S.O.S.E. Project A.Y. (23/24)

Event and ticket management

Dario D'Ercole	288643
Giovanni Spaziani	295397

Index

1. General description

The event and ticket management platform allows to:

- Create and manage events by organizers.
- Explore events and purchase tickets by customers.

2. Services involved

Service	Type	Description
EventManagement(SOA)	Provider	A SOAP service that includes: 1. Events - creating, updating and deleting events. 2. Feedback - creating, updating and deleting feedback on events. 3. Tickets - management of ticket purchases.
Authentication(SOA)	Provider	A REST service useful for authenticating two types of users within the system.
Merchandising	Provider	A REST service that coordinates and provides all information relating to merchandising items during the event.
FeedbackAggregation	Prosumer	A REST prosumer that aggregates feedback with the data of the users who left it, for use in reports.
SalesAnalysis	Prosumer	A REST prosumer that analyzes ticket sales data to provide statistics and reports to event organizers.
EventMerchAggregation	Prosumer	A REST prosumer that aggregates events with related merchandising.

3. Application client

- A web portal that allows users to explore events and purchase tickets.
- A web portal for event organizers where they can create and manage events, view sales data and user feedback.

4. Scenarios

4.1 Creating an event

Description	Services involved
An event organizer uses the system to create and manage a new event. The REST event management service is used to enter event information, such as title, description, date, location, and maximum number of participants.	1. EventManagement(SOA).

4.1.1 Interaction

1. The organizer enters the event details into the web portal.
2. The web portal sends a request to the Events service.
3. The service stores the event details in the database and returns an event ID.
4. The web portal notifies the organizer that the event was successfully created.

4.2 Viewing and purchasing tickets

Description	Services involved
A user browses available events and decides to purchase a ticket for an event. The *Ticket* SOAP service is used to process the purchase resulting in ticket payment and display of available merchandise.	1. EventManagement(SOA). 2. Merchandising.

4.2.1 Interaction

1. The organizer enters the event details into the web portal.
2. The web portal sends a request to the Events service.
3. The service stores the event details in the database and returns an event ID.
4. The web portal notifies the organizer that the event was successfully created.

4.3 Report generation and feedback

Descrizione	Servizi coinvolti
After the conclusion of an event, the organizer wants to obtain a detailed report that includes user feedback and sales statistics. The *FeedbackAggregation* and *SalesAnalysis* prosumers work together to generate this report.	1. FeedbackAggregation. 2. SalesAnalysis. 3. EventManagement(SOA).

4.3.1 Interaction

1. The organizer logs in to the web portal and requests a report for a specific event (**asynchronously**).
2. The web portal sends a request to the prosumer *FeedbackAggregation*.
3. The "FeedbackAggregation" prosumer collects event feedback from the "Feedback" service, within *EventManagement(SOA)*.
4. At the same time, the *SalesAnalysis* prosumer collects ticket sales data from the *Tickets* service, within *EventManagement(SOA)*.
5. The two prosumers synchronize and aggregate the collected data into a single report.
6. The aggregate report is sent to the web portal and displayed to the organizer.

4.4 Viewing the event catalogue and previewing merchandising

Description	Services involved
A user of the platform wants to browse the catalogue of available events and view the details of a specific event, including a preview of the merchandise items that will be available during the event. Use the web portal to search, view and get all the information needed.	1. EventManagement(SOA). 2. Merchandising. 3. EventMerchAggregation.

4.4.1 Interaction

1. The user logs into the web portal and selects the option to view the event catalogue.
2. The web portal sends a request to the *EventManagement(SOA)* service event viewing endpoint.
3. The REST service retrieves event information from the database and sends it to the web portal.
4. The web portal displays the list of available events, each with a short description.

5. The user selects an event from the list to view full details.
6. The web portal sends a request to the *EventManagement(SOA)* service event details endpoint, including the ID of the selected event.

7. The service retrieves detailed event information from the database and sends it to the web portal.
8. The web portal displays complete details of the event.

9. The web portal sends a request to the endpoint to view merchandising items from the *Merchandising* REST service, including the ID of the selected event.
10. The service retrieves information about merchandising items associated with the event from the database and sends it to the web portal.
11. The web portal displays a preview of available merchandising items.

