# Market – SPA Application – JS Apps Exam

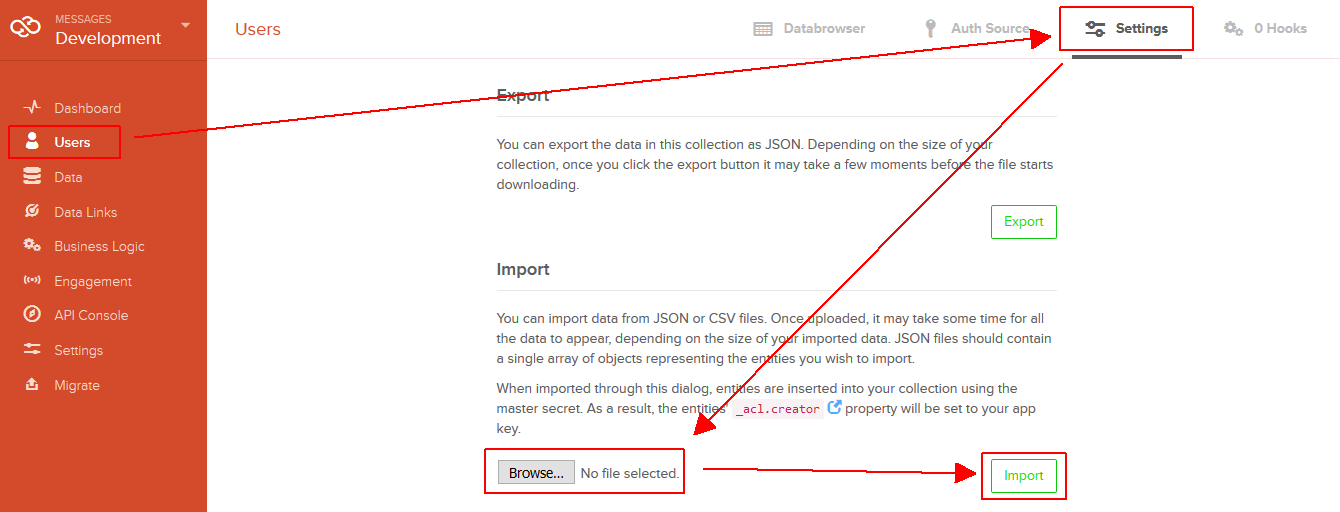
You are assigned to implement a **Shopping Web front-end application** (SPA) using HTML5, JavaScript, AJAX, REST and JSON with cloud-based backend (Kinvey). The app keeps **users** and **products**. Users can **register**, **login**, view the **products**, **purchase products** (**adding** them to their **cart**), view their **cart**, **discard products** (from their cart) and **logout**.

Using libraries like jQuery and React is allowed but is not obligatory.

## Create a Kinvey REST Service to Hold Users and Products

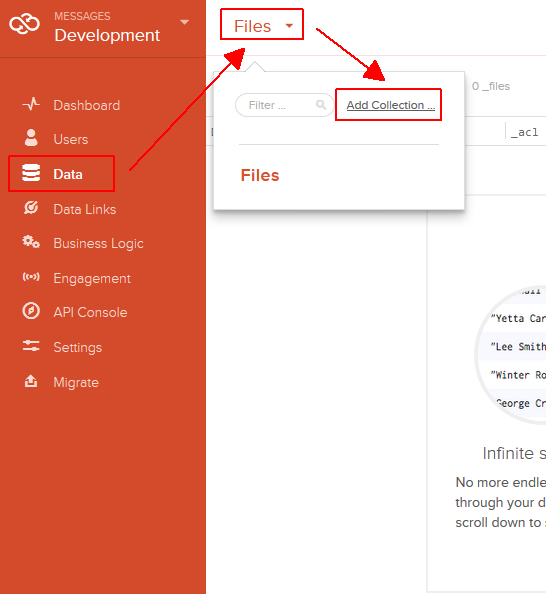
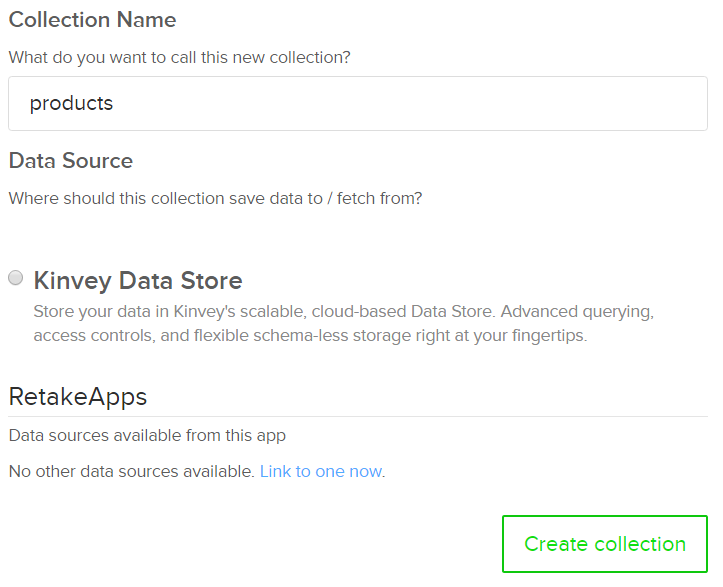
Register at **Kinvey.com** and create an application to keep your data in the cloud.

