# Activity type 1:

## Wireframes and prototypes

(Describe the tasks and the conditions)

Eg. The owner of the shop asked me to create him an eshop

For my internship project, my manager asked me to redesign and modernize his eshop. He also asked me to change the technology used as it was written in Wordpress and it is bloated with the undesired result of being slow or even stalling for tens of seconds in some cases. Also he was concerned with lowering the costs of maintenance for the eshop.

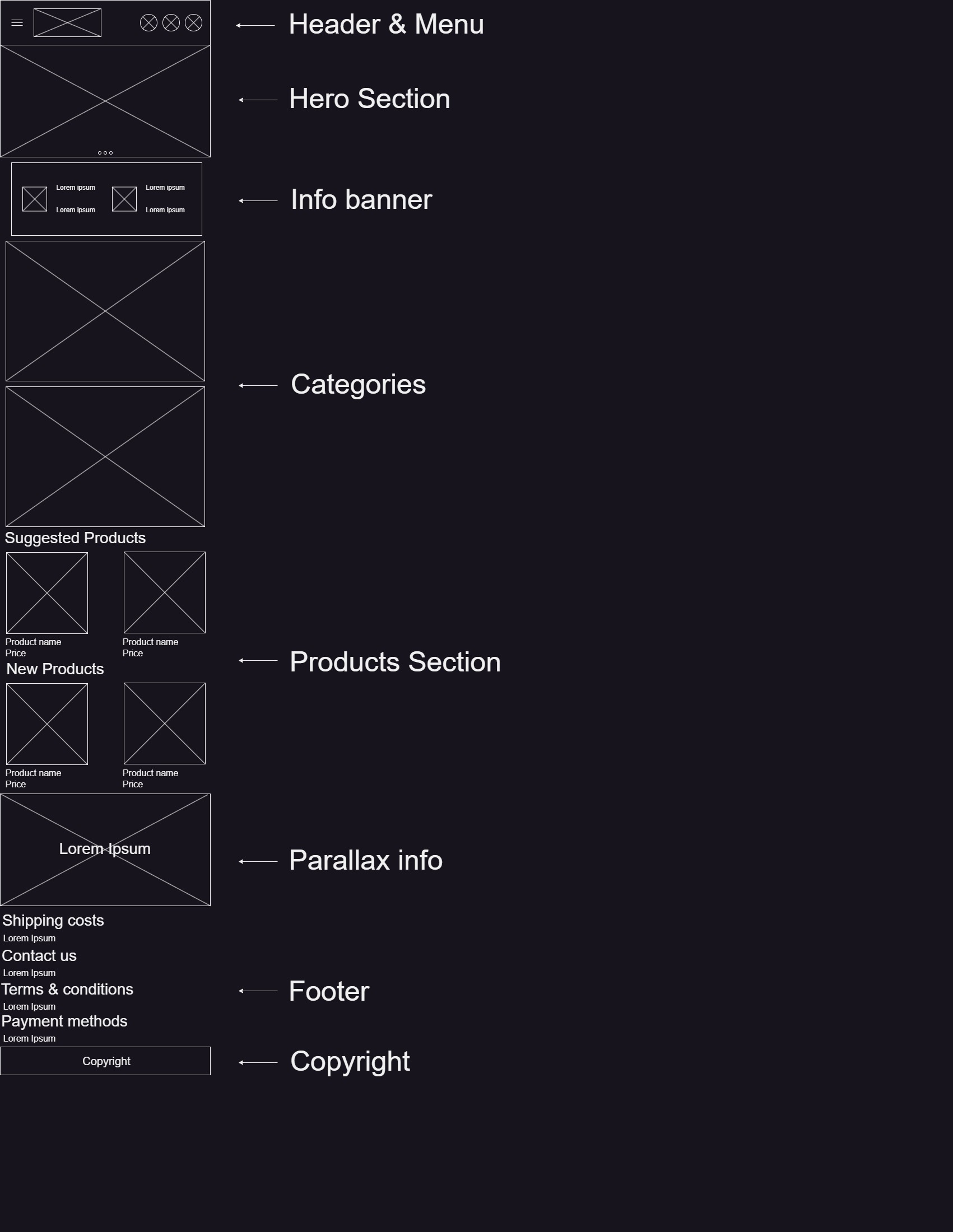
After discussing with him, I suggested the technology stack of directus for the backend and nextjs for the frontend. Since I have worked in a few projects with nextjs and have read good reviews regarding directus, we decided to go in that direction.

My manager asked for the project to have an intuitive design for the users’ navigation as well as an intuitive interface for the administration. Directus offers a very straightforward management of the database, and the data stored in it. After showcasing an example, he was convinced.

1. Wireframes

Immediately after we agreed on the desired technology, I started designing the modernized version of the website. First, I created the wireframes. They act as a guide for later development and design of the prototypes. Since modern design is mobile first in most cases, with some exceptions that do not apply here, I started with the wireframes of the mobile view (small screens). For the realization of the wireframes, I used the platform called draw.io since it is a tool I am familiar with.

Wireframe of the homepage for mobiles:



In this wireframe we can see the distinct separation of the view in three sections: the top / header section, the main / content section and the bottom / footer section.