5. Use Case Diagram

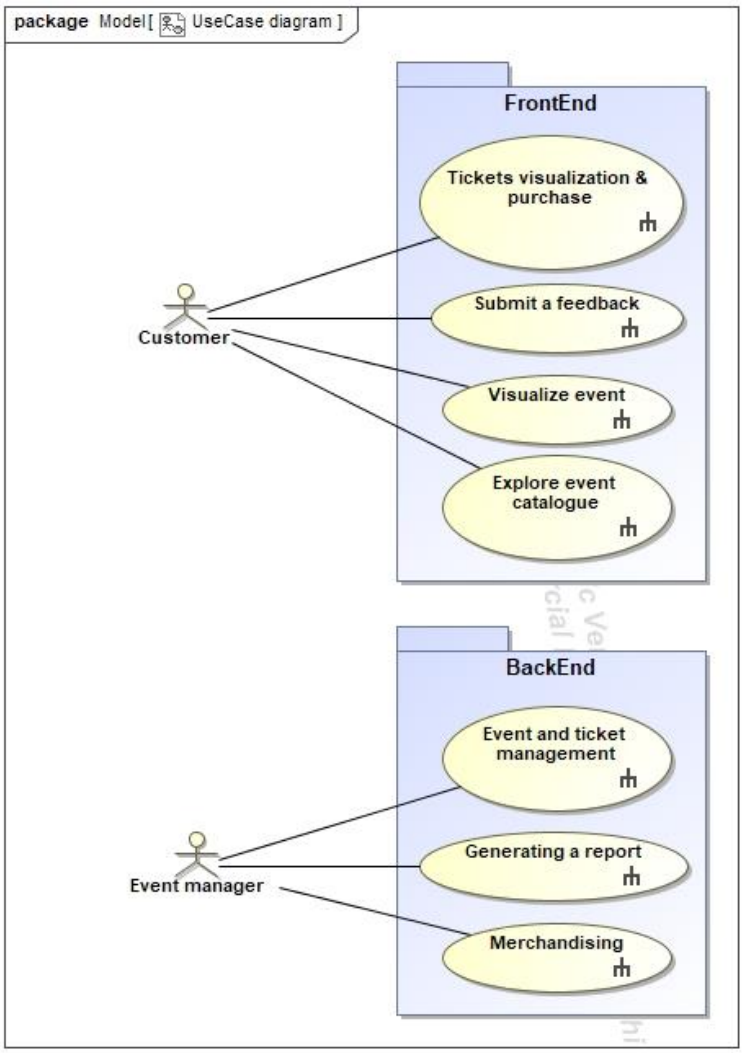

Figure 1 - UseCase Diagram

In our domain, the protagonists are therefore the *Customer* and the *Event manager*. The system has been divided into two main parts, namely *FrontEnd* and *BackEnd*. The use cases concerning the *Customer* are:

Use Case	Description
Tickets visualizazion & purchase	The *Customer* can view the tickets available for the events, and if he chooses to take them, he can proceed with the purchase operation.
Submit a feedack	The *Customer* can send feedback regarding each event they have been a part of.

Visualize event	The *Customer* can view the information of an event in detail.
Explore event catalogue	The Customer can explore the catalogue of available events.

Instead, the use cases regarding *Event manager* are:

Use Case	Description
Event and ticket management	The *Event manager* can both create an event and ticket for a given event.
Generating a report	The *Event manager* can generate a report to display certain statistics of an event.
Merchandising	The *Event manager* can manage merchandising operations linked to a specific event.

6. Profile Diagram

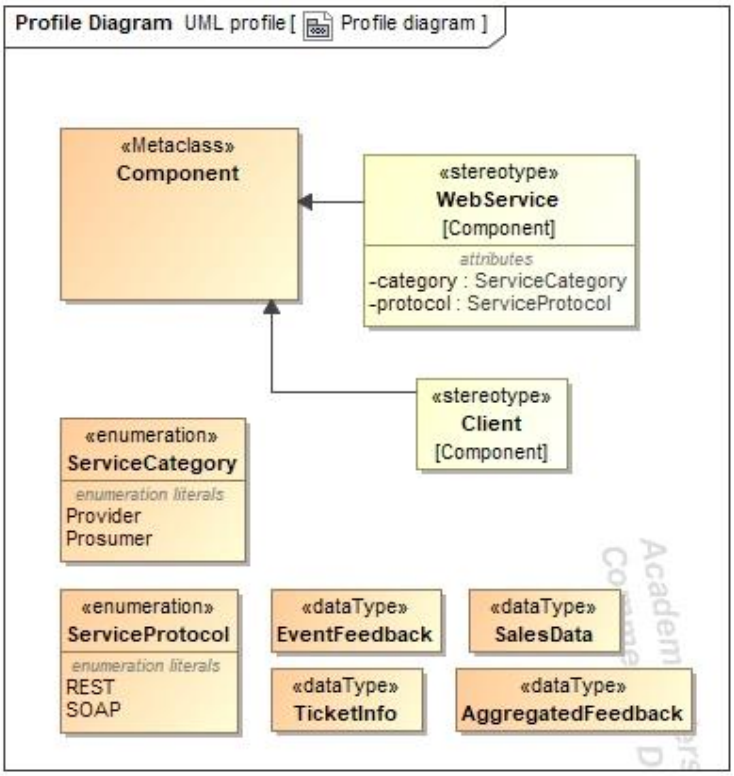

Figure 2 - Profile Diagram

Our Profile Diagram consists of:

- **Component**: UML metaclass that represents a module of the system and is used to create new elements via stereotypes.
- **WebService**: this stereotype extends the *Component* metaclass, indicating that any component of the system can be classified as a *WebService*. The attributes present within it are:
  - **Category**: this attribute uses the *ServiceCategory* enumeration, useful for classifying the web service.
  - **Protocol**: this attribute uses the *ServiceProtocol* enumeration, useful for classifying the web service protocol.
- **Client**: this stereotype represents a client using the web service.

- **ServiceCategory**: this enumeration defines two categories of services:
    - **Provider**: a service that provides data and/or functionality.
    - **Prosumer**: a service that can both provide and consume data and/or functionality.
- **ServiceProtocol**: this enumeration defines the communication protocols used by the various web services:
    - **REST**: indicates an architectural style for distributed systems and represents a data transmission system over HTTP.
    - **SOAP**: indicates a protocol for exchanging messages between software components.
- **EventFeedback**: represents event feedback data.
- **SalesData**: represents sales data.
- **TicketInfo**: represents ticket information.
- **AggregatedFeedback**: represents all the feedback, regarding each event, that has been left by users.