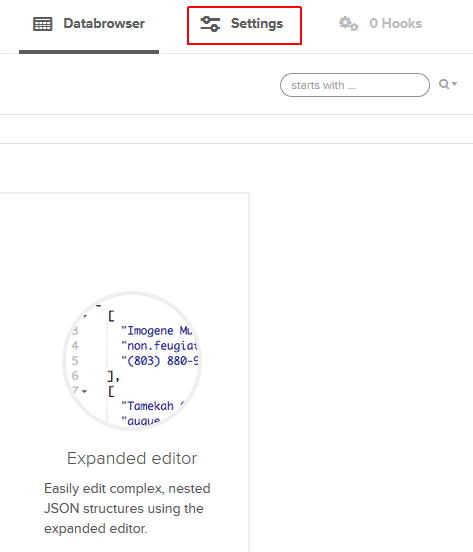
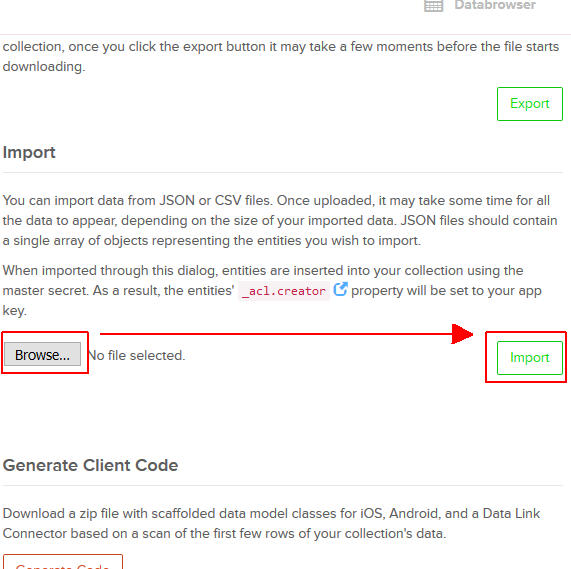
In the **Users** collection, import the provided JSON file with sample users to get started with template data. In the **Kinvey Console**, select **Users** from the navigation of the left, click **Settings** in the upper right then scroll down to the **Import** section:



Create a collection **products(name, description, price)** to hold the products. All the fields will hold text values, except the price - it will hold a numeric value. The “**\_kmd.lmt**” field is automatically created by Kinvey and will hold a date and time in the traditional for JavaScript dates ISO8601 format returned by [Date.toJSON()](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Date/toJSON).

Create a new collection and import the provided JSON file with sample **products** like shown below:

Kinvey will automatically create **REST services** to access your data.

