SOEN 342 Project

Section II

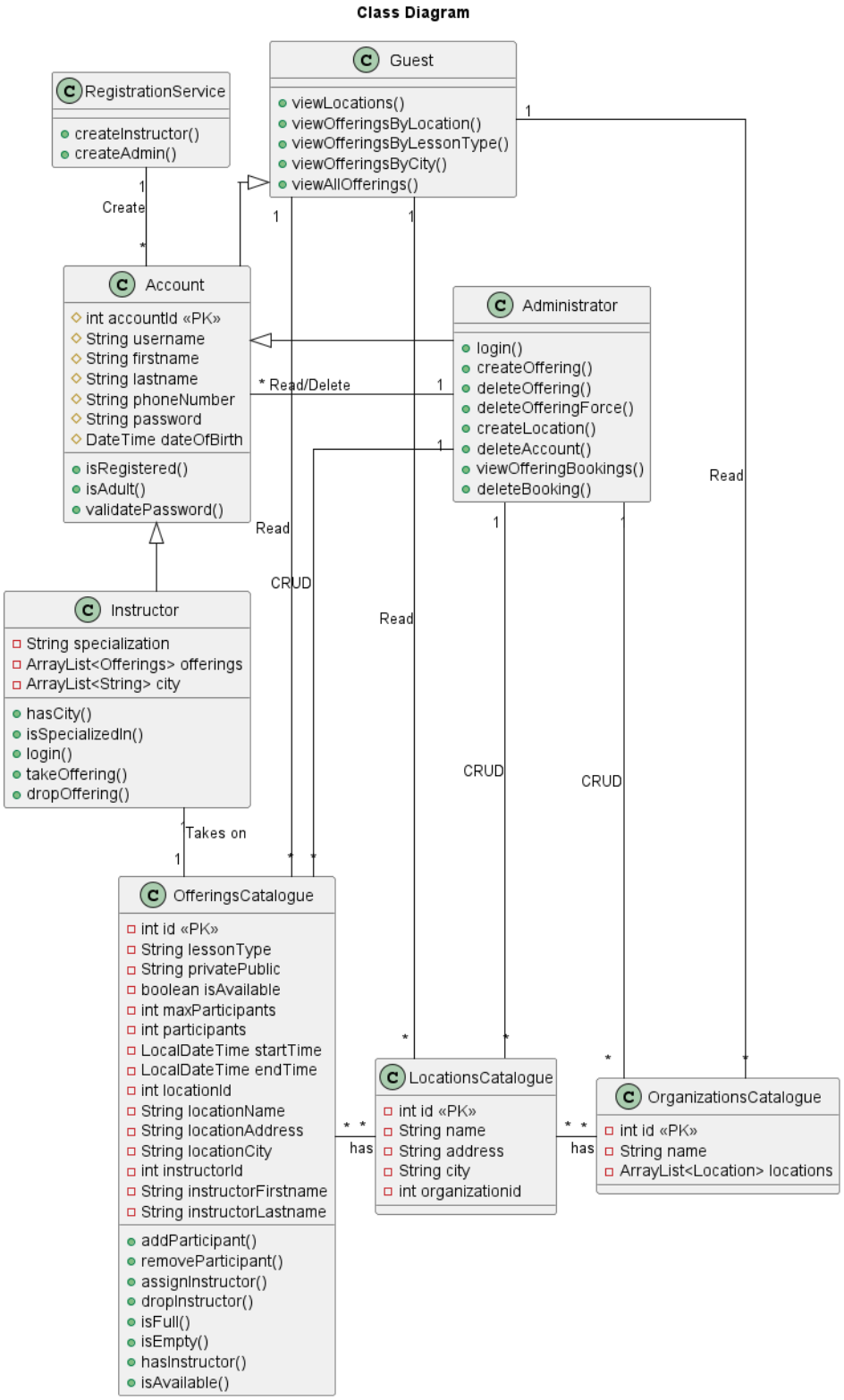
Franco Dominguez (40256199)
Ashkan Forghani (40176561)
Team #33

