Mini Project 2:

BACO Wine shop API creation

Students:

Yaokun Li (yali@itu.dk)
Matheus Valentim (matva@itu.dk)
Nikol Černá (ncer@itu.dk)
Simona Těšínská (sitc@itu.dk)

Date: 17th of April, 2023

Project recap and resource needs

Recapping from the first project, we create a web shop to sell wine. We went through our project idea and specification to lay out which values and actions were more important for the functionality of the webpage so that we could design our API accordingly.

We ended up laying out 4 main groups of resources: basket, user, product and product categories. Basket refers to an user basket which is central for how the website will keep track of what the user wants to buy. We should have API calls that allows us to change the values inside an user basket. Namely, calls for fetching the data from a basket (*/basket/:name*), updating one or multiple items from a basket(*/basket/updateAllItems* and */basket/updateOneItem*) and deleting items from a basket.

As our website allows users to log in, we had to create API resource for users. We made API infrastructure for brand new users to be created (*/user*) and for users to be retrieved by their name (*/user/:name*).

The basic unit of our website are wines and hence we need endpoints and methods for them as well. Thinking of that, we created infrastructure to get all our products(*/winedata*) and to get a specific product using its id *(/winedata/:id*).

Finally, to get these basic units in an aggregated way, and specially, to use filters that provide a better user experience, we created endpoints and infrastructure so that we can get all wine that fall under a specific category value(*/winedatatype/:type, /winedatayear/:vintage, /winedataorigin/:country, /winedataprice/:price*).

Despite creating the server and exposing the aforementioned API's we did not have time to fully implement our calls into our previous existing code. The next section explains, in detail, what each combination of endpoint and method does.

API Documentation

We have divided the document into three sections where each of them corresponds to one resource. In such sections, we clarify which paths can be reached with each of the CRUDE methods and the expected result.

Firstly, we provide a list of all the resources and the results from the CRUDE commands:

path	POST	GET	PUT	DELETE
/basket/:name	Error	Gets a basket via the user name	Error	Error
/basket/updateAllItems	Error	Error	Updates all the items in an user's basket	Error

/basket/updateOneItem	Error	Error	Adds one item to a given user basket	Error
/basket/deleteOneItem	Error	Error	Error	Deletes an item provided by the client from the basket
/winedata	Error	Gets all products	Error	Error
/winedata/:id	Error	Gets a specific product via the product id	Error	Error
/winedatacategories	Error	Gets all product categories	Error	Error
/winedataprice/:price	Error	Gets all products that have a smaller or equal price to the specified price	Error	Errror
/winedatayear/:vintage	Error	Gets all products that are from a specific year	Error	Error
/winedataorigin/:country	Error	Gets all products that come from a specific country	Error	Error
/winedatacategory/:type	Error	Gets all products from a specific wine type	Error	Error
/user/:name	Error	Gets user via the user name	Error	Error
/user	Adds a new user	Error	Error	Error

Now we will give more details on each of the paths, grouping them by the resource they relate to.

Section 1: Basket resources

Path: **/basket/:name**

- Method: **GET**
- Summary: **Gets a basket of products via the user name**

- URL Params: **name,** String that refers to an user name, required in the path.
- Body: **Not used**
- Success Response:
  - Code: 200 OK
  - Body Content: Basket of the user in JSON format.
    - Example:
      `{"wine_id":15,"wine_name":"Sicilian","number":10,"price":39.99}`
- Error Response:
  - Code: 400
  - Body Content: String written “There is no such user”
- Sample Call:

```
//defining a Getter function that takes an username and retrieves the
user's basket
async function Getter(username){
        let result = await
fetch(`http://localhost:3000/basket/${username}`,
        {method: "GET"})

        let final_result = await result.json()
        console.log(final_result)
}

//calling the function to user "abc"
Getter("abc")
```

=====

Path: **/basket/updateAllItems**

- Method: **PUT**
- Summary: **Updates the whole basket of a given user**
- URL Params: Not used
- Body: A JSON formatted object with 2 properties, **name** and **basket**. The

name property corresponds to the user whose basket will be updated. The **basket** property is the new basket that will be associated to that user. Example:

    - `{"name": "abc", "basket":{"wine_id":15,"wine_name":"Sicilian","number":10,"price":39.99} }`

- Success Response:
    - Code:
    - Body Content: Basket of the user in JSON format.
        - Example:
        - `{"wine_id":15,"wine_name":"Sicilian","number":10,"price":39.99}`
        -
- Error Response:
    - Code: 400 BAD REQUEST
    - Body Content: A string message informing that there is no such user
    - `` `updateBasketByUserName no such user ${userName}` ``

