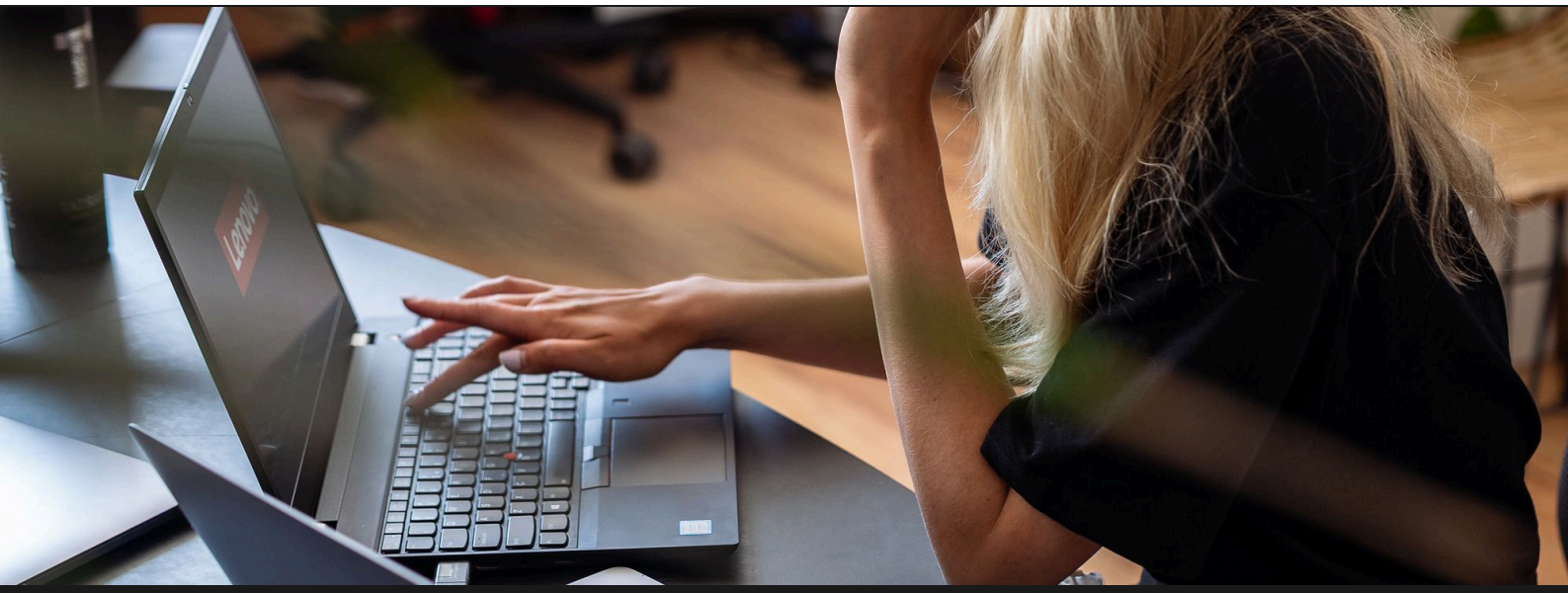


Junior Full Stack Developer test task

Welcome to the Junior Full Stack Developer test task!

Thank you for your interest and time invested in our recruitment process!

This assignment tests the minimum skills and knowledge required to successfully kick-start your career at scandiweb. The test helps us to assess your level and it helps you to check whether your level is already good enough to join us as a Junior Developer.

Please, follow the instructions to make your test compatible with the test automation script and make sure that you score all PASS before you submit it for human evaluation.

The expected outcome of the test

A simple eCommerce website with product listing, and cart functionality

General coding requirements

Creation of Back-end side:

- The BE programming language must be **PHP7.4+ or PHP8.1+**
- It is forbidden to use any Back-end frameworks(Laravel, Symfony, Slim etc)

- It is allowed to use third-party libraries(Doctrine, Dotenv, etc)
- You need to create a MySQL database with populated data provided by scandiweb.