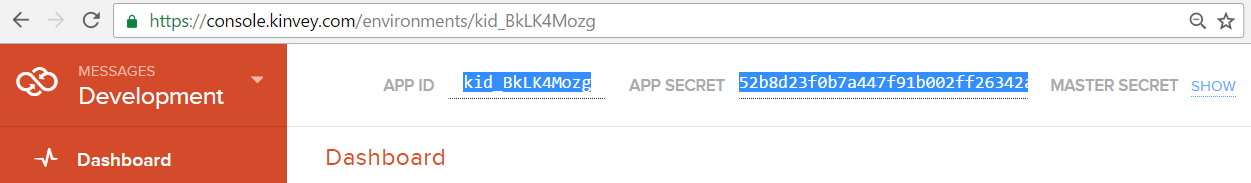
## Test the Kinvey REST Services

Using **Postman** or other HTTP client tool (you can use Kinvey’s built-in **API Console**), test the REST service endpoints:

### User Registration (Sign Up)

|  |  |
| --- | --- |
| **POST** [https://baas.kinvey.com/user/***app\_id***/](https://baas.kinvey.com/user/app_id/) | |
| Request headers | Authorization: Basic base64(app\_id:app\_secret)  Content-Type: application/json |
| Request body | {  "username": "new\_user",  "password": "pass123",  "name": "New User"  } |
| Response  201 Created | {  "\_id": "583f53bde004a9a90983f1b7",  "username": "new\_user",  "password": "pass123",  "name": "New User",  …  } |
| Error response  409 Conflict | { "error": "UserAlreadyExists", "description": "This username is already taken. Please retry your request with a different username", "debug": "" } |
| Error response  401 Unauthorized | { "error": "InvalidCredentials", "description": "Invalid credentials. Please retry your request with correct credentials", "debug": "" } |

The request needs “**Basic**” authentication. Use the Kinvey **app\_id** and Kinvey **app\_secret** as credentials.



### User Login

|  |  |
| --- | --- |
| **POST** [https://baas.kinvey.com/user/***app\_id***/login](https://baas.kinvey.com/user/app_id/login) | |
| Request headers | Authorization: Basic base64(app\_id:app\_secret)  Content-Type: application/json |
| Request body | {  "username": "new\_user",  "password": "pass123"  } |
| Response  200 OK | {  "\_id": "583f53bde004a9a90983f1b7",  "username": "new\_user",  "name": "New User",  "\_kmd": {  "**authtoken**": "**8e6471bc-3712-4cfb-b92e-50e62a0c80….Duj5fHdM /7XHIe6KdY=**"  …  },  …  } |
| Error response  401 Unauthorized | { "error": "InvalidCredentials", "description": "Invalid credentials. Please retry your request with correct credentials", "debug": "" } |

Successful login returns an “**authtoken**” which is later used to authenticate the CRUD operations.

### User Logout

|  |  |
| --- | --- |
| **POST** [https://baas.kinvey.com/user/***app\_id***/\_logout](https://baas.kinvey.com/user/app_id/_logout) | |
| Request headers | Authorization: Kinvey **authtoken** |
| Response  204 No Content |  |
| Error response  401 Unauthorized | { "error": "InvalidCredentials", "description": "Invalid credentials. Please retry your request with correct credentials", "debug": "" } |

To logout, you need to provide the “**authtoken**” given by login / register as “**Kinvey**” authorization header.

### Get All Products (Shop)

|  |  |
| --- | --- |
| **GET** [https://baas.kinvey.com/appdata/***app\_id***/products](https://baas.kinvey.com/appdata/app_id/products?query=%7b%22recipient_username%22:%22username%22%7d) | |
| Request headers | Authorization: Kinvey authtoken |
| Response  200 OK | [  {  "\_id": "5858699d4ad56c1314c48e96",  "name": "Apple",  "description": "An apple a day keeps the doctor away.",  "price": 0.5,  "\_acl": {"creator": "kid\_S1Oq7JU4g"},  "\_kmd": {"lmt": "2016-12-19T23:13:33.195Z"…}  },  {  "\_id": "585876b24ad56c1314c50eef",  "name": "Kroasan",  "description": "Chichipipikakao",  "price": 1.1,  "\_acl": {"creator": "kid\_S1Oq7JU4g"},  "\_kmd": {"lmt": "2016-12-20T00:09:22.624Z"…}  }, ...  ] |
| Error response  401 Unauthorized | { "error": "InvalidCredentials", "description": "Invalid credentials. Please retry your request with correct credentials", "debug": "" } |

### List Particular User

|  |  |
| --- | --- |
| **GET** [https://baas.kinvey.com/user/***app\_id***/](https://baas.kinvey.com/user/app_id/)***user\_Id*** | |
| Request headers | Authorization: Kinvey authtoken |
| Response  200 OK | [  {  "\_id": "5858679fe12ae039723fb628",  "username": "pesho",  "name": "Pesho Peshov",  "\_acl": {  "creator": "5858679fe12ae039723fb628"  },  "\_kmd": {  "lmt": "2016-12-20T01:09:51.514Z",  "ect": "2016-12-19T23:05:03.691Z"  },  "cart": {}  }  ] |

**NOTE:** The cart is kept in the Users collection. Every user has a cart field which corresponds to his cart. The cart is an **object** – **initally empty**. You will see how it is used later.