- Sample Call:

```
//new basketData for a user abc
let basketData = {name: "abc",
basket:{wine_id:15,wine_name:"Sicilian",number:10,price:39.99} }


async function WholeBasketUpdater(){
        let result = await
fetch(`http://localhost:3000/basket/updateAllItems`,
        {method: "PUT",
        headers: {'Content-Type': 'application/json'},
        body: JSON.stringify(basketData)
        }
        )

        //checking the status of the call
        let final_result = result.status
        console.log(final_result)
}

//calling the function to user "abc"
WholeBasketUpdater()
```

=====

Path: **/basket/updateOneItem**

- Method: **PUT**

- Summary: **Adds or updates the properties of a given item in a given user basket.**

- URL Params: Not used

- Body: A JSON formatted object with two properties: name and item info. Name will be used to locate a specific user, whose basket we want to update a product. Item info is the new info for a product that should be in that basket. Therefore, Item info is an object with the following properties' names and types: **wine_id** (of type number), **wine_name**(of type String), **number**(of type number), **price** (of type number).

- Example:
- `{name:"Lee",itemInfo:{"wine_id":15,"wine_name":"Sicilian","number":10,"price":39.99 }}`

- Success Response:
  - Code: 200 OK
  - Body Content: No content will be given back.

- Error Response:
  - Code: 400 BAD REQUEST
  - Body Content: A string message informing that there is no such user
  - `` `updateBasketByUserName no such user ${userName}` ``

- Sample Call:

```
//new product for the user Matheus' basket
let basketData = {name: "Matheus",
itemInfo:{wine_id:15,wine_name:"Sicilian",number:10,price:0.99} }


async function BasketUpdater(){
        let result = await
fetch(`http://localhost:3000/basket/updateOneItem`,
        {method: "PUT",
        headers: {'Content-Type': 'application/json'},
        body: JSON.stringify(basketData)
        }
        )

        //checking the status of the call
        let final_result = result.status
        console.log(final_result)
}

//calling the function to user "abc"
BasketUpdater()
```

=====

Path: **/basket/deleteOneItem**

- Method: **DELETE**
- Summary: **Deletes a given item in a given user's basket.**
- URL Params: Not used
-
- Body: A JSON formatted object with two properties with the following types, a **name (of type string)** and a **itemInfo (of type number).**
- Example:
- `{name:"Lee",`
- `wine_id: 15}`
- Success Response:
    - Code: 200 OK
    - Body Content: No content will be given back.
- Error Response:
    - Code: 400 BAD REQUEST
    - Body Content: There are two possible body contents in an error response. If the problem was that the user the client calls does not exist, this body content will be reported: `` `updateOneItemInBasket no such user ${userName}` ``
    - If the error was not finding the item that should be deleted, that is, there is no such item in the basket, the body content is: `` `updateOneItemInBasket no such item in Basket ${userName}`. ``
- Sample Call:

```
//Deleting a product from the user Matheus' basket, the id of the
product is 15
let basketData = {name: "Matheus",
wine_id:15 }
```

```
async function itemDeleter(){
        let result = await
fetch(`http://localhost:3000/basket/deleteOneItem`,
        {method: "DELETE",
        headers: {'Content-Type': 'application/json'},
        body: JSON.stringify(basketData)
        }
        )

        //checking the status of the call
        let final_result = result.status
        console.log(final_result)
}

//calling the function to user "abc"
itemDeleter()
```

==

Section 2: User resources

Path: **/user**

- Method: **POST**
- Summary: **Adds a new user**
- URL Params: **Not used**

- Body: User details in JSON format. The details must have 1 or 2 properties: it must contain user name, and it can contain an user basket, which is an array of products if it doesn't, an empty basket will be created.

- Examples:
  With 1 property
  {“name”: “Matheus”}

  With 2 properties, name and a basket

  {“name”:”Matheus”,
  “basket”:
  [{"wine_id":15,"wine_name":"Sicilian","number":10,"price":39.99}]
  }

- Success Response:
    - Code: 200 OK
    - Body Content: The created object in JSON format. Example: {"name":"Matheus","basket":[]}

- Error Response:
    - Code: 400 BAD REQUEST
    - Body Content: -

- Sample Call:

```
//Creating a new object newUser with property name "Li" (I could also
add a new property with the basket)
let newUser = {name: "Li"}

//Creating a function that takes that object and puts it in the data
async function userAdder(){
        let result = await fetch(`http://localhost:3000/user`,
        {method: "PUT",
        headers: {'Content-Type': 'application/json'},
        body: JSON.stringify(newUser)
        }
        )

        //This call returns what has been added
        let final_result = await result.json()
        console.log(final_result)
}

//Calling the function
userAdder()
```

========

Path: **/user/:name**

- Method: **GET**

- Summary: **Gets a specific user info**

- URL Params: **name,** String

- Body: Not used

- Success Response:
    - Code: 200 OK
    - Body Content: User details in JSON format. The user details are instances that have the properties name, password and basket.

