CS5200 Database Management

Project Requirements

Description

Here are the minimum requirements for this course's project. Pick a domain of your choice, e.g., music, education, movies, restaurants, events, productivity, healthcare, finance, etc. Create a data model that captures at least 2 types of users, 2 domain objects and their relations, e.g., playlists have songs, courses have sections, students are enrolled in sections, etc. You will identify at least 2 types of users that will use your application, e.g., a student and a faculty user, a buyer and a seller, etc. Implement a data model and user interface that will support a set of use cases and features listed in this document. If you can think of an alternative set of features we can negotiate swapping some of the features listed here for some of yours. If you have a novel project idea that escapes the confines of the requirements listed here, reach out to the instructors and we can agree on an alternate set of requirements tailored to your particular project. For the most part we expect most of you to fall within the boundaries of these general requirements.

User Roles

Software Engineer / leader

Identify several types of 2 human end users that would use your application. Example users would be

1. Online store application: sellers, buyers
2. Online project management: developers, quality assurance, project managers, business analysts
3. Online restaurant: cooks, waiters, delivery, receptionists, manager, owner
4. Online sports: player, coach, director, spectator, fan
5. Online fishing journal: fisherman, partner, boat captain
6. Online university: students, faculty, staff

The navigation and user interface should adapt depending on whether a user is logged in or not, and depending on the type or role of the user. Your application must support at least 2 user types or roles. For instance, a user can be a buyer or a seller, a user can be a book author or just a passive reader.

Admin Role Requirements

In addition to the 2 user roles, support an ***admin*** user that can administer the following: all types of users, at least one domain object and at least one type of relationship. Here are some examples of what an admin user can do for various project types

1. Manage CRUD operations on all types of users, including
    a. Create, Update, Delete users
    b. Update a user's role
    c. Update at least one other user field
2. Manage CRUD operations on all types of domain objects, including
    a. Create, Update, Delete domain objects
    b. Update at least one field of domain objects
3. Manage CRUD operations on at least one user to user relationship
4. Manage CRUD operations on at least one user to domain relationship
5. Manage CRUD operations on at least one domain to domain object relationship

End User Roles Requirements

Your application must support at least two end user roles. This is in addition to the admin and anonymous roles. User roles must have at least one use case that is specific to the role. Users can choose their role once they are logged in. It is ok for a use cases of one user role to complement with use cases of some other user role. For instance, a seller and a buyer both interact with products with various use cases. The seller can create and update a product's price and description, and a buyer can chage the quantity available by buying a particular product. Another example might be a faculty can create and update a quiz, but a student can answer the questions of a quiz.

Here are a couple of examples of how the interface might be different for different roles. The interfaces are based on the goals each actor wants to accomplish

1. Online store application - depending on their roles, users will have different user interfaces
    a. sellers can - CRUD products, stores, orders, e.g., buyers can create a store and add products to the store, then change their descriptions, change their prices, view online orders, fulfill the orders, cancel the orders, etc.
    b. buyers can - search for products, add to a shopping list, checkout, view their orders, cancel an order, review products
2. Online university application - depending on their roles, users will have different user interfaces
    a. faculty can - CRUD courses, sections, assignments, quizzes
    b. students can - search courses/sections, register for sections, take quizzes, submit assignments