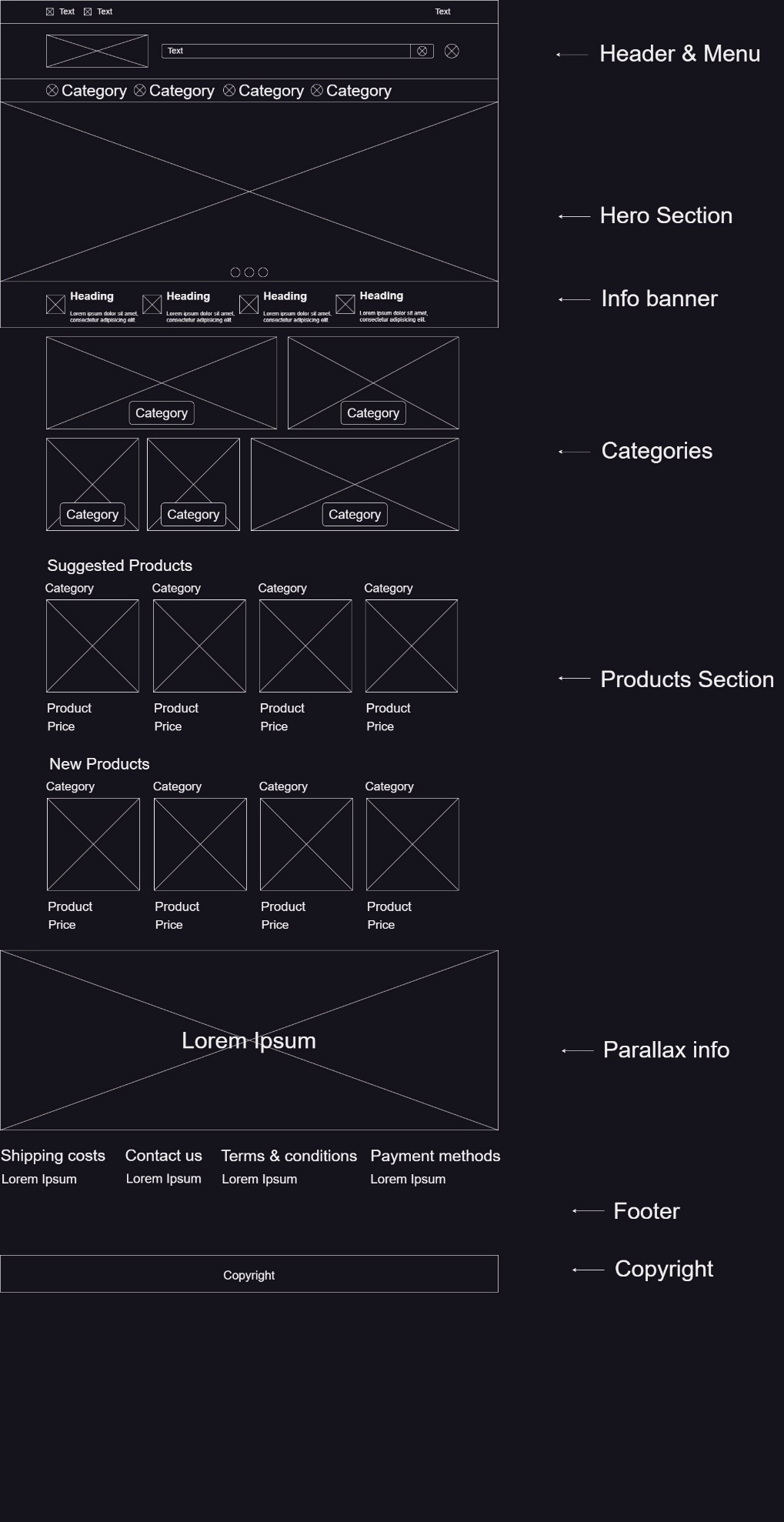
The header section serves to inform visitors and customers of the available links through the menu, any additional information required by the manager and the categories of the eshop.

The main section displays two carousels of products: suggested products and new products added to the shop.

The footer serves as submenu with additional links to legal related pages as well as information regarding payment and the copyright of the site.

The header section and footer section persist across the entire website with minor changes to the header section in order to fit the screen according to the content displayed each time.

Wireframe of the home page for desktop view:

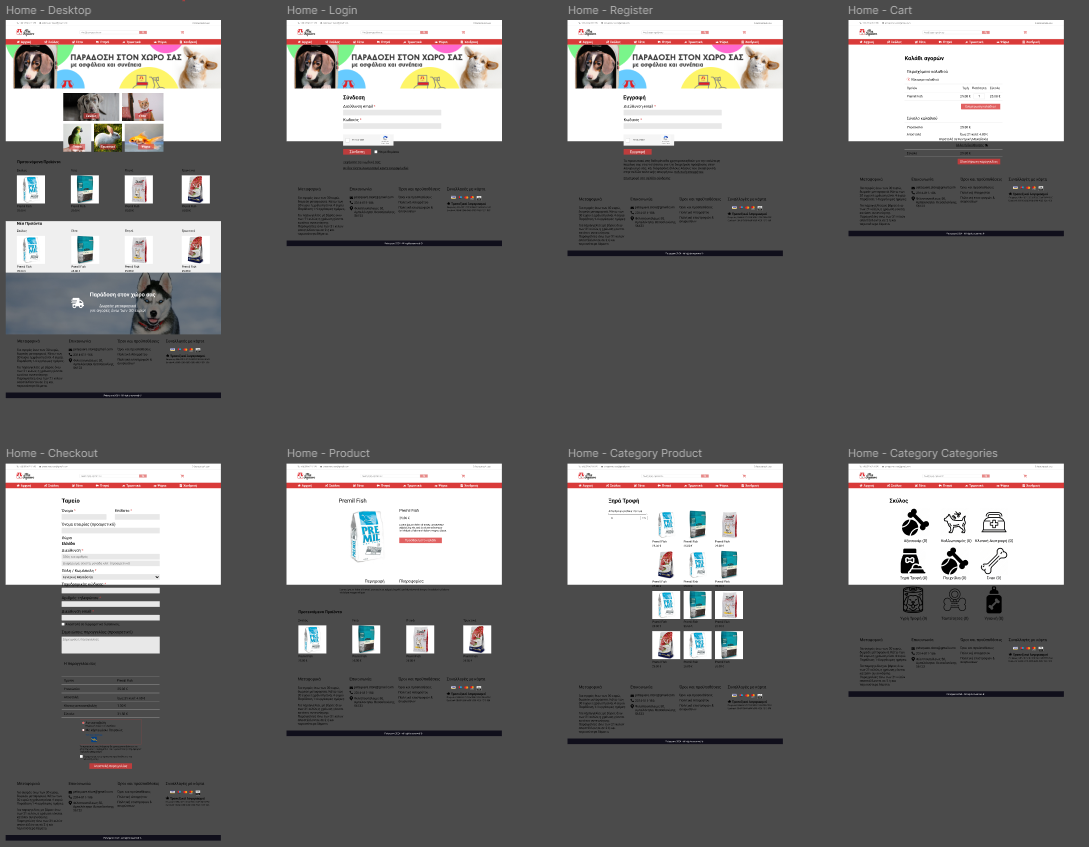
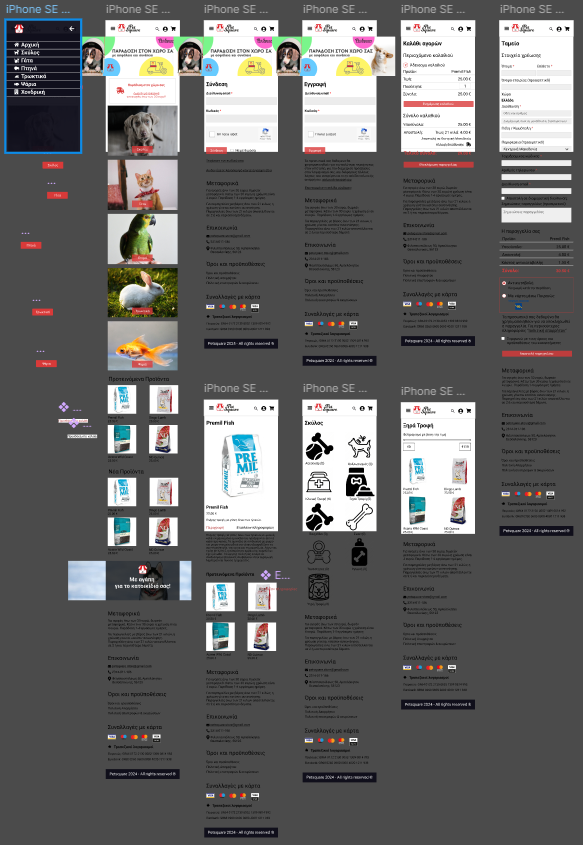