### Update Particular User

|  |  |
| --- | --- |
| **PUT** [https://baas.kinvey.com/user/***app\_id***/](https://baas.kinvey.com/user/app_id/)***user\_Id*** | |
| Request headers | Authorization: Kinvey authtoken |
| Request body | {  "\_id": "5858679fe12ae039723fb628",  "username": "pesho",  "name": "Pesho Peshov",  "\_acl": {  "creator": "5858679fe12ae039723fb628"  },  "\_kmd": {  "lmt": "2016-12-20T01:09:51.514Z",  "ect": "2016-12-19T23:05:03.691Z"  },  "cart": {  "585876e5b91c66ad2607e865": {  "quantity": "1",  "product": {  "name": "Toilet Paper",  "description": "IsSoft",  "price": "4.10"  }  }  }  } |
| Response  200 OK | [  {  "\_id": "5858679fe12ae039723fb628",  "username": "pesho",  "name": "Pesho Peshov",  "\_acl": {  "creator": "5858679fe12ae039723fb628"  },  "\_kmd": {  "lmt": "2016-12-20T01:09:51.514Z",  "ect": "2016-12-19T23:05:03.691Z"  },  "cart": {  "585876e5b91c66ad2607e865": {  "quantity": "1",  "product": {  "name": "Toilet Paper",  "description": "IsSoft",  "price": "4.10"  }  }  }  }  ] |

**NOTE:** The PUT request, **updates** the **object** in the **back-end** with whatever you send as a **body**. Note that you can send everything, but whatever you send, will **REPLACE** whatever is currently on the back-end at the **current id**.

Upon updating, make sure you first send a **GET** request, to **get the current entity**, and **change the data** you **receive**, for example the **cart**, and then **send** the **same data back**. Otherwise you might break the **back-end data**.

## Market – HTML and CSS

You аre given the Web design of the Market application as **HTML** + **CSS** files.

* Initially all views and forms are shown by the HTML. Your application may **hide** by CSS (display: none) or **delete** from the DOM all unneeded elements or just display the views it needs to display.
* You may render the views / forms / components with React, jQuery, Mustache or another UI library.

**Important**: don’t change the elements’ **class name** and **id**. Don’t rename form fields / link names / ids.

## Market Client-Side Web Application

Design and implement a client-side front-end app (SPA) for managing the **products** and **carts** of users. Implement the functionality described below.

### Navigation System

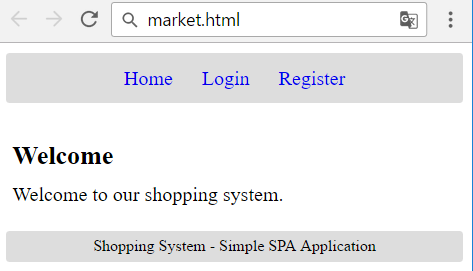
Implement a **navigation system** for the app: navigation links should correctly change the current screen (view).

* Clicking on the links in the **top navigation bar** should display the view behind the link (views are sections in the HTML code).
* Your application may **hide** by CSS (display: none) or **delete** from the DOM all unneeded elements or just display the views it needs to display.

5 score

### Home Screen

When no user is logged in, the app should display the "Home" screen holding a welcome message + three links: **[Home]**, **[Login]** and **[Register]**.

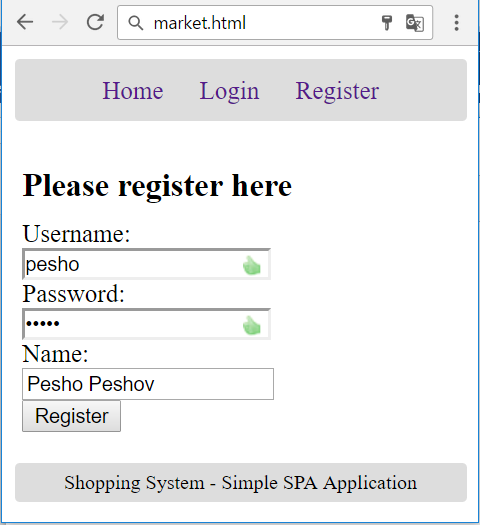


5 score

### Register User Screen

By given **username** + **password** + **name** the app should register a new user in the system.

