

Development of Web Applications

Project specification

1. General Information

- The defense of the project task solution takes place during the exam periods.
- Students apply for the defense in the same way as for other exams.
- Important rules for successful submission of the project task solution:
 - Submit the project task on time
 - The submitted zip archive must follow the specified file naming and structure
 - In your solution, fulfill at least minimum points per learning objective
 - Your solution must follow the topic approved by the professor

Submitting the task on time

- 5 days before the defense** is considered a deadline for submitting the project task solution. The project task solution must be submitted no later than the deadline. Here follows the example of the explained schedule. In the example, the 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
<i>Solving project task to have at least minimum achieved per LO</i>			<i>Improving solution to achieve more than minimum</i>				

- 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.

Submitted zip archive file naming and structure

- Submit the project task in the form of a **zip** archive to the email address of your professor/assistant. RAR, 7z and other archive formats will not be accepted.
- Archive must be named in form of **{surname}-{name}-{your project name}.zip**
Example: Smith-John-MyProjectTask.zip
- Archive must follow the structure explained in **section 3**. Incorrect archive structure will not be accepted. **Please read section 3!**

Minimum points per learning objective

- Correctly prepared project task solution is worth at least a minimal amount of points per learning objective, or 10 points for each learning objective.
- Pay attention to fulfilling at least the minimum part for each LO, otherwise you will be automatically disqualified on defense.

Improving your work and feedback

- After the deadline and before the day of the defense, students are allowed to improve their project task and submit it again.
- Students can expect feedback on their work submitted before the deadline. In some cases, students can get feedback after the deadline and before the defense but it's not guaranteed.

2. Project Task Specification

Create a single ASP.NET Core web solution that consists of two modules (projects).

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

There are two modules you need to implement:

- RESTful Service Module (Web API)
 - Covers LO1 and LO2
 - Used to retrieve data by JavaScript in e.g. static HTML page. Also meant to be used for automation (for example showing or updating the data of video content entry via API).
- MVC Module (Web Application)
 - Covers LO3, LO4 and LO5
 - Accessed by a user via web browser

2.1. Learning Objective 1 (RESTful Service Module, Web API)

Minimum (10 points): Create a 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 an additional endpoint. **Desired (10 points):** Secure your endpoints using JWT token authentication and implement common authentication functionalities.

General guides, minimum (10 points)

For your **primary entity** CRUD endpoints, use the name of your primary entity (e.g. *api/video*). Use the JSON payload in the request body where appropriate. It is important to handle errors and in HTTP responses return error codes 400, 404 and 500 where needed. Support additional endpoint for search functionality, e.g. by *Name* or *Description* or both. This endpoint must also support paging functionality using e.g. *Page* and *Count* parameters. For search functionality use an appropriate endpoint name (e.g. *api/video/search*).

For your **logs** implement an appropriate endpoint (e.g. *api/logs/get/N* - returns last N logs, where N is passed by routing parameter). Implement endpoint *api/logs/count* which returns a total number of recorded logs. For the single log, use e.g. *Id*, *Timestamp*, *Level* and *Message* attributes. Log each CRUD action for your primary entity record a meaningful message that identifies what happened to the entity instance. Examples: "Video content with id=7 has been created.", "There was a problem while updating video content with id=7." or "Cannot find Genre id=11."

Make sure you include Swagger or a similar interface to be able to easily demonstrate how your API works during the defense.

General guides, desired (10 points):

Implement the JWT token authentication for **logs** endpoints that you have implemented for LO1 minimum. Implement adding a new user (e.g. endpoint *api/auth/register*), retrieving the JWT token for the user (e.g. endpoint *api/auth/login*) and changing users' password (e.g. endpoint *api/auth/changepassword*)

Make sure your Swagger interface supports authentication to be able to easily demonstrate how your authentication works.

You can also add token authentication to other endpoints if you wish to.

2.2. Learning Objective 2 (RESTful Service Module, Web API)

Minimum (10 points): Implement the database access for your endpoints. **Desired (10 points):** Implement the static HTML pages that use JWT authentication, localStorage, and existing implemented endpoints to securely display the desired number of log records.

General guides, minimum (10 points)

Use the database for saving state using RESTful endpoint (CRUD) for your primary entity that you have implemented for LO1 minimum. Additionally, implement CRUD endpoints for 1-to-N and M-to-N entities, also supported by the database.

