A Maksimirska 58a, HR-10000 Zagreb
T (01) 2332 861
E info@algebra.hr
www.algebra.hr

ALGEBRA

Development of Web Applications

Project specification

1. General Information

- The defense of the project task solution takes place during the exam periods.
- Student applies for the defense the same as for other exams.
- Correctly prepared project task solution is worth at least minimal amount of points per learning objective, or 10 points for each learning objective.
- 5 days before the defense is considered a deadline for submitting project task solution. The project task solution must be submitted no later than deadline.
- Submit the project task in form of the zip archive to the email address of your professor/assistant.
- Students who submit their project task solution after the deadline will not have their work accepted and will not be able to attend the defense.
- Students who did not correctly prepare their project task solution before the deadline (minimal or greater amount of points per LO) will not be able to attend the defense.
- After the deadline and before the day of the defense, students are allowed to improve their project task and submit it again.

Here follows the example of the explained schedule.
In the example, defense date is set to be 9.2.

…	3.2.	4.2.	5.2.	6.2.	7.2.	8.2.	9.2.
		Deadline 23:59					**Defense**
Solving project task to have at least minimum achieved per LO			*Improving solution to achieve more than minimum*				▨

2. Project Task – General Information

Create a single ASP.NET Core web solution which consists of two modules.

When creating a solution follow the **topic** which you have chosen and which was approved by the professor.

RESTful Service Module (Web API)

It is used for both automation (for example showing or updating the data of video content entry via API) and to retrieve data by JavaScript in e.g. static HTML page.

LO1 minimum (10 points): Create RESTful endpoint (CRUD) for your primary entity, with endpoints for searching and paging. Write logs while performing these operations, and make these logs available via additional endpoint.

LO1 desired (10 points): Secure your endpoints using JWT token authentication.

LO2 minimum (10 points): Implement the database access for your endpoints.

LO2 desired (10 points): Implement the static HTML pages that use JWT authentication, localStorage and existing endpoints to securely display logs.

MVC Module (Web Application)

It is accessed by a user via web browser (for example user that accesses or updates the list of video content).

LO3 minimum (10 points): Create a visually appealing website with authentication for administrator. Implement landing page. For each entity implement CRUD pages. Implement navigation. Implement profile page that uses AJAX technique.

LO3 desired (10 points): Create a visually appealing website with authentication and authorization using two roles: administrator and user. It must include LO3 minimum functionalities. Common user must be able to register. Common user role needs items page and details page where it can perform a desired action like reserving ticket, applying for a contest, adding product to a shopping basket etc. For administrator role, also implement viewing list of users with their desired actions.

LO4 minimum (10 points): Perform model validation and labeling using annotations on the model.

LO4 desired (10 points): Implement multi-tier solution and use Automapper to map models in that solution.

LO5 minimum (10 points): Create a profile page that enables user to update its personal data using Ajax request.

LO5 desired (10 points): Enables user to perform complex paging navigation in the list page using Ajax request.

3. RESTful Service Module – LO1

In the solution, create a single ASP.NET Core Web API project.

LO1: Minimum outcome (10 points)

Create RESTful endpoint (CRUD) for your **primary entity**.

1. For the route, use name of your primary entity (e.g. *api/video*).
2. Use the appropriate JSON payload in request body where needed.
3. Handle errors and in HTTP responses return error codes 400, 404 and 500 where appropriate, with error descriptions.
4. Support additional endpoint for searching by *Name*. This endpoint must also support paging functionality using *Page* and *Count* properties
   Use route path *search* (e.g. *api/video/search*).

Create read-only endpoints for retrieving **logs**.

1. Implement endpoint *api/logs/get/N* - returns last N logs, where N is passed by routing parameter (default is N=10)
2. Implement endpoint *api/logs/count* – returns total number of recorded logs
3. For the log, use *Id*, *Timestamp*, *Level* and *Message* attributes.
4. Log each CRUD action for your primary entity with meaningful message that identifies the entity instance.
5. Examples:
6. *"Video content with id=7 has been created."*
7. *"There was a problem while updating video content with id=7."*
8. *"Cannot find Genre id=11."*

LO1: Desired outcome (10 points)

Implement the JWT token authentication for **logs** endpoints that you have implemented for LO1 minimum.