Iteration #2

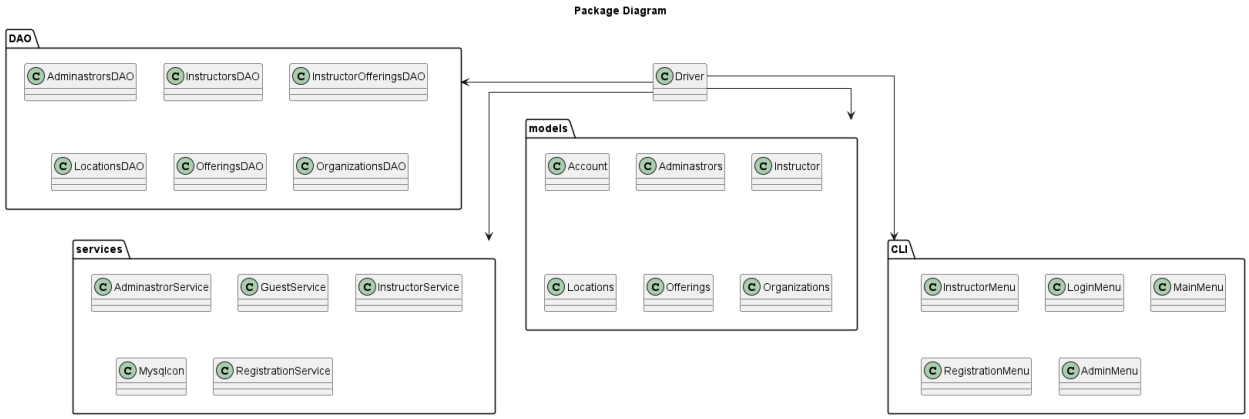
Use Case 1:

Process Offerings

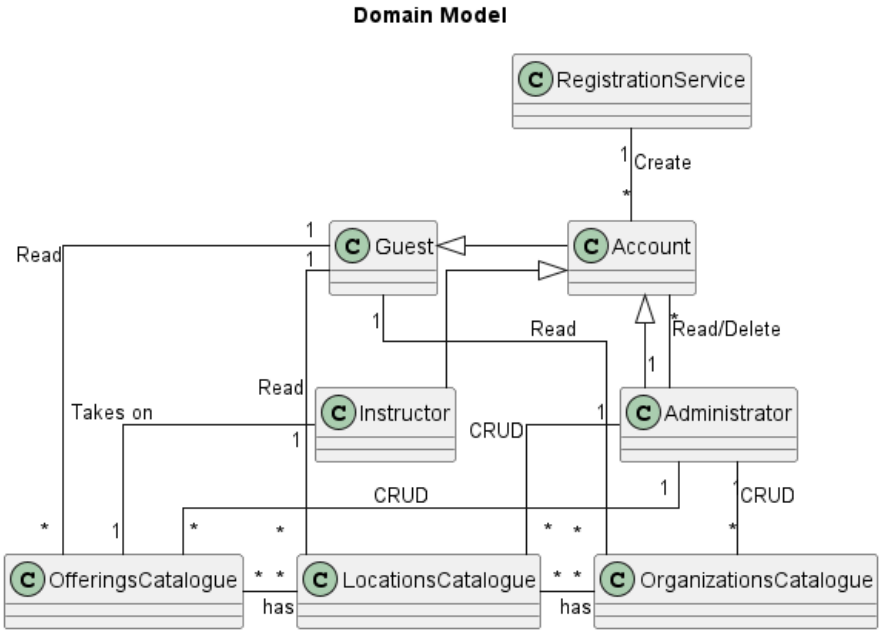
Class Diagram



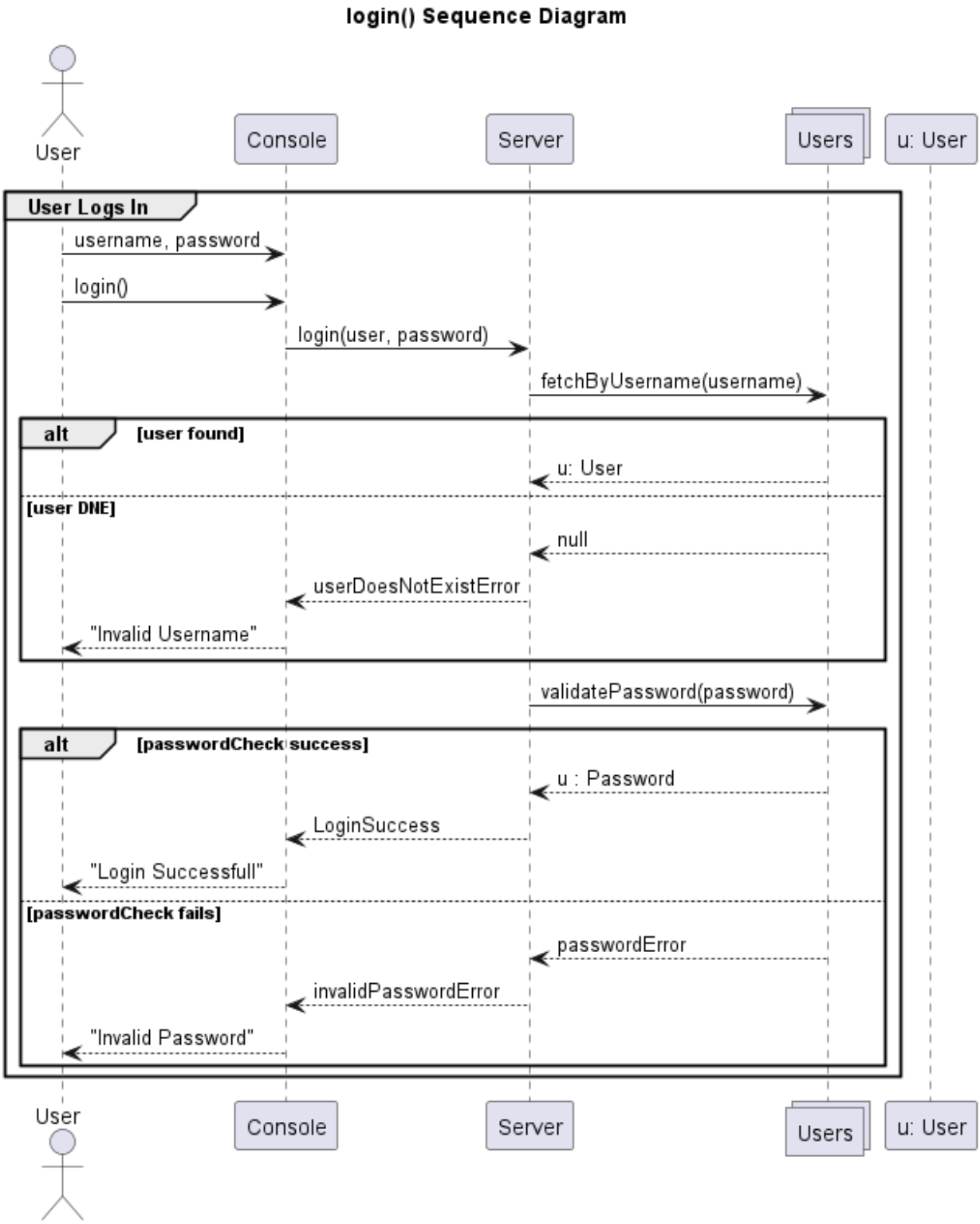