Pay attention when deleting related entities, and handle eventual errors gracefully.

General guides, desired (10 points)

Implement the static HTML pages that use JWT authentication and existing endpoints to securely display logs. Namely, these pages should be the login page and log list page. On the log list page, the user should be able to change the displayed number of logs, e.g. last 10, 25 or 50 logs, depending on what is selected by an appropriate dropdown. Use localStorage to store the authentication token. Support logout by clicking the "Logout" button.

For details on HTML pages, you can see wireframes:

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

2.3. Learning Objective 3 (MVC Module, Web Application)

Minimum (10 points): For the administrator, create a secure website that implements CRUD functionality for each of the entities. Implement meaningful and consistent navigation. **Desired (10 points):** For the user, create a visually appealing website with a landing page and a focus on the primary entity. Implement meaningful and consistent navigation. The user must be able to self-register and log in. The user needs a page where she/he can see the list of desired items, a way to open the desired item, and perform a desired action like watching a movie, reserving a ticket, applying for a contest, adding a product to a shopping basket, etc. For the administrator role, also implement viewing a list of users with their desired actions, like watched movies, reserved tickets, contest applications, or shopping basket content of a user.

General guides, minimum (10 points)

Implement the following for the administrator:

1. **Login page:** Successful login leads to the List page for the primary entity.
2. **Primary entity CRUD pages:** List, Add, Edit and Delete pages for the primary entity.

On the **List** page, there should be a search textbox and dropdown of 1-to-N items for filtering the list. Filtering is done when you click the Search button.

Clicking on the Previous or Next button navigates to the previous 10 items or the next 10 items.

3. **CRUD pages for other entities:** List, Add, Edit and Delete pages for both the 1-to-n entity and the m-to-n entity.

Pages must contain meaningful and consistent navigation. To be precise: all the mentioned pages excluding the Login page must contain navigation that leads to list pages for primary, 1-to-n, and m-to-n entities; also, each page should have a Logout button.

Display pages in a visually appealing manner.

General guides, desired (10 points)

Implement the following for the user:

1. **Landing page:** Credentials don't need to be provided to access the landing page. The landing page visually represents your topic. CTA leads to Login page.
2. **Self-register page:** The user can enter registration data (username, e-mail, password, repeat password...) and self-register.
3. **Login page:** Depending on the role, a successful login action leads to either e.g. the administrator's primary entity List page or e.g. the user's Items page.
4. **Items page:** Displays a list of primary entities to the user. There should be a search textbox (for example, the name of the movie) and a 1-to-N dropdown for filtering (for example, movie genre). Filtering is done when you click the Search button. Clicking on each item on the items page leads to the details page (for example, movie information). Clicking on the Previous or Next button navigates to the previous 10 items or to the next 10 items.
5. **Details page:** Display the primary entity attributes. Allow the user to go back to the items page.
6. **Desired action:** The user must be able to perform the desired action on that particular item's details page (watching a movie, reserving a ticket, applying for a contest, adding a product to a shopping basket, etc.)

Implement the following for the administrator:

7. Support showing a list of users and their desired actions, like movies that the user has been watching, reserved tickets, contest applications, or hers/his shopping basket content.

Display pages in a visually appealing manner.

Support image upload for the primary entity if required by the topic.

See *Addendum* section for suggested schemas and wireframes for the HTML page layouts. The schemas and wireframes are just the examples.

- 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

2.4. Learning Objective 4 (MVC Module, Web Application)

Minimum (10 points): Perform model validation and labeling using annotations on the model. **Desired (10 points):** Implement a meaningful multi-tier solution and use AutoMapper to simplify mapping models in that solution.

General guides, minimum (10 points)

Models must be validated: required fields, correct URLs, correct e-mail addresses, etc. Implement validation that prevents empty input (e.g. Name, Description...). Duplicate entity instance names are not allowed (e.g. two Genres named "Thriller" or two movies named "Die Hard"). Visible labels must be implemented using model annotations. Entity instance identifiers (or id's) must not be visible anywhere on the UI.

General guides, desired (10 points)

Make use of the multi-tier concept to simplify the structure of the solution. The idea is in the end to have Web API and MVC tiers (projects) that depend on the same common business tier (project) and the same database tier. *Note that it means that you will end up having more than two projects in your solution.*

Models: you should have a different set of models for each of the tiers (projects). For example, a database model must not be used in the view. You should not have navigation properties in viewmodels. Use AutoMapper to map models between tiers.