Similar to the design for mobile view, the design is divided into three sections: header, main and footer. The main difference here is the size of images. As larger screens can fit larger graphics, so the images are enlarged, without any loss in quality, with the addition of grid style submenu of the categories. The header and footer sections persist also with minor changes likewise.

Prototypes:

For the prototypes I used figma since it has a free tier and offers a large number of tools.

I created a functional prototype for both mobile and desktop view that imitates the effects and design of the final product. I followed strictly the instructions of the manager and the wireframe designs to create these prototypes.



## Interface with demo data

(Describe the actual work on the frontend)

Include photos of the code and the result

Frontend design with hardcoded / demo data:

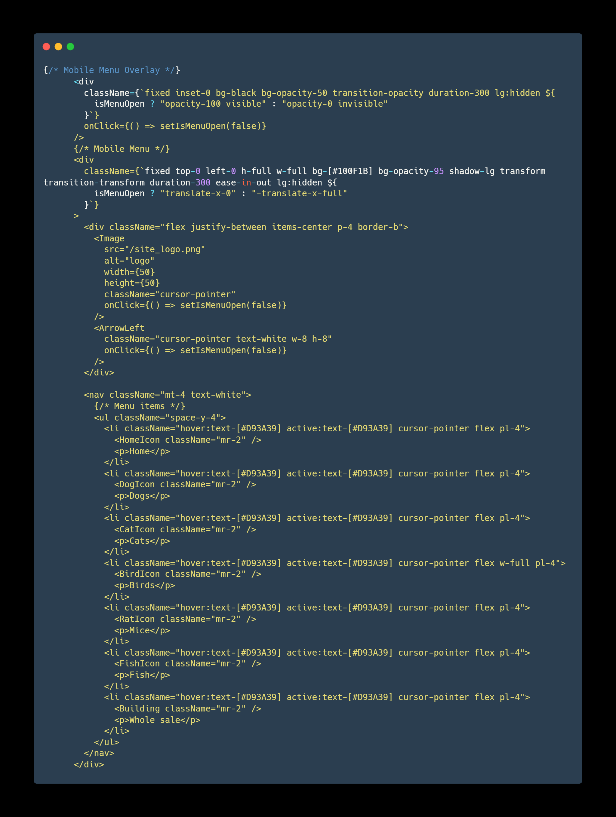
Following the wireframes and prototypes, I created the frontend. In the examples below we can see the code from the Header component of the nextjs project.