Package Diagram

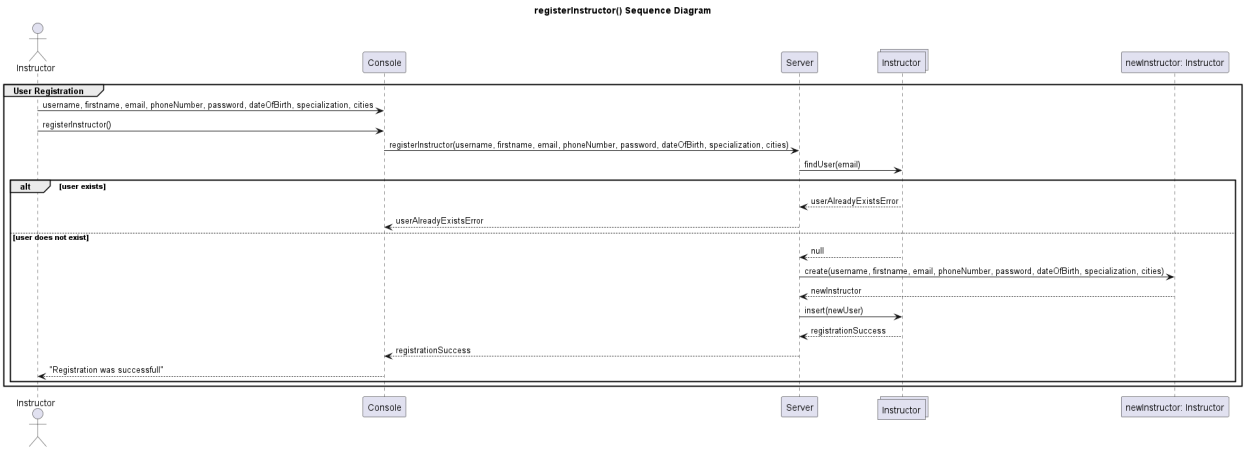


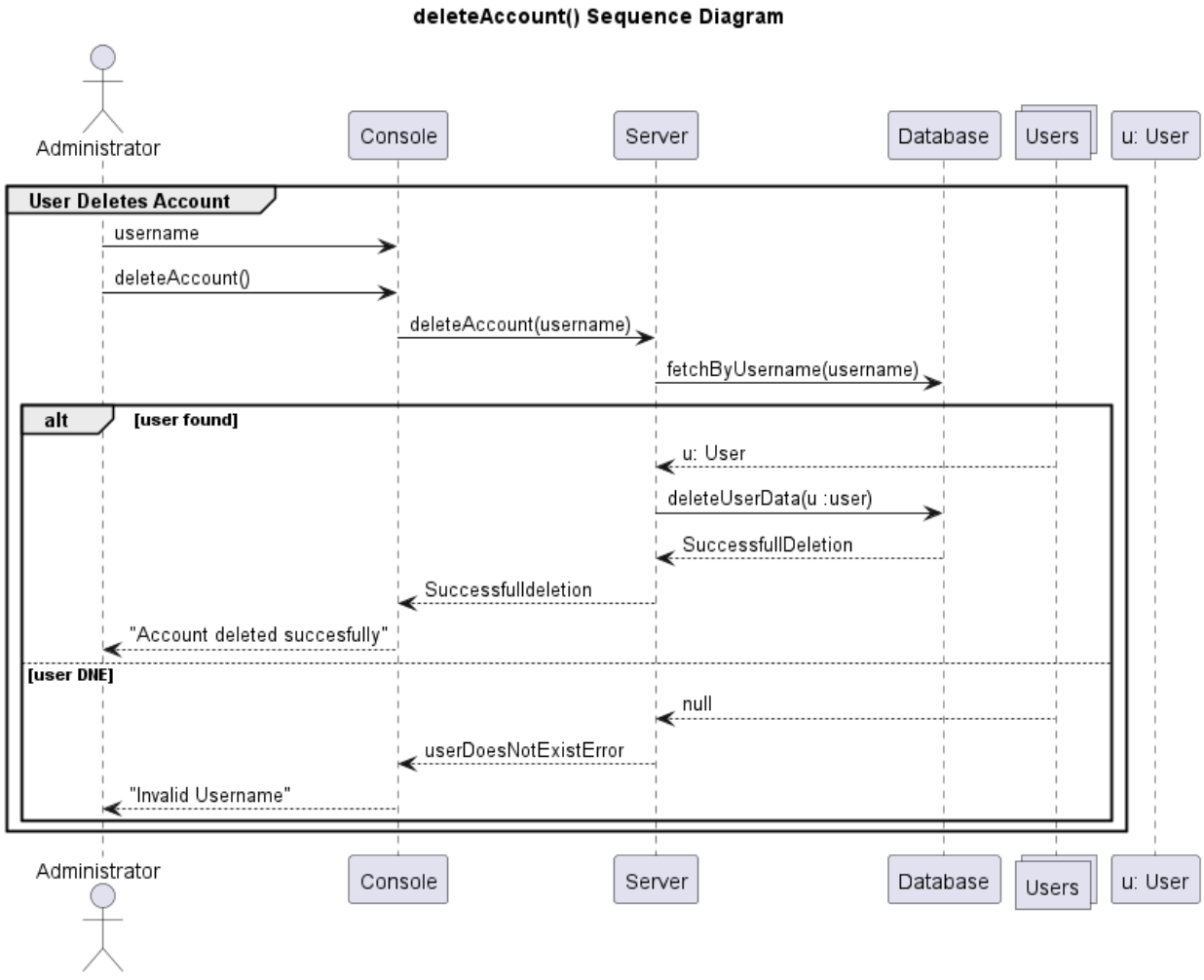
Domain Model