2.5. Learning Objective 5 (MVC Module, Web Application)

Minimum (10 points): For the administrator, implement a profile page to update hers/his personal data using AJAX requests. **Desired (10 points):** For users, implement a profile page to update their personal data using AJAX requests. Enable user to perform complex paging navigation in the primary entity list page using AJAX request.

General guides, minimum (10 points)

For the administrator, implement the **Profile page**. Administrator should be able to change email, first name, last name, phone number and other personal data. You must use AJAX requests for the implementation.

General guides, desired (10 points)

For the user, implement the **Profile page**. Users should be able to change email address, first name, last name, phone number and other personal data. You must use AJAX requests for the implementation.

Implement AJAX paging in **list page for primary entity**. The best result would be to display several pages before and after the current page (numbers like **5, 6, 7, 8**) and the **Previous** and **Next** buttons.

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

1. Image 11 - Profile page

3. Project Task Structure

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

3.1. Project task solution archive structure

The project task solution archive must be a ZIP archive with the following folder/file structure:

- ProjectTask
 - Database
 - *Database.sql* (single SQL file in the folder)
 - *SolutionName* (folder named according to the topic)
 - *SolutionName.sln* (single 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)
 - WebApp (your folder with a single Web Application project *WebApp.csproj* and its files)

You can find an example of the archive structure on Infoeduka (*ProjectTask-example.zip*).

3.2. Entity structure

The solution of the topic that you and the 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, if required by topic): 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 desired action of adding a movie, reserving a ticket, applying for a contest, adding a product to a shopping basket, etc.

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

3.3. 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 the database-first principle. That is in contrast to the code-first principle, which includes migrations and which must not be used.
 2. All the database table creation code must be in the *Database.sql* file in the archive.
 3. **Do not use database modifying (ALTER DATABASE), creating (CREATE DATABASE) or switching between databases (USE) statements in the Database.sql file.**
 4. Use table modifying (ALTER TABLE) and table creating (CREATE TABLE) statements in the *Database.sql* file, where you need them.
 5. Use statements for table record inserting, modifying, deleting and retrieving in *Database.sql* file, where you need them.
- Note that you will need to show a working example of your application during the defense. This means you will have to have values for 1-to-N and M-to-N entities in the application that has been installed on a PC in a classroom where the defense is taking place.

You can find the example of the database script in the archive structure on Infoeduka. Note that the database script contains example table and column names, not the real names that you need in your project.