1. Use following endpoints for implementation:
   - Endpoint *api/auth/register* adds a new user with password
   - Endpoint *api/auth/login* retrieves the JWT token
   - Endpoint *api/auth/changepassword* changes users' password

ALGEBRA

4. RESTful Service Module – LO2

LO2: Minimum outcome (10 points)

Implement the database access:

1. Use database for RESTful endpoint (CRUD) for your primary entity that you have implemented for LO1 minimum.
2. Support *1-to-N* and *M-to-N* entities.

LO2: Desired outcome (10 points)

Implement the static HTML pages that are able to log in and display all the logs.

See *Addendum* section for wireframes for the HTML pages:

- Image 1 – Static login page
- Image 2 – Static log list page

Implement the following in the static HTML pages:

1. **Login page**: Username and password must be provided to enter the log display.
2. **Log list page**: Logs are displayed in the central part when user clicks on "Show Logs" button.
3. **Log list page**: Log should display last 10, 25 or 50 logs, depending on what is selected by appropriate dropdown shown on the image. By default, 10 is selected.
4. **Log list page**: Support logout by clicking "Logout" button.

5. MVC Module – LO3

In the solution, create a single ASP.NET Core MVC project. Create website for viewing and editing primary entities. Distribute the application interface elements meaningfully and visually polish as good as you can.

LO3: Minimum outcome (10 points)

Implement the following for administrator:

1. **Landing page**: Username and password don't need to be provided to access the Landing page. Landing page visually represents your topic. CTA leads to Login page.
2. **Login page**: Administrator provides username and password to enter the secured content. Successful login leads to the List page for primary entity.
3. **Primary entity CRUD:** List, Add, Edit and Delete pages for the primary entity. For **List** page there should be a search textbox and dropdown of 1-to-N items for filtering primary list. Filtering is done when you click the Search button. Clicking on Previous or Next button navigates to previous 10 items or next 10 items.
4. **1-to-n entity CRUD:** List, Add, Edit and Delete pages for the 1-to-n entity. No filtering or paging is required.
5. **m-to-n entity CRUD:** List, Add, Edit and Delete pages for the m-to-n entity. No filtering or paging is required.
6. **All the mentioned pages** except Login page must include navigation that leads to list pages (primary, 1-to-n, m-to-n) and a Logout button that supports logout action.
7. Display pages in a visually appealing manner.

LO3: Desired outcome (10 points)

Implement the following for user:

1. **Register page:** User can enter registration data (username, e-mail, password, repeat password…) and register. Secret data (password) is hashed. Succesful registration leads to Register confirmation page.
2. **Register confirmation page:** This page is displayed with confirmation (Ok) button that leads back to Login page.
3. **Login page:** Depending on the role, login leads to either administrator's primary entity List page or users Items page.
4. **Items page**: There should be a search textbox and 1-to-N dropdown for filtering items. Filtering is done when you click the Search button. Clicking on each item in the items page leads to details page. Clicking on Previous or Next button navigates to previous 10 items or to next 10 items.
5. **Details page**: Display all the primary entity attributes in a visually appealing manner. Allow user to click on the desired action related to the primary entity. Allow user to go back to items page.
6. **Confirm desired action page**: Allow user to either confirm desired action or cancel and go back to items page.
7. **Desired action confirmed page**: Display confirmation of the desired action. Page is displayed with confirmation (Ok) button that leads back to Items page.
8. Support image upload for primary entity
9. ***For administrator:*** Support showing list of users with desired actions (support in navigation).
10. Display pages in a visually appealing manner.

See *Addendum* section for schemas and wireframes for the HTML page layouts:

- Image 3 - Schema for CRUD web pages
- Image 4 - Navigation schema of web application
- Image 5 - Login page
- Image 6 - Landing page
- Image 7 - Primary entity list page (variant 1)
- Image 8 - Primary entity list page (variant 2)
- Image 9 - Primary entity details page
- Image 10 - Primary entity add/edit page

Note there are two variants of the (primary entity) list page that you can use. One could be more appropriate for minimum, and another for desired outcome. Use them as you find appropriate.

6. MVC Module – LO4

LO4: Minimum outcome (10 points)

Implement the following:

1. Models should be validated (required fields, correct URLs and correct e-mail addresses…)
    - For example, validate required fields, correct URLs and correct e-mail addresses
2. No entity should be allowed an empty input (e.g. Name, Description…)
3. No primary entity should be allowed to accept existing Name.
4. Every primary entity must be in relation to one 1-to-N entity.
5. Every primary entity must be in relation to one or more M-to-N entities.
6. Labels must use annotations on the model.

