InternProducts - iOS School

1. INTRODUCTION

As a graduation project from Endava's "School of iOS 2022" programme you are tasked with creating an application that communicates with a backend and displays to their user a list of products. You will be provided with a bundled server that will run locally and accept requests at `http://localhost:8080` which will act as the backend, along with a series of resources describing the requirements of the `InternProducts` app.

The development will be done in a `MVP → MVP + bonus features` manner. That is, the app must implement a minimum set of functionalities in order to be considered a viable usable product (Minimum Viable Product). Only after these requirements are met you may start implementing/adding new functionalities which we consider to be `bonus features`. In other words, do not start working on bonus features if the minimum viable product is not yet ready~

The purpose of this project is to test & verify your ability to organize yourself and your project. Design your architecture in a clear, concise, easy to understand and maintainable manner. Follow any code best-practices that you know, use any architectural patterns that you think may help.

The design and design metrics that you will be presented with should be considered as guidances at most. That is, the visual elements and the order they appear in your app's screens should resemble what is shown in the design screenshots, but they do not have to be pixel-perfect (i.e. you should not spend too much time on trying to match the design perfectly; your main focus should be the functionalities and the way in which you implement them).

It is assumed that at this point you have the technical knowledge to implement the project requirements. However you may continue to call out for mentors to help you whenever you encounter a strange bug, or need some more explanation on how to use any `UIKit` / iOS feature in general, but the mentors shall not handle any questions regarding your app architecture(i.e. “Should I put these in a struct, class or keep the properties separate?”, “Should I use a tableView or a collectionView?”, “Should I make a class for network requests?”, etc…). These are for you to decide.

2. BACKEND description

Attached in the package there is a zip named `LocalBackend.zip`. Unzip it and you will find an executable named `LocalBackend`. Double-click to run it and give any permissions for file access that you are prompted to. Due to the way OSX treats foreign executables & binaries you will be prompted multiple times to allow the system to open the app, and even afterwards you will have to manually give permissions in the “System Preferences → Security ” for both “LocalBackend” and the associated framework. Follow the instructional video “*LocalBackendPermissions.mov*”. If all goes well you should see a console window stating “[INFO] GCDWebServer started on port 8080 and reachable at http://<ipAddress>:8080/”. You can ignore the actual ip address listed there, as in your code you will use the `*localhost*` alias for it.

The server will respond to 3 HTTP requests, for each request the response will be a JSON string, described as follows:

1. The `Login` request:
   Http method: GET
   Path: /login
   Parameters: “username” and “password” that will be integrated as url query items.
   For example, the url to make a login request for the “admin” / ”1234” credentials will look like
   `http://localhost:8080/login?username=admin&password=1234”

   The are 2 types of responses that you may receive in this request:
   a) Error response: this will appear if there is no user with the presented username or the password is wrong. It will be a JSON string in the form:

```
{"status":"FAILED","message": “<some error message
here, type string>” }
```

b) Successful response: when the username and password match for an existing user:

```
{"status":"SUCCESS", "loginToken": "<login token
for the current user, type string>"}
```

2. The `Register` request:
Http method: GET
Path: /register
Parameters: “username” and “password” that will be integrated as url query items.
For example, the url to register a user with the “admin” / ”1234” credentials will look like
`http://localhost:8080/register?username=admin&password=1234”

You should expect the same types of responses that you receive in the `Login` request as well.

3. The `Products` request:
Http method: GET
Path: /products
Parameters: “loginToken” integrated as a url query item.

This request accepts a valid `loginToken` that you must have received either in a previous /login or /register call.
For example, the url to make a products request for the “123456” login token looks like:
`http://localhost:8080/products?loginToken=123456”

The types of responses you should expect:
1) Error response: this will appear if provided loginToken is invalid/does not match any existing user’s loginToken

```
{"status":"FAILED","message": “<some error message
here, type string>” }
```

2) In the successful case you will receive a JSON which has a list of the products at the key “products” like so:

```
{
    "status": "SUCCESS",
    "products": [
        {
            "tags": <list of tags associated with the product, encoded as
array<string> >,
            "title": “<“product title, string>”,
            "image": “<product’s display image encoded as a base64 string,
string>”,
            "description": “<“product description, string>”,
            "date": <product date encoded as a UNIX timestamp since 1970, int>
        }, ..]
}
```

Special consideration for the “image” and “date” elements. You will most probably have to convert them into appropriate objects like “UIImage” and “Date” in your code in order to manipulate/display them properly to the user. Everything you need for the image reconversion process is covered in the available methods and initializers of the Swift “String” and “Data” and “UIImage” types.

