THEMEDIMENSION

Mobile Native Shopping iOS Documentation v1.0

CONFIDENTIAL

ABOUT

Ivory is a mobile application that shows the products sold by a shop to its clients online. The app takes a JSON feed of products and displays it to its users in an easy and interactive way, allowing them to choose their favorites products and place orders according to their needs.

INSTALLATION

How to run the project:

If you don't have cocoapods installed on your mac, please follow the next steps:

- Type *Terminal* in Search, open it and type the following command:
  `sudo gem install cocoapods`

After you install cocoapods, follow the next steps:

- Type *Terminal* in Search, open it and type *cd*.
- Drag and drop the *IvoryiOS* folder in Terminal and press Enter.
- Type *pod install*, press Enter and wait until all the frameworks are downloaded.
- Go to *IvoryiOS* folder of the project and open *IvoryiOS.xcworkspace*.

Note: If you want to run the project on a device (and not on simulator) you will have to create a developer account on https://developer.apple.com and follow the steps from Apple Documentation.

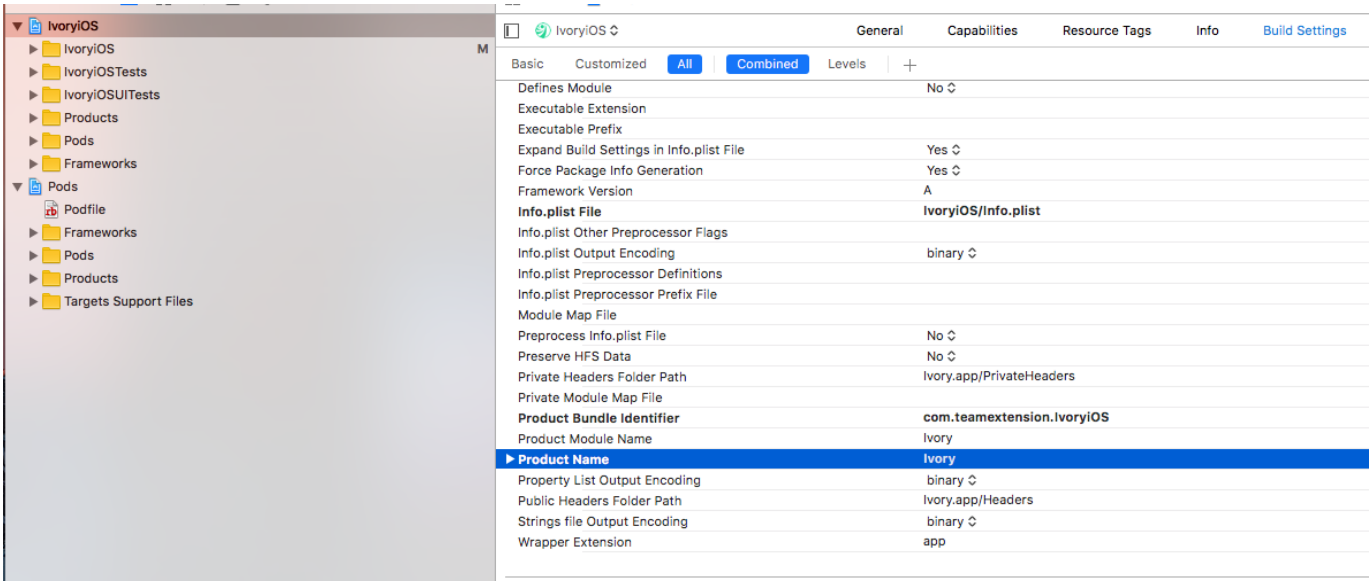
CUSTOMIZATION

BASIC

How to change the app's name:

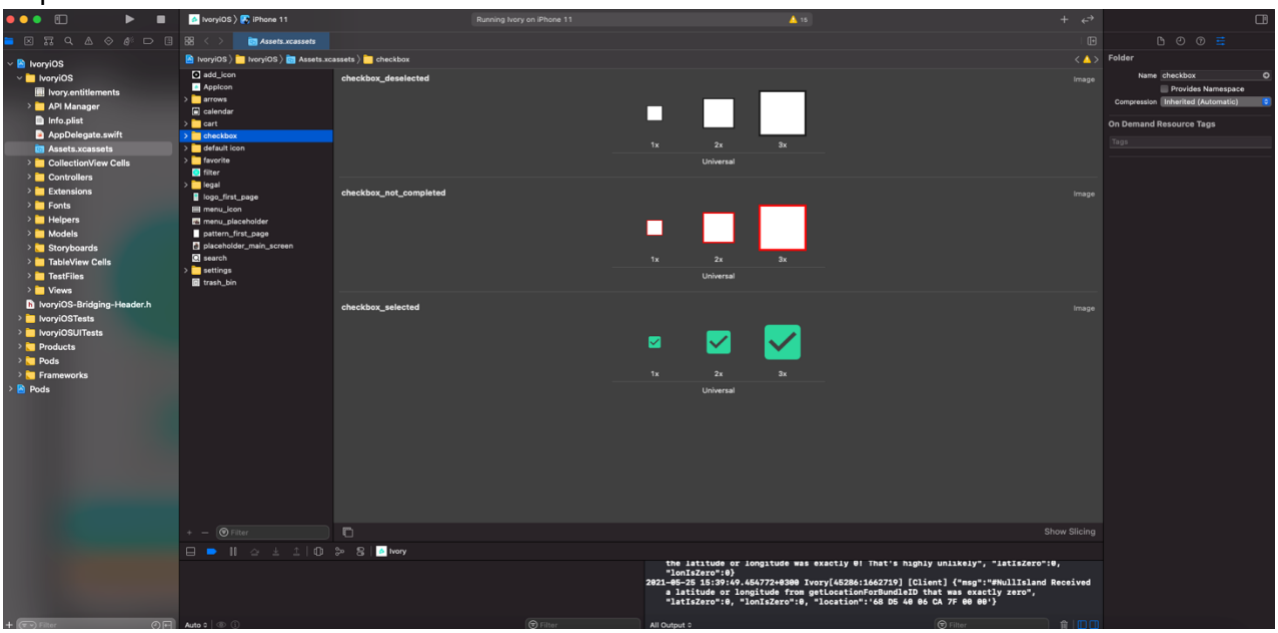
To change the application's name you have to select the root of the project (IvoryiOS), select Build Settings from the upper menu and search for Product Name. There you can change the text "Ivory" with the name of your application.

Note: Make sure that the filter form the upper left is set to "All" and "Combined".



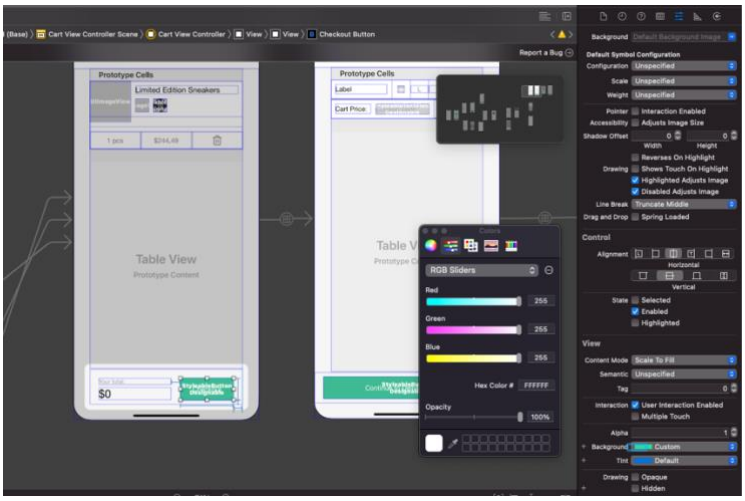
How to change images & app's icon:

All the images from the project can be found in Assets.xcassets. Feel free to change any of it, but make sure that you don't change the name of the image packages and that you add the required sizes for iOS (@1x, @2x, @3x).
To change the app's icon you have to go to Assets.xcassets in AppIcon set and add all the required sizes.



How to change colors:

All the colors used in the code can be found in *Extensions/Basic/UIColorExtension.swift* file. You can change the hexcode of any color to be the one that suits you the best. There are also colors set from *Storyboards/Main.storyboard*. To change a color from storyboard, click on the element, notice the menu from the right and select *Attributes inspector*. There you can change the colors, the fonts and other properties of an element.



