An Online Store

Groups: **3 students**.

An online store is a retail business that sells different products or services to the public.

Overview

You are a software developer and one of your clients hired you to create an online store. You can create the details for this store, such as the products and or services it sells. For instance, It may be an online:

- Supermarket
- PetShop (selling products/services)
- Glasses
- Real estate (Houses, apartments and land)

Requirements

- The system must have 2 types of users: Clients and Administrators
    - Administrators are responsible for registering/managing administrators, customers, and products/services provided. The application already comes with an account *admin* with password *admin*.
    - Customers are users who access the system to buy products/services.
- The admin record includes, at least: name, id, phone, email.
- Each customer's record includes, at least: name, id, address, phone, email
- Product/services records include, at least: name, id, photo, description, price, quantity (in stock), quantity sold.
- Your store may sell products, services or both (you decide)
- Selling Products (or services): Products are selected, their quantity chosen, and are included in a cart. Products are purchased using a credit card number (any number is accepted by the system). The quantity of product sold is subtracted from the quantity in stock and added to the quantity sold. Carts are emptied only on payment or by customers.

- Product/Service Management: Administrators can create/update/read/delete (crud) new products and services. For example, they can change the stock quantity.
- Your functionality: Create a functionality that is specific to your application. It does not have to be something complicated. For instance, if you are selling cars, you may allow users to use an accelerator to hear how each car engine roars up and down.
- The system must provide accessibility requirements and provide good usability. The system must be responsive, meaning that it should complete assigned tasks within a reasonable time.

GitHub

GitHub will be used to house student projects. Reviewers will clone each project in GitHub after the deadline and review it (no need to send code to the reviewers).

Peer Review

Peer review is the evaluation of work by one or more people with similar competencies as the producers of the work (peers). It functions as a form of self-regulation by qualified members of a group within a field.

In our course, peer review groups will help other groups building their Online Stores. Each group will review the work of another group. Their job is to act as the “manager” of the project being reviewed. Reviewers can contact the group directly to help solve any problems/questions. These interactions must be documented in the review document (they count towards the reviewers’ grade). However, the revised group does not have to accept all the reviewers' opinions (but they have to document it in their project report).

The final version of the assignment will also be graded using the Peer Review method. Each group’s grade will be based on the quality of its assignment and its review of another group’s work. The reviewer’s opinions about a group’s work can be challenged. However, a group grade is not determined by its review. For the final version, groups can contact the teacher/monitor and inform them that they disagree with something, said by the reviewers (adding a justification).

Project Milestone 1: Store Mockups

The most common use of mockups in software development is to create user interfaces that show the end-user what the software will look like without having to build the software or the underlying functionality. UI mockups can range from very simple hand-drawn screen layouts, through realistic bitmaps, to **semi-functional user interfaces** developed in a software development tool.

Objectives:

- Develop a mockup (medium-high fidelity) of the Online Store application's graphical user interface (client-side). It should be, based on the requirements listed in the following sections.
- Practice mockup, HTML5 and CSS3 languages

What should be ready (in your GitHub Project)

- GitHub's README.md file with the Project Report (see below). The report should include the following items:
- A navigation diagram for your application. You must use the Single-Page Application style. There is no need for fancy diagrams. You may even draw them on paper and photograph them using a cellphone. You can also use the Marvel tool to simulate the interaction between screens.
- Mockups for all major system screens. Your mockup pages are static snapshots of your application at different points. Create HTML5/CSS3 files for, **at least**, the main app screen and 2 others. For the remaining screens, you can use HTML5/CSS3 files, mockup tools (e.g. Figma), or good quality drawings. Tip: some mockup tools generate HTML.

Project Report

The Project Report should begin with the group identification and the names and USP numbers of all students in the group. The Project Report has to have the following sections:

1. **Requirements**: The requirements are given in the assignment, but you have to add any new requirements needed by your particular store implementation.
2. **Project Description**: Describe how your project implements the functionality in the requirements. Diagrams can help a lot here.

3. **Comments About the Code**: Any comment you may want to add to help understand your code. This is good programming practice.
4. **Test Plan**: Text describing the tests that will be performed. If an automatic test tool/framework is used (ex: Selenium, JUnit, Spock), the code for the tests can be used.
5. **Test Results**: Text describing the test results. If an automatic test tool/framework is used, its output can be used.
6. **Build Procedures**: A step-by-step guide to run your code. You should start telling how to install whatever software you need, then how to download/build your program, and finally how to set up the environment to run it. Imagine that someone installing will just follow these commands (nothing more).
7. **Problems**: List any major problems you had.
8. **Comments**: Any comments you wish to add.

At this first milestone, you will probably only have material for the first two topics. But include the other topics, even if you leave them empty, and, if possible, write how you plan to implement them. For instance, for testing, you may say I intend to use the postman to test the back end, etc.