LO4: Desired outcome (10 points)

Implement the following:

1. Implement multi-tier application.
2. Having a single model for all the tiers is not allowed.
3. Having navigation properties in viewmodels is not allowed.
4. Use Automapper to map models between tiers.

7. MVC Module – LO5

See *Addendum* section for wireframes for the HTML page layouts:

1. Image 11 - Profile page

LO5: Minimum outcome (10 points)

Implement the following:

1. Profile page: user should be able to change username, email, first and last name and phone using Ajax request.

LO5: Desired outcome (10 points)

Implement the following:

1. Primary list page: implement paging on primary list page using Ajax requests.

8. Project Task Structure

For your project to be accepted, you have to follow the **structure** described here. The structure consists of project task solution archive structure, entity structure and database creating SQL script structure.

8.1. Project task solution archive structure

Project task solution archive must be ZIP archive with the following folder/file structure:

- ProjectTask
    - Database
        - *Database.sql* (<u>single</u> SQL file in the folder)
    - *SolutionName* (folder named according to the topic)
        - *SolutionName.sln* (<u>single</u> solution file in the folder, name it according to the topic, e.g. *VideoContentManager.sln*)
        - WebAPI (your folder with a single Web API project *WebAPI.csproj* and its files)
        - WebApplication (your folder with a single Web Application project *WebApplication.csproj* and its files)

You can find the example of the archive structure on Infoeduka.

8.2. Entity structure

The solution of the topic that you and professor agreed upon needs to have some entities defined. These entities are as follows:

- **Primary entity** (for example, for the topic Video content management system, the main entity would be **Video**)
- **Additional entities**
    - **1-to-N entity**: for example, **Genre**
    - **M-to-N entity**: for example, **Tag**; M-to-N relation implies having both entity and bridge tables in the database, like **Tag** and **VideoTag**
    - **Application user entity**: for example, **User**
    - **Image entity** (desired): for example, **Image**; that entity is used to represent the image of the primary entity
    - **User M-to-N entity** (desired): bridge table that records users desired action, like reserving ticket, applying for a contest, adding product to a shopping basket etc.

All the entities must have *Name* as an attribute. Primary entity must have additionally at least 3 other attributes except keys and *Name* (e.g. *Duration*, *Description*, *VideoUrl*).
All the tables must be named same as their entities. For example, you are not allowed to name the table *Video* and use it as *Product* entity (e.g. in a *Web shop* project).
All the table names must be singular.

8.3. Database creating SQL script structure

1. Database creation script file is a mandatory requirement. That is why the database part of the solution must follow database-first principle.
2. All the database table creation code must be in *Database.sql* file in the archive.
3. Do not use database modifying (ALTER DATABASE), creating (CREATE DATABASE) or switching (USE) statements in *Database.sql* file.
4. Use table modifying (ALTER TABLE) and creating (CREATE TABLE) statements in *Database.sql* file, where you need them.
5. Use table data inserting, modifying, deleting and retrieving statements in *Database.sql* file, where you need them (e.g. to have values for 1-to-N and M-to-N entities).
6. Before each CREATE TABLE statement you need to have a single line comment with token that describes the entity-table mapping for your created table.
   Tokens are *Primary*, *1-to-N*, *M-to-N*, *M-to-N-bridge*, *User-M-to-N-bridge*, *Image* and *User*.

   Example code:

   ```
   -- 1-to-N
   CREATE TABLE [dbo].[Genre]( ... )
   GO
   ```

You can find the example of the database script in the archive structure on Infoeduka. Note that database script contains example table names, not the real names that you need in your project. Also, it does not contain creation of all the columns that you require in your project.