7. Component Diagram

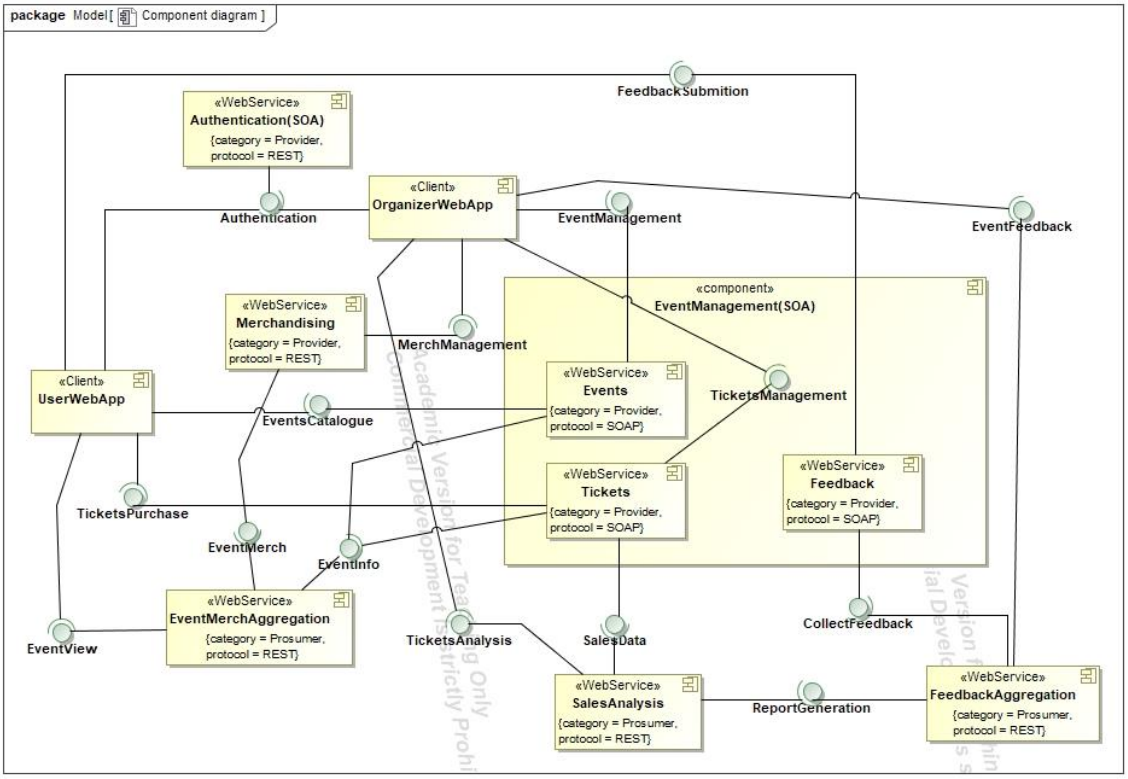

Figure 3 - Component Diagram

The component diagram is composed of:

- **Authentication(SOA)**: this component is responsible for authenticating users. It is classified as a *WebService*, is part of the *Provider* category and uses the *REST* architectural style.
- **Merchandising**: this component manages all merchandising operations. It has been classified as a *WebService*, is part of the *Provider* category and uses the *REST* architectural style.
- **EventManagement(SOA)**: manages all operations relating to events, tickets and feedback.
- **Events**: this component manages event information. It is classified as a *WebService*, is part of the *Provider* category and uses the *SOAP* protocol. Inside it contains the *fetchEvents* operation (with *sortBy* and *page* parameters) inherited from the interface connected with *UserWebApp*, which is used to retrieve the events in ascending/descending order for both the ID and the alphabetical sequence of letters. It is also possible to retrieve events based on the pages in which they were cataloged.

- **Tickets**: this component manages ticket information and operations. It is classified as a *WebService*, is part of the *Provider* category and uses the *SOAP* protocol. Inside it contains the *availableTickets* operation (with *eventId* parameter), which is used to check the remaining tickets relating to a specific event.
- **Feedback**: this component manages user feedback on events. It is classified as a *WebService*, is part of the *Provider* category and uses the *SOAP* protocol.
- **OrganizerWebApp**: it is an interface for *Event manager*, and is used to manage events, various merchandise items and collect feedback. Inside it contains the *formatReport* operation (with *aggregatedFeedback* and *salesData* parameters), which is used to give a structure to the report, collecting all the feedback for an event, together with its sales data.
- **UserWebApp**: it is an interface for *Customer*, and is used to view events, purchase tickets and send feedback.
- **EventMerchAggregation**: this component aggregates data relating to events and available merchandising. It is classified as a *WebService*, is part of the *Prosumer* category and uses the *REST* architectural style.
- **SalesAnalysis**: this component analyzes event sales data. It is classified as a *WebService*, is part of the *Prosumer* category and uses the *REST* architectural style. Inside it contains the *analyze* operation (with *ticketList* parameter), which is used by the organizer to retrieve the list of tickets sold for an event, useful in the future for analyzing ticket sales data.
- **FeedbackAggregation**: this component aggregates all user feedback regarding an event. It is classified as a *WebService*, is part of the *Prosumer* category and uses the *REST* architectural style. Inside it contains the *aggregate* operation (with *feedbackList* parameter), which is used to return a list of all the aggregated feedback to the organizer.