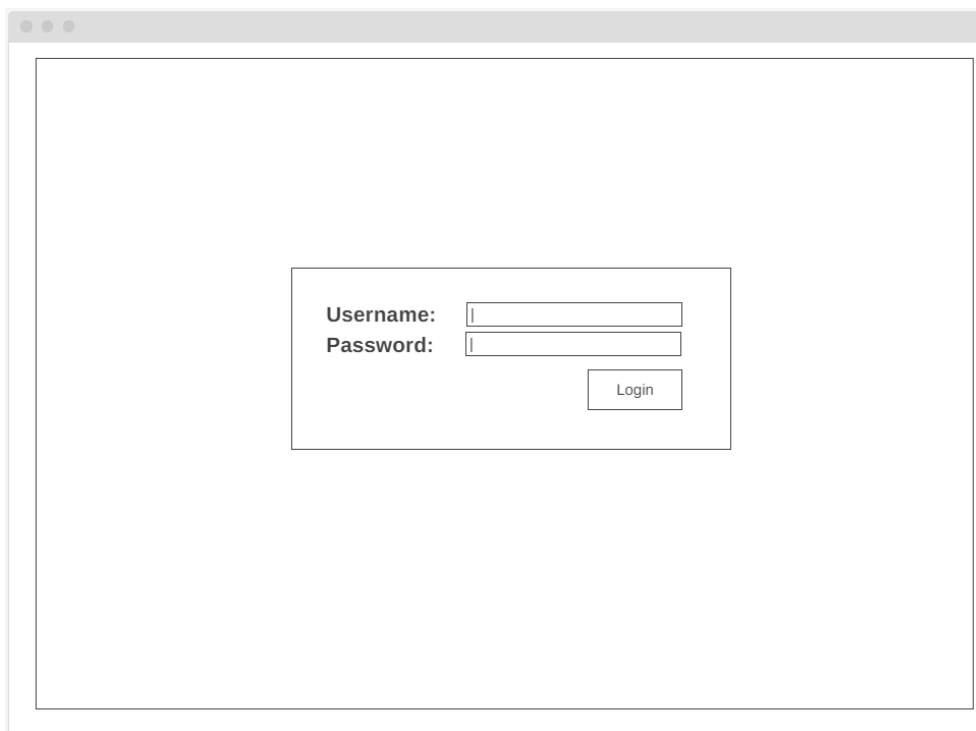
3.4. Important notes

1. As an archive format, use ZIP. RAR, 7z and the other archive formats files are not supported and will not be accepted.
2. The archive must not contain **bin** and **obj** folders due to the limitations of antivirus email filters. The same goes for other folders that contain the artifacts (for example **packages**). Make sure to delete them before you create the ZIP archive. Note that you have to close Visual Studio to avoid automatic re-creating of these folders.

If your ZIP archive is too large (> 10MB or such) or you receive an error when submitting the archive through e-mail, you should check if your archive contains bin, obj, packages or other such folders with artifacts.

3. **Hardcoding of a connection string is forbidden.** The connection string must be loaded from the configuration (appsettings.json). Note that simply adding a connection string to the appsettings.json file doesn't solve the issue, you need to also reference the configuration from the code.
4. Use the **specified .NET version** for development. It should be the same version you have been using in your workshops and tasks. If you use another .NET version, your application cannot be installed and verified with an automatic check and you cannot receive feedback before the defense.

4. Addendum: Schemas and wireframes



A wireframe of a static login page. It features a central box containing a 'Username:' label with an adjacent input field, a 'Password:' label with an adjacent input field, and a 'Login' button positioned below the password field.

Image 1 – Static login page



A wireframe of a static log list page. The page has a header area with a 'Log list' title and a 'Logout' button. Below the header is a list of ten items, each represented by a bullet point and a horizontal bar of varying length. At the bottom right, there is a pagination control showing '25' with a dropdown arrow, and a 'Show Logs' button.