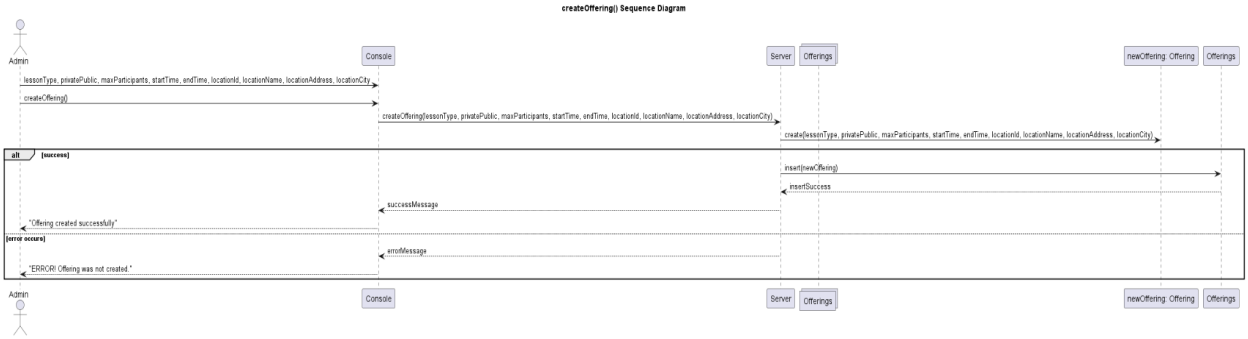


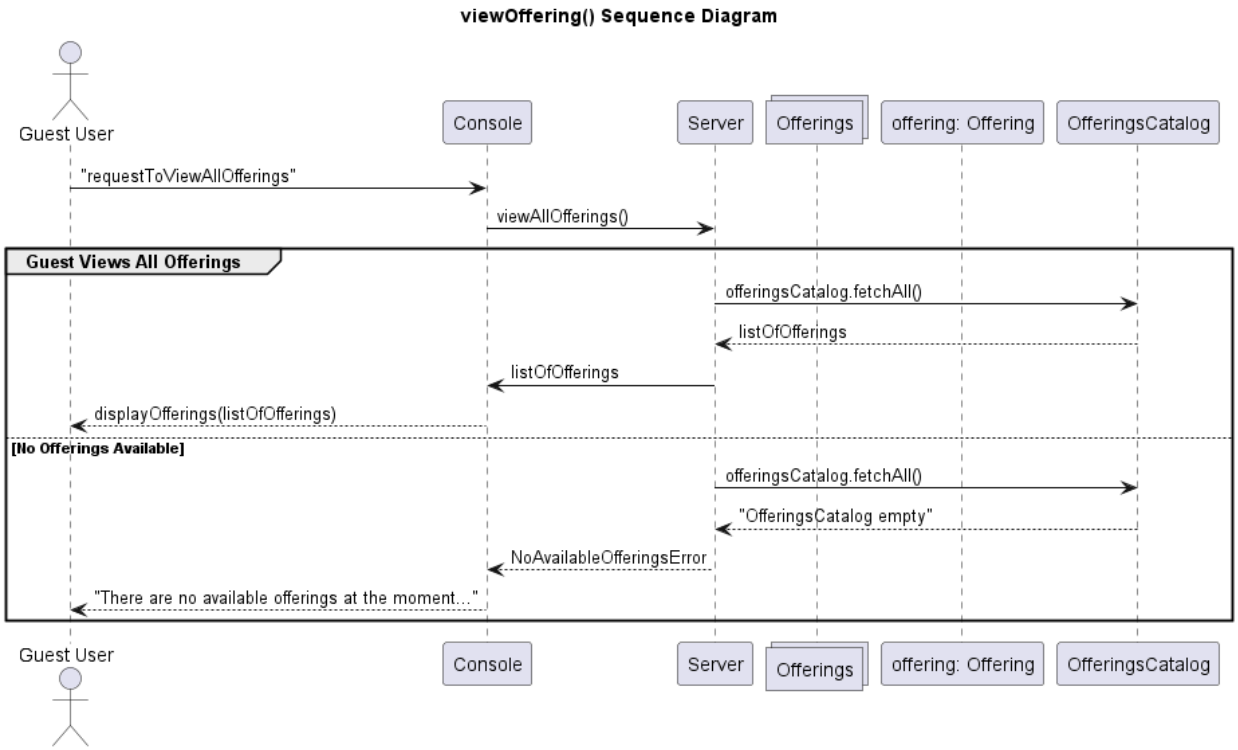
System Sequence and Interaction Diagrams