Ex:
Change color in UIColorExtension.swift:

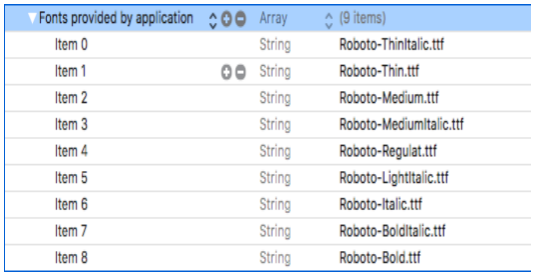
We will change the color *"ivoryGreen"* (#2cdca0) with a warmer color *"orangeColor"* (#f4af09). We just have to change the value stored at *"ivoryGreen"* with the value *#f4af09* . Now, anywhere the app uses the former green color, will now be displayed a nice orange color.

How to change texts:

All the texts used in the app can be found in the code written between “<<text here>>”. Also, there are some texts set from *Storyboards/Main.storyboard*. You can change any text with one that suits you better.

If you want to change the font of a text you have to go to *Helpers/Constants.swift* and change the name of the font that you wish to change inside *BasicFonts*. This will change only the fonts that are set from the code. Most of the fonts are set in *Main.storyboard* and you can change it the same way that you can change the colors from storyboard (mentioned above).

Note: Please be aware if you want to add a new font that can’t be found in the default list of fonts provided by Apple, you have to add the font with the extension .ttf inside the *Fonts* folder (Ex: Roboto-Medium.ttf) and also add it as a new item in *Fonts provided by application* property from *Info.plist*.



Fonts provided by application	Array	(9 items)
Item 0	String	Roboto-ThinItalic.ttf
Item 1	String	Roboto-Thin.ttf
Item 2	String	Roboto-Medium.ttf
Item 3	String	Roboto-MediumItalic.ttf
Item 4	String	Roboto-Regular.ttf
Item 5	String	Roboto-LightItalic.ttf
Item 6	String	Roboto-Italic.ttf
Item 7	String	Roboto-BoldItalic.ttf
Item 8	String	Roboto-Bold.ttf

How to change the Legal Link & Terms and Conditions Link:

```
func setWebView(){
    if let url = URL(string: "https://www.google.ro"){
        let urlRequest = URLRequest(url: url)
        self.webView.loadRequest(urlRequest)
    }
}
```

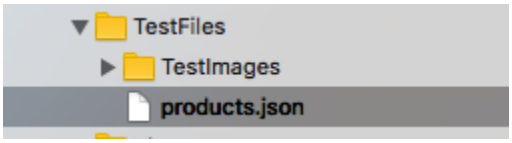
The “Legal” section is meant to send the user to the privacy policy of the app. To add your own privacy link navigate to *Controllers/Menu/LegalViewController.swift* and inside that class change the link "https://www.google.ro" with your own legal link.

The “Terms and conditions” section is meant to send the user to the terms and conditions of the app. To add your own terms and conditions link navigate to *Controllers/Login/TermsAndConditionsViewController.swift* and follow the same steps as specified above.

How to change the source data displayed in the app:

Note: Currently, the application has demo data for testing so all data from the application is saved locally inside the app.

The app supports source data to be read from a json file located in the *TestFiles/products.json* and it follows the JSON format ({ "productList": [jsonProduct, ..., jsonProduct] }). The data must be inserted in the file *JSON* and it reads the list of products in a JSON format as it follows:



Product:

```
{
        "productName":"WomenClothes",
        "productId":1,
        "productColor":"#00aabb",
        "productPrice":20,
        "sizes":
            [
              {
                "id":1,
                "name":"XS",
                "quantity":0
              },
            .....
              {
                "id":5,
                "name":"XL",
                "quantity":25
              },

              {
                "id":6,
```

```
                "name":"XXL",
                "quantity":25
              }
            ],
        "productAddedDate":"05-11-2015",
        "productGender":"Women",
        "productCategory":"Clothes",
        "productImage":"blouse_blue.png",
        "sibilings": [2, 3, 4, 5]
},
…..
```

Note 1: The example above represents a single product in ***JSON*** format. In order for the app to work as expected, you should follow the instructions as stated above.
Note 2: If you have a server that you what the app to use, please check the server-side communication documentation further down this document.
Note 3 : In order to change all the source data at once, be sure to follow the following instructions:
1. All the data should be enclosed between *{* and *}* .

