Sequence Diagram: User Registration

The sequence diagram presented in Fig. 1 illustrates the dynamic interactions between the application layers (Presentation, Business Logic, and Persistence) when a user attempts to register a new account.

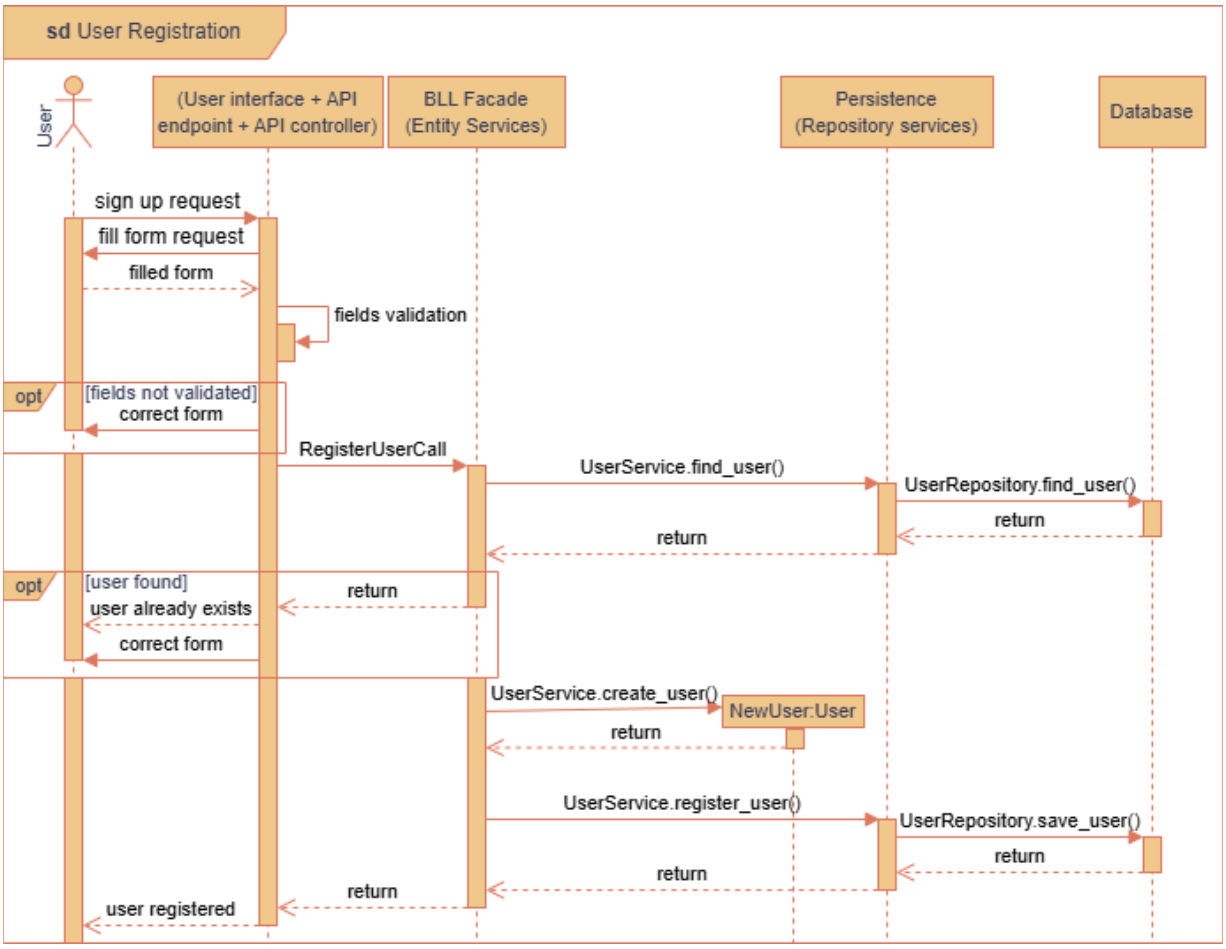

Fig. 1: user registration sequence diagram.

Actors, System Elements and Lifelines

- **User**: The external actor who interacts with the system
- **User Interface + API Endpoint + API Controller (Presentation Layer)**: Responsible for receiving user requests, performing simple input validation, data deserialization/serialization, and calling the Business Logic Layer (BLL)

- **BLL Facade (Entity Services)**: Entry point for business logic, orchestrator of complex business operations, interactions with the presentation layer and delegation of persistence operations to repository services
- **Persistence Layer (Repository Services)**: Represents the data access component responsible for direct interactions with the database, handling the storage and retrieval of specific entity data
- **Database**: The data storage system

Registration (Sign-Up) Process

Sign-Up Request:

- **Sender**: User
- **Receiver**: User Interface + API Endpoint + API Controller
- **Message**: sign-up request
- **Description**: The user initiates the registration process via the user interface
- **Data**: Signal

Fill Form:

- **Sender**: User Interface + API Endpoint + API Controller
- **Receiver**: User
- **Message**: fill form request
- **Description**: The user interface prompts the user to fill in the registration information
- **Data**: Prompt first name, last name, email, and password

Filled Form:

- **Sender**: User
- **Receiver**: User Interface + API Endpoint + API Controller
- **Message**: filled form
- **Description**: The user sends back the form filled with the required registration information (first name, last name, email, password)
- **Data**: First name, last name, email, and password

First Option Fragment:

  - **Condition** [fields not validated]
  - **Description**: The fields contain errors (invalid email format, password is too short or doesn't contain special characters, etc.)
  - **Implied Flow:** The user interface displays an error message and the user is asked to fill out the form demanding acceptable inputs

Register User Request:

- **Sender**: User Interface + API Endpoint + API controller
- **Receiver**: BLL Facade (Entity Services)
- **Message**: RegisterUserCall
- **Description**: The interface asks the BLL Facade to verify if the user exists (if there is a user account associated with the provided email) and to register it as a new user otherwise
- **Data**: first name, last name, email, and password

Find User Service Call:

- **Sender**: BLL Facade (Entity Services)
- **Receiver**: Persistence Layer (Repository Services)
- **Message**: UserService.find_user()
- **Description**: The BLL Facade asks the Persistence Layer to find the user
- **Data**: email

Find User in Database:

- **Sender**: Persistence Layer (Repository Services)
- **Receiver**: Database
- **Message**: UserRepository.find_user()
- **Description**: The Persistence Layer makes a request to the Database to check if the user already exists
- **Data**: email

Find User Return:

- **Sender1**: Database
- **Receiver1**: Persistence Layer (Repository Services)
- **Sender2**: Persistence Layer (Repository Services)
- **Receiver2**: BLL Facade (Entity Services)
- **Message**: return
- **Description**: The Persistence Layer confirms the user did or did not already exist to the BLL
- **Data**: user_id/empty

Second Option Fragment:

- **Condition** [user found]
- **Description**: Notifies the user that an account associated with its email already exists
- **Implied Flow:** If the user is not found, the flow continues with the User creation service call

User Creation Service Call:

- **Sender**: BLL Facade (Entity Services)
- **Receiver**: User class
- **Message**: UserService.create_user()
- **Description**: The BLL Facade creates an instance of User containing the user information
- **Data**: first name, last name, email, and password

User Creation Return:

- **Sender**: NewUser
- **Receiver**: BLL (Entity Services)
- **Message**: return
- **Description**: The User class confirms the creation (or not) of a new User instance to the BLL and returns its reference
- **Data**: NewUser reference

New User Registration:

- **Sender**: BLL Facade (Entity Services)
- **Receiver**: Persistence Layer (Repository Services)
- **Message**: UserService.register_user()
- **Description**: The BLL Facade asks the Persistence Layer to register the new user
- **Data**: NewUser reference

Data Storage:

- **Sender**: Persistence Layer (Repository Services)
- **Receiver**: Database
- **Message**: UserRepository.save_user()
- **Description**: The Persistence Layer requests the Database to save the new user. Returns a validated creation message to the user
- **Data**: NewUser reference

Register User Return:

- **Sender1**: Database
- **Receiver1**: Persistence Layer (Repository Services)
- **Sender2**: Persistence Layer (Repository Services)
- **Receiver2**: BLL Facade (Entity Services)
- **Sender3**: BLL Facade (Entity Services)
- **Receiver3**: User Interface + API Endpoint + API Controller
- **Message**: return
- **Description**: The Persistence Layer confirms to the interface that the user has been created
- **Data**: Signal

Sign-Up Return:

- **Sender**: User Interface + API Endpoint + API Controller
- **Receiver**: User
- **Message**: user registered
- **Description**: The user receives confirmation of registration success
- **Data**: Message

Sequence Diagram: Place Creation

The sequence diagram presented in Fig. 1 illustrates the dynamic interactions between the application layers (Presentation, Business Logic, and Persistence) when a registered and logged user attempts to create a new place.