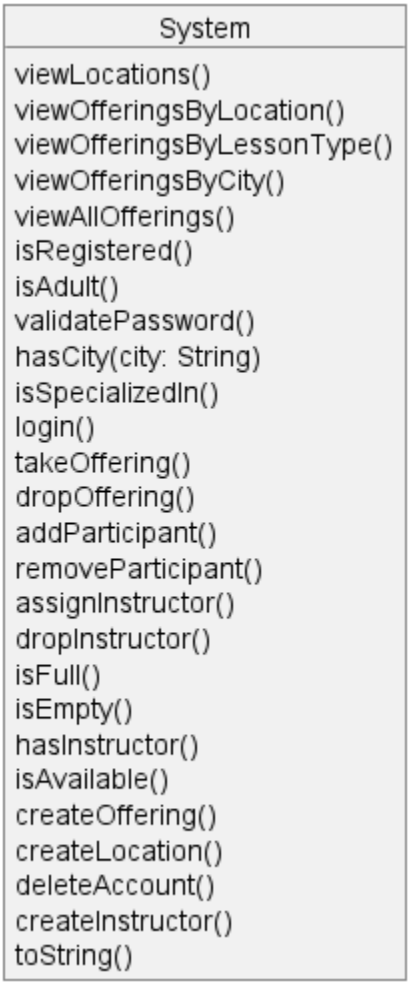

registerInstructor() Sequence Diagram
Instructor
Console
Server
Instructor
newInstructor: Instructor
User Registration
username, firstname, email, phoneNumber, password, dateOfBirth, specialization, cities
registerInstructor()
registerInstructor(username, firstname, email, phoneNumber, password, dateOfBirth, specialization, cities)
findUser(email)
alt
[user exists]
userAlreadyExistsError
userAlreadyExistsError
[user does not exist]
null
create(username, firstname, email, phoneNumber, password, dateOfBirth, specialization, cities)
newInstructor
insert(newUser)
registrationSuccess
registrationSuccess
"Registration was successfull"
Instructor
Console
Server
Instructor
newInstructor: Instructor


deleteAccount() Sequence Diagram
Administrator
Console
Server
Database
Users
u: User
User Deletes Account
username
deleteAccount()
deleteAccount(username)
fetchByUsername(username)
alt
[user found]
u: User
deleteUserData(u :user)
SuccessfullDeletion
Successfulldeletion
"Account deleted succesfully"
[user DNE]
null
userDoesNotExistError
"Invalid Username"
Administrator
Console
Server
Database
Users
u: User