Ex:
```
{
    ...data...
}
```
2. The first line between the *{}* symbols must be *"productList":* and immediately after it, a block of products in JSON format should follow between the square brackets *[...]* . You must be sure to close all the brackets used and state where a new product is entered, using the ' , ' symbol.

Ex:
```
{
    "productList":
    [
        {
            Product1InJSONFormat
        },
        {
            Product2InJSONFormat
        },
        ...
        {
            ProductNInJSONFormat
        }
    ]
}
```

Note: Make sure that there is a *.png* file with the same name available for the *"productImage:..."* inside the *TestFiels/TestImages* folder . In case not, the app will load its data successfully, but the image of the product will be a default one.

For further information about the JSON format and how to use it, please follow the link below:
http://json.org/example.html

If you already have knowledge about the format stated above, here are some helpful links to format the data using JSON: JSONLint, JSON Formater & Validator and JSON Viewer

How to change the currency for charging users:

In order to change the currency that the app uses to charge its users, you will need to change the string value of the constant *CURRENCY_UNIT* from *$* to any other currency that you would like to use. The constant is located in *Helpers/Constants.swift*.

SERVER

How to link the application to a server:

In order for the app to fulfill its objectives, it is recommended that a server connection should be created so that the user can enjoy the full Ivory experience and real-time communication to be enabled. To help you achieve this, the app comes with helping Swift classes designed to represent the link between app, server and any necessary databases.

You can link your server using the constant strings stored in the *APIManager/APIRouter.swift* class.

Note that the *APIRouter* is just the pattern for server-side communication, and any additional implementation needed must be developed in order for the app to work accordingly.

How to change the URL to an existing server:

In order to link-up the app to an existing server, you need to navigate to *APIManager/APIRouter.swift* and replace the *basePath* string to the URL used by the server.

Ex: *static var basePath: String{*
return "https://secure-server-domain.web-extension/ "
}

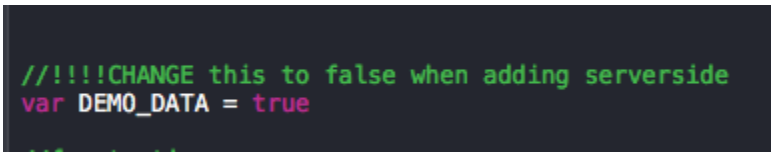
Now, if your server does not need to suffer any additional editing, and neither does the app itself, your users should be able to correctly send requests and get responses from the server.

Note that the app might need extra configuring in order to properly sustain server-side communication.

For different API calls (such as registration, login, etc.) the app will need to be provided with the correct *basePath* and url variations .

basePath: The domain of your server (followed by a "/")
headers: Contains the required headers to make a server request (such as “apikey” which will make all the server requests secure).
imagesPath: The extension needed in order to reach the image resource storage location of your server
requestPath: The extension needed in order to reach the data JSON to use as source data.
bodyParameters: Here is how the parameters that are sended to the server should be format. It is specially used for POST requests.
headerParameters: Contains all parameters needed to be send in the header for a server request. It is specially used for GET requests.

Note1: You will need to provide the URL to the endpoint that you link to the app.
Note2: After all the setup with server side, you will need to change the DEMO_DATA constant from *Helpers/Constants.swift* to be false.



STRIPE

How to enable Stripe:

In order to enable Stripe as the main card payment method, you have to navigate to *Helpers/Constants.swift* and change the *STRIPE_AVAILABLE* flag to true. Once you do that, the app will automatically change its card input method accordingly, and the layout will change at the same time.

What is more, in order for the app to correctly finish payment requests, an online server must be added, else the app will only get a failure response.

Note that you must change the *SRIPE_PUBLISHABLE_KEY* to your own keys provided by Stripe. This constant is located in *Helpers/Constants.swift*.

Basic workflow of the payment method:

- The app prompts the user with the Stripe CardInputWidget.
- The user enters the card credentials into the widget.
- The app sends a request to the Stripe servers to convert the sensitive data into a token.
- The Stripe servers return the token to the app.
- The app sends the token further to its server for any other processing.

For more details about the Stripe API and how it can be used, please follow the reference below:
https://stripe.com/docs/mobile/ios
Note that you can also find useful information about how the app manages payments in the source code.