* After a **successful registration**, a notification message “**User registration successful.**” should be displayed and the user home screen should be displayed.
* In case of **error**, an appropriate error message should be displayed and the user should be able to try to register again.
* **Form validation** is already implemented in the HTML, so you don’t need to add it.
* Keep the user session data in the browser’s **session storage**.

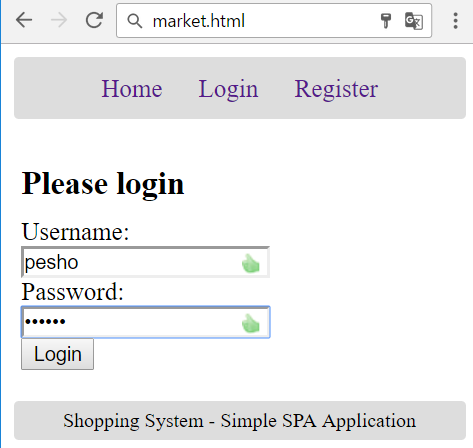


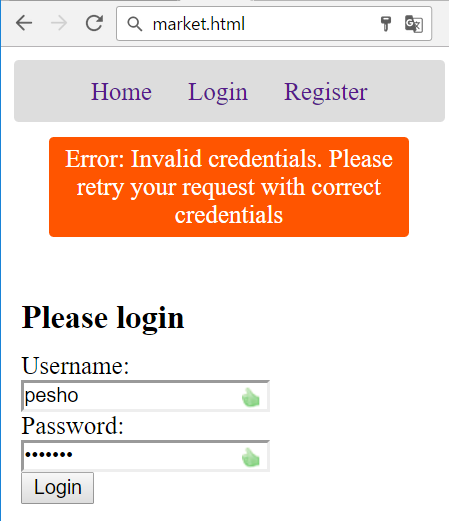
10 score

### Login User Screen

By given **username** and **password** the app should be able to login an existing user.

* After a **successful login**, a notification message “Login successful.” should be displayed and the user home screen should be displayed.
* In case of **error**, an appropriate error message should be displayed and the user should be able to fill the login form again.
* **Form validation** is already implemented in the HTML, so you don’t need to add it.
* Keep the user session data in the browser’s **session storage**.





5 score

### Logout

Successfully logged in user should be able to **logout** from the app.

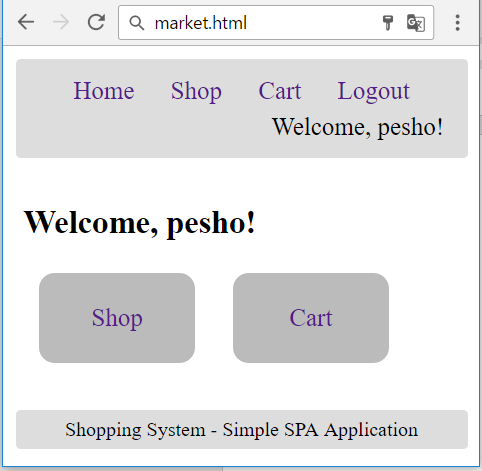
* After a **successful** logout, a **notification** message “Logout successful.” should be displayed.
* After successful logout, the **Home screen** should be shown.
* The **“logout” REST service** at the back-end should be obligatory called at logout.
* All local information in the browser (**user session data**) about the current user should be deleted.

5 score

### User Home Screen

After successful login, the app should display the **user's home screen**.

* It should hold a message **“Welcome, ” + the username** of the current user.
* At the top navigation bar display the navigation links **[Home]**, **[Shop]**, **[Cart]** and **[Logout]** + “**Welcome, {username}**”.
* At the main view area display navigation boxes **[Shop]**, **[Cart]**.