  data.json 26.1KB
- Using MySQL: ^5.6 is mandatory
- Utilize OOP and use OOP Approach in the backend
  - We expect to have no procedural outside of the initial application bootstrap, which could be registering routes/graphQL handlers, initiating database or router, and initiating config
  - We expect a demonstration of meaningful usage of OOP features, like inheritance, polymorphism provision, and clear delegation of responsibilities to each class
- Utilize models for categories and products and attributes
  - Each model should leverage polymorphism. We expect an abstract class for each model which has different types, with differences between types handled in their own sub-classes
  - Any differences handled in different types for Products and Attributes shall not require the usage of switches/if statements but handled in their own classes
- We expect the use of the provided carcass as the base implementation of GraphQL support in the backend application, the implementation of actual schemas/mutations, etc shall be done by you. The carcass can be found **HERE**
- You need to create a GraphQL schema for categories/products and their fields
  - We expect to see attributes as part of Product Schema however they should be implemented as a type of their own and resolved through their own set of classes and not directly on the Product Schema/Resolver
- You need to create a GraphQL mutation for inserting orders

Creation of the Front-end side that should work as SPA (Single Page Application):

- Creating CRA application
- It is forbidden to use ReactJS-based frameworks (NextJS, Remix, etc)
- It is forbidden to use component libraries (Material UI, Chakra, React-Bootstrap, etc)

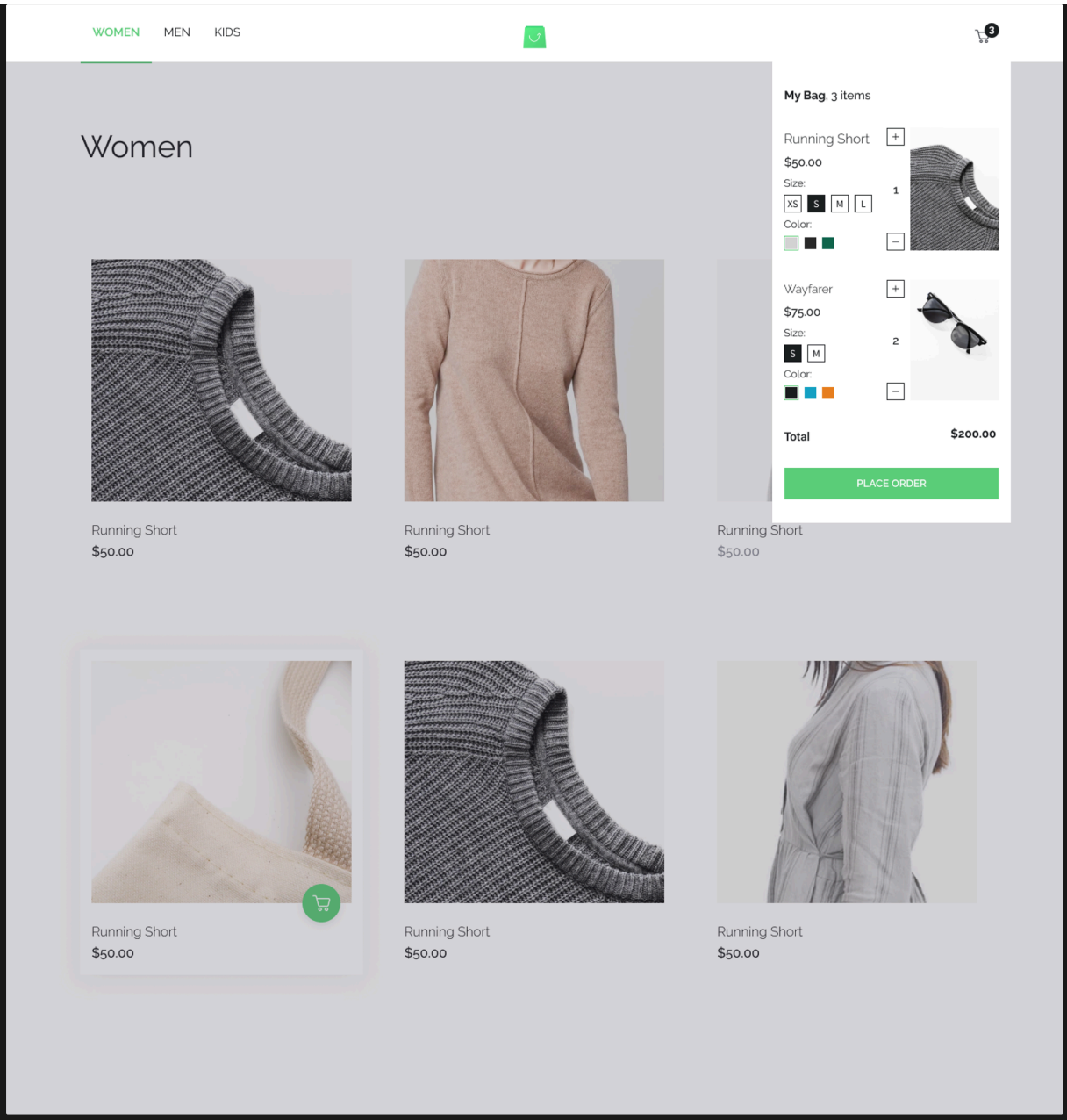
- It is allowed to use:
  - Plain CSS
  - CSS Preprocessors (SASS, SCSS, LESS, etc)
  - CSS Frameworks (Bootstrap, TailwindCSS)
  - CSS-in-JS (Emotion, etc)
  - styled-components
- Creating GraphQl requests for information gathering from the Back-end server.
- Creating Front-end as per design (Not pixel perfect).
  - Full Stack Test Designs www.figma.com
- Displaying data from the Back-end as per design
- Implementing required functionality (explained in the next paragraph)
- Using **ReactJS i**s mandatory
- Class components only

