ADR 003: Use the BFF pattern

We have a microservice architecture with several REST services and different types of frontends: Web application, iOS application, Android application, public API clients (for the future), chatbot (also for the future). Different frontends may require slightly different message formats, message structures, headers, etc. Farmacy Food is a start-up with limited resources to configure, deploy, and govern more sophisticated middleware solutions, such as a full-fledged API gateway product.

Decision

We will use the BFF pattern and have BFF services for each type of frontend as the central point of interaction with the Farmacy Food frontend apps. Moreover, instead of a single BFF service (for each frontend type) that interacts with *all* backend services, we will create separate BFF services per subdomain: - **BFF for User** and account management (one version for each frontend type) - **BFF for Catalog** and browsing services available (one version for each frontend type) - **BFF for Order**, including checkout and order processing (one version for each frontend type)

Rationale

- The BFF pattern prescribes a different BFF edge service that handles the specificities of each type of customer. It is simple to implement and deploy, especially if you use the same development and runtime platforms used for other services in the solution.
- We have separate BFFs for Android and iOS for different reasons:
    - to allow the iOS and Android app teams to be responsible for their own BFFs;
    - to allow different deployment configurations for each BFF. For example, if we get 10x more request from Android devices, we can scale out the Android BFFs.
    - to more easily handle app communication details that are platform specific, such as push notifications.

Status

Proposed

Consequences

- Frontend apps will call BFFs and frontend devs need to discuss with backend devs the BFF contracts (endpoints, message formats, etc.).
- New server-side functionality should be exposed via BFF endpoints.