HINT: You may experiment and inspect responses returned by the server by simply creating the appropriate login/register/products urls with your custom values in the username / password / loginToken fields and pasting those links into your browser’s navigation bar and accessing them. The browser by default makes GET requests for any url in the navigation bar. You should then see the textual response in the page.

3. MVP description

Overview: at the very least the user must be able to register a new account on the app, login with that account and see the list of products that they receive from the server.

3.1) Login and registration

The app must always start with the “Login” screen presented to the user. On the login screen the user can navigate back and forth towards the “Register” screen by pressing the “Register” button. (see *LoginRegisterNavigation.mov*)

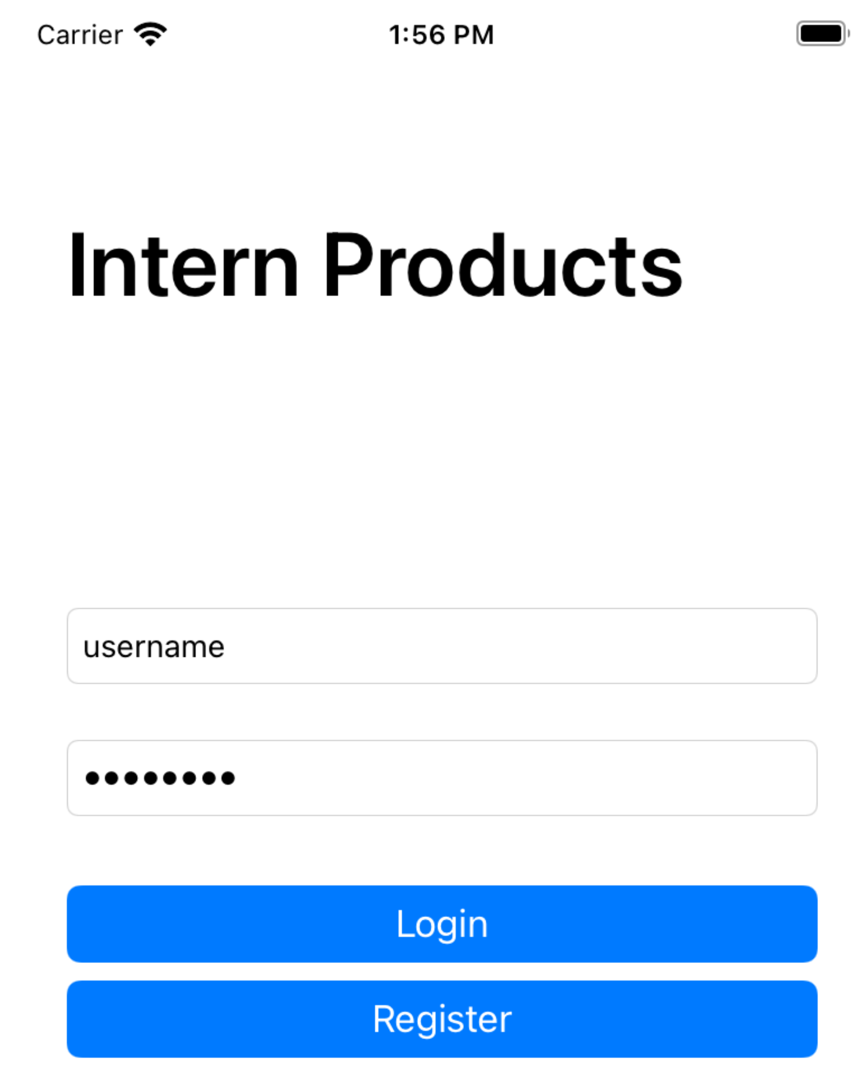
Upon pressing the “Login” button a http request should be made to the server to log in the user. If the server returns a “status: FAILED” with an error message then that error message should be displayed to the user in the form of an alert. See attached videos (“Login_passwordsMismatch”, “Login_unknownUser”)
If however the log in is successful then the user should be redirected to the products screen (explained in the next sections).