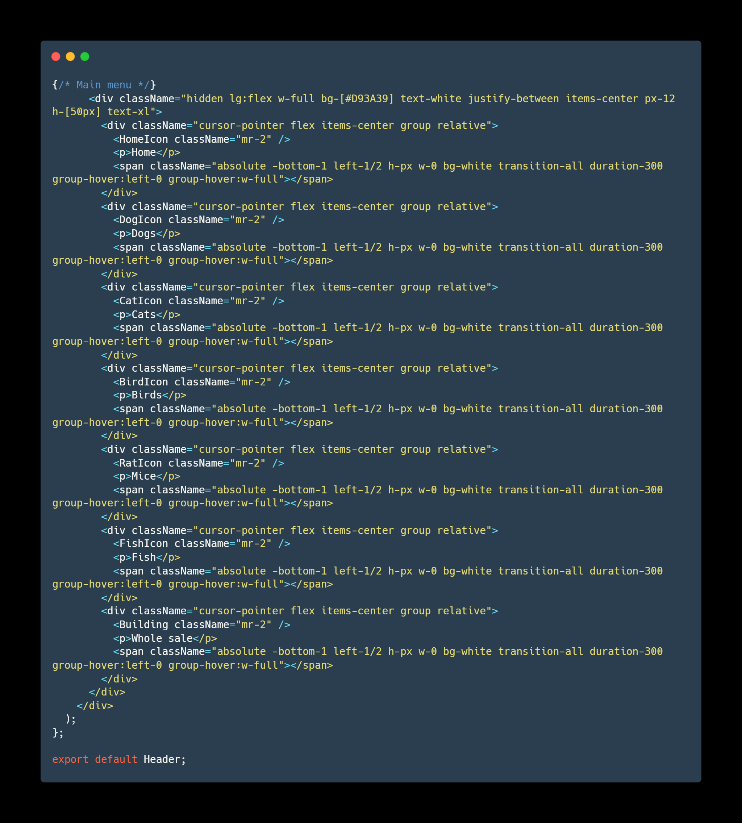
At the very top of the file, I import all the necessary libraries and utilities that are needed for this component. Since it is using React hooks, it needs to be rendered client side and therefore at the first line of the code I define “use client”. The rest of the imports are the mentioned hooks and the exported object React from the react library, icons from lucide-react, the component Image of next and my SearchComponent.

At the definition of the component, I create two variables that require to use state as they will store temporary information based on user interactions. I call the useEffect hook so React will not keep re-render the page when the values of the variables change which in turn will cause the variables to be reset and I inject the variables themselves in the dependency array so it keeps track of changes made to the particular variables.

In the rendering of the component, I create the wrapping element div for the entire component which will have display flex with direction of column, and it will expand to the whole width of the viewport. Then, I define the element that will hold the upper portion of the menu. It will also be of display flex but this time the elements inside it will have the direction of row, with content justification of between, leaving even spaces between the elements and item justification of center, meaning the element will be centered. When the minimum screen changes to the breakpoint of large screens (@media (min-width: 1024px)) the direction changes to column, putting the items inside as a vertical stack. On the top level, by default, at screens with max-width less than 1024px the highest menu is hidden. On the breakpoint “lg” the element has a display flex again, the main change is its height which is now set to 50px. The main theme of the component is the use of display flex which handles alignment of elements in one dimension, either horizontally or vertically.

The handling of the breakpoints is left to the standardized values set by tailwind. We have to keep in mind that tailwind has a “mobile-first” approach meaning its queries are of “@media (min-width: …px) and therefore we design starting from smaller screens.

Another part that is worth mentioning is that of the mobile menu in contrast to the main menu.



On smaller screens the main menu is hidden. In its place we see a hamburger icon which upon interaction forces the mobile main that is offset to the left to come in view with a smooth transition. The menu itself contains the same categories as the main menu. The wrapping element of the mobile menu blocks vertical navigation of the site in order to avoid confusion of the user.

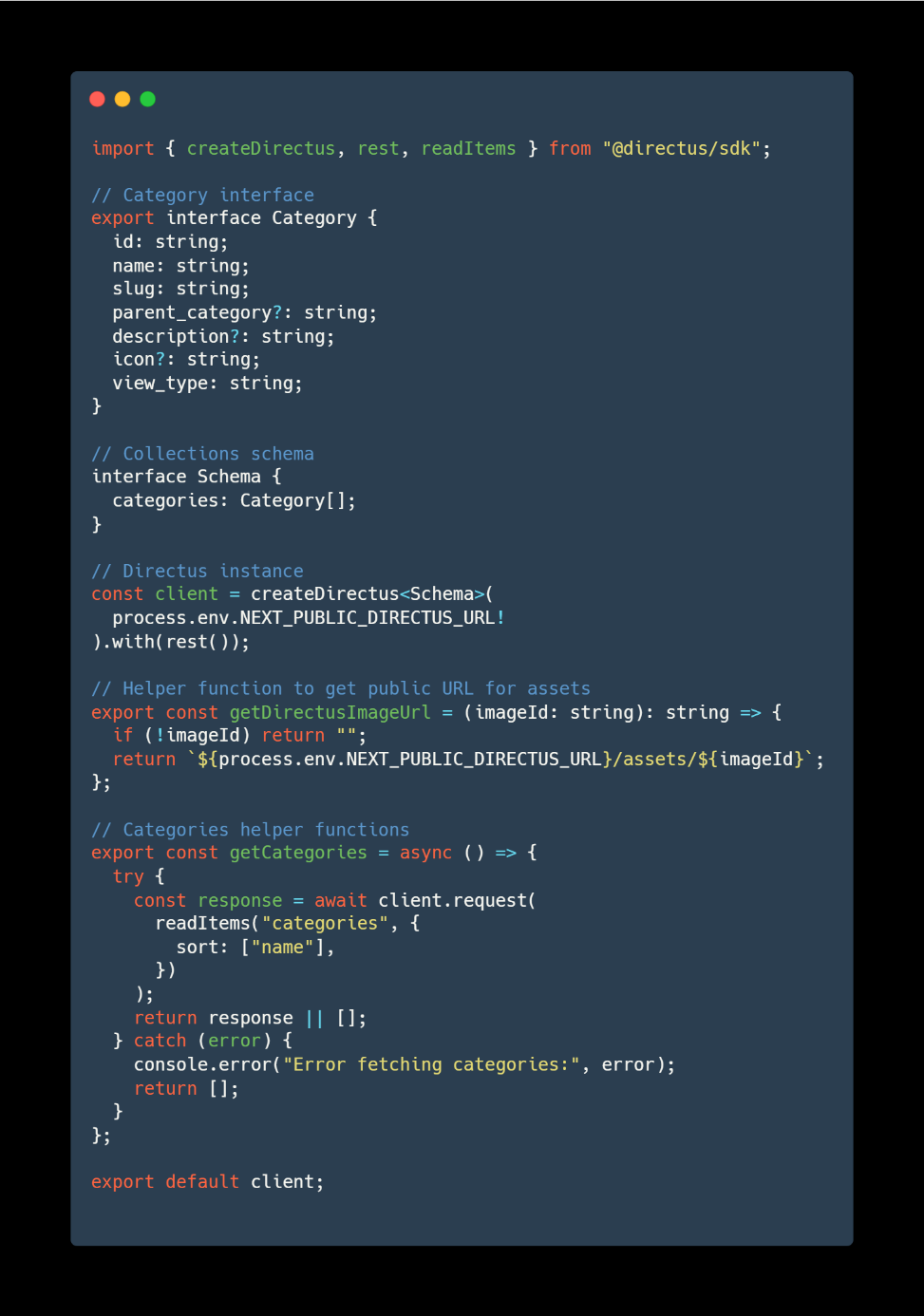
Upon larger screens, we see that the mobile menu and the hamburger icon are hidden. In their place we see the actual main menu which itself has smooth effects when interacting with a mouse (hover, click, area leave).