Image 2 – Static log list page

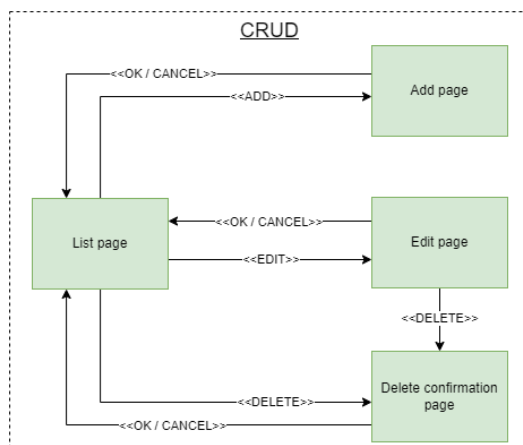


Image 3 - Schema for CRUD web pages

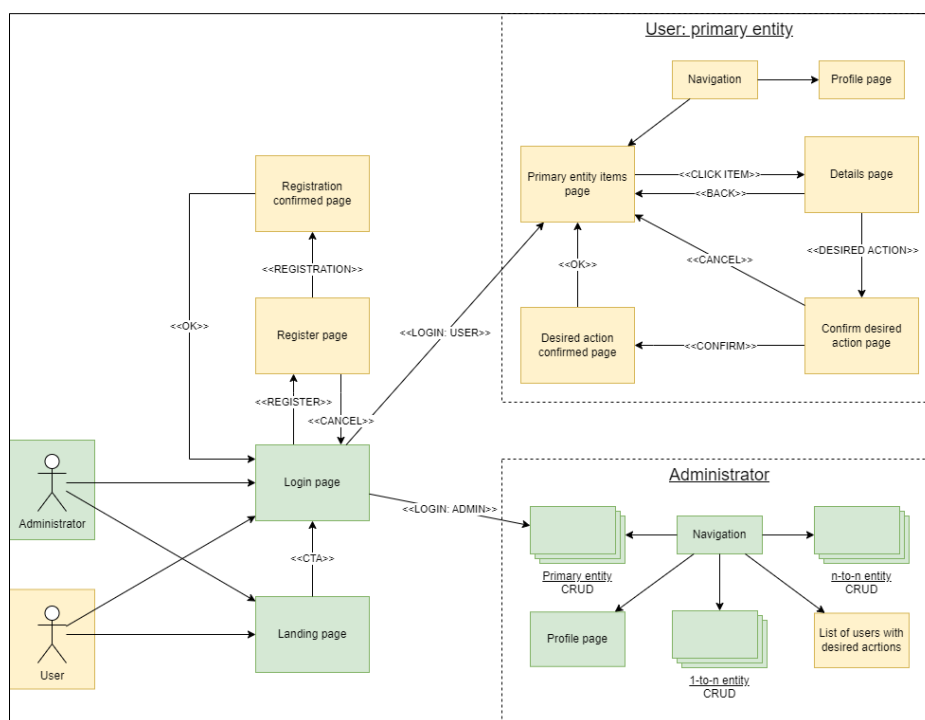


Image 4 - Navigation schema of web application

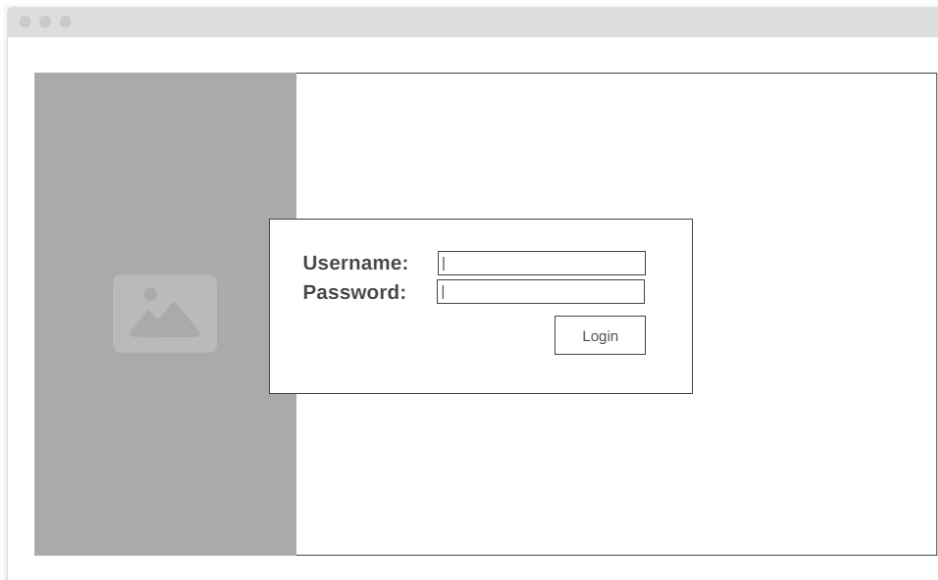
A mockup of a login page within a browser window. On the left is a dark grey sidebar with a white image placeholder. The main content area is white and contains a login form. The form has two input fields: 'Username:' and 'Password:'. Below the password field is a 'Login' button.