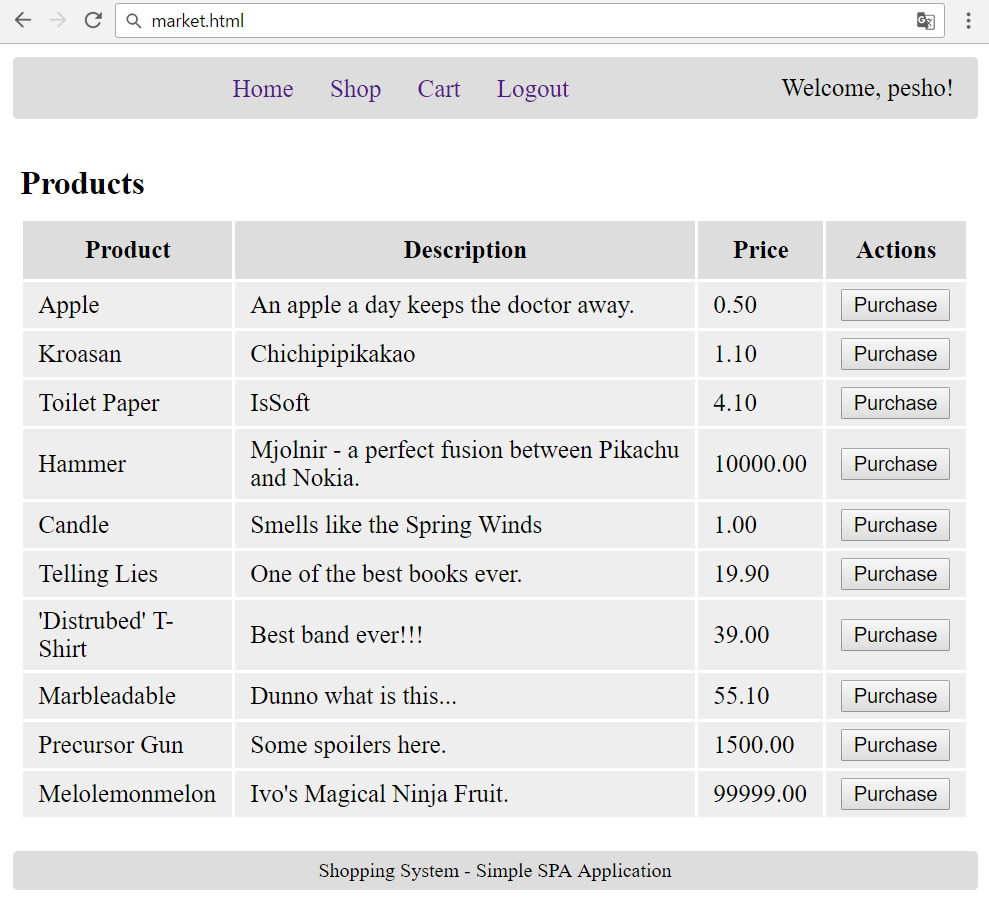
* Ensure you handle properly all HTML **special characters**, e.g. the username could be "*<pesho><br>*".

5 score

### Shop Products Screen

Successfully logged users after clicking the **[Shop]** link should be able to view all products.

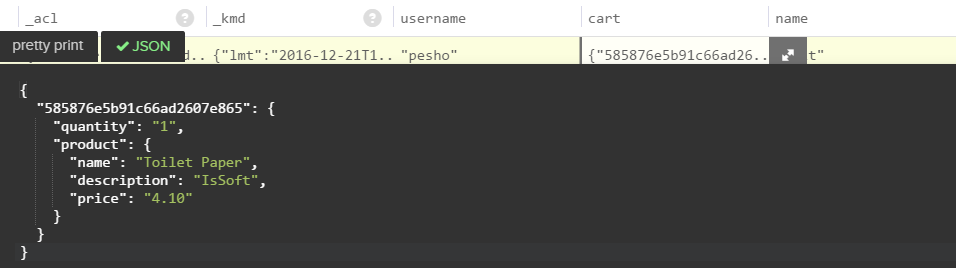
* The products should be listed in the **format** as shown in the Web design (see the screenshot below).
* In case of **error** (e.g. Internet connection lost), an error message should be displayed.
* Display **[Purchase]** button for each product in the shop. The button will purchase the product, adding it to the **user’s cart**.
* Thought this case is quite impossible… In case of **no products**, display an empty table (header row only).
* All prices should be **rounded**, the **default way** (0.505 == 0.51, 0.504 ==0.50) to the **second digit** after the **decimal point**, and printed the same way.