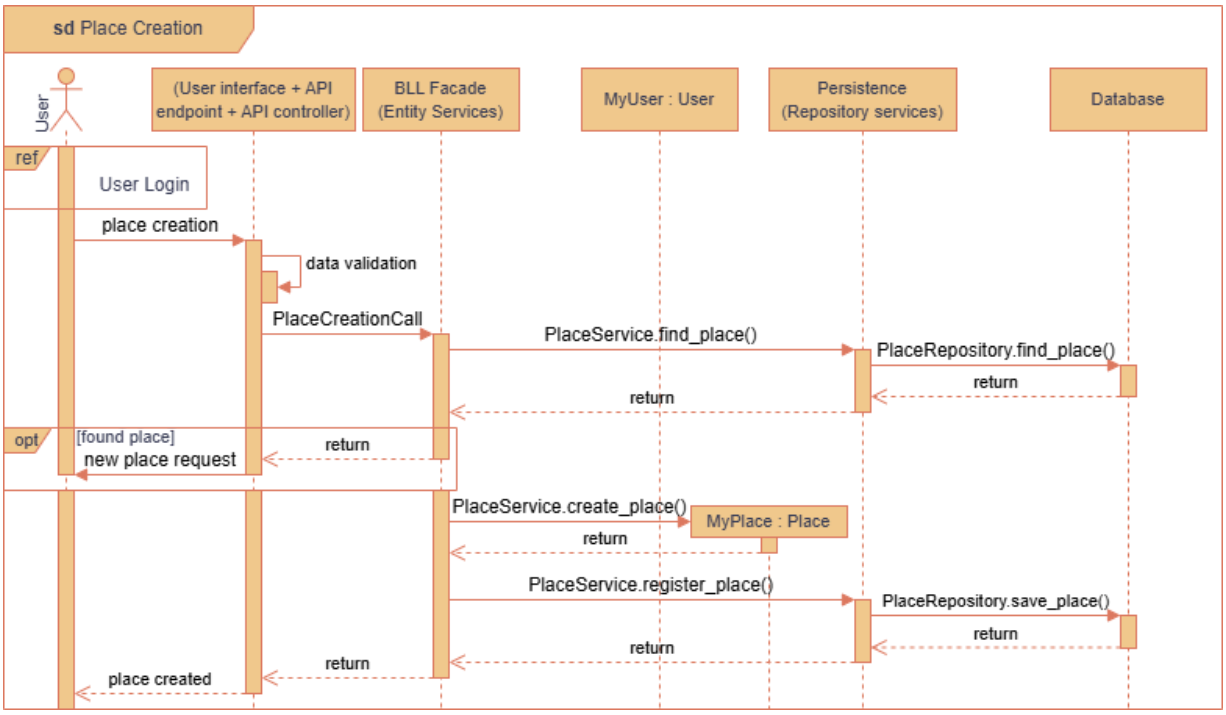

Fig. 1: place creation sequence diagram.

Actors, System Elements and Lifelines

- **User**: The external actor who interacts with the system
- **User Interface + API Endpoint + API Controller (Presentation Layer)**: Responsible for receiving user requests, performing simple input validation, data deserialization/serialization, and calling the Business Logic Layer (BLL)
- **BLL Facade (Entity Services)**: Entry point for business logic, orchestrator of complex business operations, interactions with the presentation layer and delegation of persistence operations to repository services
- **MyUser : User**: Represents an instance of the User class, an object MyUser of class User
- **Persistence Layer (Repository Services)**: Represents the data access component responsible for direct interactions with the database, handling the storage and retrieval of specific entity data
- **Database**: The data storage system

Ref User Login

Refers to the User Login sequence diagram and its flow. It calls to the fact that the user needs to be logged-in to perform the Place Creation operation, which is also why the object NewUser exists and is available at the business layer.

Place Creation Process

Place Creation Request:

- **Sender**: User
- **Receiver**: User Interface + API Endpoint + API Controller
- **Message**: place creation
- **Description**: The user makes a request, via the interface, to create a place
- **Data**: Place Data (title, description, etc)

Register Place Call:

- **Sender**: User Interface + API Endpoint + API controller
- **Receiver**: BLL Facade (Entity Services)
- **Message**: PlaceCreationCall
- **Description**: After simple validation of the PlaceData introduced by the user, the interface asks the BLL Facade to verify if the place exists and register a new one with the provided information otherwise
- **Data**: PlaceData

Find Place Service Call:

- **Sender**: BLL Facade (Entity Services)
- **Receiver**: Persistence Layer (Repository Services)
- **Message**: PlaceService.find_place()
- **Description**: The Facade asks the Persistence Layer to find the place
- **Data**: PlaceData

Find Place in Database:

- **Sender**: Persistence Layer (Repository Services)
- **Receiver**: Database
- **Message**: PlaceRepository.find_place()
- **Description**: The Persistence Layer requests the Database to verify if the described place already exists (compare title, description, location)
- **Data**: PlaceData

Find Place Return:

- **Sender1**: Database
- **Receiver1**: Persistence Layer (Repository Services)
- **Sender2**: Persistence Layer (Repository Services)
- **Receiver2**: BLL Facade (Entity Services)
- **Message**: return
- **Description**: The Persistence Layer confirms the place did or did not already exist to the BLL
- **Data**: place_id/empty

Option Fragment

- **Condition**: [found place]
- **Description**: If a place with the provided details is found in the database, an error message is returned to the user, indicating that the place already exists. The process of creating a new place is then halted, and the user is prompted to provide different details or an alternative action
- **Implied Flow**: If the place is *not* found (meaning it's a new place), the flow proceeds normally to the "Place Creation" step, as this part of the process is outside the `opt` fragment's explicit condition

Place Creation Service Call:

- **Sender**: BLL Facade (Entity Services)
- **Receiver**: Place class
- **Message**: PlaceService.create_place()
- **Description**: The BLL Facade creates an instance of Place containing the new place information
- **Data**: PlaceData, MyUser.id

Place Creation Return:

- **Sender**: MyPlace
- **Receiver**: BLL (Entity Services)
- **Message**: return
- **Description**: The Place class confirms the creation of a new Place instance to the BLL and returns its reference
- **Data**: MyPlace reference

New Place Registration:

- **Sender**: BLL Facade (Entity Services)
- **Receiver**: Persistence Layer (Repository Services)
- **Message**: PlaceService.register_place()
- **Description**: The BLL Facade asks the Persistence Layer to register the new place
- **Data**: MyPlace reference

Data Storage:

- **Sender**: Persistence Layer (Repository Services)
- **Receiver**: Database
- **Message**: PlaceRepository.save_place()
- **Description**: The Persistence Layer asks the Database to save the new place. Returns a validated creation message to the user
- **Data:** MyPlace reference

Register Place Return:

- **Sender1**: Database
- **Receiver1**: Persistence Layer (Repository Services)
- **Sender2**: Persistence Layer (Repository Services)
- **Receiver2**: BLL Facade (Entity Services)
- **Sender3**: BLL Facade (Entity Services)
- **Receiver3**: User Interface + API Endpoint + API Controller
- **Message**: return
- **Description**: The Persistence Layer confirms to the interface that the place has been created
- **Data**: Signal

Place Creation Return:

- Sender: User Interface + API Endpoint + API Controller
- Receiver: User
- Message: place created
- Description: The user receives confirmation of place registration success
- Data: Message

Sequence Diagram: Review Submission

The sequence diagram presented in Fig. 1 illustrates the dynamic interactions between the application layers (Presentation, Business Logic, and Persistence) when a registered and logged user attempts to submit a review of a chosen place.

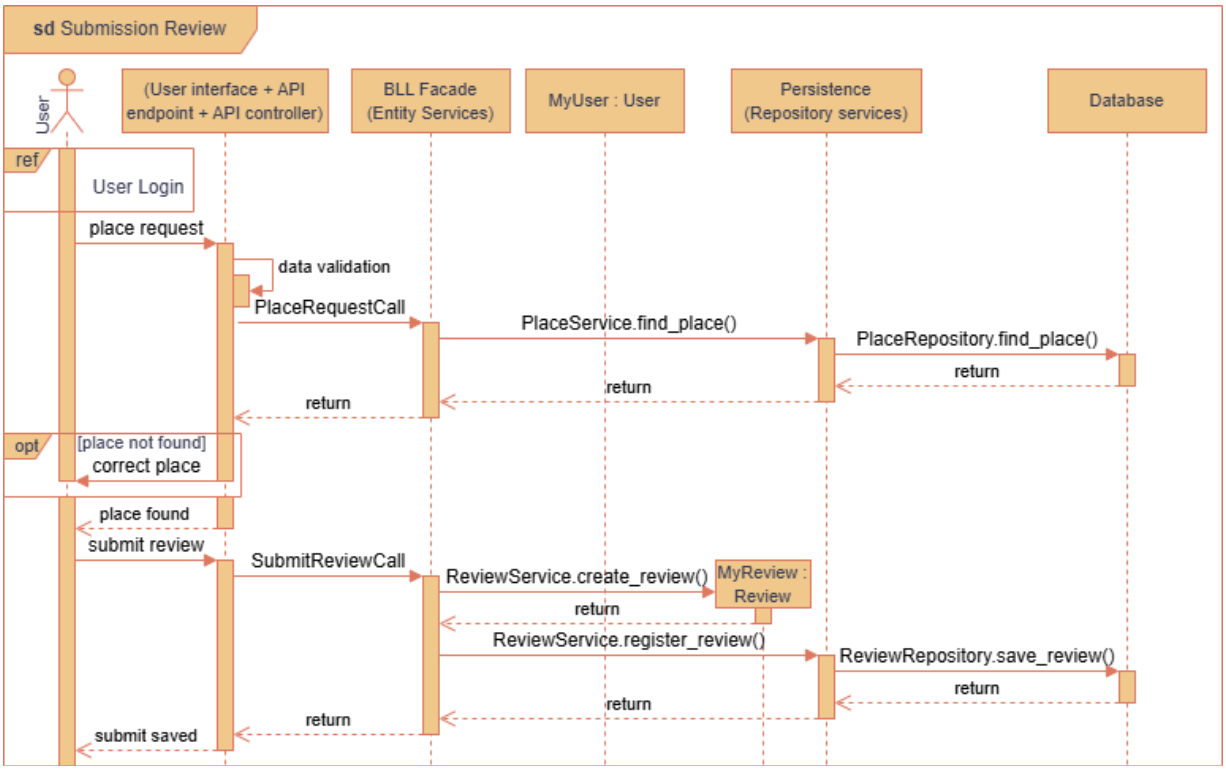

Fig. 1: review submission sequence diagram.