The interfaces that have been inserted are:

- **Authentication**: it is an interface connected to the *Authentication(SOA)*, *OrganizerWebApp* and *UserWebApp* components, and is used to authenticate users to access all the services made available. Inside it contains the operations *LogIn*, *SignUp* and *SignOut*.
- **FeedbackSubmition**: it is an interface connected to the *UserWebApp* and *FeedbackAggregation* components and is used for submitting feedback on events. Inside it contains the *submit* operation (with parameters *userId*, *eventId*, *rating* and *body*), which is used to submit feedback by entering the identifying ID, the event id to to whom the feedback is addressed, the rating from 1 to 5, and the comment on the feedback.
- **EventManagement**: it is an interface connected to the *OrganizerWebApp* and *Events* components and is used for creating events. Inside it contains the *createEvent* operation (with parameters *name*, *startDate*, *endDate*, *location*, *description* and *nrTickets*), which is used to provide a general overview of the newly created event.
- **EventFeedback**: it is an interface connected to the *OrganizerWebApp* and *FeedbackAggregation* components and is used to request all the feedback for a specific event from the feedback aggregator. Inside it contains the *requestFeedback* operation (with *eventId* parameter).
- **TicketsManagement**: it is an interface connected to the *OrganizerWebApp* and *Tickets* components and is used for creating tickets for events. Inside it contains the *createTickets* operation (with parameters *eventId*, *referenceDate*, *price* and *amountNumber*).
- **CollectFeedback**: it is an interface connected to the *Feedback* and *FeedbackAggregation* components and is used to collect all the feedback relating to an event. Inside it contains the *fetchEventFeedback* operation (with *eventId* parameter), which is used to retrieve the feedback of an event, passing the ID of the latter for identification.

- **ReportGeneration**: it is an interface connected to the *SalesAnalysis* and *FeedbackAggregation* components and is used to combine data from sales data and feedback, to generate the final report for a specific event.
- **SalesData**: it is an interface connected to the *Tickets* and *SalesAnalysis* components and is used to collect all ticket sales data. Inside it has the *fetchTicketInfo* operation (with *eventId* parameter), which is used to fetch all the ticket information, identifying the event to which it refers via an identifying ID.
- **TicketsAnalysis**: it is an interface connected to the *OrganizerWebApp* and *SalesAnalysis* components and is used to carry out ticket sales analysis. Inside it contains the *requestSalesAnalysis* operation (with *eventId* parameter), which is used by the organizer's interface to request the analysis of the sale of an event, which is passed via identifying ID.
- **EventInfo**: it is an interface connected to the *Events*, *Tickets* and *EventMerchAggregation* components and is used to retrieve event information. Inside it contains the *fetchInfo* operation (with *eventId* parameter).
- **EventMerch**: it is an interface connected to the *Merchandising* and *EventMerchAggregation* components and is used to retrieve the merchandise associated with an event. Inside it contains the *fetchEventMerch* operation (with *eventId* parameter).
- **EventsCatalogue**: it is an interface connected to the *UserWebApp* and *Events* components and is used by the customer to explore scheduled events. Inside it contains the operations *fetchEvents* (with *sortBy* and *page* parameters) and *viewEvent* (with *eventId* parameter). The first is used to retrieve events in ascending/descending order both for the ID and for the alphabetical sequence of letters. It is also possible to retrieve events based on the pages in which they were cataloged; the second instead is simply used to display the event, passing the ID to which it refers.
- **MerchManagement**: it is an interface connected to the *OrganizerWebApp* and *Merchandising* components and is used to correlate an event with the related merchandise. Inside it contains the operations *addMerchItem* (with parameters *name*, *description* and *barCode*) and *addMerchToEvent* (with parameters *merchId* and *eventId*). The first is used to add merchandise objects, passing the name, the description of the object, and its barcode. The second operation instead is used to correlate the merchandise to an event.
- **TicketsPurchase**: it is an interface connected to the *UserWebApp* and *Tickets* components and is used for the purchase of one/more tickets. Inside it contains the operations *openPurchaseMenu* (with parameter *eventId*), *purchaseTickets* (with parameters *userId*, *eventId*, *quantity*, *date*), and *ticketsRefund*. The first is used to open the ticket purchase menu by passing the event ID. The second is used to purchase the ticket(s), passing the various identification IDs, the quantity of tickets, and the reference date of the event. Finally, the last operation is used to refund the purchased ticket(s).
- **EventView**: it is an interface connected to the *UserWebApp* and *EventMerchAggregation* components and is used for displaying events. Inside it contains the *loadEventPage* operation (with *eventId* parameter), which is used to open the page with all the details of the event, passed via identifying ID.