Required Data Model

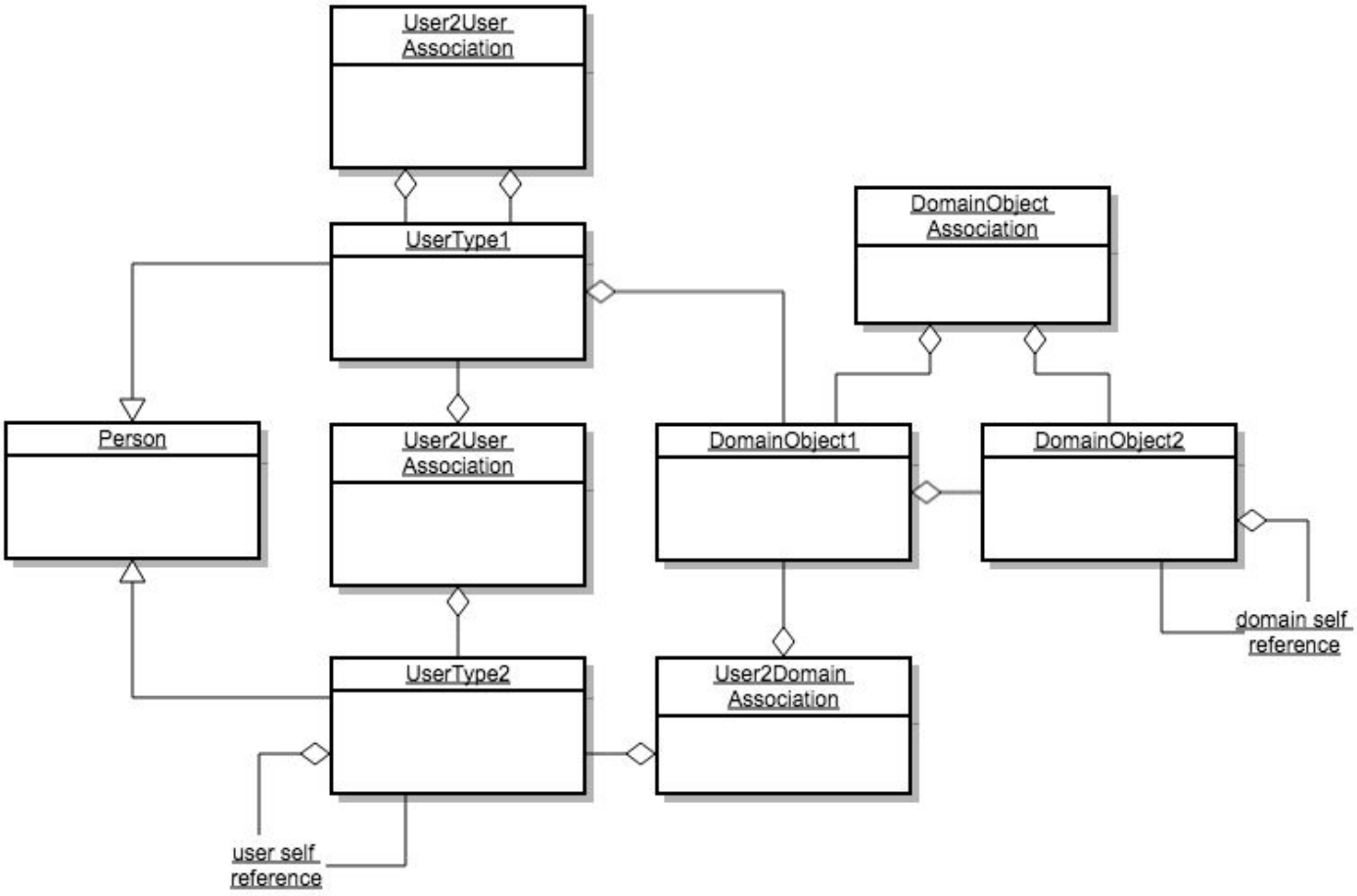
Your data model must represent the following entities and relationships

1. ***User*** - allows capturing data such as username, password, date of birth, first name, last name, phone, address, email, and other personal information. There should be at least two distinct types of users with at least 1 use case that is specific to that user
2. ***Domain object*** - represents data for the particular domain you chose. Domain objects are non human concepts specific to the domain you choose, e.g., recipes, sports team, tasks, fishing trips, products, movies, songs. There should be at least two distinct types of domain objects. Do not confuse domain objects with relationship objects. For instance, a MovieReview is an object that relates a Movie and the User who wrote the review
3. ***User to user relationship*** - model at least one user to user relation. Capture an association relation between users, e.g., user follows a user, a user sends a message to another user
4. ***User to domain object relationship*** - model at least one user to domain object relation. Capture an association relationships between users and a domain objects, e.g., user creates a recipe, user likes a movie, user reviews an article
5. ***Domain to domain relationship*** - model at least one domain to domain object. Represent association relationships between domain objects, e.g., a category has many products, a fishing trip has many fish, a team has many players, a blog has many posts, a portfolio has many stocks
6. ***One to one relationship*** - model at least one one to one relation
7. ***One to many relationship*** - model at least one one to many relation. Represent one to many relationships such as fishing trip has many fish, a team has many players, a product has many reviews, a blog post has comments
8. ***Many to many relationships*** - model at least one many to many relation. Represent many to many relationships such as a student can be in many sections and a section can have many students, a