Do not forget to include in the Project Description:

- The functionalities you are going to implement.
- A navigation diagram for your application linked to the screen mockups. This link can be just the name of the mockup (i.e. Mockup1, Mockup2, etc) or an HTML link.
- The information you are going to save in the server (no need to decide how that information will be saved).

At this stage, do not focus on producing beautiful user interfaces (UIs) but rather think about what kinds of interfaces you will need and what information each interface needs to show and to ask from the user. Also, think about how the users should navigate your app to accomplish their tasks (such as buying a product, registering, etc).

Layout Suggestions for Screens

These are just suggestions. Feel free to use other combinations:

- Homepage with a description of services/products offered and login area (any type of user)
- If the user is a customer:
    - Screen with actions: buy a product, edit your account.
- If you are an administrator

- Screen with actions: register administrators/customers/products etc.) in a menu
- Product Inventory Management Screen (Add, Update, Delete, Consult)

Evaluation Criteria

Guiding Questions	**Minimum**	**Effort and Satisfactory Performance**	**Above and Beyond**
Do the screens cover all program functionality? (50% of the grade)	**<2 stars**: more than 50% of the functionality is covered by the screens.	**<4 stars**: All features are covered. The screens are well organized.	**<=5 stars**: All features are met. Screens are well organized and use appropriate widgets
HTML5 and CSS3. Is the web interface well structured? (50% of the grade)	**<2 stars**: the interface is broken due to poor HTML or CSS structuring/misuse of navigational elements	**<4 stars**: the interface is not broken, navigability is reasonable and well structured / browser compatibility	**<=5 stars**: elements were used that made the interface look “clean” and easy to navigate, and may allow compatibility between devices of different sizes

Project Milestone 2: Client Functionality

After developing the store client-side mockup, you should now implement it.

Objectives:

- Develop the client-side of the application, based on the requirements outlined in the previous milestone
- Apply the HTML5, CSS3, and JavaScript expertise learned in the course

What should be ready (in your GitHub Project)

- All client program interfaces and features should be functional. Meaning that they should respond to user input, but not necessarily do something useful.
- HTML5, CSS3, and Javascript files with the client code. There must be an index.html file and the client should work by placing these files on an HTTP server and pointing the browser to that file.
- Test results (see below).

There is no need to develop a server-side. Server functionality can be implemented in the client through mock objects. Mock objects are “fake” objects that simulate the behavior of a “real” class or object so that we can focus the test on the code being tested. For example, application data can be saved locally (in memory or the browser's local storage). This allows you to simulate (for a single client) all server database functionality.

The Test Plan and Test Results topics of your Project Report must include a test script of the program functionalities and the results of the tests performed by the team.

The allowed javascript frameworks are jQuery, materializecss, React, and Vue.

Tips:

- Update all Project Report topics to reflect the current state of your project.
- Make sure your Build Procedures are easily understandable to avoid problems with reviewers unable to run your program.
- If you do not use testing tools, just describe the tests performed and their results.

- The Comments About the Code, Problems, and Comments topics are not mandatory (just write No comments or No problems).

Evaluation Criteria

Guiding Questions	**Minimum**	**Effort and Satisfactory Performance**	**Above and Beyond**
Do screens cover all program functionality? (50% of grade)	**<2 stars**: More than 50% of functionality is covered by screens with some bugs.	**<4 stars**: All features are covered. The functionality of the screens has few bugs.	**<=5 stars**: All features are met. The screens are well organized with no important bugs detected. Appropriate widgets are used.
Is the web interface well structured? (50% of the rating)	**<2 stars**: interface is poorly functional due to poor HTML5 / CSS3 / JavaScript code. The code does not always use ECMA6 or has poor quality.	**<4 stars**: interface is functional, navigability is reasonable. The code is well structured and uses good quality ECMA6 script code	**<=5 stars**: elements were used that made the interface look pleasant and easy to navigate. Well-made code without major compatible problems across browsers.

Tip

You can use a JavaScript object or `window.localStorage` to save data (in JSON) that will be read/written to the server in the final version. You can simulate a server call using the function below. It returns a promise and adds a 100ms delay to the read/write operation.

```
function delay(){
    return new Promise(function(resolve) {
```

```
        setTimeout(resolve, 100);
    });
}
```

For instance, you can read a product using an id:

```
function async readProductById(id){
    await delay();
    return JSON.parse(localStorage.products)[id];
}
```

When the server-side functionality is working, you can change it to:

```
function async readProductById(id){
                              const       resp       =       await
fetch("http://localhost:3000/products/"+id);
    return resp.json();
}
```

Final Version: Fully Functional Application

Objectives:

- Fully develop the Online Store application, based on the requirements outlined in the previous tasks
- Applying HTML5, CSS3, JavaScript, Node.js and NoSQL expertise learned in the course

After developing the client-side of the Online Store, now the full application should be deployed, including server and database sides.