9. Addendum: Schemas and wireframes

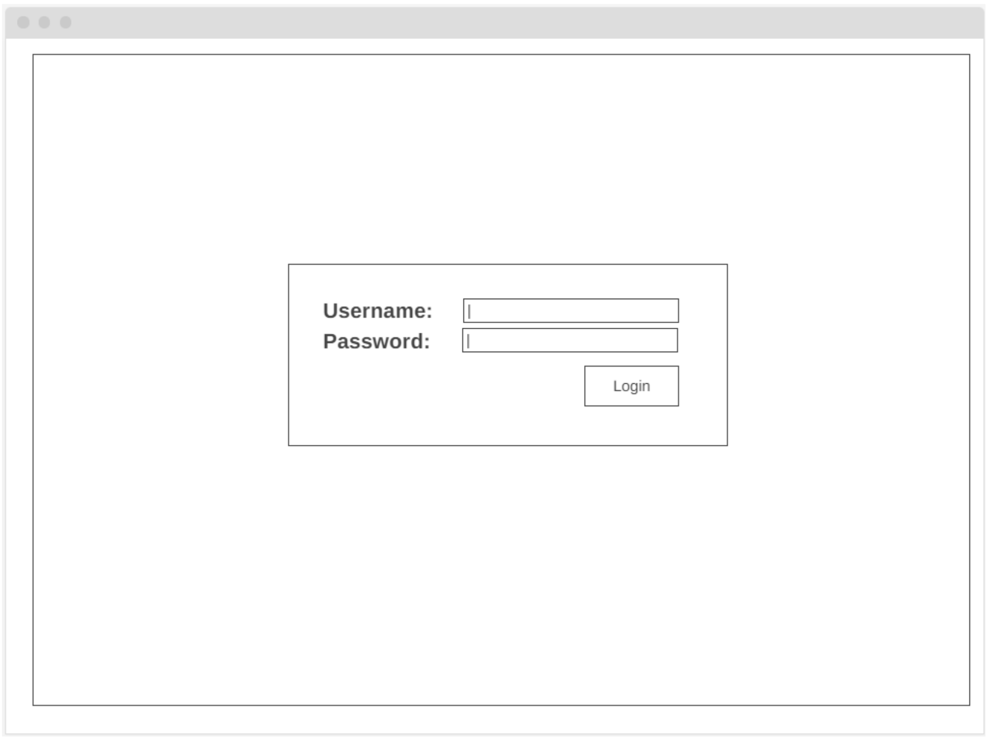


Image 1 – Static login page

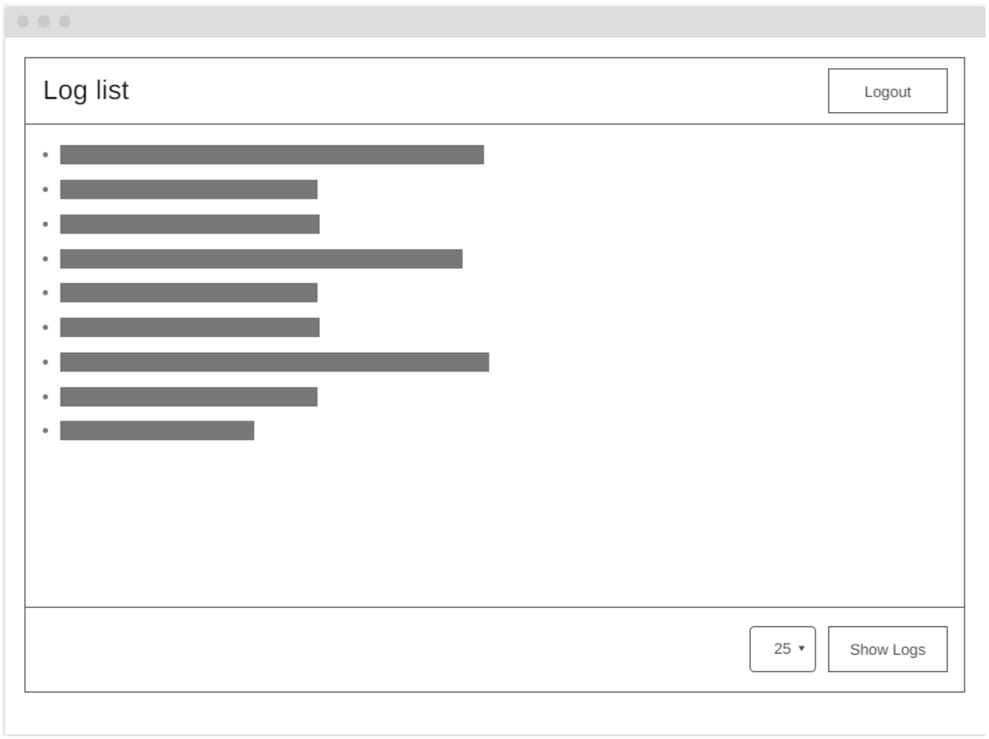


Image 2 – Static log list page

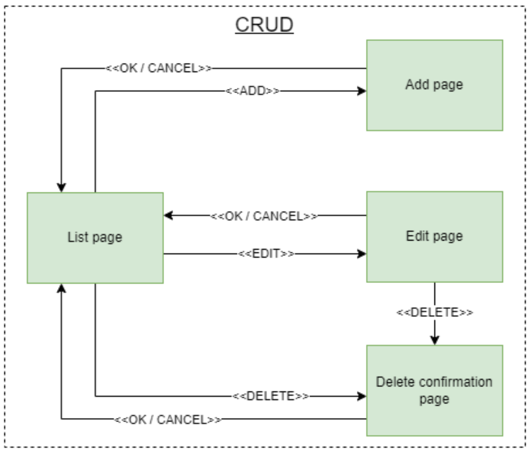


Image 3 - Schema for CRUD web pages