createOffering() Sequence Diagram
Admin
Console
Server
Offerings
newOffering: Offering
Offerings
lessonType, privatePublic, maxParticipants, startTime, endTime, locationId, locationName, locationAddress, locationCity
createOffering()
createOffering(lessonType, privatePublic, maxParticipants, startTime, endTime, locationId, locationName, locationAddress, locationCity)
create(lessonType, privatePublic, maxParticipants, startTime, endTime, locationId, locationName, locationAddress, locationCity)
alt
[success]
insert(newOffering)
insertSuccess
successMessage
"Offering created successfully."
[error occurs]
errorMessage
"ERROR! Offering was not created."
Admin
Console
Server
Offerings
newOffering: Offering
Offerings


viewOffering() Sequence Diagram
Guest User
Console
Server
Offerings
offering: Offering
OfferingsCatalog
"requestToViewAllOfferings"
viewAllOfferings()
Guest Views All Offerings
offeringsCatalog.fetchAll()
listOfOfferings
listOfOfferings
displayOfferings(listOfOfferings)
[No Offerings Available]
offeringsCatalog.fetchAll()
"OfferingsCatalog empty"
NoAvailableOfferingsError
"There are no available offerings at the moment..."
Guest User
Console
Server
Offerings
offering: Offering
OfferingsCatalog

System Operations



Operation Contracts

Contract CO1: viewLocations
Operation: viewLocations()
Cross References: Use Case: Get available offerings
Preconditions:

- The specified location exists and is active in the system

Postconditions:

- A list of all active locations is retrieved and displayed to the user.

- Only locations that are open and available for offerings are included.

Contract CO2: viewOfferingsByLocation
Operation: viewOfferingsByLocation(location: Location object)
Cross References: Use Case: View Offerings by Location
Preconditions:
- The specified location exists and is active in the system.

Postconditions:
- A list of offerings available at the specified location is retrieved.
- Only offerings with available time slots are included.
- Offerings that are fully booked are excluded.

Contract CO3: viewOfferingsByLessonType
Operation: viewOfferingsByLessonType(lessonType: String)
Cross References: Use Case: View Offerings by Lesson Type
Preconditions:
- The specified lesson type is available in the system.
- Bookings of the specified lesson type exists

Postconditions:
- A list of offerings for the specified lesson type is retrieved.
- Only offerings with available time slots are included.
- Offerings that are fully booked are excluded.

Contract CO4: viewOfferingsByCity
Operation: viewOfferingsByCity(cityName: String)
Cross References: Use Case: View Offerings by City
Preconditions:
- Bookings exists that are in the specified City

Postconditions:
- A list of offerings for the specified city is appears for the user
- Only offerings with available time slots are included.
- Offerings that are fully booked are excluded.

Contract CO5: viewAllOfferings
Operation: viewAllOfferings()
Cross References: Use case: User wants to view all offerings
Preconditions:
- There exists one or many available offerings

Postconditions:
- A list of all available offerings is displayed to the user

Contract CO6: isRegistered
Operation: isRegistered()
Cross References: Use case: To check if an account is registered

Preconditions:

- Account must exist in the DB.

Postconditions:

- Return True or False if the account is or is not registered.

Contract CO7: isAdult
Operation: isAdult()
Cross References: Use case: To check if an account is over 18 years old.
Preconditions:

- The account exists in the DB.

Postconditions:

- Returns true or false if the account holder is or is not over 18 years of age.

Contract CO8: validatePassword
Operation: validatePassword(password: String)
Cross References: Use case: To validate password when logging into an account.
Preconditions:

- The account exists in the DB.

Postconditions:

- Returns true or false if the password matches or does not match the account password in questions.

Contract CO9: hasCity
Operation: hasCity(city: String)
Cross References: Use case: Check if the instructor is registered to a specific city.
Preconditions:

- Instructor account is registered

Postconditions:

- Return true or false if the city matches or does not match the city the instructor has registered on their account.

Contract CO10: isSpecializedIn
Operation: isSpecializedIn(specialization: String)
Cross References: Use case: Check if an instructor has a specific specialization.
Preconditions:

- Instructor account exists in the system.

Postconditions:

- Returns true or false if the specialization matches or does not match the specialization registered on the instructor's account.

Contract CO11: login
Operation: Login()
Cross References: Use Case: User logs into the system.
Preconditions:

- The username and password must match a registered account in the system.