## Dynamic data

(Describe the integration of data from directus)

In order for my frontend (Next.js) to get data from the backend (Directus), I had to make changes to my frontend to accommodate the Directus api. I had to start by adding the information in regards to my frontend in the docker-compose.yml file.

Then, I had to install the npm package that contains the directus sdk by using the command `npm install @directus/sdk`. After the installation was finished, I created a new file named directus.ts in my libraries folder in my frontend project. In the file, I imported the necessary methods, created an interface for the categories, as those were the first data that I integrated to the site, followed by a schema interface describing the structure of the database. In this case, as I only used the Category collection, I simply included that table. I created an instance of directus and some helper functions in order to get the data I required from my database and finally I exported the directus instance.



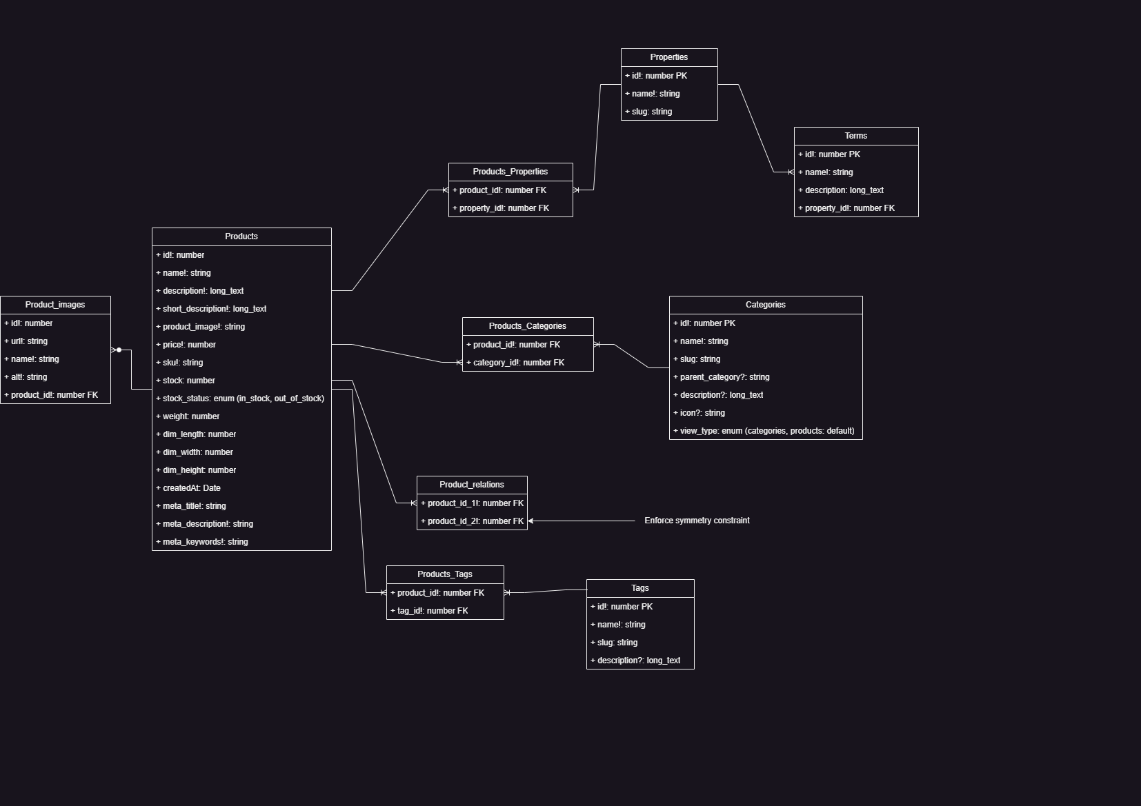
Since I wanted to show my categories in my menu, in my Header component I imported the getCategories helper function and the Category interface from the directus.ts file, created a function that gets the data from the getCategories function and then called it in a useEffect hook in order to only execute the request once. I defined a new variable in order to save my returned data. I also created a hash map for my icons, that way I can print the correct icon with the corresponding category. Finally, in my jsx, where the menu items are printed, I created a map that returns the contents of each category.



# Activity type 2:

## Diagram for database

In regards to the database, I created the diagram using draw.io.



The database includes numerous tables related to eshops such as a products table, categories table, a tags table and connecting and related tables. It is worth noting that many to many relations require connecting tables while one to many only required foreign keys.

For the products, product images (gallery) are required and therefore I created another table, connecting to the products table that contains the links (url), the name and the alt for the relating product as well as the foreign key that connects the image to the correct product.

The products table contains the standard fields such as name, id, description, price and sku with the addition of stock number and stock status for better management, weight and dimensions for shipping management as well as meta information for helping with searching on site and on platforms such as google search.

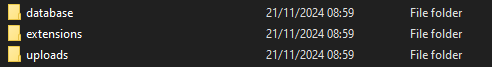
Tags and categories tables are used for better management and easier searches of products.