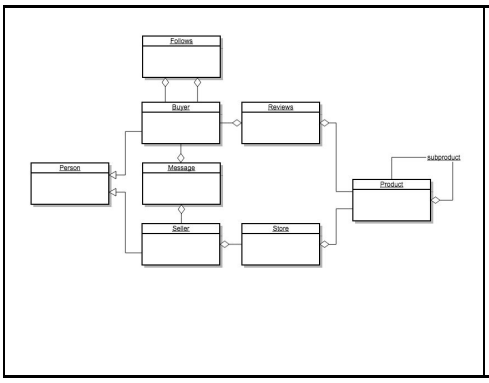
developer can be in a project and a project can have many developers, a user reviews many products and a product is reviewed by many users

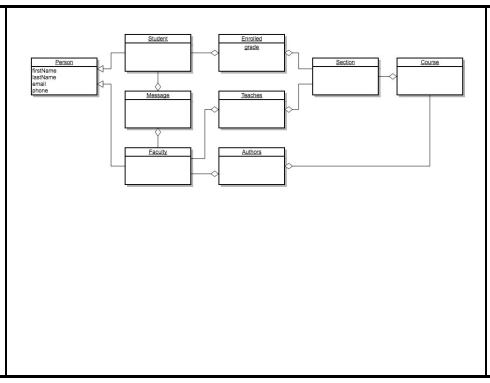
9. ***Self reference*** - model at least one self reference.
10. ***Generalization*** - model at least one generalization.
11. ***Enumeration*** - model at least one enumeration.
12. ***Aggreggation*** - at least one aggregation
13. ***Composition*** - at least one composition

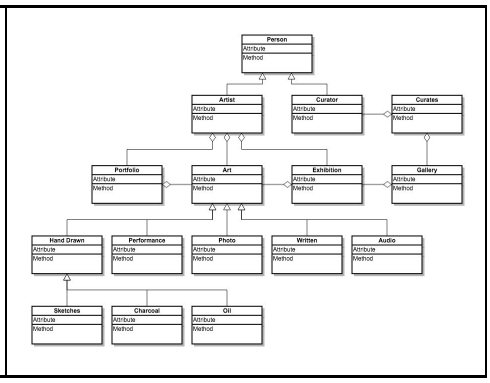
Here's a generic UML diagram capturing the user, domain objects, and several types of relationships. Use this as a guide to create your own data model



Here are a couple more examples data models. Click to expand.







User Interface Requirements

All applications need some form of user interface. You are free to use any technology you are comfortable with to implement your user interface. The UI can be implemented with Java, C#, Mobile, Desktop, or Web application. The whole application can be implemented in a single screen or some complex navigation between screens.

Feature Requirements

The following set of features are required. You can provide various features in a single user interface or decide to spread the feature across several interfaces. You are free to layout the user interface and implement navigation any which way you want as long as you implement the features below.