8. Sequence Diagram “Event and ticket management”

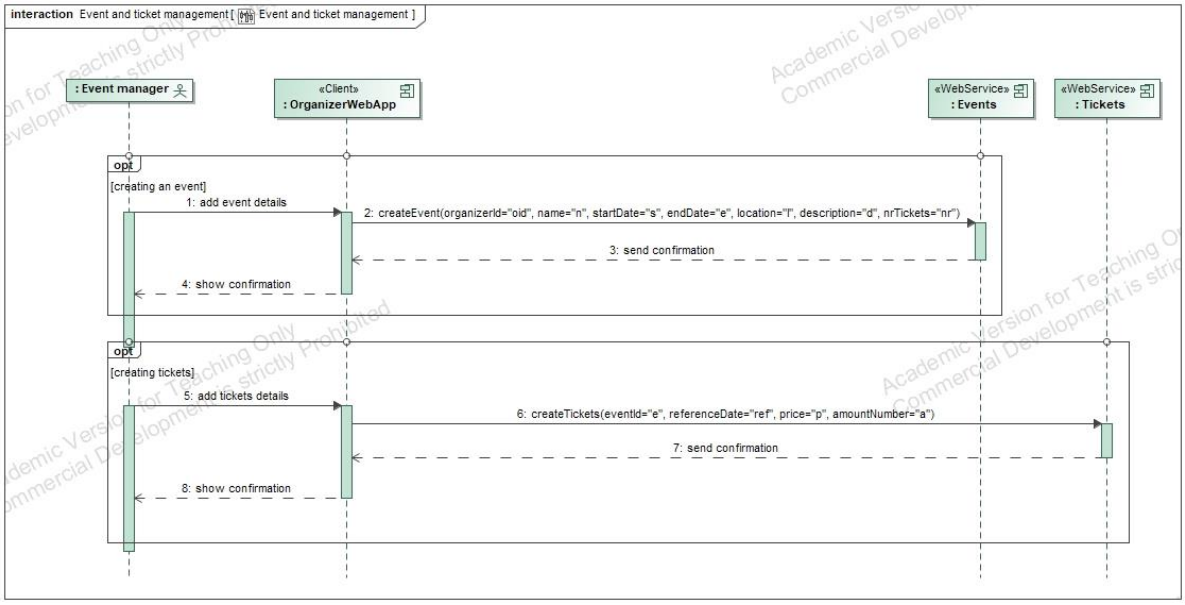

Figure 4 - SD Event and ticket management

This Sequence Diagram starts with two optionals:

- If the *Event manager* wants to create an event, it enters the first block. First, it adds the details of an event to the *OrganizerWebApp* interface (1). Through this interface, the event is created (2), passing as parameters: *organizerId*, event *name*, event *startDate*, event *endDate*, event *location*, *description*, and *nrTickets* available for the event. The creation confirmation is returned to the organizer’s interface (3). Finally, the interface takes care of showing the *Event manager* user confirmation of the addition of the event (4).
- If the *Event manager* wants to create tickets for an event, the procedure is almost the same as the block above. First, the *Event manager* adds the ticket details to the *OrganizerWebApp* interface (5). Through this interface, the ticket is created (6), passing as parameters: *eventID*, event *referenceDate*, ticket *price* and *amountNumber* of tickets available. Confirmation of ticket creation is returned to the organizer's interface (7). Finally, the interface takes care of showing the *Event manager* user confirmation of the addition of the ticket (8).

9. Sequence Diagram “Explore event catalogue”

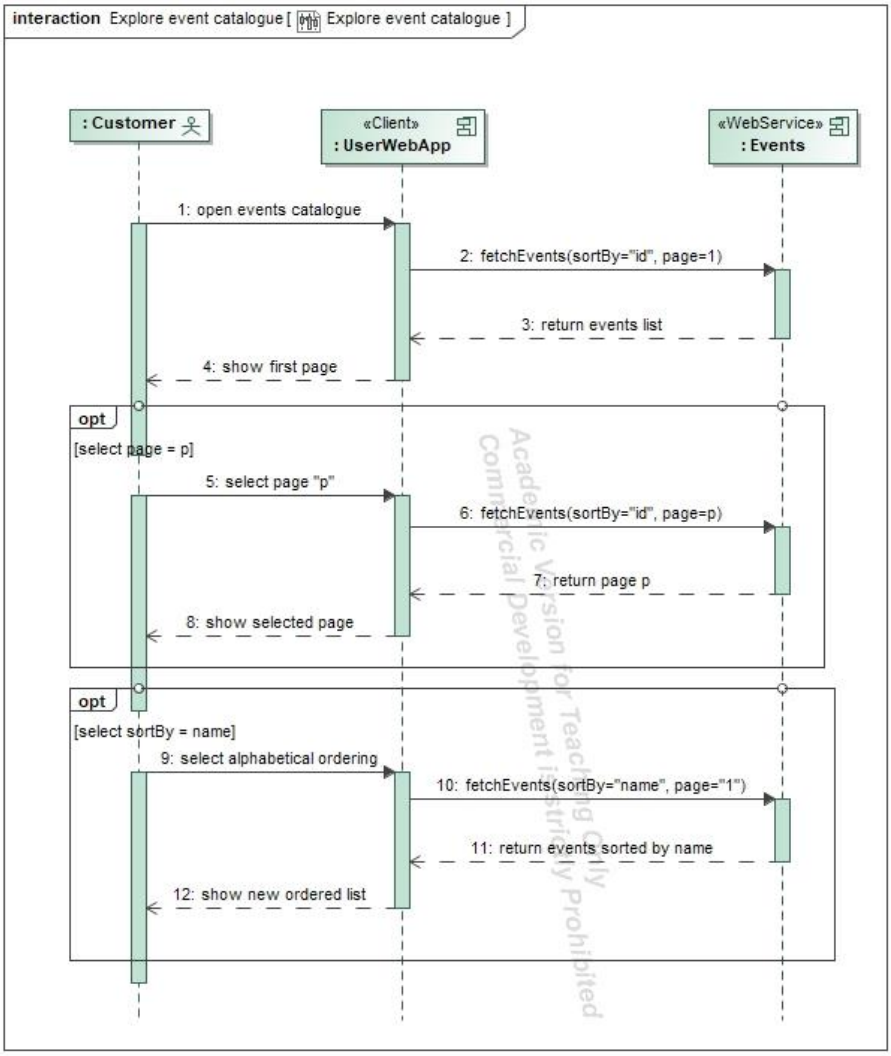

Figure 5 - SD Explore event catalogue

This Sequence Diagram begins with the *Customer*'s request to the *UserWebApp* interface to open the event catalogue (1). The interface takes care of contacting the *Events* component to retrieve all the events, sorted by *id*, and viewable by *page* (2). The *Events* component takes care of returning the list of events to the interface (3). The latter then returns the first page showing the events to the *Customer* (4). At this point there are two options:

- The *Customer* can choose to view another page. He starts by selecting the desired page via the interface (5), the interface takes care of retrieving all the events present on that page sorting them by *id* (6). The *Events* component then returns the events present on that page to the interface (7). Finally, the interface returns the selected page to the *Customer* (8).
- The Customer can choose to view the events by sorting them alphabetically. He starts by selecting through the interface to show all events in alphabetical order (9). The interface takes care of contacting the *Events* component to retrieve events (10). The latter returns the list of events sorted alphabetically to the Customer interface (11). Finally, the interface shows the *Customer* the new ordered list of events (12).