    - Example:
```
{"name":"abcd",
"password":"abc",
"basket":[{"wine_id":15,"wine_name":"Sicilian","number":10,"
price":39.99}]}
```

- Error Response:
    - Code: 400 BAD REQUEST
    - Body Content: A string with the value “no such user: “ followed by the user name: `` `no such user: ${name}` ``

- Sample Call:

```
//defining an async function to retrieve info from a user "user"
async function Getter(user){
    let result = await fetch(`http://localhost:3000/user/${user}`,
    {method: "GET"})

    let final_result = await result.json()
    console.log(final_result)
}
//calling the function to get info on a user called Matheus
Getter("Matheus");
```

=============================

Section 3: Products resources

Path: /winedata

- Method: **GET**

- Summary: **Gets all the wine data**

- URL Params: Not used

- Body: Not used

- Success Response:
  - Code: 200 OK
  - Body Content: An array of JSON formatted wine objects. That is an array of objects, in JSON format, with the following properties: **name, region, country, type, brand, vintage, price, details.**

  - Example:
    This is in the same format but only has two wine objects in the array, and their detail property was abbreviated for better comprehension.

    ```
    [{"name":"Chateau de la
    Lune","region":"Bordeaux","country":"France","type":"Champag
    ne","brand":"Château de la Belle
    Vue","vintage":2015,"price":19.99,"details":"This Champagne
    has a bright, golden color with aromas of freshly baked
    bread and ripe fruit"},{"name":"Luminous
    Blanc","region":"Napa
    Valley","country":"USA","type":"White","brand":"Coastal
    Vines
    Winery","vintage":2016,"price":29.99,"details":"Luminous
    Blanc is a crisp and refreshing white wine made from 100%
    Chardonnay grapes grown in the sun-drenched vineyards of
    California."}]
    ```

- Error Response:
  There are two possible error situations: if the error code is `"ENOENT",` then these are the characteristics of the error response:
  - Code: 400 BAD REQUEST
  - Body Content: Empty array. Example: []

  If, however, the error code is not `"ENOENT",` then the response will have the following:

  Code: 400 BAD REQUEST
  - Body Content: -

- Sample Call:

```
//defining an async function
async function Getter(){
        let result = await
fetch('http://localhost:3000/winedata',
        {method: "GET"})
```

```
        let final_result = await result.json()
        console.log(final_result)
}
//calling the function
Getter();
```

Path: /winedata/:id

- Method: **GET**
- Summary: **Gets data on a specific product via its id.**
- URL Params: **id, a string formatted number** that indexes a product in our data. Example: 2, in which the endpoint would be /winedata/2.
- Body: Not used
- Success Response:
    - Code: 200 OK
    - Body Content: A JSON formatted wine object. That is, an object, in JSON format, with the following properties: **name, region, country, type, brand, vintage, price, details.**
    - Example:

```
[{"name":"Luminous Blanc","region":"Napa
Valley","country":"USA","type":"White","brand":"Coastal
Vines
Winery","vintage":2016,"price":29.99,"details":"Luminous
Blanc is a crisp and refreshing white wine made from 100%
Chardonnay grapes grown in the sun-drenched vineyards of
California."}]
```

- Error Response:
  This call will only have a Body Content if the error is not finding the product of the given index. In this case the Body Content would have a string with the message: `` `Wine with ID:${wineId} doesn't exist` ``
    - Code: 400 BAD REQUEST
    - Body Content: In case of invalid index, a string `` `Wine with ID:${wineId} doesn't exist` ``

- Sample Call:

```
//defining a getter that takes an wineid parameter
async function Getter(wineid){
        let result = await
fetch(`http://localhost:3000/winedata/${wineid}`,
        {method: "GET"})

        let final_result = await result.json()
        console.log(final_result)
}
//Calling the Getter with id 2
Getter(2)
```

==============================

Section 4: Product categories resources

Path: **/winedatacategories**

- Method: **GET**

- Summary: **This function retrieves a JSON object in which the properties represent distinct categories, and the corresponding values denote the feasible options within each category.**