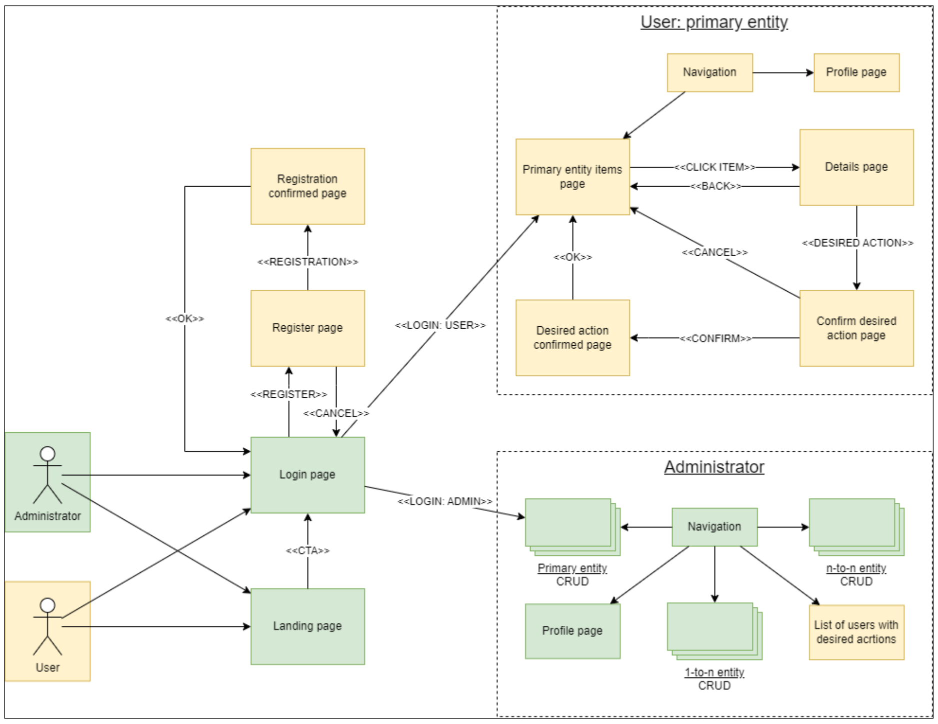


Image 4 - Navigation schema of web application

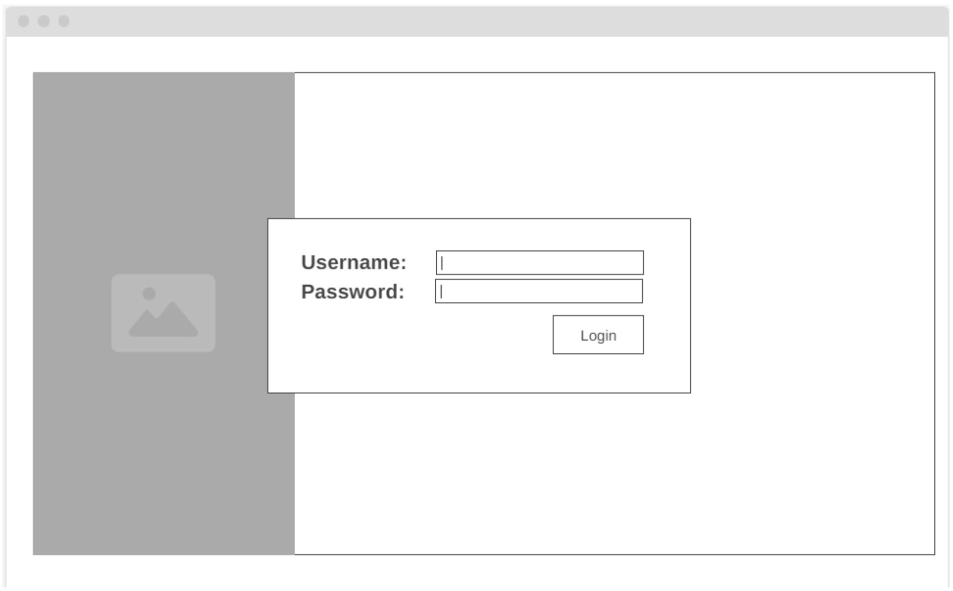


Image 5 - Login page

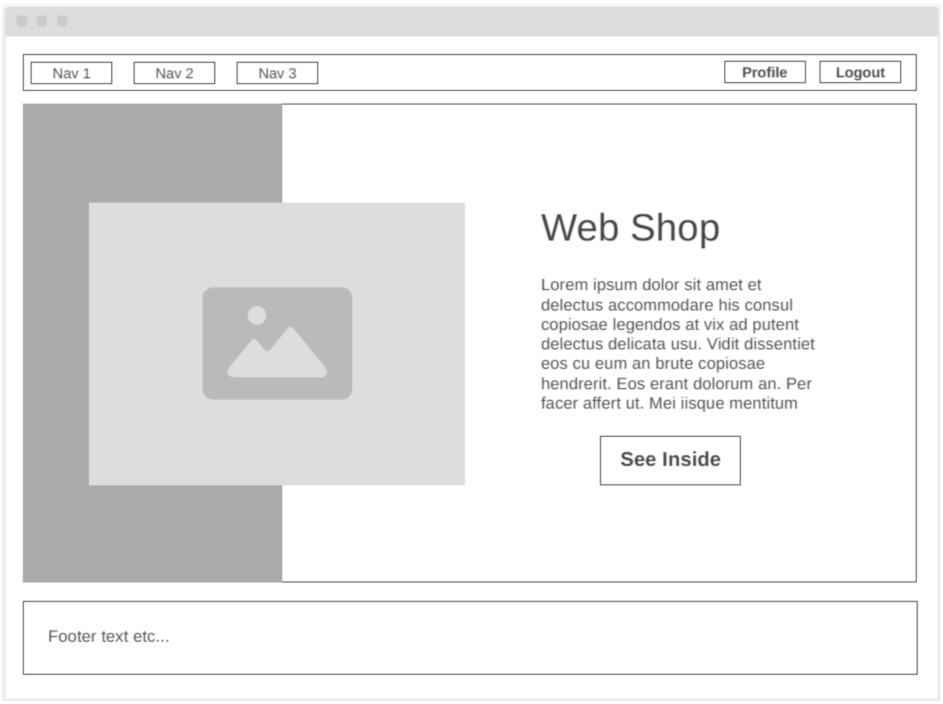

Nav 1
Nav 2
Nav 3
Profile
Logout
Web Shop
Lorem ipsum dolor sit amet et delectus accommodare his consul copiosae legendos at vix ad putent delectus delicata usu. Vidit dissentiet eos cu eum an brute copiosae hendrerit. Eos erant dolorum an. Per facer affert ut. Mei iisque mentitum
See Inside
Footer text etc...