10. Sequence Diagram “Generating a report”

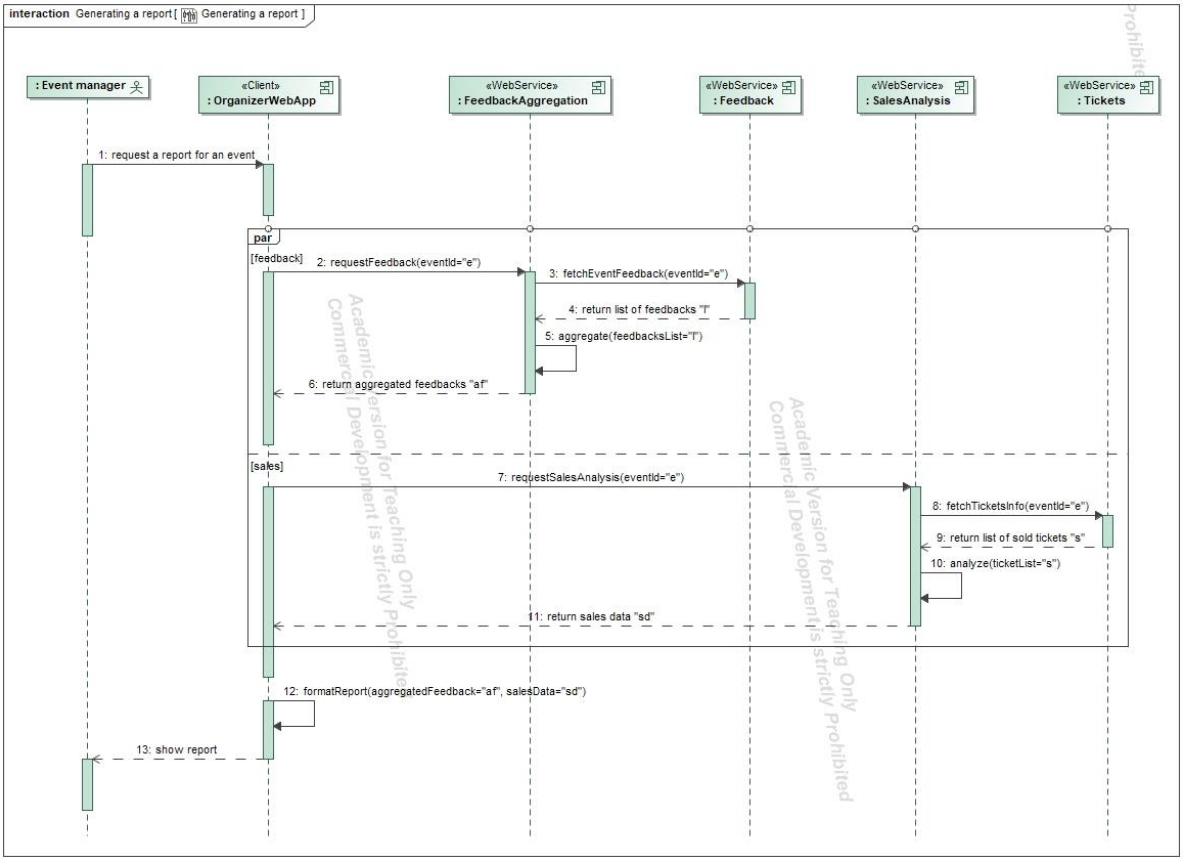

Figure 6 - SD Generating a report

This Sequence Diagram begins with the *Event manager*'s request to report for an event (1). At this point, two sequences of operations are performed in parallel:

- The first sequence of operations deals with getting all the feedback for an event. It starts with the organizer interface to request feedback for a selected event via *Id* (2). *FeedbackAggregation* asks the *Feedback* component to retrieve the feedback for that event, always passing its identifying *id* (3). A list of feedback (4) is then returned to the *FeedbackAggregation*. It then scrolls through all the feedback, counts the keywords that the organizer asks it to count, calculates the average age of those who reviewed the event, and finally calculates the average rating for that event (5). Finally, the list of all aggregated feedback is returned to the organizer's interface (6).
- The second sequence of operations deals with getting all the sales data for an event. It starts with the request from the *OrganizerWebApp* interface to acquire the sales data for an event, identified with an *Id* (7). The prosumer asks to retrieve the ticket information, taken via the *eventId* (8). Subsequently, a list of tickets sold (9) is returned to him. The *SalesAnalysis* prosumer at this point analyzes all the data, i.e. the list of tickets sold, age distribution of buyers, count of tickets sold on a date (if an event has multiple dates), count of the genders of those who purchased the tickets, calculation of the average age of buyers (10). Finally, all ticket sales data is returned to the *OrganizerWebApp* interface (11).

At this point in the Sequence Diagram, the interface formats all the data it receives (12), and then shows the report to the *EventManager* user (13).