Actors, System Elements and Lifelines

- **User**: The external actor who interacts with the system
- **User Interface + API Endpoint + API Controller (Presentation Layer)**: Responsible for receiving user requests, performing simple input validation, data deserialization/serialization, and calling the Business Logic Layer (BLL)
- **BLL Facade (Entity Services)**: Entry point for business logic, orchestrator of complex business operations, interactions with the presentation layer and delegation of persistence operations to repository services.
- **MyUser : User**: Represents an instance of the User class, an object MyUser of class User
- **Persistence Layer (Repository Services)**: Represents the data access component responsible for direct interactions with the database, handling the storage and retrieval of specific entity data

- **Database**: The data storage system

Ref User Login

Refers to the User Login sequence diagram and its flow. It calls to the fact that the user needs to be logged-in to perform the Place Creation operation, which is also why the object NewUser exists and is available at the business layer.

Review Submission Process

Place Request:

- **Sender**: User
- **Receiver**: User Interface + API Endpoint + API Controller
- **Message**: place request
- **Description**: The user makes a request for a place's id
- **Data**: PlaceData

Place Request Call:

- **Sender**: User Interface + API Endpoint + API controller
- **Receiver**: BLL Facade (Entity Services)
- **Message**: PlaceRequestCall
- **Description**: After simple validation of the PlaceData introduced by the user, the interface asks the BLL Facade to find the reference id of the place to which the user provided information
- **Data**: PlaceData

Find Place Service Call:

- **Sender**: BLL Facade (Entity Services)
- **Receiver**: Persistence Layer (Repository Services)
- **Message**: PlaceService.find_place()
- **Description**: The Facade asks the Persistence Layer to find the chosen place
- **Data**: PlaceData

Find Place in Database:

- **Sender**: Persistence Layer (Repository Services)
- **Receiver**: Database
- **Message**: PlaceRepository.find_place()

- **Description**: The Persistence Layer requests the Database to retrieve the id of the selected place
- **Data**: Place Data

Place Request Return:

- **Sender1**: Database
- **Receiver1**: Persistence Layer (Repository Services)
- **Sender2**: Persistence Layer (Repository Services)
- **Receiver2**: BLL Facade (Entity Services)
- **Sender3**: BLL Facade (Entity Services)
- **Receiver3**: User Interface + API Endpoint + API Controller
- **Message**: return
- **Description**: The Persistence Layer provides the id of the place if it found it or an empty value otherwise
- **Data**: place_id/empty

Option Fragment

- **Condition**: [place not found]
- **Description**: If the requested place is not found in the database, an error message is returned to the user, prompting them to enter a correct place. The main flow (submitting the review for a valid place) would then not proceed
- **Implied Flow**: If the place is found, the flow continues normally with the submit_review message (this part is outside the opt fragment)

Submit Review:

- **Sender**: User
- **Receiver**: User Interface + API Endpoint + API Controller
- **Message**: submit review
- **Description**: The user makes a request to submit a review
- **Data**: Review Data, place_id

Submit Review Call:

- **Sender**: User Interface + API Endpoint + API controller
- **Receiver**: BLL Facade (Entity Services)
- **Message**: SubmitReviewCall
- **Description**: The interface transmits the review data to the BLL Facade for review creation
- **Data: ReviewData, place_id**

Review Creation Service Call:

- **Sender**: BLL Facade (Entity Services)
- **Receiver**: Review class
- **Message**: ReviewService.create_review()
- **Description**: The BLL Facade creates an instance of Review containing the comment made by the user regarding the place
- **Data**: ReviewData, place_id, MyUser.id

Review Creation Return:

- **Sender**: MyReview
- **Receiver**: BLL (Entity Services)
- **Message**: return
- **Description**: The Review class confirms the creation of a new Review instance to the BLL and returns its reference
- **Data**: MyReview reference

New Review Registration:

- **Sender**: BLL Facade (Entity Services)
- **Receiver**: Persistence Layer (Repository Services)
- **Message**: ReviewService.register_review()
- **Description**: The BLL Facade asks the Persistence Layer to register the new review
- **Data**: MyReview reference

Data Storage:

- **Sender**: Persistence Layer (Repository Services)
- **Receiver**: Database
- **Message**: ReviewRepository.save_review()
- **Description**: The Persistence Layer asks the Database to save the new review. Returns a validated creation message to the user
- **Data**: MyReview reference

Register Review Return:

- **Sender1**: Database
- **Receiver1**: Persistence Layer (Repository Services)
- **Sender2**: Persistence Layer (Repository Services)
- **Receiver2**: BLL Facade (Entity Services)
- **Sender3**: BLL Facade (Entity Services)
- **Receiver3**: User Interface + API Endpoint + API Controller

- **Message**: return
- **Description**: The Persistence Layer confirms to the interface that the review has been created and stored
- **Data**: Signal

Review Submission Return:

- Sender: User Interface + API Endpoint + API Controller
- Receiver: User
- Message: submit saved
- Description: The user receives confirmation that its review has been successfully submitted
- Data: Message

Sequence Diagram: List of Places

The sequence diagram presented in Fig. 1 illustrates the dynamic interactions between the application layers (Presentation, Business Logic, and Persistence) when a registered and logged user attempts to display the list of places obeying certain criteria.