Image 6 - Landing page

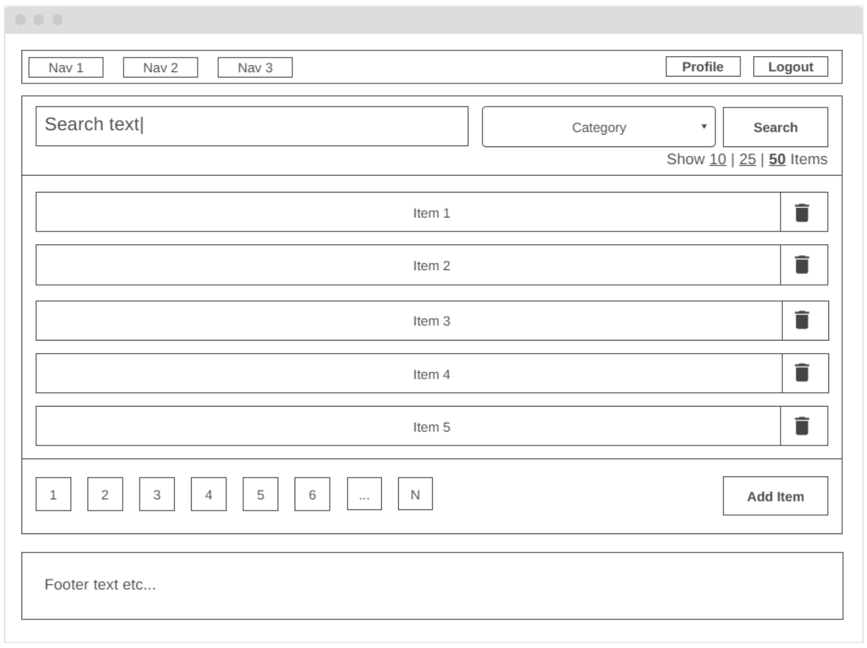


Image 7 - Primary entity list page (variant 1)

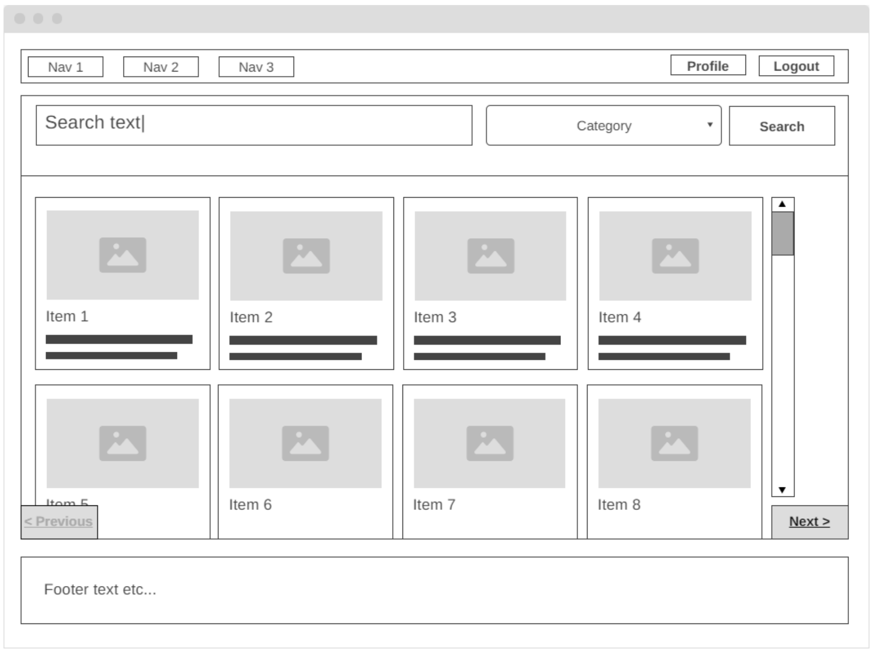


Image 8 - Primary entity list page (variant 2)