1. ***Home*** - this will be the landing page of your project. It should look spectacular. I should be proud to share the URL with colleagues. The purpose of the Website should be clear
2. ***Login / Register*** - allow users to register and login. Users should be able to use the application without login in. Only when accessing private or sensitive information should the application force users to login. Or if the application needs to know the identity of the user to execute an action.
3. ***Profile*** - allow users to edit personal information and navigate to objects related to the user, e.g., users they are following, domain objects they are related to, e.g., fishing trips they created, fish they caught, products they bought, sports team they are following, articles they wrote. Profiles can have a public and a private version
4. ***Search / Search Results*** - allow users to search for domain objects. Results are shown as a summarized list. Clicking on a search result navigates to a details view for the domain object. Retrieve data from a public Web service. For instance, if you chose movies as your domain objects, then users should be able to search for movies from a public API. Users should be able to see a summary of the search results and navigate to a detail page that shows a detail view of the result
5. ***Details*** - shows a detailed view of a domain object including any relationships it has with other objects, e.g., a product shows a detail description, rating, and reviews by users. Clicking on the user that posted the review navigates to that user's profile. Retrieve details from a public Web service
6. ***Web Service*** - use a public Web service such as Yelp, Amazon, Best Buy, Walmart, or Weather.com, to implement the search and details pages. Search for public Web services from services such as http://www.programmableweb.com. You may download data from data services and import it into your database instead of using the remote service. The amount of data you import into your database should be substantial enough to realistically demonstrate the use cases
7. ***User Management*** - a user with admin credentials, e.g., admin/admin, must be able to create, update, and remove all other users. They should also be able to see a list of all the users in the system. Admin users should be able to search for a user and then view and update the details for that user. The navigation should adapt depending on the whether a user is admin or not, and depending on their role in the application

Deliverables

Deliver your project in a private github repository named cs5200-f19-lastname-firstname, where lastname and firstname are your last name and first name. Invite the TAs and instructor as collaborators. Submit the url of your github repository in blackboard. In your GitHub wiki, create a web page called ***Design***. Using a UML tool,

create a class diagram that captures users, domain objects, and relations between users and domain objects. Here are some free UML tools

1. Visual Paradigm
2. Lucid Chart
3. UMLet
4. Violet UML

Class diagram should include

1. Cardinality
2. Class(es) modeling user(s)
3. Class(es) modeling additional user roles, if applicable based on team size and graduate level
4. Class(es) modeling domain object(s)
5. At least one one to many relation
6. At least one many to many relation
7. A relation between users
8. A relation between domain objects
9. A relation between users and domain objects

Embed the UML diagram in your Design wiki page providing a description of your design. When you finish your project create the following accounts, using the username/password shown

1. admin/admin - user with admin role
2. alice/alice - user with role 1
3. bob/bob - user with role 2
4. charlie/charlie - user with role 3 (optional)
5. dan/dan - user with role 4 (optional)

In your GitHub wiki, create a page called ***Testing***. In the Testing page, create detailed steps to test your application. The tests should validate that you have met the requirements as described by this document. Write DETAILED steps that test the following

1. A user related to another user, e.g., a fan follows their favorite cricket player, a manager gives an employee a raise, an airbnb guest leaves a message for their host, etc.
2. A user searches for a list of domain objects that match a criteria, e.g., search for movies with a title, search for events around boston, search for restaurants near me, etc.
3. A user views details of a particular domain object listed in the search results, e.g., clicking on a particular movie displays more details of the movie, clicking on a particular restaurant displays more details for that restaurant, etc.
4. A user views all domain objects related to the user, e.g., a movie critic sees all their movie reviews listed in their profile page, a buyer sees all orders and/or items listed in their profile page, etc.
5. A user views all other users related to the user, e.g., a fan sees all their favorite football (soccer) players in their profile page, a social network user sees all other users they are following and sees users that are following them, etc.
6. A user related to a domain object, e.g., a user bookmarks their favorite book, a user creates a playlist, a buyer buys a product
7. A domain object related to another domain object, e.g., add a song to a playlist, an order that contains several products, a recipe that contains several ingredients, etc.
8. An admin creates a user
9. An admin lists all users
10. An admin edits/updates a particular user

11. An admin removes a user

Create a 5 to 10 minute video on YouTube where you present and demonstrate your project. Submit a link to your video. Provide the link of your video in your GitHub wiki. Your video should include the following:

1. 1 minute introduction
2. 1 minute problem statement
3. 1 minute solution statement
4. 1 minute architecture and technology stack
5. 1 minute data model
6. 5 to 10 minute demo: briefly show how to use your Web application and highlight the various features

Submit a link to your source code repository and make sure it is accessible to the instructors