## Creation of database

For the backend I decided to go with directus, a well known and easy to use backend system based on nodejs. For the installation I followed the guide of directus docs for self-hosting.

I started by downloading docker, as the project needs to be dockerized in order to run. The process is self explanatory as all the user has to do is click on next until the installation is complete.

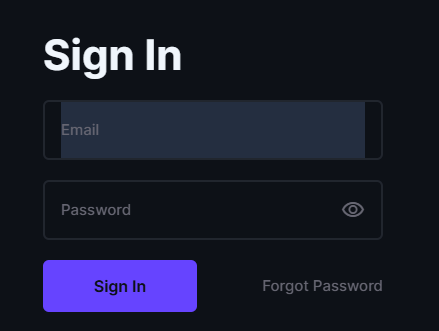
After the installation of docker, I needed to create the directus folder for my project in which I included three folder: database, uploads and extensions.



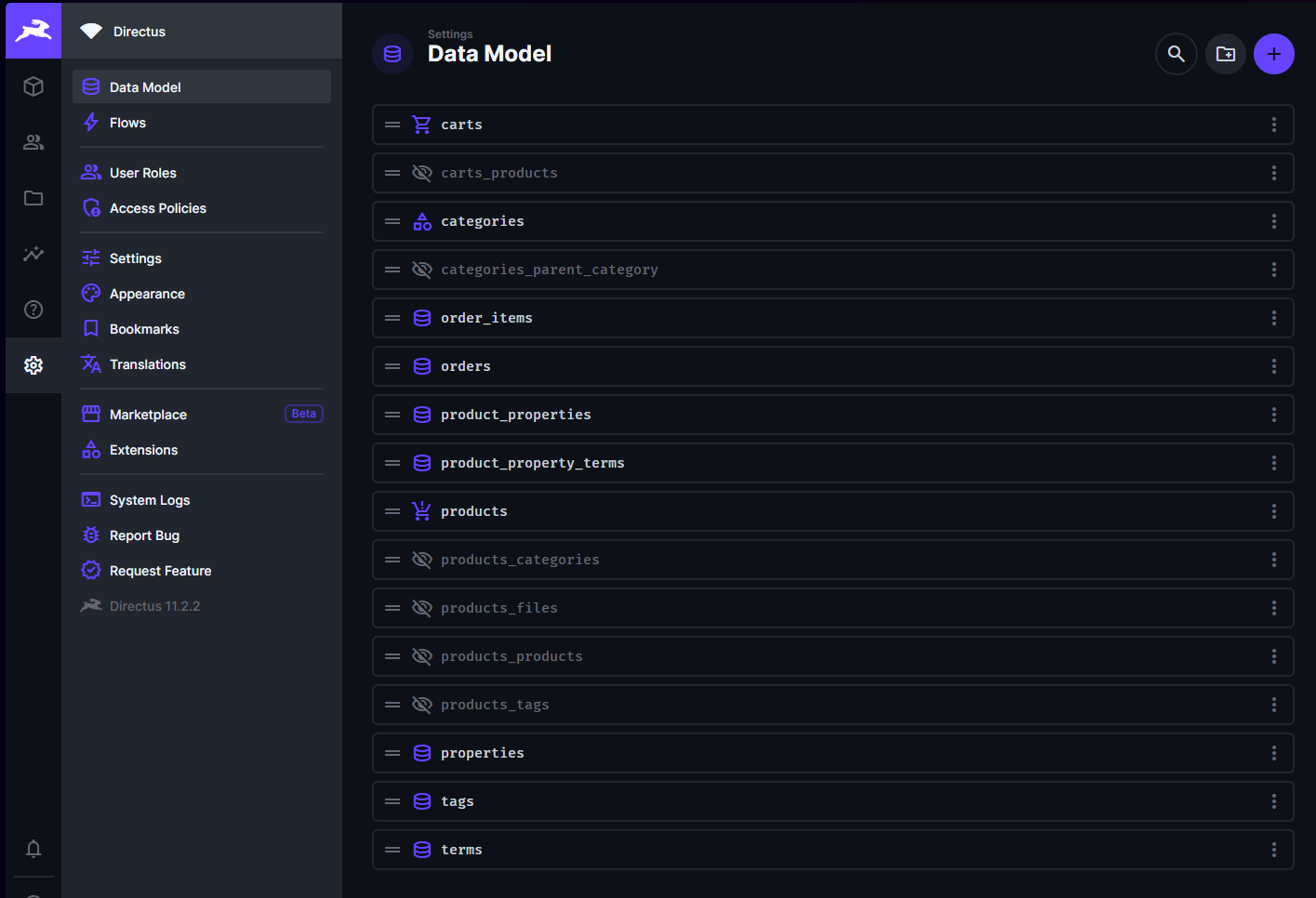
In the main folder I had to create a file called docker-compose.yml containing a set of instructions for docker. Finally I had to run the docker container by navigating to the folder containing my files and executing a command line with the command `docker compose up`. After the download was complete, I had access to my backend using either the link `http://localhost:8055` or `http://127.0.0.1:8055`.