The general appearance of the “Login” screen should resemble what is displayed in the attached screenshot. In essence we have the following visual elements:

1) The title label, contents: “InternProducts”. You may pin it at 25 points spacing top, 20 points spacing to leading to the container view. Font: System Semibold 40

2) Username and password textfields. Both of them pinned at 20 points leading and trailing to the container view. No height constraints, as they have intrinsic size. Vertically spaced with 25 points between them. Vertical positioning: relatively above the centerY of their container view at 10 points.
   The password textfield must be set up with secure text entryi, i.e the user should see dots not actual characters. The user must see the “Username” and “Password” placeholders when the textfields are empty.

3) “Login” and “Register” buttons. Both of them pinned to 20 points trailing and leading to their container view. No height constraints as they have intrinsic size. Vertically spaced with 25 points between them. Vertical positioning: Relatively below the centerY of their container view, at 10 po

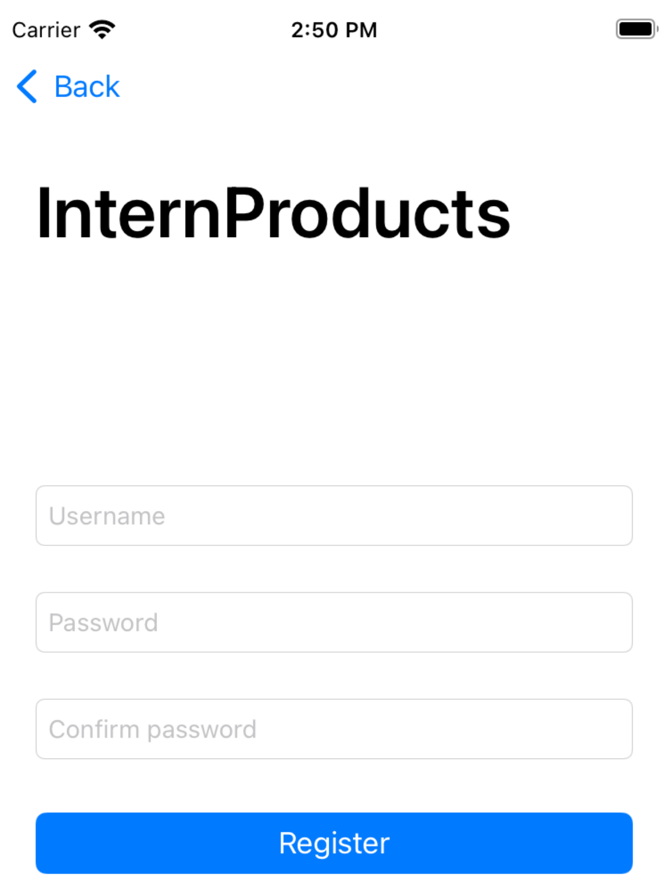

Carrier
1:56 PM
Intern Products
username
•••••••••
Login
Register

The general appearance of the “Register” screen should resemble what is displayed in the attached screenshot. In essence we have the following visual elements:

4) The title label, contents: “InternProducts”. You may pin it at 25 points spacing top, 20 points spacing to leading to the container view. Font: System Semibold 40

5) Username, password, password confirmation textfields. All of them pinned at 20 points leading and trailing to the container view. No height constraints, as they have intrinsic size. Vertically spaced with 8 points between them. Vertical positioning: relatively above the centerY of their container view at 10 points. The user must see the “Username” / “Password” / “Confirm password” placeholders and the passwords textfields must have secure text entry.

6) “Register” button. Pinned to 20 points trailing and leading to its container view. No height constraints as they have intrinsic size. Vertical positioning: Relatively below the centerY of the container view, at 10 points.
7) Upon pressing the “Register” button you must verify that the password and confirmation password are equal.
   a) If so, then proceed by making a register http call to the backend. Similarly to the “Login” screen, if the server replies with a “success: FAILED” case, display the error message in an alert. Otherwise proceed to display the products screen.
   b) If not, then display an appropriate error message under the confirmation password textField, with a red color like in the (“Register_password_error.mov”) video.