- URL Params: Not used

- Body: Not used

- Success Response:
    - Code: 200 OK
    - Body Content: All product categories and their possible results in JSON format.
        - Example:
          ```
          {"prices":[19.99,29.99,39.99,49.99,59.99],"countries":["France","USA","Spain","Italy"],"types":["Champagne","White","Rosé","Red"],"vintages":[2015,2016,2011]}
          ```

- Error Response:
    - Code: 400
    - Body Content: -

- Sample Call:

```
//defining a Getter function that retrieves all categories and their
possible values
```

```
async function Getter(username){
        let result = await
fetch(`http://localhost:3000/winedatacategories`,
        {method: "GET"})

        let final_result = await result.json()
        console.log(final_result)
}

//calling the function to user "abc"
Getter()
```

Path: **/winedataprice/:price**

- Method: **GET**
- Summary: **Gets a JSON formatted array of wine products that either match the specified price or have a lesser price**
- URL Params: Not used
- Body: Not used
- Success Response:
  - Code: 200 OK
  - Body Content: An array of wine products that have the specified price or a smaller price as their price property. In other words, it returns an array of objects with the properties **name, region, country, type, brand, vintage, price, details** where the **price** property equals :price.
    - Example: array with only one wine, the only one that has price equal or smaller than 19.99

```
      [{"name":"Chateau de la
Lune","region":"Bordeaux","country":"France","type":"Champag
ne","brand":"Château de la Belle
Vue","vintage":2015,"price":19.99,"details":"This Champagne
has a bright, golden color with aromas of freshly baked
bread and ripe fruit. On the palate, it is rich and complex
with flavors of citrus, green apple, and a hint of honey.
The finish is long and elegant, with a refreshing acidity
that lingers on the tongue."}]
```

- Error Response:
  - Code: 400
  - Body Content: -
- Sample Call:

```
//defining a Getter function that retrieves the all wines that cost
49.99
async function Getter(username){
        let result = await
fetch(`http://localhost:3000/winedataprice/49.99`,
        {method: "GET"})

        let final_result = await result.json()
        console.log(final_result)
}

//calling the function to user "abc"
Getter()
```

Path: **/winedatayear/:vintage**

- Method: **GET**
- Summary: **Gets a JSON formatted array of wine products that match the specified vintage**
- URL Params: Not used
- Body: Not used
- Success Response:
    - Code: 200 OK
    - Body Content: An array of wine products that have the specified vintage as their vintage property. In other words, it returns an array of objects with the properties **name, region, country, type, brand, vintage, price, details** where the **vintage** property equals :vintage.
        - Example: array with wines from 2011, details property shortened for better reading

```
      [{"name":"Barcelona Blanca","region":"Ribera del
Duero","country":"Spain","type":"White","brand":"Bodega del
Sol","vintage":2011,"price":39.99,"details":"Barcelona
Blanca is a bright and fresh white wine made from a blend of
traditional grape varieties grown in the hills surrounding
Barcelona."},{"name":"Sangria Sunset","region":"Ribera del
Duero","country":"Spain","type":"Red","brand":"Cava del
Rey","vintage":2011,"price":49.99,"details":"The wine has a
deep ruby color with aromas of ripe red fruit, blackberry,
and a hint of spice. On the palate, it is smooth and juicy
with flavors of cherry, raspberry, and a touch of cinnamon.
The wine finishes with a light sweetness and a subtle
acidity."},{"name":"Marbella Blush","region":"Ribera del
Duero","country":"Spain","type":"Rosé","brand":"Rioja Real
Vineyards","vintage":2011,"price":49.99,"details":"The wine
finishes with a lively acidity and a long, pleasant
```

```
        aftertaste."}]
```

- Error Response:
  - Code: 400
  - Body Content: -

- Sample Call:

```
//defining a Getter function that retrieves the all wines that have
vintage = 2011
async function Getter(username){
        let result = await
fetch(`http://localhost:3000/winedatayear/2011`,
        {method: "GET"})

        let final_result = await result.json()
        console.log(final_result)
}

//calling the function to user "abc"
Getter()
```

Path: **/winedataorigin/:country**

- Method: **GET**

- Summary: **Gets a JSON formatted array of wine products values that come from the specified country**