Functionality Explanation

The cart overlay button shall be included in the header and visible on all pages

- **Item count bubble**
  - Shall be visible on the cart overlay button only if there are products in the cart
- **Total items count**
  - If only one item is in the cart, it should be shown as `1 Item` . If 2 or more plural form should be used: `X Items`

- **Product List in the cart overlay**
  - Have to display the product name and main picture
  - Have to display currently selected product options (like size or colour) and the other available options
  - If a user adds the same product with **different** option(s) selected they have to be displayed separately in the cart overlay view. However, if the user adds same product with the **same** option(s) selected it has to be displayed with respective quantity
  - Clicking on “+” should increase quantity, and clicking on “-” decrease. If the product’s quantity is equal to 1, clicking on “-” should remove this product from cart
  - Product options should not be clickable in the cart overlay
- **Cart Total**
  - Cart total have to be presented as a total of all items currently in the cart, if none it shall still be present while showing 0 in total
- **Place order button**
  - It has to perform respective GraphQL mutation that as a result will create a new order in DB
  - Once order is placed, cart should be emptied
  - If a cart is empty the button shall be greyed out and disabled
- **Page Behaviour**
  - When the cart overlay is open the whole page except the header shall be greyed out. Refer to designs for a visual example
- **Save Behaviour**
  - The cart doesn’t need to be saved and doesn’t need to be persistent, it should only be persistent through a single user session at a minimum. This means saving it in the frontend states and local storage is enough

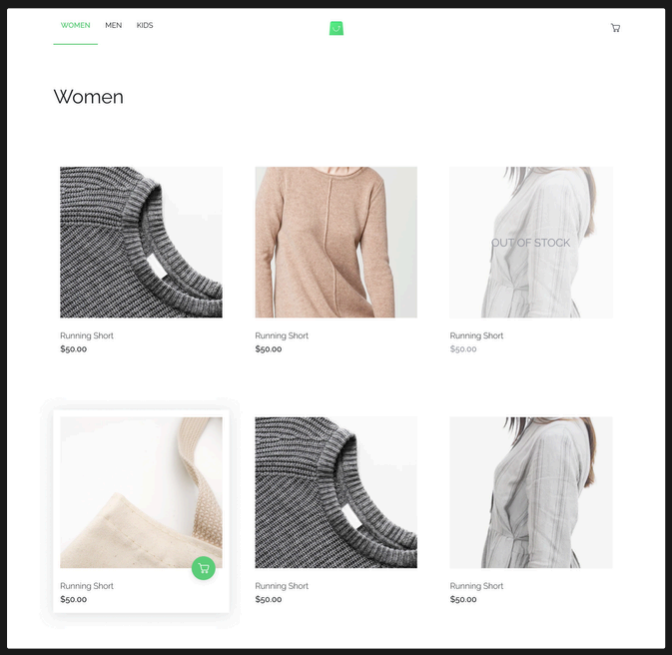


Product Listing Pages (Categories)