15 score

### Cart Products Screen

The **Cart functionality** is quite simple for **implementing**. When you must store products in the cart, upon purchase... You should store in the cart field, which is an **object**, the **id** **of the particular product**, as **a key** (**property**) and the **product’s quantity**, and **product data**, as a **value** (of the **property**). In the end it would look like this.

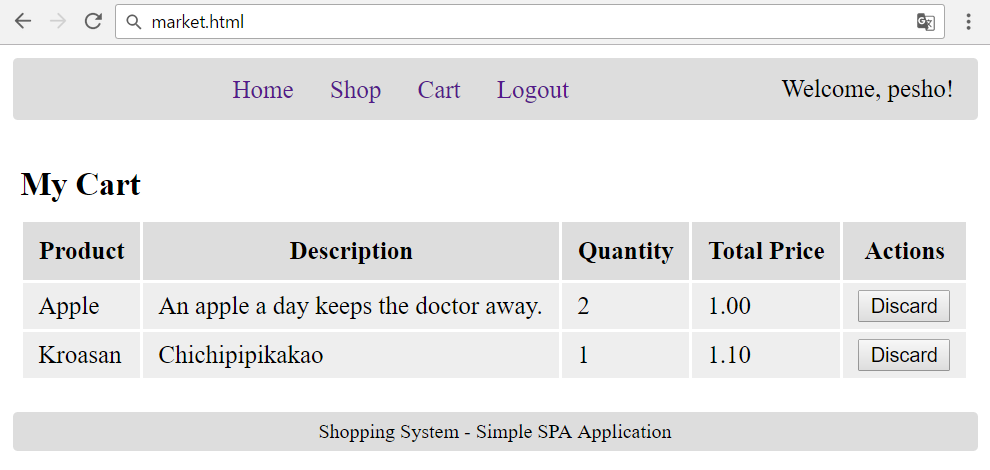


If you have multiple products, they are to be stored, each with its **id** (**as the key**), and the **quantity** and **product data** as an **object** (**as the value**). You have been given several users with multiple products in their carts. Use them as test data.

That is why you were given an Update end-point. So you could **update** the **cart** of a **user**, **adding** or **removing** a **property** from the cart **object**, or just **increasing** its **quantity**.

Successfully logged users after clicking the **[Cart]** link should be able to view all products, purchased by the **current user**, i.e. **products** which are currently in the user’s **cart**.

* The **products** should be listed **as shown in the Web design** (see the screenshot below).
* In case of **error** (e.g. Internet connection lost), an error message should be displayed.
* Display **[Discard]** button for each product in the cart. The button will discard the product, removing it from the **user’s cart**.
* In case of **no products**, display an empty table (header row only).
* All prices should be **rounded**, the **default way** (0.505 == 0.51, 0.504 ==0.50) to the **second digit** after the **decimal point**, and printed the same way.



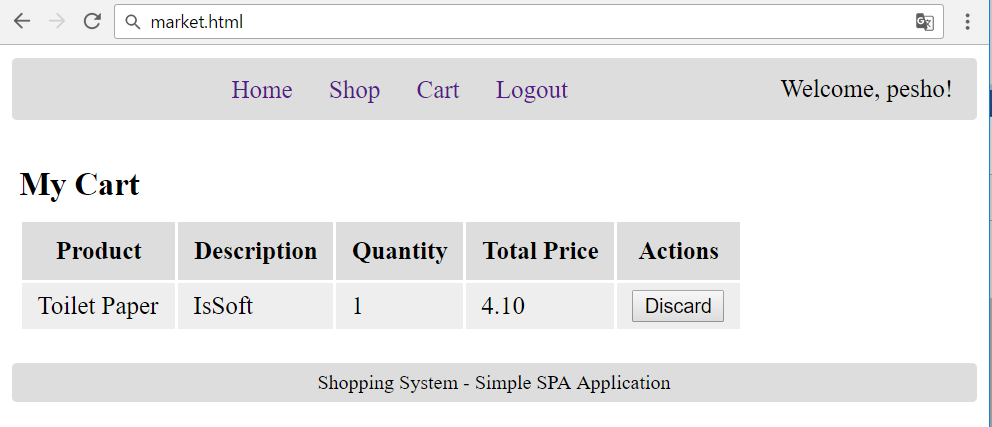
20 score

### Purchase Product

Successfully logged in users should be able to **purchase products** by choosing a **product**, from the Shop products, and clicking the **[Purchase]** button.