- URL Params: **country,** String, country used to filter the wine

- Body: Not used

- Success Response:
  - Code: 200 OK
  - Body Content: An array of wine products that have the specified country as their country property. In other words, it returns an array of objects with the properties **name, region, country, type, brand, vintage, price, details** where the **country** property equals **:country**.
    - Example: array with wines from France, details property shortened for better reading

```
      [{"name":"Chateau de la
Lune","region":"Bordeaux","country":"France","type":"Champag
ne","brand":"Château de la Belle
Vue","vintage":2015,"price":19.99,"details":"This Champagne
has a bright, golden color with aromas of freshly baked
bread and ripe fruit. On the palate, it is rich and complex
with flavors of citrus, green apple, and a hint of honey.
The finish is long and elegant, with a refreshing acidity
that lingers on the tongue."},{"name":"Rosé de
Reims","region":"Bordeaux","country":"France","type":"Champa
gne","brand":"La Maison des
Vignes","vintage":2015,"price":49.99,"details":"Champagne
Rosé de Reims is a refined and elegant sparkling wine made
from a blend, of Pinot Noir, Chardonnay, and Pinot Meunier
grapes grown in the world-renowned Champagne
region."},{"name":"Provence
Pink","region":"Bordeaux","country":"France","type":"Rosé","
brand":"Domaine de la
Roche","vintage":2015,"price":49.99,"details":"Provence Pink
is a delicate and refreshing rosé wine made from a blend of
Grenache, Cinsault, and Syrah grapes grown in the
picturesque vineyards of Provence. The wine has a beautiful
pale pink color with aromas of fresh red fruit, peach, and a
hint of citrus. On the palate, it is light and crisp with
flavors of strawberry, raspberry, and a touch of minerality.
The wine finishes with a bright acidity, and a clean,
refreshing aftertaste."}]
```

- Error Response:
  - Code: 400
  - Body Content: In case of invalid country, a string “no such origin:”
- Sample Call:

```
//defining a Getter function that retrieves the all wines that come
from France
async function Getter(username){
        let result = await
fetch(`http://localhost:3000/winedataorigin/France`,
        {method: "GET"})

        let final_result = await result.json()
        console.log(final_result)
}

//calling the function to user "abc"
Getter()
```

Path: **/winedatacategory/:type**

- Method: **GET**

- Summary: **Gets a JSON formatted array of wine products values that have the specified**

- URL Params: **type**, String, type of the wine.

- Body: Not used

- Success Response:
    - Code: 200 OK
    - Body Content: An array of wine products that have the specified type as their type property. In other words, it returns an array of objects with the properties **name, region, country, type, brand, vintage, price, details** where the **type** property equals **:type**.
        - Example: array with wines from France, details property shortened for better reading

```
    [{"name":"Sangria Sunset","region":"Ribera del
Duero","country":"Spain","type":"Red","brand":"Cava del
Rey","vintage":2011,"price":49.99,"details":"Sangria Sunset
is a fruity and refreshing red wine made from a blend of
Tempranillo, Garnacha, Bobal, and Monastrell grapes grown in
the sunny vineyards of Valencia. The wine has a deep ruby
color with aromas of ripe red fruit, blackberry, and a hint
of spice. On the palate, it is smooth and juicy with flavors
of cherry, raspberry, and a touch of cinnamon. The wine
finishes with a light sweetness and a subtle
acidity."},{"name":"Napa Noir","region":"Napa
Valley","country":"USA","type":"Red","brand":"Sonoma Hills
Vineyard","vintage":2016,"price":59.99,"details":"Napa Noir
is a full-bodied and elegant red wine made from a blend of
Cabernet Sauvignon, Merlot, and Cabernet Franc grapes grown
in the world-renowned Napa Valley region of California. The
wine has a deep ruby color with aromas of dark fruit, black
cherry, and a hint of vanilla. On the palate, it is complex
and rich with flavors of blackberry, cassis, and a touch of
tobacco. The wine finishes with silky tannins and a long,
lingering aftertaste."},{"name":"Sicilian
Sunset","region":"Tuscany","country":"Italy","type":"Red","b
rand":"Montepulciano
Estate","vintage":2016,"price":39.99,"details":"Sicilian
Sunset is a rich and robust red wine made from the Nero
d'Avola grape, grown in the sunny vineyards of Sicily. The
wine has a deep ruby color with aromas of ripe black fruit,
blackberry, and a hint of vanilla. On the palate, it is
full-bodied and flavorful with notes of black cherry, plum,
and a touch of spice. The wine finishes with firm tannins
and a long, lingering aftertaste."}]
```

- Error Response:
    - Code: 400
    - Body Content: In case of invalid country, a string “no such origin:”
- Sample Call:

```
//defining a Getter function that retrieves the all wines that are Red
wines
async function Getter(username){
        let result = await
fetch(`http://localhost:3000/winedatacategory/Red`,
        {method: "GET"})

        let final_result = await result.json()
        console.log(final_result)
}

//calling the function to user "abc"
Getter()
```