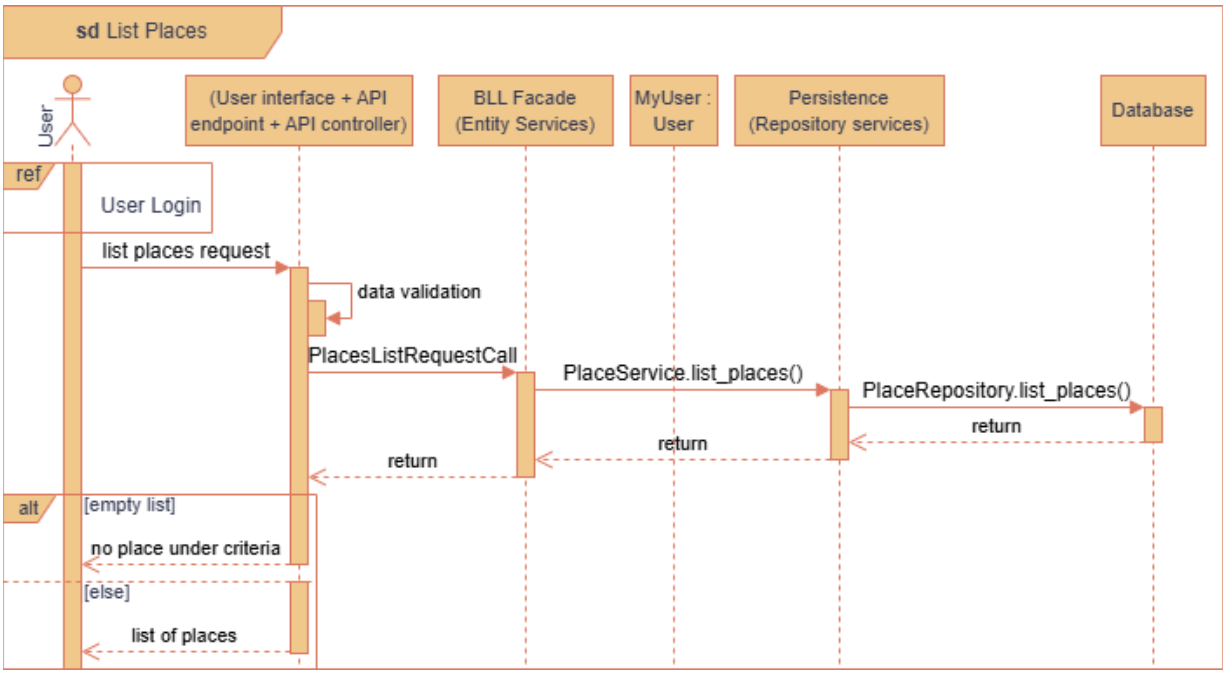

Fig. 1: list of places sequence diagram.

Actors, System Elements and Lifelines

- **User**: The external actor who interacts with the system
- **User Interface + API Endpoint + API Controller (Presentation Layer)**: Responsible for receiving user requests, performing simple input validation, data deserialization/serialization, and calling the Business Logic Layer (BLL)
- **BLL Facade (Entity Services)**: Entry point for business logic, orchestrator of complex business operations, interactions with the presentation layer and delegation of persistence operations to repository services
- **MyUser : User**: Represents an instance of the User class, an object MyUser of class User
- **Persistence Layer (Repository Services)**: Represents the data access component responsible for direct interactions with the database, handling the storage and retrieval of specific entity data
- **Database**: The data storage system

Ref User Login

Refers to the User Login sequence diagram and its flow. It calls to the fact that the user needs to be logged-in to perform the Place Creation operation, which is also why the object NewUser exists and is available at the business layer.

