events>
</events>
```

4. Export all bookmarked Events by a User as a Calendar file (UC09-01)

- HTTP GET: http://localhost:8081/export-service/user/1/export/bookmarked-events

Expected outcome:

An **events.ics** file should be downloaded through browser

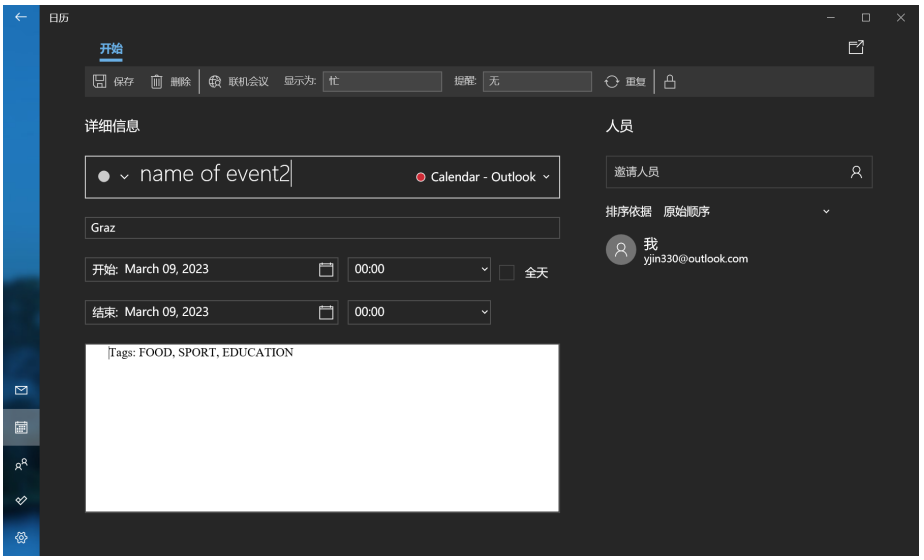


Figure 5.6.8: events.ics file when opened

5. Export all bookmarked Events by a User as a JSON file (UC09-01)

- HTTP GET: http://localhost:8081/export-service/user/1/export/bookmarked-events/JSON

Expected outcome:

An **events.json** file should be downloaded through browser and should contain the following information:


```
[{"id":2,"globalId":2,"name":"name of
event2","location":"Graz","date":"2023/3/9","tags":["FOOD","SPORT","EDUCATION"]}]
```


6. Export all bookmarked Events by a User as a XML file (UC09-01)

- HTTP GET: http://localhost:8081/export-service/user/1/export/bookmarked-events/XML

Expected outcome:

An **events.xml** file should be downloaded through browser and should contain the following information:


```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<events>
    <events>
        <date>2023/3/9</date>
        <globalId>2</globalId>
        <id>2</id>
        <location>Graz</location>
        <name>name of event2</name>
        <tags>FOOD</tags>
        <tags>SPORT</tags>
        <tags>EDUCATION</tags>
    </events>
</events>
```


Use-Case demonstrations SR8:

0. Prerequisite:

- Execute the following command inside the implementation folder to build and start the Docker containers.

```
docker compose -f docker-compose-d.yml up
```

- 10 example events and 10 example attendees are created by default for documentation purposes
- Attend one of the created events via HTTP GET request, e.g.:

```
curl -X GET http://localhost:8088/attendees/1/event/attend/1
```

1. UC08-01: Attendees can add a feedback and rate events

- Send a HTTP POST request to the according endpoint, e.g.:

```
curl -X POST \
  http://localhost:8088/feedbacks/new \
  -H 'Content-Type: application/json' \
  -d '{
      "comment": "This is an example feedback comment.",
      "eventId": 1,
      "attendeeId": 1,
      "locationrating": 2,
      "descriptionrating": 1
}'
```

- The Feedback Entity including it's id will be returned
- The feedback can be access via its ID and a HTTP GET request, e.g.:

```
curl -X GET http://localhost:8088/feedbacks/1
```

- - All existing feedbacks can be displayed via HTTP GET request:

```
curl -X GET http://localhost:8088/feedbacks
```

1.1 UC08-01: Example Output

```
[
   {
       "id": 1,
       "comment": "This is an example feedback comment.",
       "locationrating": 2,
       "descriptionrating": 1,
       "attendeeId": 1,
       "eventId": 1
   }
]
```

Use-Case demonstrations SR10:

0. Prerequisite:

- Execute the following command inside the implementation folder to build and start the Docker containers.

```
docker compose -f docker-compose-d.yml up
```

- 1000 example Feedbacks with randomized location and accuracy of description rating (between 1-5) will be added to 10 different events (Eventid: 1-10) for demonstration purposes
- 10 example events will be created (eventId 1-10)

1. UC10-01: Generate Event Reports

- Event based reports can be accessed via HTTP GET request, e.g.:

```
curl -X GET http://localhost:8090/reports/1
```

- Event analytics including the corresponding event id will be responded

1.1 UC10-01: Example Output

```
{
   "event_id": 1,
   "eventName": "Wer Wieso Der Event",
   "beginDate": "2023-09-21T00:00:00Z",
   "endDate": "2023-09-26T00:00:00Z",
   "numberOfAttendees": 167,
   "numberOfBookmarks": 461
}
```

2. UC10-02: Generate and View Feedback Analytics

- Average Rating analytics per event can be accessed via HTTP GET request, e.g.:

```
curl -X GET http://localhost:8090/analytics/event/3
```

- Feedback analytics for the according event will be responded

2. UC10-02: Example Output

```
{
   "locationAvg": 3.17,
```

```
    "descriptionAvg": 3.09,
    "event_id": 3
}
```

How-to SR12

For the notification service the mockup implementation was kept to a minimum. Also, in the design phase I create an endpoint for you to test the mockup implementation, in the final the notification service will not have any REST endpoints but will only receive messages from the message broker.

To try out the notification service, navigate in the parent project to the notification service and start the spring boot application locally. This is the endpoint:

```
Fritz Ziernhöld
@PostMapping(path = "/notify")
public ResponseEntity<String> sendNotification(@RequestParam String notification, @RequestParam String email) {
    return notificationService.sendNotification(notification, email);
}
```

To test the endpoint, use the curl request below. Replace the email with the email of your choice and you should receive the email a mockup email from our group email shortly.

```
curl--location --request POST
'localhost:8092/notifications/notify?notification=test%20notificati
on&email=ase.team0601%40gmail.com'
```