Web Development

Project Requirements

# Description

Here are the minimum requirements for this course's project. You will identify at least 2 types of users that will use your application, e.g., a student and a faculty user. 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 me and we can agree on an alternate set of requirements tailored to your particular project. For the most part I expect most of you to fall within the boundaries of these general requirements.

## User Roles

Identify at least 2 types of human users that would use your Website. 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 role of user. Your Website must support at least two user roles (one for undergraduates). For instance, a user can be buyer and a seller, a user can be a book author or just a passive reader. A user can be a regular user or an admin. The user interface should adapt to the user role by hiding certain links, or entire pages and functionality. For instance, admin users might have access to admin pages that would otherwise be inaccessible to the rest of the users of the application. A particular user can have multiple roles, but they can only be in a particular role at any one time, or, if you prefer, the Website can provide a mechanism to toggle between roles, but the interface must clearly show that the user is in that new role.

### Admin Role Requirements

In addition to the two user roles, support an ***admin*** user that can administer all the entities of the Website including other users, domain objects, and all relations. Each user role should have their dedicated user interface, although they might share some common use cases and interfaces. Admin CRUD operations on domain objects and relations are optional for undergraduates. Admin CRUD operations on users is required by all students. Here are some examples of what an admin user can do for various project types

1. Online store application - admins should be able to (CRUD = Create/Read/Update/Delete)
   1. CRUD users - List/create/remove/update sellers and buyers
   2. CRUD domain objects and relations - List/create/remove/update online orders, favorite products, etc. (optional for undergraduates)
2. Online university application - admins should be able to
   1. CRUD actors - List/create/remove/update faculty and students
   2. CRUD domain objects and relations - List/create/remove/update courses, sections, enrollments, etc. (optional for undergraduates)
3. Online movie social network - admins should be able to
   1. CRUD actors - List/create/remove/update fans, actors, critics, directors
   2. CRUD domain objects and relations - List/create/remove/update movies, reviews, playlists, favorites, etc. (optional for undergraduates)

### Anonymous User Requirements

Your project should support some functionality for anonymous users and only force users to login if user identity is required to fulfill a service. For instance, in an online store, anonymous users should be able to search for products, view product details, read product reviews, etc. If a user would like to bookmark a product, or comment on a product, or add a product to a shopping cart, then, and only then, would the Website ask the user to identify themselves or register.

### End User Roles Requirements

Your Website should support multiple roles. Graduate students are required to support at least 2 roles. Undergraduate students are required to support at least one role. One additional role is required for each additional team member. 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
   1. 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.
   2. 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
   1. faculty can - CRUD courses, sections, assignments, quizzes
   2. students can - search courses/sections, register for sections, take quizzes, submit assignments

### Additional Role Requirements

Students are allowed and encouraged to join into teams of at most 3 students. The final project complexity must reflect the added team capacity, therefore one additional user role must be supported for each additional team member. The following are all the

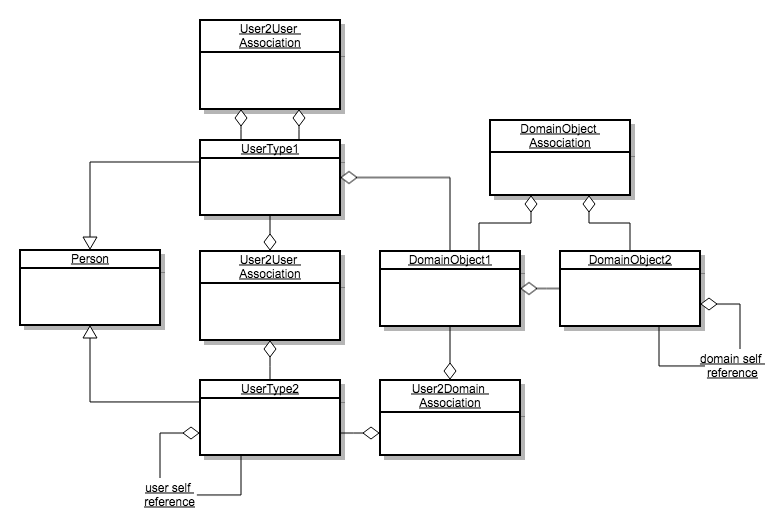
|  |  |  |
| --- | --- | --- |
| **Number of**  **Team Members** | **Required Roles** | |
| **Undergraduates** | **Graduate** |
| 1 | Admin, Anonymous, Role 1 | Admin, Anonymous, Role 1, Role 2 |
| 2 | Additional Role 2 | Additional Role 3 |
| 3 | Additional Role 3 | Additional Role 4 |

## Required Data Model

Your data model should 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 types of users
2. ***Domain object*** - represents data for the particular domain you chose, e.g., recipes, sports team, tasks, fishing trips, products, categories, movies, songs. There should be at least two types of domain objects
3. ***User to user relationship*** - 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*** - 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*** - 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 many relationships*** - 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
7. ***Many to many relationships*** - 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

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.

|  |  |  |
| --- | --- | --- |
| seller-buyer.png | student-faculty.png | art-gallery.png |

## Feature Requirements

The following set of features are required. You can provide various features in a single Web page or decide to spread the feature across several pages. You are free to layout the pages and implement navigation any which way you want as long as you implement the features below. Pages must be phone and tablet friendly.