Image 5 - Login page

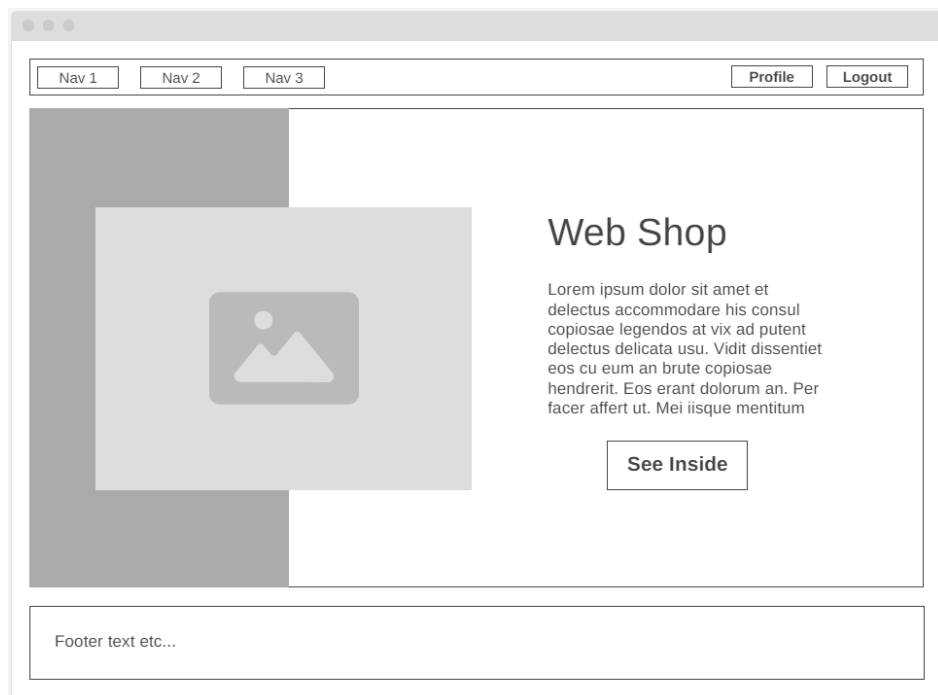
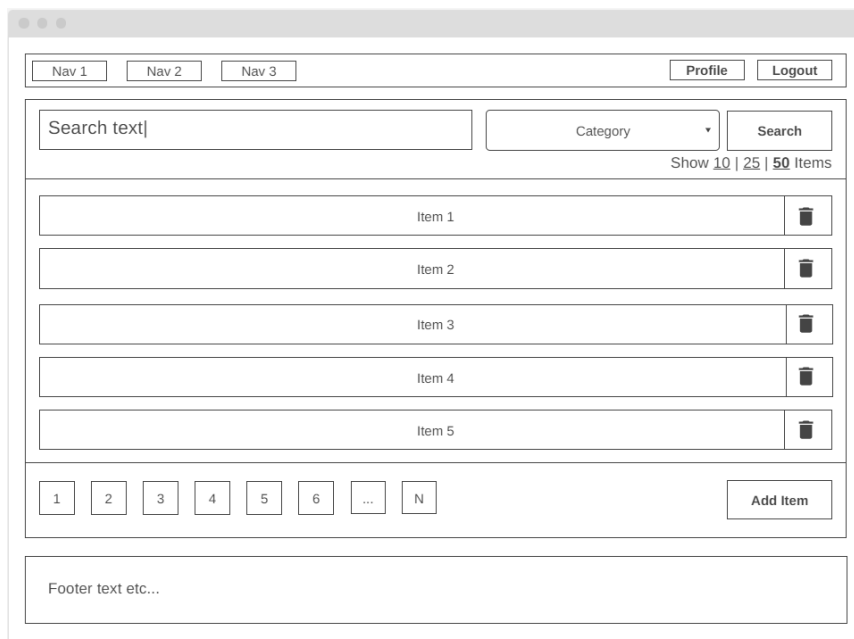
A mockup of a landing page within a browser window. The page has a dark grey sidebar on the left with a white image placeholder. The main content area is white. At the top, there is a navigation bar with three buttons: 'Nav 1', 'Nav 2', and 'Nav 3'. On the right side of the navigation bar are 'Profile' and 'Logout' buttons. The main content area features a large image placeholder on the left and a 'Web Shop' section on the right. The 'Web Shop' section includes a paragraph of Lorem Ipsum text and a 'See Inside' button. At the bottom of the page is a footer area with the text 'Footer text etc...'.