PAYPAL

How to enable PayPal:

In order to activate the PayPal Payments inside the app, you need to navigate to *Helpers/Constants.swift* and change the value for the *PAYPAL_AVAILABLE* from **false** to **true**. This will make the app load the button (inside the Checkout screen) that enables the user to create an order using his / her PayPal account.

Note1: You must change the PAYPAL_CLIENT_ID_PRODUCTION *& PAYPAL_CLIENT_ID_SANDBOX* with your own keys provided by PayPal.

Note2: Currently, PayPal is set to work in sandbox environment so you can test it. If you will launch the app make sure that you change

***PayPalMobile.preconnect(withEnvironment: PayPalEnvironmentSandbox)* to *PayPalMobile.preconnect(withEnvironment: PayPalEnvironmentProduction)* located in *Controllers/Menu/Cart/PaymentViewController.swift*.**

Basic workflow of the payment method:

- The app prompts the user with the PayPal UI.
- The user enters PayPal account credentials.
- The API processes the PayPal account and returns a PaymentConfirmation object to the app.
- The app sends the confirmation to the server for further processing.

Note that in order for the app to correctly finish payment requests, an online server must be added, else the app will only get a failure response.

For further details about the PayPal API integration and PayPal documentation, please refer to the link below:
PayPal iOS SDK Documentation

Logging in and Register

(This section only applies to the Demo version -> no backend)

The app supports users to login with an existing account, create a new account, and choose to skip the registration screen. In order to work accordingly to expectations, the app offers a demo version (with no backend communication) which simulates the server-side communication.

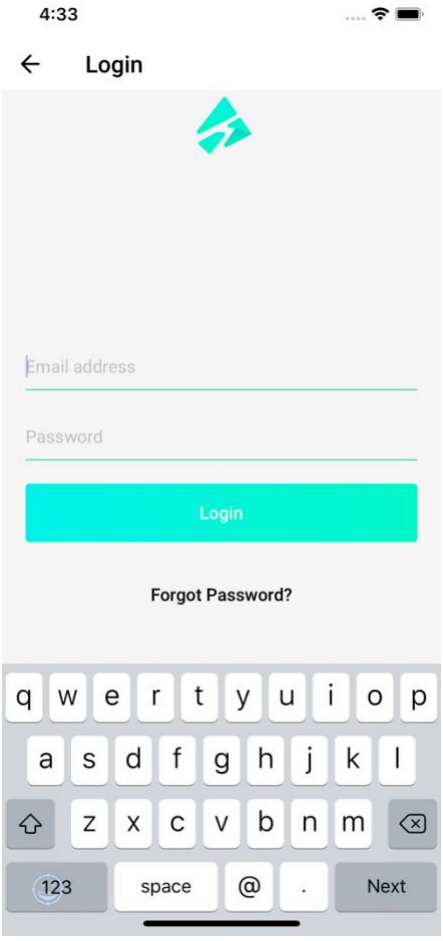
Note that once a backend is added, the following guideline will not be taken into consideration.

Login: The user tries to login with an already existing account.

As the app does not feature a backend to respond to the request, the existing accounts are stored in an external file named ***users.json*** which contains all the accounts created by the app on the running device.

Once a Login action happens, the email and password are tested to see if they appear in the external file. In case of a positive result, all the data used by the user to create that account will be retrieved and used across the app.

Note: The code for login screen can be found in *Controllers/Login/LoginViewController.swift* and *Extensions/Controllers Extensions/LoginViewControllerExtension.swift*.



Forgot Password: The user wants to reset the password for an account.
For the DEMO part this feature is not working, as the app doesn't send an email.

The functions for the server side are created and if they are setup accordingly to the current documentation it will work with no problems.