1. ***Home page*** - 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 Web site without login in. Only when accessing private or sensitive information should the application force users to login. Or if the Web application needs to know the identity of the user to execute an action. Allow users to login with credentials from third parties such as Google, Facebook, or Twitter
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 page 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 Page*** - 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. ***Social Networking*** - provide social networking features such as discovering other users, following other users, replying to other's comments, sharing a subset of profile/personal information, inviting other users
7. ***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>
8. ***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

## Styling Requirements

1. Responsive - the Web application must be usable in a desktop, tablet or phone.
   1. Unnecessary horizontal scrollbars must be avoided, unless reasonably justified such as scrolling through tables or large images.
   2. Unnecessary wrapping must be avoided. As the application is resized, elements should not wrap unnecessarily. If elements do not fit on smaller screens, they should collapse into other elements that allow access through other UI means.
   3. Font sizes and UI elements should take full advantage of larger screens, and make best use of smaller screens. For instance, on smaller screens buttons and input fields should be reasonably larger to make it easier to touch on smaller screens
2. Bootstrap - use bootstrap CSS, JavaScript, and HTML templates. Feel free to override the look and feel to achieve a unique style. Alternatively you can use [Materials](https://material.angularjs.org) or [Foundations](http://foundation.zurb.com/)
3. White spacing - use white spacing to properly format space around and between content
   1. Padding - use padding to avoid content being flush with container elements
   2. Margin - use margin to add space between content
   3. Justification - use justification to format left, center, right, or justified content depending on the data type, e.g., numbers should justify right, whereas text should justify left. Use center justification where it makes sense
   4. Wrapping - avoid unnecessary wrapping
4. Look and Feel - the web application should feel professional and have a polished, finished look. You should be proud to showcase your Web application and brag about to a potential employer. Use a color scheme that provides enough contrast between elements to distinguish them, especially foreground and background elements
5. Tables - only use tables for tabular data. Do not use tables for laying out HTML elements. Avoid large tables on small screens, if not, use responsive tables where they can be scrolled right/left within a fixed container DIV

## Best Practice Requirements

Throughout the semester we covered several best practice. Your project should make use of all best practices, of which here are a few

1. Immediately invoked function expressions (IIFE) - use IIFEs to avoid littering the namespace
2. Dynamic content loading - load HTML content dynamically using client side routing, views, and templates to avoid reloading the entire page, e.g., use SPAs
3. Controllers - implement controllers to handle user interaction, provide data to the view. Avoid the use of $scope, use controller instance binding instead, declare event handlers at the top, encapsulate initializer code in an init() function, access data services using Web client services
4. Web service client - centralize Web service and data access in services that can be shared across the application, declare interfaces at the top
5. File structure - organize your files into a consistent file structure as shown in class and assignments
6. Use a consistent naming convention, use camel casing, use indentation to make code readable, choose meaningful variable and function names

## Deliverables

1. Project Description (project1)
   1. Visit [programmable web](https://www.programmableweb.com/) and research an API that might interest you. If you can't find an API of interest, search the web for an API for a domain of interest
   2. In your GitHub wiki, create a web page called ***Project***
   3. Provide a 300 to 500 word description of your project
   4. Include a problem statement
   5. Include a high level description of your proposed solution
   6. Describe at least 3 potential domain objects
   7. Describe at least 3 potential human users for your domain
   8. For each human user, list at least 3 goals the user could accomplish using your Website
   9. For each human user, list at least 2 relations with other users
   10. For each human user, list at least 2 relations with domain objects
   11. For each domain object, list at least 2 relations with other domain objects
2. Project Design (project2)
   1. In your GitHub wiki, create a web page called ***Design***
   2. 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](https://www.visual-paradigm.com/features/uml-and-sysml-tools/)
      2. [Lucid Chart](https://www.lucidchart.com/)
      3. [UMLet](http://www.umlet.com/)
      4. [Violet UML](http://alexdp.free.fr/violetumleditor/page.php)
   3. 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
   4. Embed the UML diagram in your Design wiki page providing a description of your design
3. Project Proof of Concept (project3)
   1. For your project create experiments that test the API of your choice
   2. Start creating the data models and services to interact with your API. Create client (and/or server) side services that can
      1. Query for a list of domain objects based on a search criteria
      2. Query for a particular domain object instance based on some primary key/unique identifier
   3. Start creating client (and/or server) side experiments that exercise your API. Create at least the following two experiments
      1. A search page with at least a single input field and a search button, results are render below as a list of elements, e.g., search a movie database API for a particular movie title, list all movies that match the movie title
      2. A details page linked from the search page. Clicking on a search result navigates to a details page that renders additional details from the API, e.g., on the search page above, click on a particular movie, navigate to a separate details page, retrieve more details for that particular movie, e.g., cast, plot, poster pictures, reviews, etc.,
      3. Any other experiment you think you need to make sure the API meets your stated goals and solves the problem statement
   4. These pages don't need to be pretty, but must demonstrate your understanding of the API of choice
   5. In your GitHub wiki, create a web page called ***POC*** (Proof of Concept)
   6. Provide links to the experiments from the POC wiki page describing how to use the experiments
4. Project Prototype (project4)
   1. In your remote server start creating the profile page, user to user relations and user to domain object relations
   2. Get started with the login, registration and profile page with their related templates, controllers, and services
   3. Get started with the user to user relations, e.g., while logged in as alice, navigate to bob's profile and follow bob, then show in alice's profile that alice is following bob, and in bob's profile show that bob is followed by alice
   4. Get started with the user to domain object relations, e.g., while logged in as alice, search for a movie, navigate to a movie detail page, and book mark the movie, then show in alice's profile that alice bookmarked a movie. Clicking on the movie should navigate to the movie detail page
   5. In your GitHub wiki, create a web page called ***Prototype***. Describe
      1. the user to user relations and how to test them
      2. the user to domain relations and how to test them
5. Project Implementation (project5)
   1. Finish your project by adding security, finishing persistent storage, cleaning up the code, white spacing, wrapping, responsiveness, styling in general
   2. Create the following accounts, as needed, 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
      5. dan/dan - user with role 4
   3. In your GitHub wiki, create a page called ***Testing***
   4. In the Testing page, create detailed steps to test your application. The tests should validate that you have met the functional requirements of your project. 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 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 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
6. (Optional) 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
7. Your project must be accessible from your home page. Host your project on a hosting service of your choice. Submit a link to your home page and your project
8. Upload your project's source code to a source control repository such as Github or Bitbucket. Submit a link to your source code repository and make sure it is accessible to the instructors