11. Sequence Diagram “Merchandising”

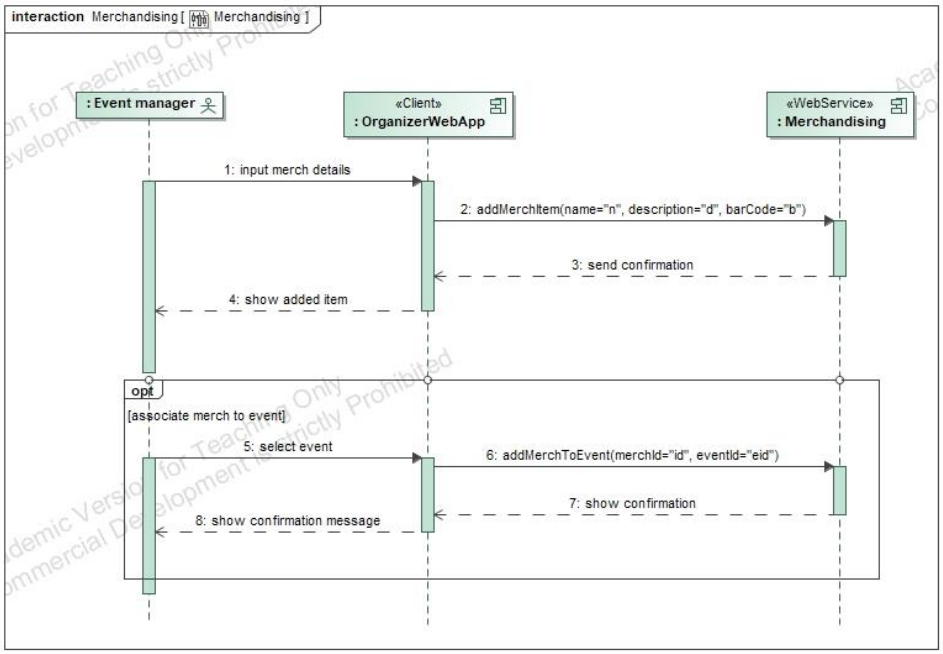

Figure 7 - SD Merchandising

This Sequence Diagram begins with the *Event manager* user who enters the merchandise details via the *OrganizerWebapp* interface (1). The interface contacts the *Merchandising* provider to add a merchandise item, writing the item *name*, *description* and *barCode* (2). Then, the provider returns the confirmation of insertion of the merchandise (3). Finally, the interface takes care of showing the inserted object to the *Event manager* user (4). At this point there is an optional: if the *Event manager* user wants to associate the merchandise with an event, he continues with the selection of the event to which to associate the objects (5). The *OrganizerWebApp* interface takes care of requesting the association of the merchandise with the event (6). Subsequently, the addition confirmation (7) is returned. Finally, the *Event manager* user receives the confirmed association message (8).

12. Sequence Diagram “Visualize event”

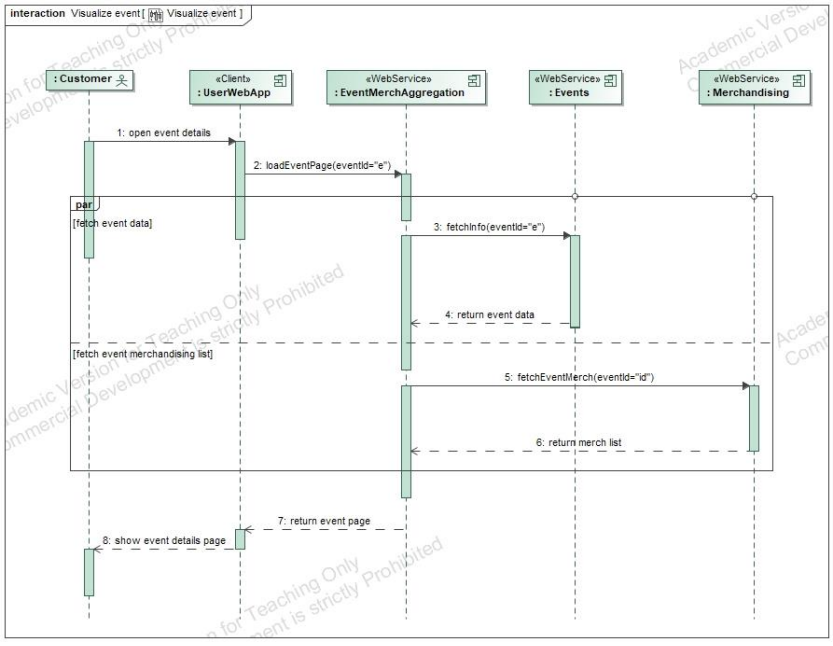

Figure 8 - SD Visualize event

This Sequence Diagram begins with the *Customer*'s request to open the detail page of an event (1). Via the *UserWebApp* interface, the event page is loaded (2). At this point, two sequences of operations are performed in parallel:

- The first sequence of operations deals with recovering the event data. It starts with the request from *EventMerchAggregation* to retrieve the information, passing the *eventId* (3). The event data is then returned to the *EventMerchAggregation* (4).
- The second sequence of operations deals with recovering the list of merchandise for that event. It therefore starts with the request from *EventMerchAggregation* to recover the merchandise associated with the event, passed through *Id* (5). The list of merchandise associated with that event is then returned to the *EventMerchAggregation* (6).

The penultimate operation instead concerns the return to the event page interface (7). Finally, the interface takes care of showing the *Customer* user the event details page (8).