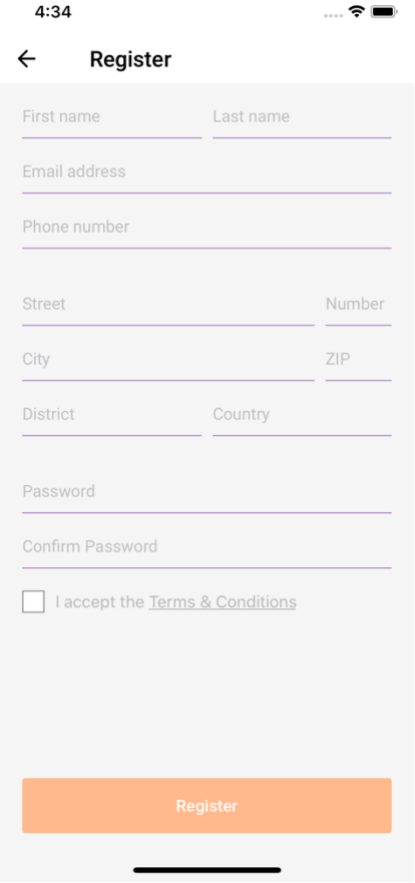
Note: The code for “Password Recovery” screen can be found in *Controllers/Login/PasswordRecoveryViewController.swift* and *Extensions/Controllers Extensions/PasswordRecoveryViewControllerExtension.swift*.

Register: The user creates a new account.

In order to create a new account, the data that the user wants to use in order to register inside the app will also be stored in the ***users.json*** external file.

The workflow is a little different than in case of a Login action, as the email is searched inside the file and in case it is found, the app will let the new user know that the email has already been used. Moreover, in case the app can create the new user, it will write the user's data inside the file and will take him to the home screen as a logged in user.

Note: The code for register screen can be found in *Controllers/Login/RegiserViewController.swift* and *Extensions/Controllers Extensions/RegiserViewControllerExtension.swift*.



Skip registration: The user skips any type of authentication method using the Skip button.

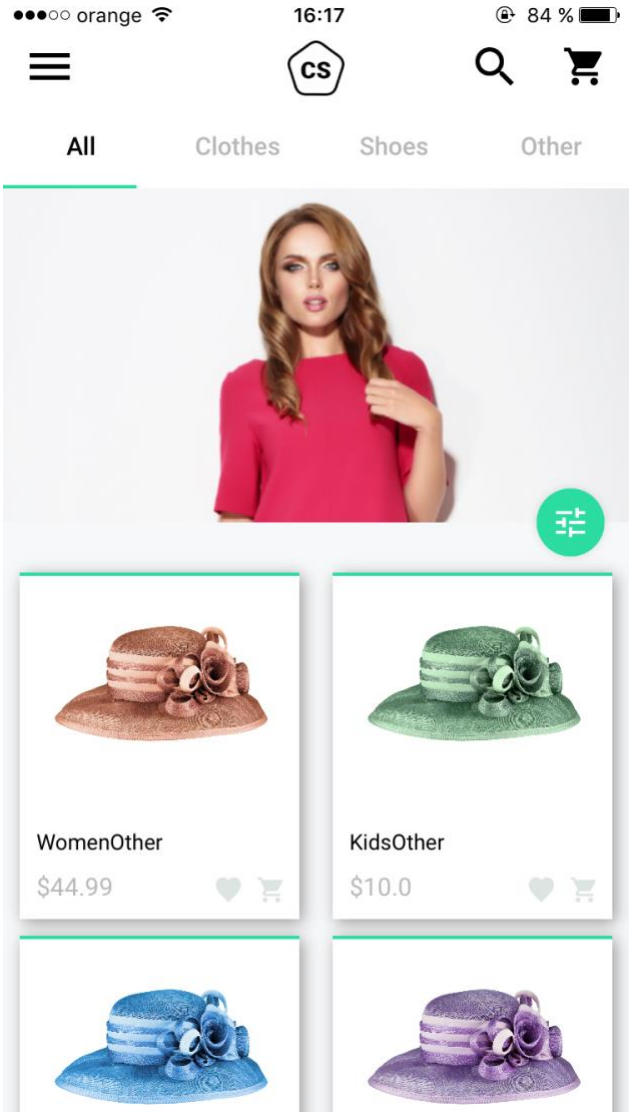
In case of a Skip action, the app will use default data to instantiate an anonymous user with the ID -1 . This means that, in case a backend is used, the server will be correctly provided with the necessary data in order to make orders (if it is the case); all other fields are left empty.

For the user to make orders, he will need to provide all the data needed (Address, email, phone, etc.) later in the app. (if it is the case)

USAGE

The main page shows all the products items from the feed.

Note: The code for main page can be found in *Controllers/MainViewController.swift* and *Extensions/Controllers Extensions/MainViewControllerExtension.swift.*



How to add products to favorites:

You can add your products to favorites by pressing the heart shaped icon located next to the product's price, or by entering the product's details screen, where you will find the same icon on the top of the screen.