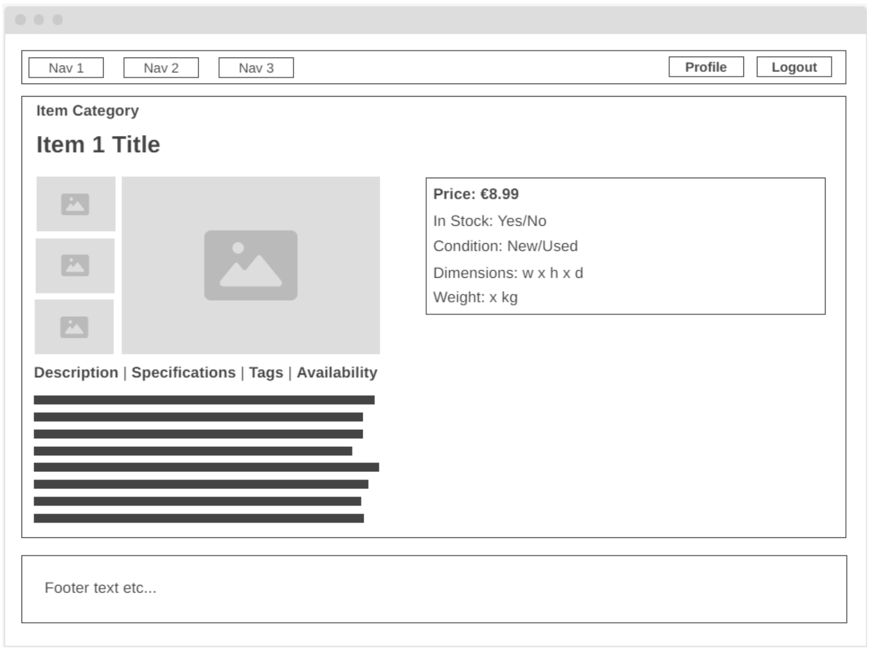


Image 9 - Primary entity details page

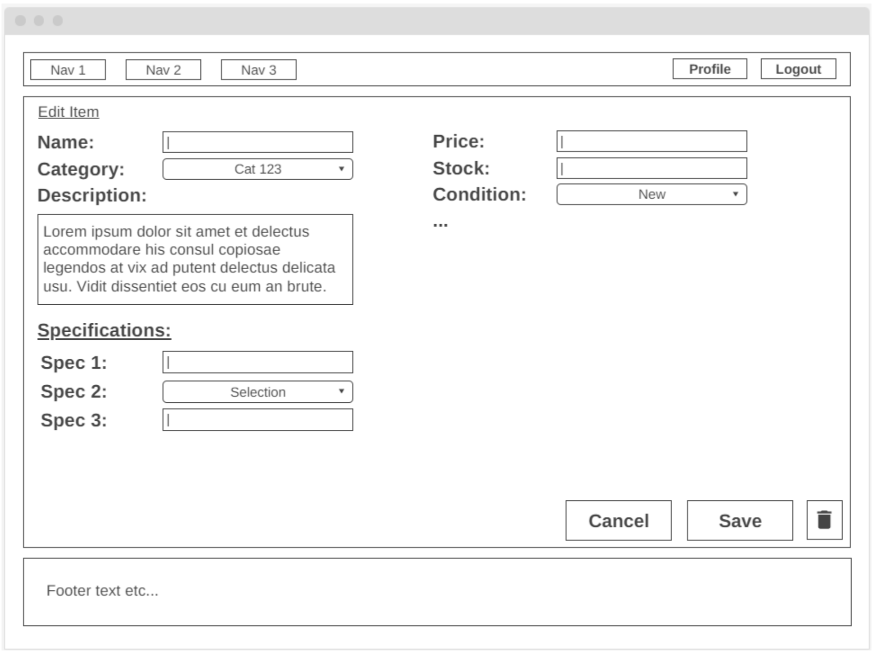
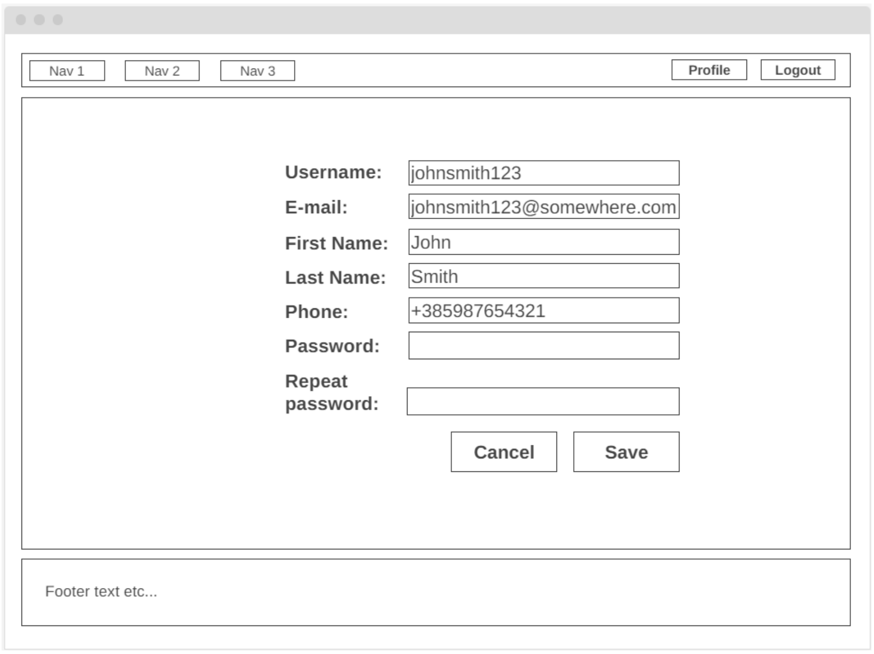


Image 10 - Primary entity add/edit page



Username:	johnsmith123
E-mail:	johnsmith123@somewhere.com
First Name:	John
Last Name:	Smith
Phone:	+385987654321
Password:	
Repeat password:	

Footer text etc...

Image 11 - Profile page