What should be ready (in your GitHub Project)

- All interfaces and features of the program must be ok (client and server).
- Server functionality must be implemented using node.js and the NoSQL databases (CouchDB or MongoDB).
- The allowed javascript frameworks are Vue, jQuery, Vuetify, and React.
- The source code must be properly formatted and commented on.
- The code should be compiled in a distributable form. There must be an index.html file.
- All Project Report topics must be finished.

Tips:

- Update all Project Report topics to reflect the current state of your project.
- Check again your Build Procedures and make sure they work for the client and server-side. Do not forget to initialize the database. Avoid problems with reviewers unable to run your program.
- If you do not use testing tools, just describe the tests performed and their results.
- The Comments About the Code, Problems and Comments topics are not mandatory (just write No comments or No problems).

Evaluation Criteria

Guiding Questions	**Minimum**	**Effort and Satisfactory Performance**	**Above and Beyond**
Does the application cover all program functionality? Does it follow the standards for Single-Page Applications (SPAs)? (50% of rating)	**<2 stars**: Over 50% of the features are covered with some bugs.	**<4 stars**: All features are covered. Features (including interfaces) have few bugs	**<=5 stars**: All features are met. The interfaces are well organized. There are no major bugs. Well-made code without major compatible problems across browsers.
Is the server implemented correctly? Are web services well structured? (50% of grade)	**<2 stars**: Web services do not follow RESTful standards or their interface is poorly functional.	**<4 stars**: Web services follow RESTful standards. The interface is reasonably structured. Code uses ECMA 6	**<=5 stars**: Well-designed web services, suitable for SPAs. JavaScript code is well done.

Reviews

Each group will review the work of another group. That will be done for each milestone and the final work. All 3 reviews will have the same format. Each review is divided into the following topics:

1. The quality of the software design.
2. The quality of the code generated.
3. Could your group have done something better/different?

In topic 1, the reviewers must comment:

- On each of the 7 topics on the report (1. Requirements, 2. Project description, 3. Comments about the code, 4. Test plan, 5. Test results, 6. Build Procedures, 7. Problems),
- On any interactions, they have with the group (if any), and
- About the general quality of the material produced by the team.

Not all reviews will have all project topics available. For instance, the first milestone may have only topics 1 and 2 ready.

Topic 2 covers only the code. It should include any issues/problems found in the code. For topics 1 and 2, the reviewers should attribute up to 5 stars to them (There are evaluation criteria for each phase).

Topic 3 is about the reviewers' take on the project. What would you do differently? You can comment here on things that you would implement differently (even if the implementation presented in the project is fine). These comments should not affect the project's stars (given in topics 1 and 2). Comments about bad/wrong decisions in the project should go in the other review topics.

In all cases, the reviewers have to justify their points. For instance, if they say the code is not well documented, they should list points in the code that need comments.

Try to **suggest improvements** at the 1st and 2nd Milestones, so the team can improve their project (and final grade).

Grades

There is just one final grade for the Online Store. At each milestone, the group will have a grade that can change in the next phase. For the 2nd Milestone, the group will retain 30% of the 1st Milestone grade. For the Final Version, the group will retain 40% of the 2nd Milestone. As an example, if you get:

1st Milestone	2nd Milestone	Final Version
10	**5** `0.3*10 + 0.7*5 =` **`6.5`**	**3** `0.4*6.5 + 0.6*3  =` **`4.4`**
3	**5** `0.3*3 + 0.7*5 =` **`4.4`**	**10** `0.4*4.4 + 0.6*10 =` **`7.8`**

The review's grade will depend on how well the reviewers judge the assignment. If they complain of no existing problems or fail to point out existing ones, their grades will go down.

Notice

- The grade of a group's work is not given solely by the review of that work. The review serves as a tool to help the teacher/monitor to detect flaws in the projects. These failures are what determine the grades of the groups.
- **ALL** project/review documents must have the name and USP number of all students in the group.

Good luck!

Suggestion for Review Template

Peer Review

Review Group:
Name: NUSP:
Name: NUSP:
Name: NUSP:

Group Being Revised:
Name: NUSP:
Name: NUSP:
Name: NUSP:

1. **Project Quality:**
   Comment the topics:
   a) Requirements:

Requirements	Situation	Comments
Client		
Admin		
Products		
Product Sale		
Product Management		
The group Functionality		

   b) Project Description:
   c) Comments About Code: *If the group made some*
   d) Plans and Test Results:
   e) Build Procedures:
   g) *Problems, General Work Quality, etc.*
   f) Group Interactions: *Describe and comment on your interactions with the group*

2. Code Quality:

   a) Code Design and frameworks' use:

b) Code Organization:

c) How the code works:

d) Code Documentation:

3. What you would make differently:

Anything you would make differently (It does not necessarily mean that the group implemented these things badly).

You may include, in any topic, screenshots, links or code lines to illustrate any point being discussed.