Note1: The code for product's details screen can be found in *Controllers/ProductViewController.swift* and *Extensions/Controllers Extensions/ProductViewControllerExtension.swift*

Note2: If there is no user logged in, the option to add a product to favorite won't be available as it can't be associated with a user.

How to add products to cart:

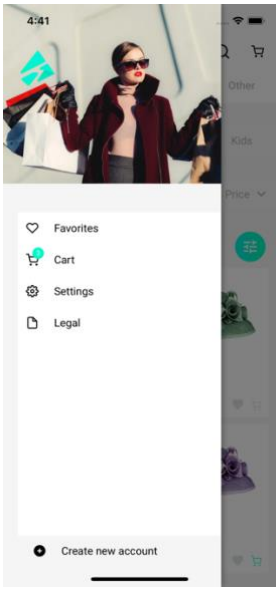
You can add products to cart by accessing the product's details screen. To add items to cart, you need to first select a size. You can modify the number of products to add to cart by pressing the "+" and "-" symbols located in the screen.

Note: The code for product's details screen can be found in *Controllers/ProductViewController.swift* and *Extensions/Controllers Extensions/ProductViewControllerExtension.swift*

How to search products:

You can use the search function to find articles that match your search. Access the search function by clicking the search icon on top of the page.

How to access the menu:



You can access the menu by clicking the menu button. You can dismiss the menu by clicking outside of the menu or by dragging the right margin in the left. In the top of the menu you will see a list from where you can access the Favorite products screen, the Cart screen, the Settings screen or the Legal screen.

Note: The code for menu screen can be found in *Controllers/Menu/MenuViewController.swift* and *Extensions/Controllers Extensions/MenuViewControllerExtension.swift*

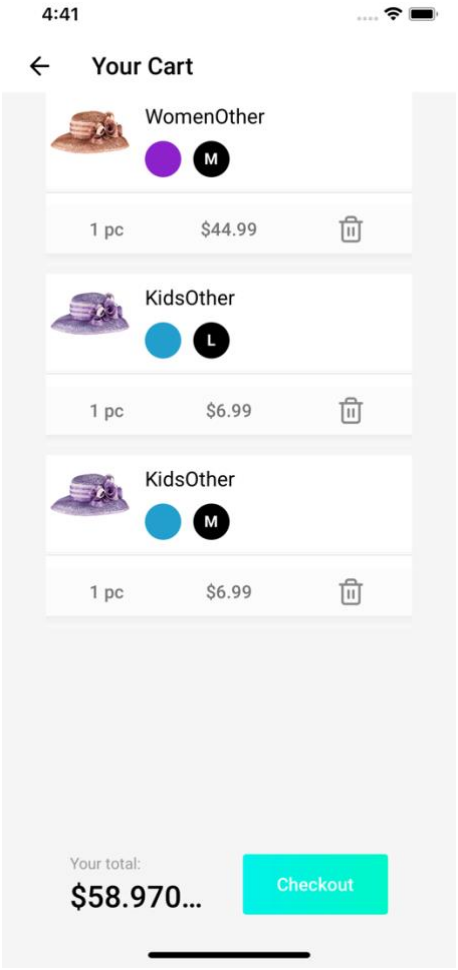
How to remove articles from “Favorites”:

You can modify the list of favorite products by clicking heart icon from the respective product. The icon’s color will change accordingly. This action will remove the article from your list.

How to remove articles from “Cart”:

You can modify the cart’s content by clicking the remove icon in the cart screen.

Note: The code for cart screen can be found in *Controllers/Menu/Cart/CartViewController.swift* and *Extensions/Controllers Extensions/CartViewControllerExtension.swift*



How to access the settings:

You can access the settings by clicking on the “Settings” section from the menu. The settings will contain all the user information completed at register.

Note: If the user is not logged in the Settings section is not available.

How to modify the settings:

You can modify the settings by changing the data saved in the respective fields and then clicking the button “SAVE”.

Note: The code for settings screen can be found in *Controllers/Menu/SettingsViewController.swift* and *Extensions/Controllers Extensions/SettingsViewControllerExtension.swift*

GIVE FEEDBACK

You can give us feedback in order to help improve and create higher quality template and applications at:

contact@themedimension.com