Carrier
2:50 PM
Back
InternProducts
Username
Password
Confirm password
Register

3.2) The 'Products' screen

The products screen contains a title label and a list of visual elements each displaying a particular product.

a) Title label has the "Products" text in it, font System Semibold 40. It is pinned at 25 points from top, centered horizontally in the container view
b) Scrollable list of product views, pinned to the leading, bottom and trailing edges of the containerView, with a vertical spacing of 50 points to the title label.

This screen is presented to the user after a successful login or registration. When the screen appears for the first time a network call must be made to the backend in order to retrieve and display the list of products to the user. You must pass in the loginToken in this call that you have received from the previous login or registration request. The network might be slow, so until the response is received the user should see an activity indicator spinning in the center of the screen. Once the list of products is parsed and ready to be displayed then hide or remove the activity indicator. See "LoginToProductsScreen.mov"

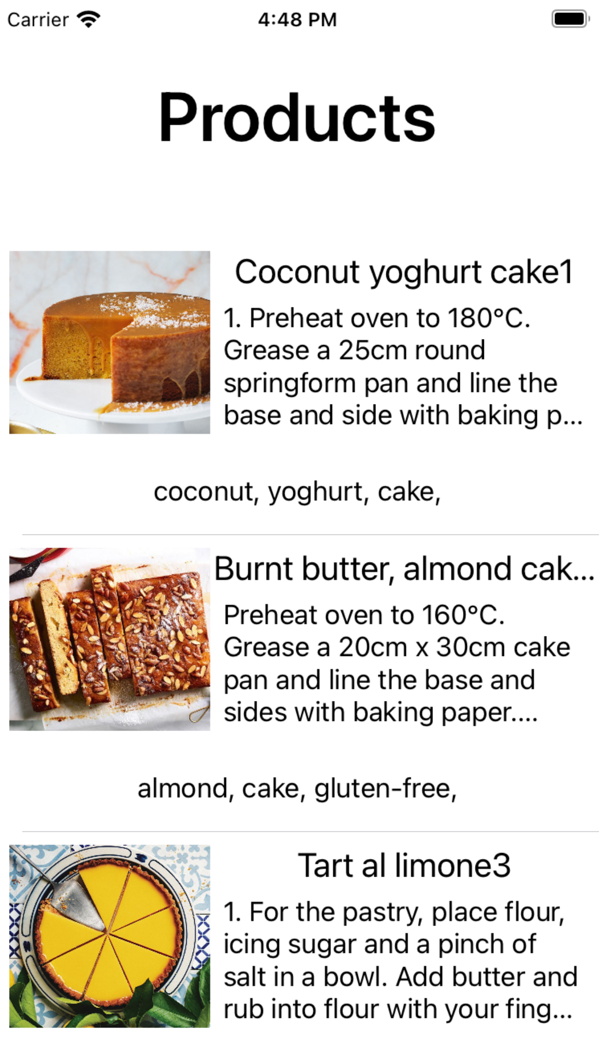
The general appearance of a product view in the list should match the following screenshot. It should have a height of 185 points when displayed in the list. In essence the visual elements are:

a) Product image, with fixed width=125, height=115 points. Positionally spaced to 8 points leading and top to the container view. Aspect filling mode

b) Product title label, font System 21. Vertical positioning: top-aligned to the product image. Horizontal positioning: 8 points spacing leading towards product image, 8 points spacing towards container view trailing.

c) Product description label, font System 14. Vertical positioning: 4 points top space to the title label, bottom-aligned to the products image. Horizontal positioning: 4 points leading space to products image trailing, 4 points trailing space to the container view trailing.

d) Tags label, font System 14. Pinned to the trailing, leading, bottom edges of the container view. Top space 8 points to the product image bottom. The product tags must be displayed here, with a “,” separator between them


Carrier
4:48 PM
Products
Coconut yoghurt cake1
1. Preheat oven to 180°C. Grease a 25cm round springform pan and line the base and side with baking p...
coconut, yoghurt, cake,
Burnt butter, almond cak...
Preheat oven to 160°C. Grease a 20cm x 30cm cake pan and line the base and sides with baking paper....
almond, cake, gluten-free,
Tart al limone3
1. For the pastry, place flour, icing sugar and a pinch of salt in a bowl. Add butter and rub into flour with your fing...