Image 6 - Landing page



Nav 1 Nav 2 Nav 3 Profile Logout

Search text| Category Search

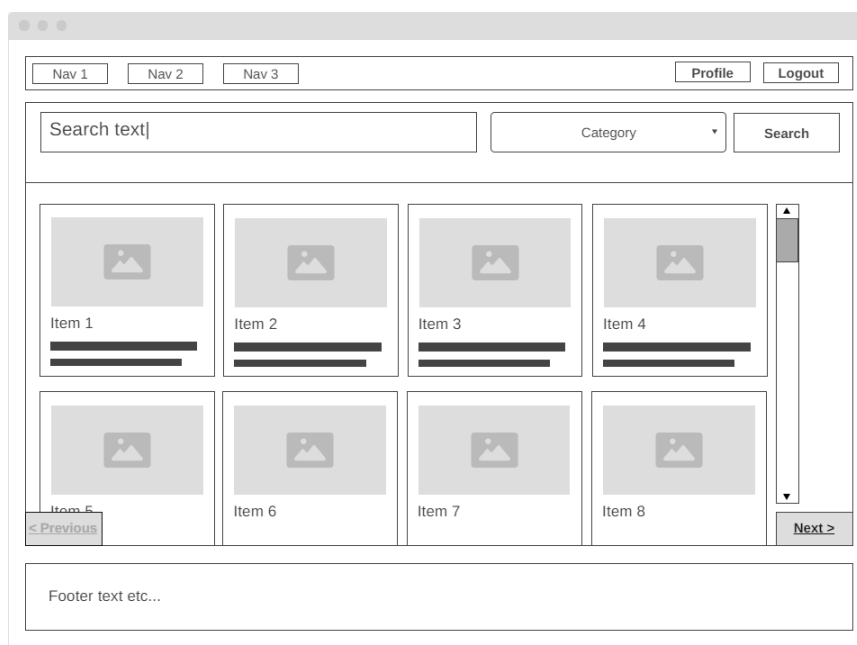
Show 10 | 25 | 50 Items

Item 1	
Item 2	
Item 3	
Item 4	
Item 5	

1 2 3 4 5 6 ... N Add Item









Footer text etc...

Image 7 - Primary entity list page (variant 1)



Nav 1 Nav 2 Nav 3 Profile Logout

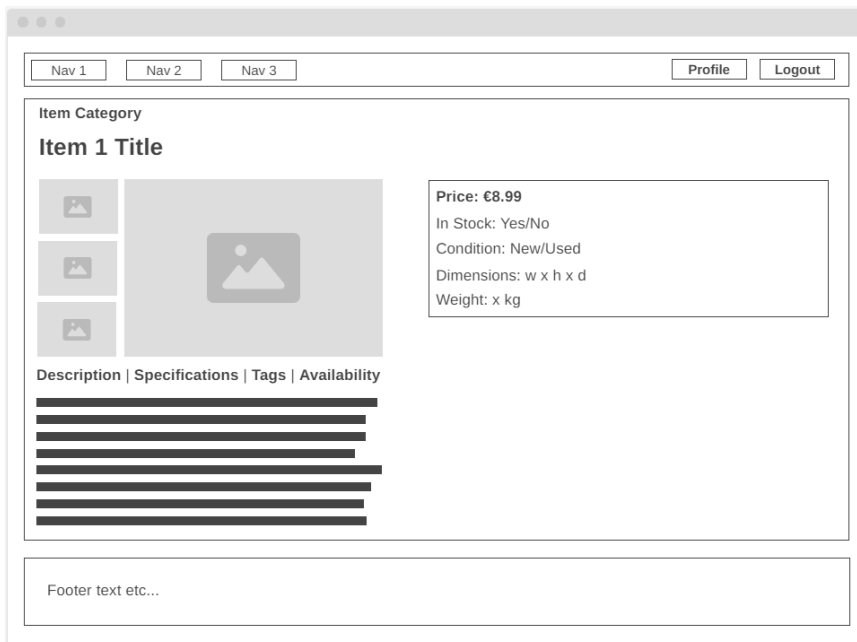
Search text| Category Search

 Item 1	 Item 2	 Item 3	 Item 4
 Item 5	 Item 6	 Item 7	 Item 8

< Previous Next >

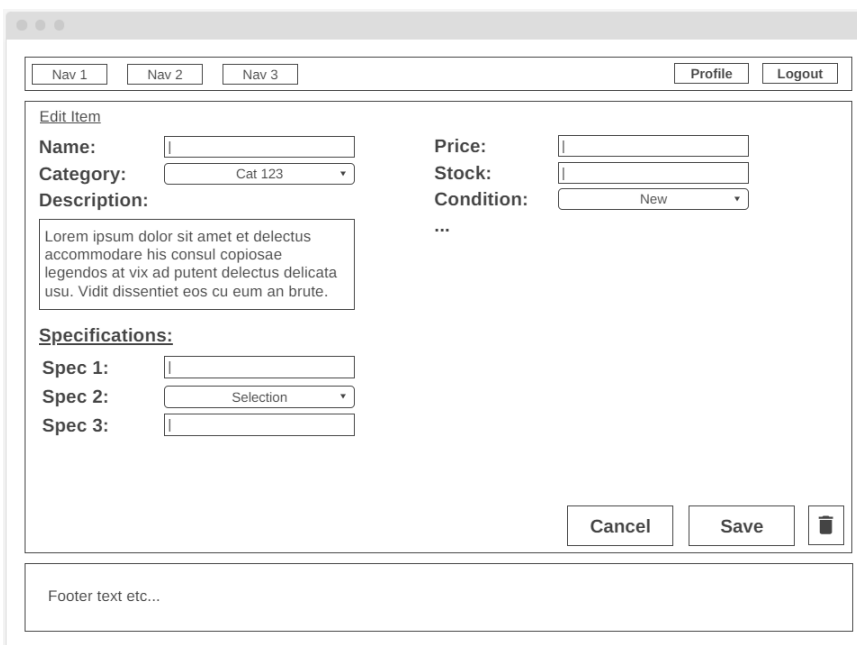
Footer text etc...