Process of Fetching a List of Places

List of Places Request:

- **Sender**: User
- **Receiver**: User Interface + API Endpoint + API Controller
- **Message**: list places request
- **Description**: The user makes a request for a list of places
- **Data**: Criteria Data

Places List Request Call:

- **Sender**: User Interface + API Endpoint + API controller
- **Receiver**: BLL Facade (Entity Services)
- **Message**: PlaceListRequestCall
- **Description**: After simple validation of the Criteria Data introduced by the user, the interface asks the BLL Facade to retrieve a list of places that satisfy such criteria
- **Data**: Criteria Data

List Places Service Call:

- **Sender**: BLL Facade (Entity Services)
- **Receiver**: Persistence Layer (Repository Services)
- **Message**: PlaceService.list_places()
- **Description**: The Facade asks the Persistence Layer to find the places fulfilling the criteria in order to list them
- **Data**: Criteria Data

Database Verification:

- **Sender**: Persistence Layer (Repository Services)
- **Receiver**: Database
- **Message**: PlaceRepository.list_places()
- **Description**: The Persistence Layer requests the Database to find the places in the database obeying the conditions in Criteria Data
- **Data**: Criteria Data

Places List Request Return:

- **Sender1**: Database
- **Receiver1**: Persistence Layer (Repository Services)
- **Sender2**: Persistence Layer (Repository Services)
- **Receiver2**: BLL Facade (Entity Services)
- **Sender3**: BLL Facade (Entity Services)
- **Receiver3**: User Interface + API Endpoint + API Controller
- **Message**: return
- **Description**: The Persistence Layer provides a list with the id's of all places satisfying the demanded criteria, it may return an empty list if no places were found to satisfy the criteria
- **Data**: list[place_id]/list[empty]

Alternative Fragment:

- **Conditions**:
    - **[empty list]**: No places corresponding to the provided criteria have been found in the database, a message is displayed saying as much.
    - **[else]**: At least one place was found to satisfy the provided criteria and the list of all such places is presented to the user via the interface.

Sequence Diagram: UserLogin

The sequence diagram presented in Fig. 1 illustrates the dynamic interactions between the application layers (Presentation, Business Logic, and Persistence) when a user attempts to log in.

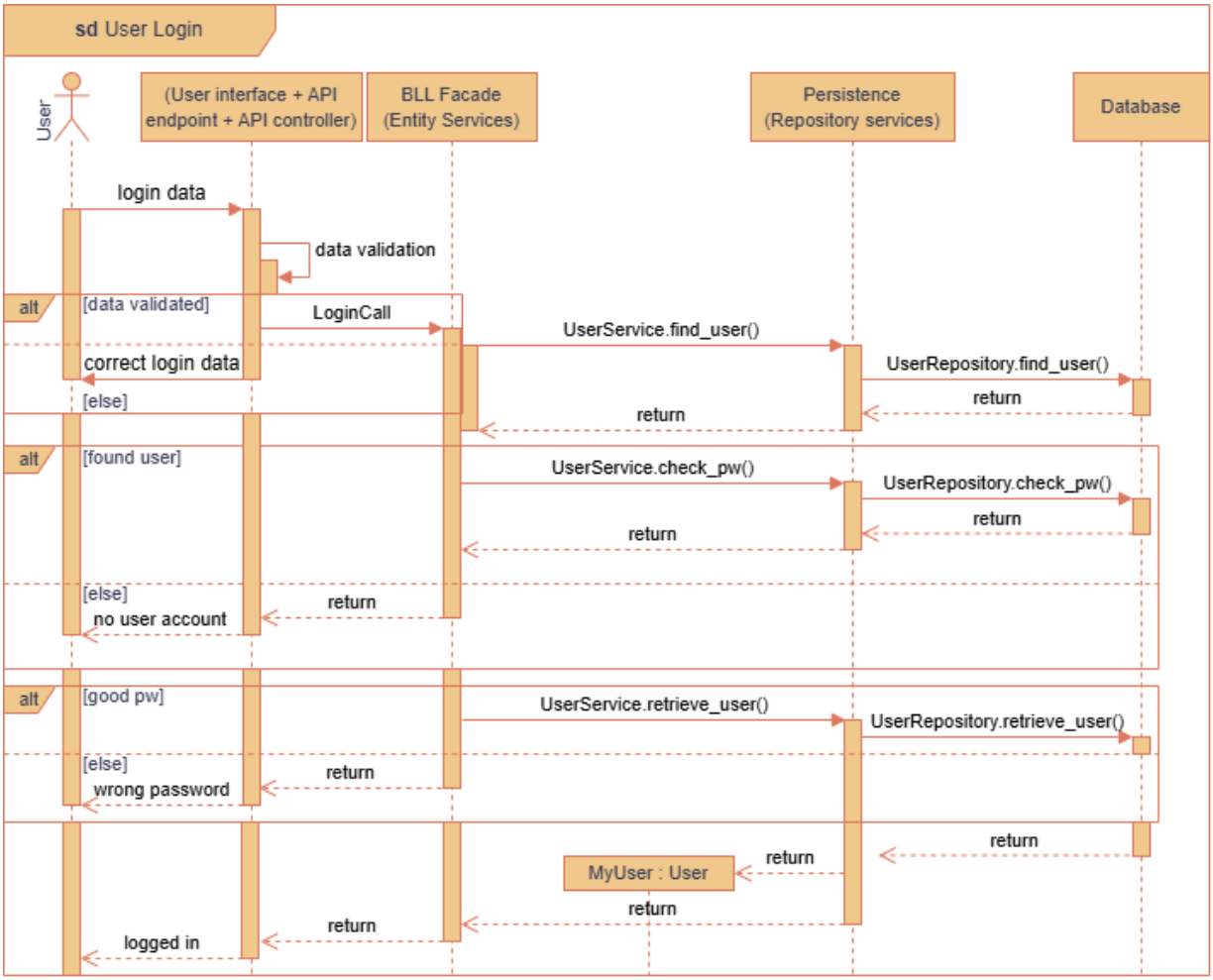

Fig. 1: user login sequence diagram.