The order in which the elements appear in the list must be “most recent to least recent”, i.e. older products appear at the bottom, according to their date.

Tapping on an element should open up the “ProductDetails” screen for that particular product (see “ProductsToDetailsScreen.mov”)

3.3) The ‘ProductsDetails’ screen

This screen is presented once the user taps on a product item in the “Products” screen. It displays the details for that particular product in a more expanded manner.

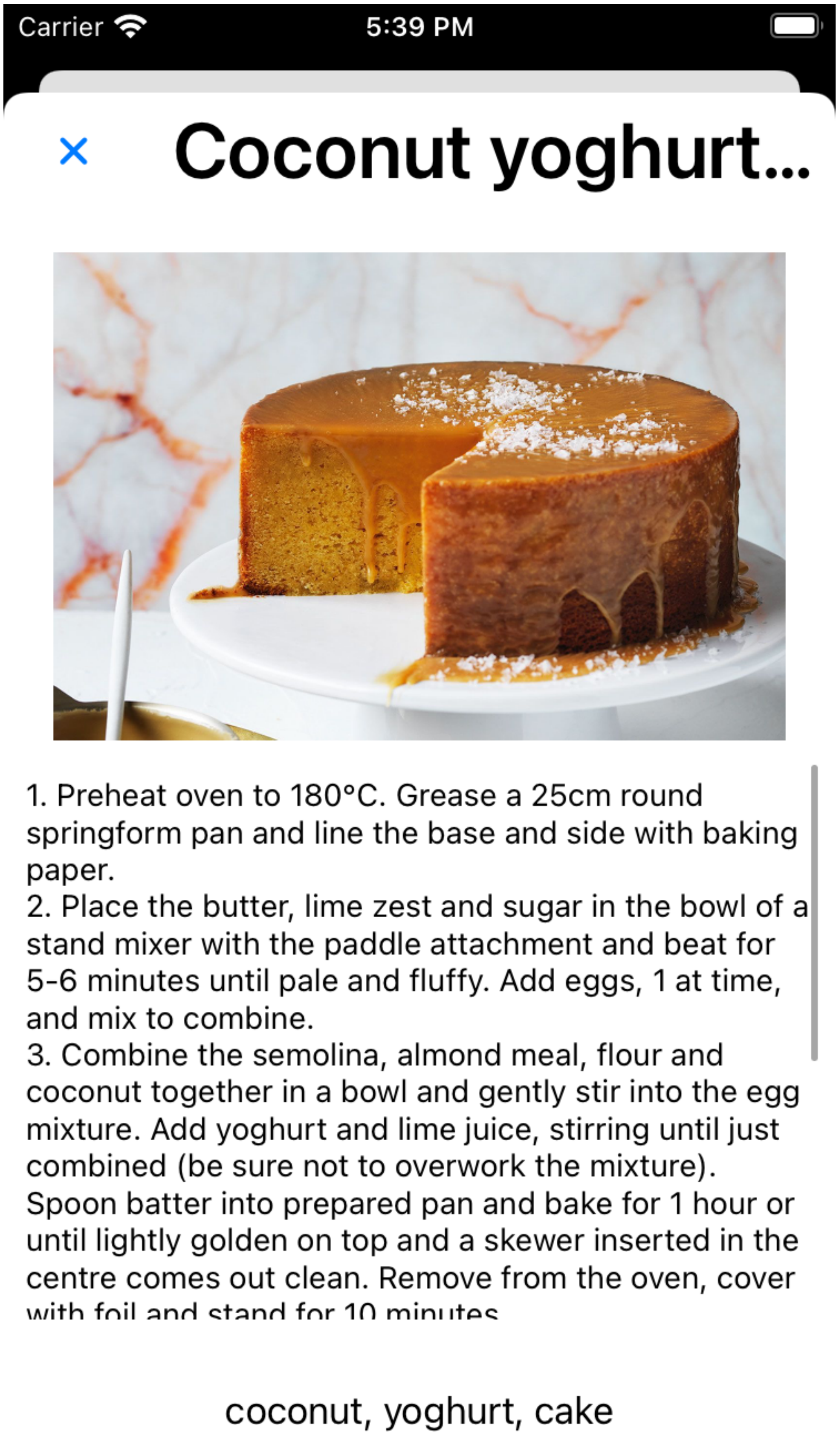
In essence the visual elements are:
Close button; Fixed size of 56x56 width and height, centered vertically to the product title label, 8 points leading spacing to the container view. When the user taps on this button the screen disappears and the user is returned to the product list screen.