Image 8 - Primary entity list page (variant 2)



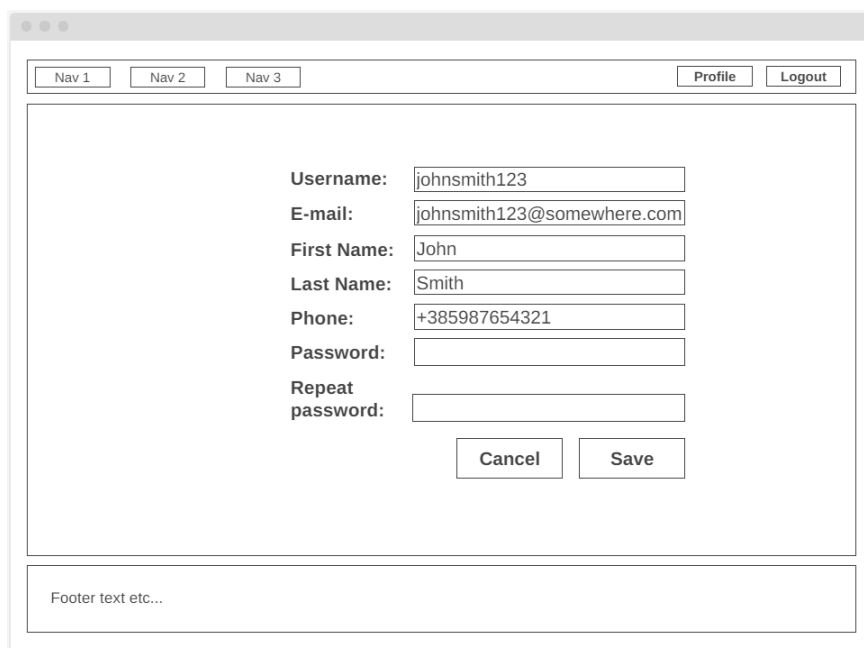
The screenshot shows a web application interface for viewing item details. At the top, there is a navigation bar with three tabs: 'Nav 1', 'Nav 2', and 'Nav 3'. On the right side of the navigation bar are two buttons: 'Profile' and 'Logout'. Below the navigation bar, the main content area is titled 'Item Category' and 'Item 1 Title'. It features a grid of image placeholders on the left and a box on the right containing item details: 'Price: €8.99', 'In Stock: Yes/No', 'Condition: New/Used', 'Dimensions: w x h x d', and 'Weight: x kg'. Below the image grid, there are tabs for 'Description', 'Specifications', 'Tags', and 'Availability'. The 'Description' tab is active, showing several lines of placeholder text. At the bottom of the page, there is a footer section labeled 'Footer text etc...'.

Image 9 - Primary entity details page



The screenshot shows a web application interface for adding or editing an item. At the top, there is a navigation bar with three tabs: 'Nav 1', 'Nav 2', and 'Nav 3'. On the right side of the navigation bar are two buttons: 'Profile' and 'Logout'. Below the navigation bar, the main content area is titled 'Edit Item'. It contains several form fields: 'Name' (text input), 'Category' (dropdown menu with 'Cat 123' selected), 'Description' (text area with placeholder text), 'Price' (text input), 'Stock' (text input), 'Condition' (dropdown menu with 'New' selected), and 'Specifications' (three text inputs labeled 'Spec 1:', 'Spec 2:', and 'Spec 3:'). At the bottom right of the form area, there are three buttons: 'Cancel', 'Save', and a trash icon. At the bottom of the page, there is a footer section labeled 'Footer text etc...'.

Image 10 - Primary entity add/edit page



The image shows a web browser window with a profile page. At the top, there are three navigation tabs labeled 'Nav 1', 'Nav 2', and 'Nav 3'. To the right of these tabs are two buttons: 'Profile' and 'Logout'. The main content area contains a form with the following fields and values:

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

At the bottom of the form are two buttons: 'Cancel' and 'Save'. Below the form is a footer section with the text 'Footer text etc...'.

Image 11 - Profile page