Actors, System Elements and Lifelines

- **User**: The external actor who interacts with the system
- **User Interface + API Endpoint + API Controller (Presentation Layer)**: Responsible for receiving user requests, performing simple input validation, data deserialization/serialization, and calling the Business Logic Layer (BLL)
- **BLL Facade (Entity Services)**: Entry point for business logic, orchestrator of complex business operations, interactions with the presentation layer and delegation of persistence operations to repository services.

- **MyUser : User**: Represents an instance of the User class, an object MyUser of class User
- **Persistence Layer (Repository Services)**: Represents the data access component responsible for direct interactions with the database, handling the storage and retrieval of specific entity data
- **Database**: The data storage system

Ref UserLogin

Refers to the User Login sequence diagram and its flow. It calls to the fact that the user needs to be logged-in to perform the Place Creation operation, which is also why the object NewUser exists and is available at the business layer.

User Login Process

Login Data Transfer:

- **Sender**: User
- **Receiver**: User Interface + API Endpoint + API Controller
- **Message**: login data
- **Description**: The user sends, via the interface, its login credentials
- **Data**: Login Data (email, password)

Login Call:

- **Sender**: User Interface + API Endpoint + API controller
- **Receiver**: BLL Facade (Entity Services)
- **Message**: LoginCall
- **Description**: After simple validation of the LoginData introduced by the user, the interface asks the BLL Facade to verify if there is a user registered that corresponds to those credentials and retrieve its user_id if that's the case.
- **Data**: LoginData

Find User Service Call:

- **Sender**: BLL Facade (Entity Services)
- **Receiver**: Persistence Layer (Repository Services)
- **Message**: UserService.find_user()
- **Description**: The Facade asks the Persistence Layer to find the user corresponding to the given credentials
- **Data**: LoginData

Find User in Database:

- **Sender**: Persistence Layer (Repository Services)
- **Receiver**: Database
- **Message**: UserRepository.find_place()
- **Description**: The Persistence Layer requests the Database to verify if the user is, in fact, registered (user associated to email exists)
- **Data**: LoginData

Find User Return:

- **Sender1**: Database
- **Receiver1**: Persistence Layer (Repository Services)
- **Sender2**: Persistence Layer (Repository Services)
- **Receiver2**: BLL Facade (Entity Services)
- **Message**: return
- **Description**: The Persistence Layer informs the BLL it did or did not find the user
- **Data**: user_id/empty

Alternative Fragment:

- **Conditions**:
    - **[found user]**: The user corresponding to the login credentials was found and its id retrieved. The BLL can then proceed to check if the introduced password matches that of the user. It returns success/failure to the BLL after verification
    - **[else]**: The user was not found, the BLL returns a failure signal to the user interface which, in turn, displays a “no user account” message
- **Implied Flow**: If the user was *not* found the flow is interrupted, the user would have to restart the process providing different login credentials. The flow proceeds normally only when the user is found

Alternative Fragment:

- **Conditions**:
    - **[good pw]**: The password provided by the user matches that of the registered account. The BLL can then proceed to request the retrieval of the User object from the Persistence Layer by providing the previously obtained user_id
    - **[else]**: The password was not correct, the BLL returns a failure signal to the user interface which, in turn, displays a “wrong password” message
- **Implied Flow**: If the password was incorrect the flow is interrupted, the user would have to restart the process providing different login credentials. The flow proceeds normally only when the correct password is supplied

Retrieve User from Database Return:

- **Sender**: Database
- **Receiver**: Persistence Layer (Repository Services)
- **Message**: return
- **Description**: The Persistence Layer retrieves the appropriate User object from the database
- **Data**: MyUser object

Retrieve User Return:

- **Sender**: Persistence Layer (Repository Services)
- **Receiver**: BLL (Entity Services)
- **Message**: return
- **Description**: The Persistence Layer makes the appropriate User object available to the BLL and sends a success signal to it
- **Data**: MyUser reference, signal

Login Call Return:

- **Sender**: BLL (Entity Services)
- **Receiver**: User Interface + API Endpoint + API Controller
- **Message**: return
- **Description**: The BLL signals the success of the login data API call to the User Interface
- **Data**: Signal

Login Data Return:

- **Sender**: User Interface + API Endpoint + API Controller
- **Receiver**: User
- **Message**: logged in
- **Description**: The User Interface informs the user that it is logged in
- **Data**: Message