Product title; Font System Semibold 35, 8 points top spacing to the container view, 8 points leading spacing to the close button.

Product image; Fixed height of 220 points, pinned to the leading and trailing edges of the container view, 25 points top spacing to the title.

Product description; Scrollable text area, 8 points leading and trailing spacing to its container view. 8 points top spacing to the product image and 8 points bottom spacing to the tags label.

Product tags label; Pinned to the leading, trailing and bottom edges of the container view, with a fixed height of 66 points. The product tags must be displayed here, with a “,” separator character.



Coconut yoghurt...

1. Preheat oven to 180°C. Grease a 25cm round springform pan and line the base and side with baking paper.
2. Place the butter, lime zest and sugar in the bowl of a stand mixer with the paddle attachment and beat for 5-6 minutes until pale and fluffy. Add eggs, 1 at time, and mix to combine.
3. Combine the semolina, almond meal, flour and coconut together in a bowl and gently stir into the egg mixture. Add yoghurt and lime juice, stirring until just combined (be sure not to overwork the mixture). Spoon batter into prepared pan and bake for 1 hour or until lightly golden on top and a skewer inserted in the centre comes out clean. Remove from the oven, cover with foil and stand for 10 minutes

coconut, yoghurt, cake

In essence, your app should be able to exhibit the same behaviour displayed in both the “HappyFlow_Login.mov” and “HappyFlow_Register.mov” videos, to be considered a minimum viable product. Once these criteria are met you may request the next document containing the bonus features to be implemented for extra credit points.

Regarding the transitions between screens, you may implement these in the manner you are most comfortable with. It does not matter to us whether you use navigationControllers or modal presentations, what is important is that the flow is respected as in the videos, i.e the user can go from screen A to screen B, or back to screen A from screen B, if it is required.

The key aspects that we prioritize when evaluating your projects will be in the following order:

1) Functionality:
    a) As stated previously, your app must be able to exhibit the same functional features as presented in the happy flow videos, with as less bugs/inconsistencies as possible. Between two students who have submitted their final projects, the one who has implemented more functionalities will have a higher grade.
2) UI:
    a) As stated previously, the UI needs not to be pixel perfect but it must match what is presented in the screenshots/ videos, as otherwise it will be hard for the user to access the functionalities. In particular as well, if you decide to manage the layout using AutoLayout we should as less constraint conflicts as possible.
3) Code quality:
    a) At the end of the day your task is to implement functionalities within reasonably designed UI and these are the main criteria that an app user will judge you for. However, between two students who have submitted their final project and implemented the set of functionalities + UI in the same level (i.e. as described, very few bugs, UI reasonably matches the design) the

student who has organized their code/architecture in a cleaner manner will have the higher grade. As stated in the beginning it is important that you follow the best practices that you know regarding class/struct organization, access control, SOLID principles etc.. when writing your project. In the end, if you aren't careful in this aspect it will be easier for bugs and inconsistencies to slip through, therefore ruining the 1) and 2) evaluations.

If you find yourself stuck or spending too much time implementing the backend communication in your code, you may skip it and instead implement the following:

- The login and register screens will skip directly into the product list screen only for the "endavaStudent" username and "schoolOfiOS" password. For any other user or password display the error message "Unknown user" in an alert.

- Then integrate the "productsResponse.json" file into your project and in the products list screen you should deserialize and display the product elements in there instead of making a network call to the backend. Use the following code snippet to read the data from the file:
    - ``

```
let path = Bundle.main.path(forResource: "productsResponse", ofType: "json")!
let data = try! Data(contentsOf: URL(fileURLWithPath: path))
```

``

If you do not succeed in creating an image from the textual representation that you receive from the backend or you find yourself spending too much time on it then simply put a background color ".red" into the "UIImageView" wherever the image was supposed to be shown and carry on with the next tasks.

BONUS FEATURES:

[will be completed once the student has finished their MVP]