## Rubric

This is the rubric used in the previous semester. Use this as a checklist

|  |  |  |  |
| --- | --- | --- | --- |
| Functional | Register | page | allows registering with username/password, e.g., jose/jose |
| Login | page | allows login in with same username/password, e.g., jose/jose |
|  | stays logged in even after page reload (maintains session) |
| Home Page | page | shows a nice/pleasing landing page for the home page. Short description of project. Should know what it's about from home page, or link. can include/merge with search/results page |
| Search | page | allows searching with either single input field or dropdowns. navigates to or shows results page after searching. can be in same page as search results. 3rd party API is mandatory for grads (cs5610) |
| Search Results | page | shows summary results, e.g., thumbnails, titles, links to details page |
| Details | page | shows details from links from search results page. aggregates details data from 3rd party API (grads) plus local database, e.g., comments, ratings, reviews, etc. |
| Profile page |  | shows current values |
|  |  | can update |
| Second User |  | provides different ui |
| Relationships | User/Domain | e.g., review|comment |
|  | User/User | e.g., follows|message|group |
| Back | button | remembers state/results when you press back button |
| Styling | White Space |  | padding, margins, not too much, not too little, no unnecessary |
| Wrapping |  | content should not wrap unintentionally |
| Responsive | desktop | resize window slowly on each page, should still be usable, look acceptable |
|  | phone | each page, should still be usable, look acceptable |
| Pinch/Zoom |  | shouldnt be able to pinch and zoom |
| Scrollbars | No Horizontal | there shouldnt be horizontal scrollbars, except for images, or responsive table |
|  | Not Nested | there shouldnt be nested scrollbars, except maybe responsive tables, images, etc. |
| Alignment |  | elements should line up, not jagged |
| Consistency |  | fonts/colors/layout etc should be consistent |
| Contrast |  | should be able to read text against background |
| Pleasant |  | pages should be generally pleasant |
| All Angular Controllers | event handlers at top |  | all event handlers must be declared at top of controller |
| vm, !$scope |  | controllers should declare local variable to which all data and event handlers are bound to |
| model |  | declare model in config |
|  |  | use model in templates |
| init() |  | all initialization should happen in an init function |
| !$http |  | controllers should not access server data directly, should use services instead |
| promises |  | service functions should be asynchronous and handled with promises |
| All Angular Services | declare API at top |  | apis should be declared at the top |
| uses promises |  | $http calls should be returned immediatedly |
| POST |  | to create |
| GET |  | to get all |
| GET |  | to get one |
| PUT |  | to update |
| DELETE |  | to remove |
| REST |  | all URLs should use REST syntax |
| All Web Services | declare API at top |  | apis should be declared at the top |
| uses promises |  | $http calls should be returned immediatedly |
| POST |  | to create |
| GET |  | to get all |
| GET |  | to get one |
| PUT |  | to update |
| DELETE |  | to remove |
| REST |  | all URLs should use REST syntax |
| All Schemas | User to User |  |  |
| User to Domain |  |  |
| Domain to Domain |  |  |
| All Model | API at top |  |  |
| Create |  |  |
| Find One |  |  |
| Find All |  |  |
| Update |  |  |
| Delete |  |  |