# Creating tables and relations

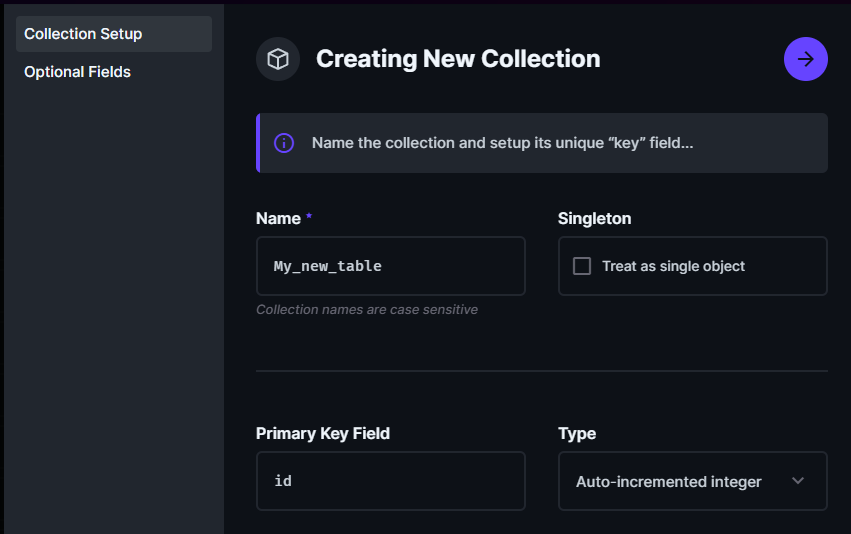
After I created the Directus backend in docker I had to create the database’s tables (Directus calls them collections) and relations. In order for Directus to be launched, the docker container needs to be running. I use the docker interface to launch my container but that is not the only solution to running the project as mentioned in the previous chapter.  
  
After the docker is launched, I log in by accessing the localhost url and entering my credentials.



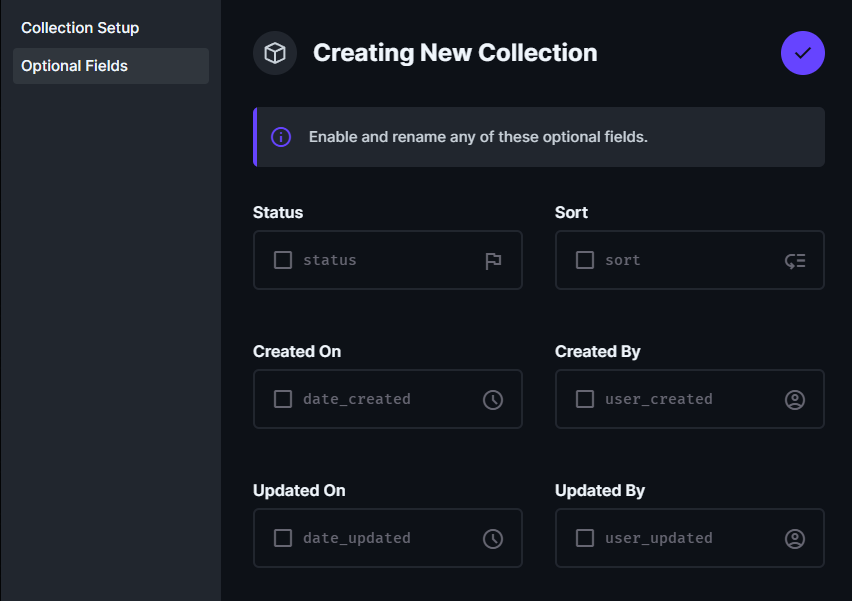
After a successful login, in order to create a new table I have to navigate to the bottom of the left side menu and choose “Settings”, access the “Data Model” and click on the “+” icon in order to create a new collection / table.

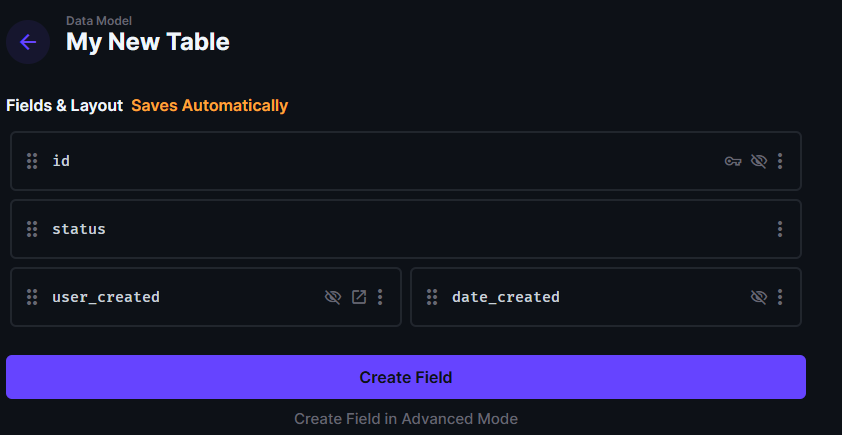


In the next section, I can choose a name for the collection, whether I want it to be treated as a single object and finally set the primary key.