Postconditions:

- If credentials are valid, the user is logged into the system and a session is created.
- If credentials are invalid, the login attempt fails, and no session is created.

Contract CO12: takeOffering
Operation: takeOffering(offering: Offering object)
Cross References: Use Case: User enrolls in a selected offering.
Preconditions:

- The instructor is registered and logged into the system.
- The offering exists and has available slots.

Postconditions:

- The instructor is enrolled in the offering, reducing available slots by one.
- If the offering is full, it is marked as unavailable

Contract CO13: dropOffering
Operation: dropOffering(offering: Offering object)
Cross References: Use Case: User drops a previously taken offering.
Preconditions:

- The instructor is registered and logged into the system.
- The instructor has the offering in question.

Postconditions:

- The instructor is removed from the offering, increasing available slots by one.
- If the offering has open slots, it is marked as available.

Contract CO14: addParticipant
Operation: addParticipant()
Cross References: Use case: Update participant count in given offering.
Preconditions:

- Offering already exists.

Postconditions:

- Offering's participant count is update by +1.

Contract CO15: removeParticipant
Operation: removeParticipant()
Cross References: Use case: Remove a participant from a specific offering.
Preconditions:

- The participant is enrolled in the specified offering.

Postconditions:

- The participant is removed from the offering, increasing available slots by one.

Contract CO16: assignInstructor

Operation: assignInstructor()
Cross References: Use case: Assign an instructor to an offering.
Preconditions:

- The instructor and offering both exist in the system.

Postconditions:

- The instructor is assigned to the offering, making it available for public enrollment.

Contract CO17: dropInstructor
Operation: dropInstructor()
Cross References: Use case: Remove an instructor from an offering.
Preconditions:

- The instructor is currently assigned to the specified offering.

Postconditions:

- The instructor is removed from the offering, which may affect the offering's availability.

Contract CO18: isFull
Operation: isFull()
Cross References: Use case: Check if an offering is fully booked.
Preconditions:

- The offering exists in the system.

Postconditions:

- Returns true if the offering has no available slots; false otherwise.

Contract CO19: isEmpty
Operation: isEmpty()
Cross References: Use case: Check if an offering has no participants.
Preconditions:

- The offering exists in the system.

Postconditions:

- Returns true if the offering has no participants; false otherwise.

Contract CO20: hasInstructor
Operation: hasInstructor()
Cross References: Use case: Check if an instructor is assigned to an offering.
Preconditions:

- The offering exists in the system.

Postconditions:

- Returns true if the offering has an assigned instructor; false otherwise.

Contract CO21: isAvailable
Operation: isAvailable()
Cross References: Use case: Check if an offering is available for enrollment.
Preconditions:

- The offering exists in the system.

Postconditions:

- Returns true if the offering has available slots and is open for enrollment; false otherwise.

Contract CO22: createOffering
Operation: createOffering()
Cross References: Use case: Admin creates a new offering.
Preconditions:

- The admin is logged into the system.

Postconditions:

- A new offering is added to the system with available slots.

Contract CO23: createLocation
Operation: createLocation()
Cross References: Use case: Admin creates a new location.
Preconditions:

- The admin is logged into the system.

Postconditions:

- A new location is added to the system and is available for offerings.

Contract CO24: deleteAccount
Operation: deleteAccount()
Cross References: Use case: Admin deletes a client account.
Preconditions:

- The account exists in the system.

Postconditions:

- The account is removed from the system, and associated bookings are canceled.

Contract CO25: createInstructor
Operation: createInstructor()
Cross References: Use case: Admin creates a new instructor account.
Preconditions:

- The instructor information entered is valid.

Postconditions:

- A new instructor account is created in the system.

Contract CO26: createAdmin
Operation: createAdmin()
Cross References: Use case: Super admin creates a new admin account.
Preconditions:

- The admin information entered is valid.

Postconditions:

- A new admin account is created in the system and info is stored in DB.

Contract CO27: toString
Operation: toString()
Cross References: Use case: General object string representation.
Preconditions:

- The object exists.

Postconditions:

- A string representation of the object is returned, detailing its attributes.