13. Sequence Diagram "Tickets visualization & purchase"

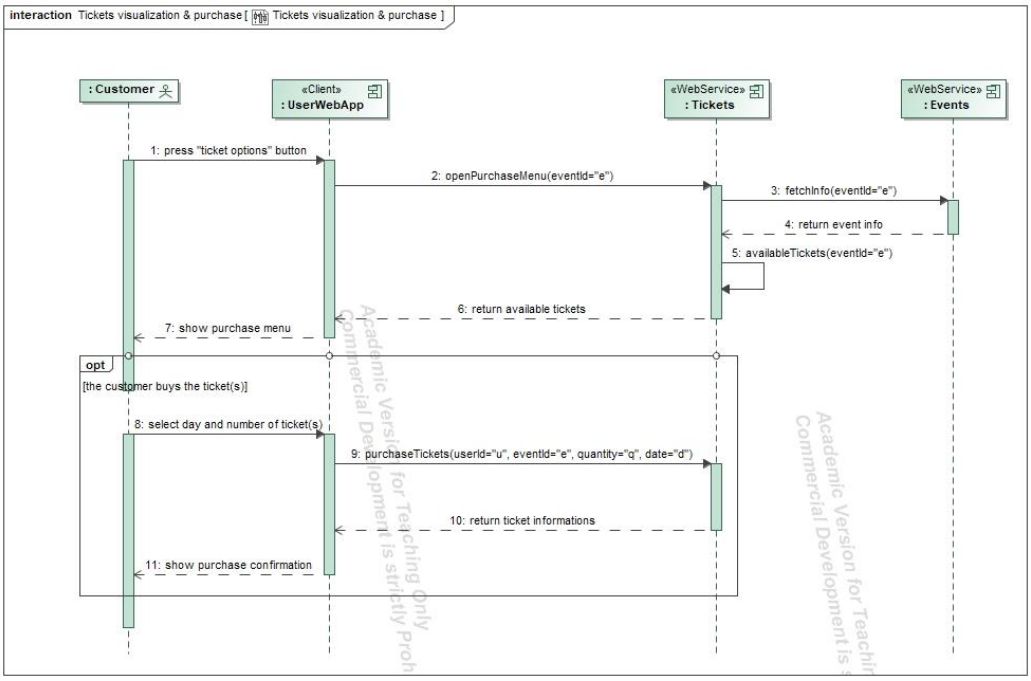

Figure 9 - SD Tickets visualization & purchase

This Sequence Diagram starts with the *Customer* pressing the button to view the ticket options (1). It continues with the *UserWebApp* interface which asks the *Tickets* provider to open the ticket purchase menu, passing the *eventId* (2) as a parameter. The *Tickets* provider at this point contacts the *Events* provider to retrieve the event information (3). *Events* then returns the event information to *Tickets* (4). The *Tickets* provider now takes all available tickets for the event passed through *Id* (5) and returns the available tickets to the interface (6). The *Customer* is then shown the ticket purchase menu (7). At this point of the Sequence Diagram, there is an optional, which concerns the case in which the *Customer* decides to buy the ticket(s). First, the *Customer* selects the day and number of tickets to purchase (8). Through the *UserWebApp* interface, the purchase operation of the ticket(s) is started, passing the *userId*, the *eventId*, the *quantity* of tickets to be purchased and the *date* of the event (9). The information of the ticket(s) is returned to the *UserWebApp* interface (10). Finally, the purchase confirmation is returned to the *Customer* user (11).

14. Sequence Diagram “Submit a feedback”

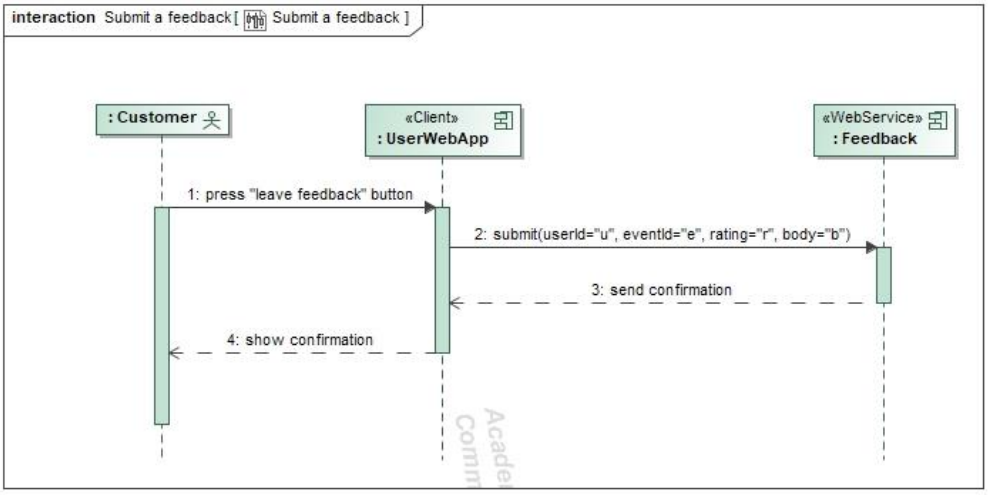

Figure 10 - SD Submit a feedback

This last Sequence Diagram begins with the *Customer* pressing the button to leave feedback for an event (1). Through the *UserWebApp* interface, the feedback release operation is initialized, passing the *userId*, the *eventId*, the *rating* for the event, and the feedback *body* (2). At this point the *Feedback* provider returns the confirmation of the added feedback to the interface (3), which finally returns the confirmation of submission of the feedback to the *Customer* (4).

However, for greater clarity, all images at optimal resolution are located within the Docs/Images folder.