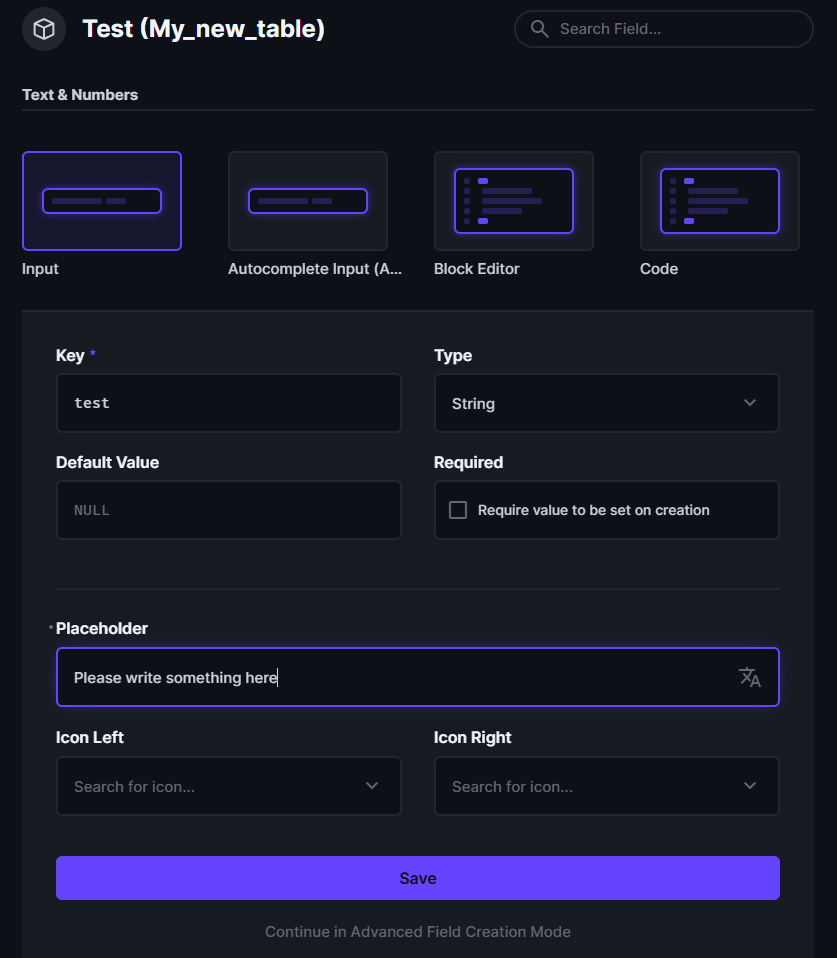


Then, I add any extra fields I might want by accessing the “Optional Fields” tab. After choosing the optional fields, I click on the accept icon.



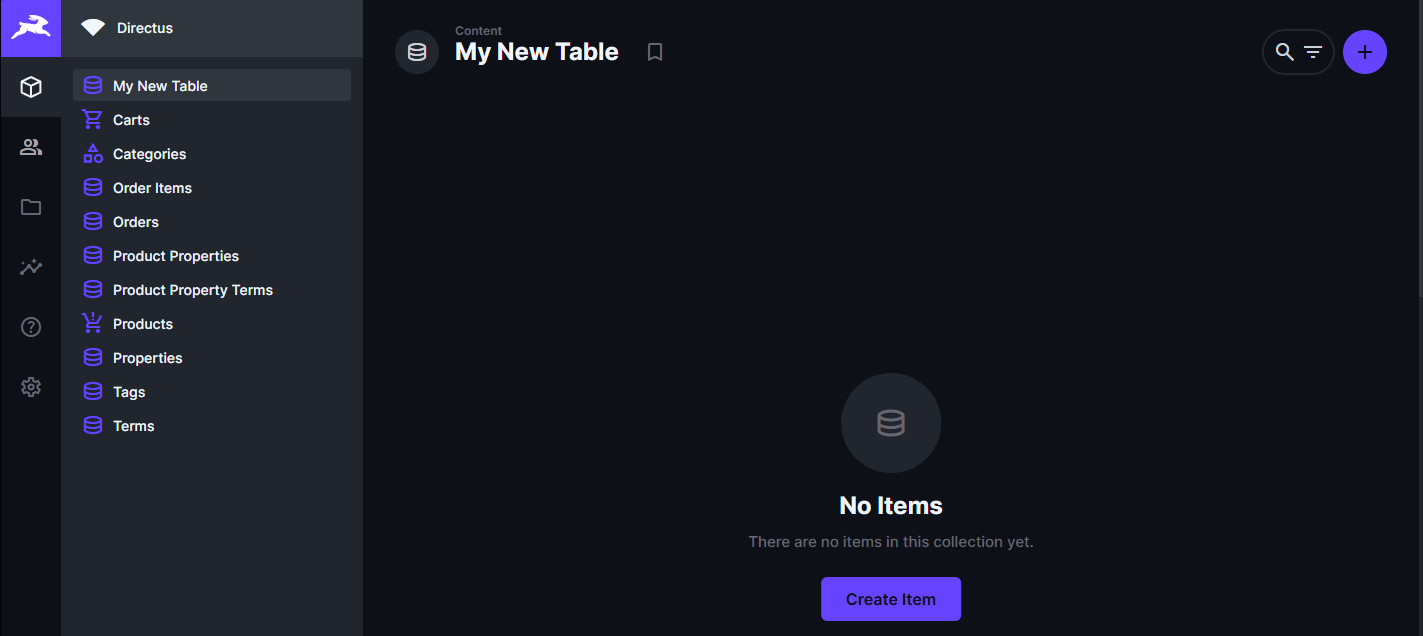
I then click on my newly created collection and click on the “Create Field” button to add any new fields I need. This is where relations are also created, so it is an important step.  
  


In the next section, I choose the type of field (for this example I will choose a simply input but the process is similar no matter what the type of the field is), I then fill in the necessary information and finally I save.

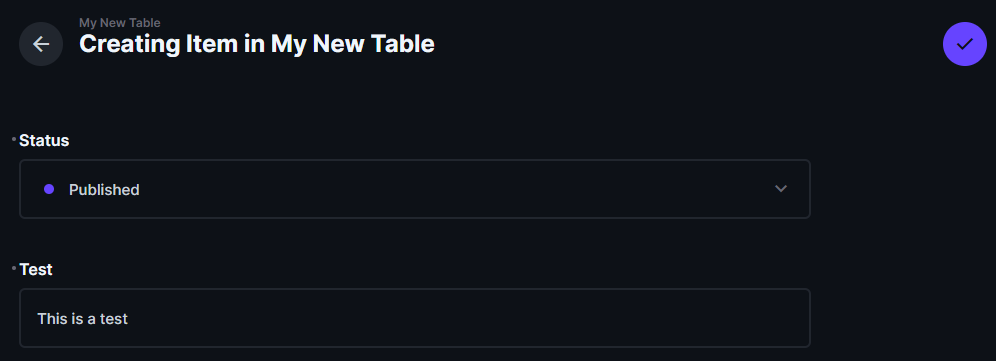


Now my new collection is ready to be used. If any other modifications are necessary, I can repeat the process, edit a field or even add complementary information like automatic sorting based on a field of my choice.

In order to add content to my new collection, I simply have to navigate to the “Content” tab of the left side menu choose my collection and click on “Create Item” button.



Finally, I choose the status of my item, fill in the fields and save by clicking on the save button.



This is a typical workflow I followed while developing my internship project.