Shopware React Storefront Manifest

The background

ScandiPWA team is known for building storefronts for e-commerce. We focus on developer experience enhancement. We achieve it by making our projects extensible, modular, themeable and easy to get into. We want to bring our extensibility patterns to the community of Shopware, because we believe in Shopware as a product.

We are the team of passionate developers who value extensibility, ease of maintenance, and low boundary of project entry.

The goals

Our goal for the next week (10th May – 15th of May) is to **deliver an MVP of a React "blank theme" storefront for Shopware, deploy it's demo and publish it Open Source**.

The requirements – architecture

As the requirements for our MVP implementation, we highlight a standalone-module approach, which **must organize frontend logic into isolated entities**. Each module must contain the entity request logic, the entity context, the layout definitions for entity presentation and the entity integration into the application. The standalone entities (modules) planned in MVP scope are: CMS, Category, Product, Auth, Customer, Checkout, Cart. The frontend separation into entities is important, as it allows for long-term flexibility and ease of maintenance.

The flexibility is achieved by ensuring that modules not only contain the entity implementation, but also it's integration into the application. This means that the module could be disabled or swapped at any moment. This allows applications to integrate third-party services, migrate to newer API, while still working in a scope of a single module / entity.

Modules are standalone. Thus, for contribution or maintenance, you are not required to understand the big-picture of the application. This also allows for easier test-coverage, as now we can test each entity behaviour in isolation, and we can also test each specific integration, while still working in a scope of a single module. The test-coverage is out of scope for our MVP "blank theme" product implementation, due to our time constraints.

The modules must be abstract and extensible, thus they will not contain styles or any unnecessary markup, but will define the consistent extensibility points and logic for further extension. We call the process of building modules such a way – a "blank theme" creation.

The **“blank theme” will serve as an abstract basis** for all further developments. Other themes will be built on top of the blank theme with inheritance. Thus, in opposition to composition, the abstraction of the base layer is required to ensure further extensibility. Integration must happen on the abstraction level of “blank theme”, and not in the final application.

We call this a Mosaic architecture. It is made possible by our build-toolchain – tilework/mosaic. It provides an inheritance-based theme mechanism and an ability to build completely standalone modules.

The requirements – technology stack

The project must allow for presentation layer flexibility, as it will serve the base for other themes in the future. Thus, **we believe it is important not to limit developers with the tools and libraries of our choice**. Thus, the third-party dependency list must be as short as possible.

We plan on having just 2 third-party dependencies:

1. React – because our team has a level of expertise in it, plus we developed a list of utilities and approaches that might help us deliver the project faster.
2. NextJS – because of SEO and speed, which is hard to achieve without server-side rendering. We believe that baking the server-side rendering into the application is better, rather than forcing people to use third-party rendering services.

We are not planning to use any third-party state managers. Our application will utilize the built-in-one: React context. We are also not planning to use any third party clients to communicate with the back-end.

We are avoiding third-party solutions, because:

a) Most third-party solutions contain features that we will never actually use, thus creating a dead-code in our application, which will increase the bundle size.
b) We have no control over the third-party libraries. If used, they will require an abstraction over them, if we want to keep them extensible. This would grow our bundle size even more.
c) Introduction of any third-party library will require a developer to learn it. This would create an entry threshold that will stay there forever. We are aiming to set this threshold as low as possible to make the product accessible to the widest possible range of developers.

Having a UI layer is important, because it is a building block of a “blank-theme”. Writing a completely custom UI is a very difficult task, that creates a huge entry threshold. We believe that developers should be able to choose the UI they want: Bootstrap, Material, etc. Thus, we are going to introduce **the Virtual UI – an abstraction over the whole UI layer that would allow developers to use the UI library of their choice**.

The community input

If you like the idea and would like to contribute, **reach out to us via the YouTube live-stream commentary section**. We will be happy to greet and meet you live on a live stream, discuss your task, review the PR and assist the implementation.

We are building the application in standalone modules, however, the MVP scope only covers the limited amount of entities. In the later stage of development (starting from 11th May) the roadmap of smaller entity-implementation modules will become available. This may include entities like: Pricing, Tier-pricing, Totals, Taxes, Media. We would appreciate these modules' contributions.

We are aware that some modules are larger than others. We might not be able to complete the whole CMS module without the input from the community in time. We will focus on it’s most sensitive parts, while leaving out many section, block and element types. We would appreciate these CMS module parts contributions.

Our UI is virtual, this means it might have multiple implementations. If you don’t like our original choice, of supporting the Material UI, or have a personal preference, feel free to reach out to us! We will be happy to see multiple UI implementations, as this would strongly affect the flexibility and delivery time of styled theme creation.

We put a strong emphasis on developer experience enhancement, thus, we want as much feedback from the community as possible. The feedback: negative and positive, will help us to create a product that satisfies community needs the most! We hear your opinion.