* After a **successful** product purchase, a notification message “**Product purchased.**” should be displayed and **the cart** (**user’s cart**) should be shown.
* In case of **error**, an appropriate error message should be displayed.
* Users are allowed to **purchase a product**, **more** than **once**, which is why the Quantity parameter stays in the table of the **Cart** view. The Total Price should be equal to the product\_price \* quantity.

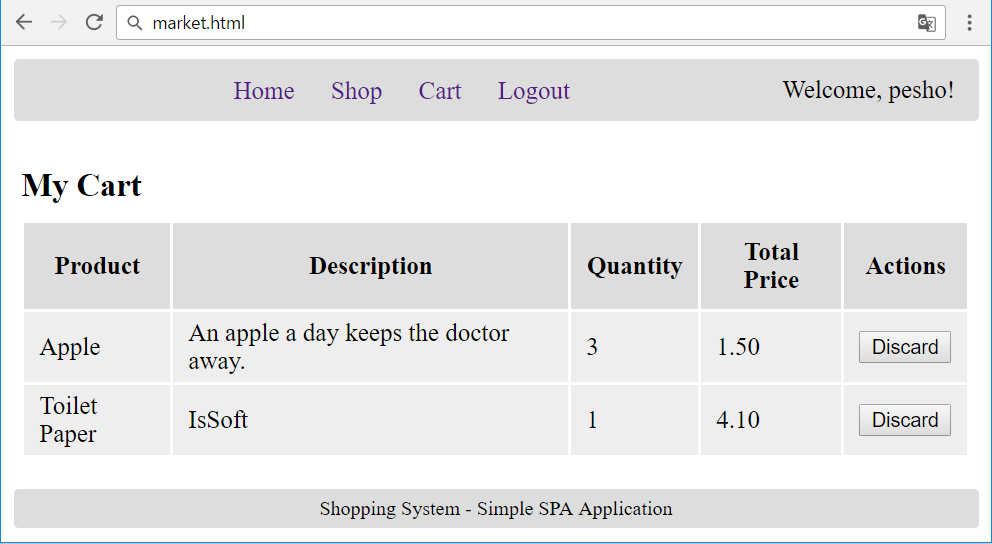


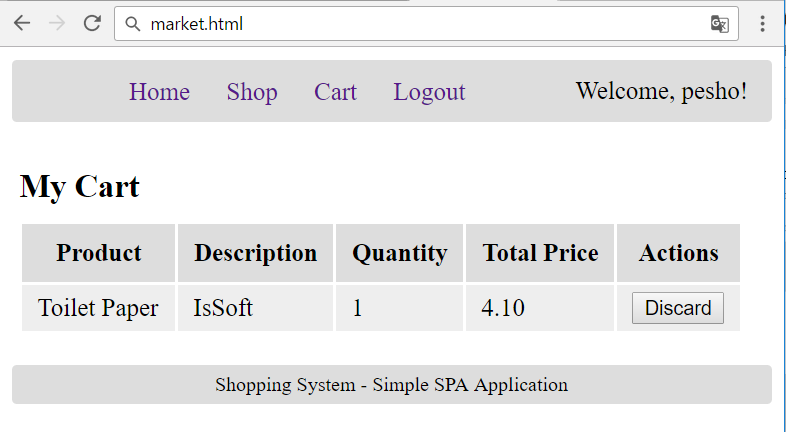
15 score

### Discard Product

Successfully logged in users should be able to **discard the products they purchased** by clicking on the **[Discard]** button in the table of product in the **Cart** view.

* After **successful** product discard a notification message “**Product discarded.**” should be shown.
* In case of **error** (e.g. Internet connection lost / unauthorized request / missing **product**), an error message should be displayed.
* The Deletion, should delete the **whole product**, **regardless** of its **quantity**.





15 score

### Notifications

The application should notify the users about the result of their actions.

* In case of successful action an **informational (green) notification message** should be shown, which disappears automatically after 3 seconds or manually when the user clicks it.



* In case of **error**, an **error notification message** (red) should be shown which disappears on user click.



* During the AJAX calls a **loading notification message (blue)** should be shown. It should disappear automatically as soon as the AJAX call is completed.



Good luck!