These pages shall be shown whenever a category is chosen and it’s the default view of the website, the very first category is always shown as the website's default view

- **Product Cards**
  - Each Product Card have to display following: the product's main image, product name, product price
  - Product Price have to be in the correct format (2 digits after the dot)

- **In-stock products**
  - Have to be clickable and lead to the Product Details Page (PDP)
- **Out-of-Stock Products**
  - The Product Image have to be greyed out
  - an Out of Stock message have to be visible on the Product Image
  - The Quick Shop button (The green cart button) must not be visible
  - Product card have to be clickable and lead to the product's main page. However, add-to-cart functionality must not be possible
- **Quick Shop**
  - Clicking on the quick shop button (The green cart button) have to add a product with its default (first in each options array) options to cart

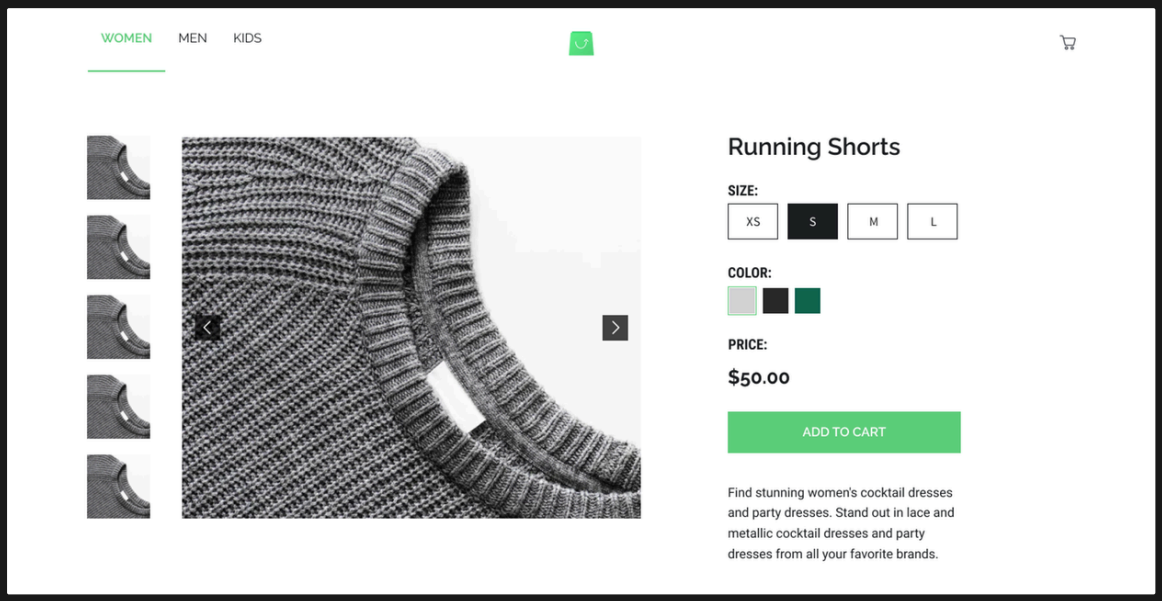


Product Details Page (PDP)

The page have to be showing all Product details, images, and a button to add it to the cart, the user should be able to configure their product on this page before adding it to the cart

- **Product Details**
  - Product Name
  - Product Attributes should be visible and select-able with their name clearly visible as in designs
    - Size Attributes should show swatch buttons for each size type
    - Color Attributes should show swatch buttons for each colour type designed so each type is a square of this color. Refer to provided designs for visual example
  - Product price have to be visible and formatted correctly according to currency formatting rules: 2 digits after dot
  - Product description have to be visible and added last under Add to Cart Button, HTML tags should be parsed. It is forbidden to use **dangerouslySetInnerHTML**
  - A Gallery of product images have to be visible
- **Product Gallery**
  - It have to be presented as an image carousel if images should fit the main image height, max-height should be set, and scroll allowed
  - Available images have to be visible on the left-hand side of the currently visible image
  - It should be possible to click on the any of images to switch to it
  - Arrows for sliding the images have to be visible on top of the main image

- **Add to cart button**
  - Have to be greyed out and disabled until the user selected the necessary options for this product
  - Clicking the button have to add the product to the Cart and open up Cart Overlay to show added products



How to submit?