avensia

PRODUCT DESCRIPTION

2021-11 v2

Avensia Nitro accelerates the start-up of e-commerce projects and enhances our customers' business value and the consumers' experience.

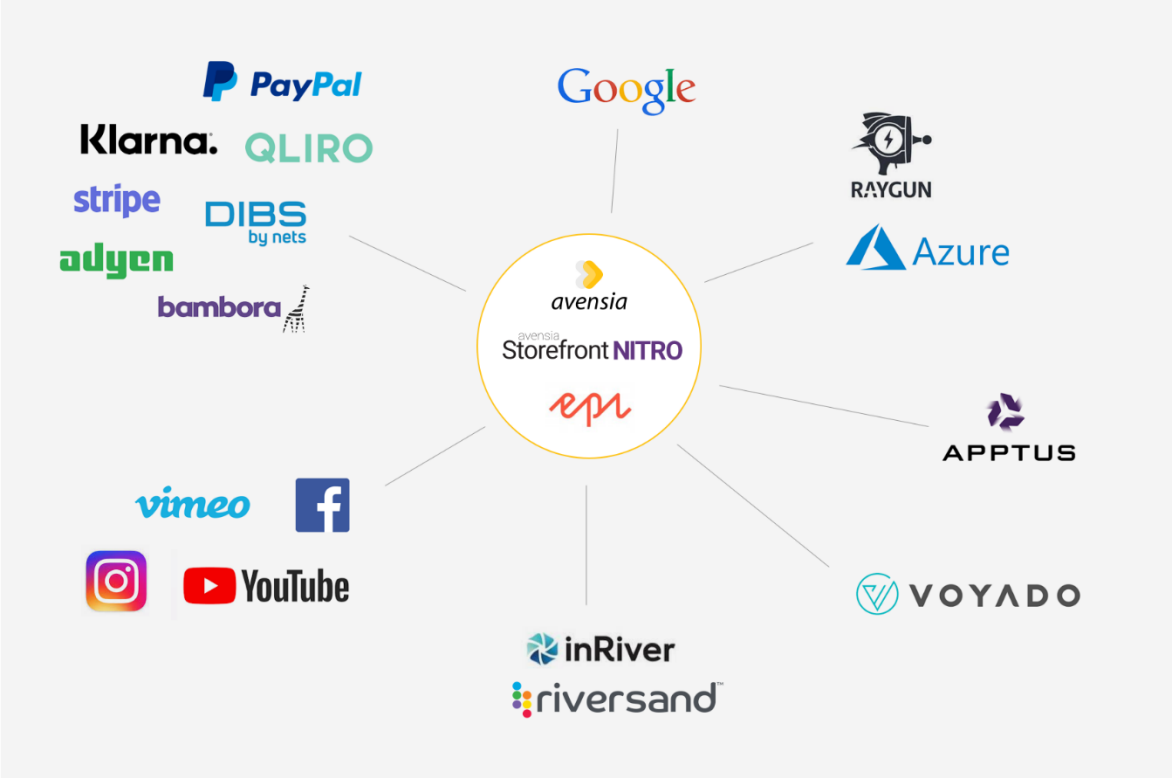
avensia

Contents

1 Avensia Nitro Solution Framework

Avensia Nitro gives you the following key benefits:

- **Faster time to market** for complex projects. Allows for early focus on customer value rather than technology.
- **Smooth customer experience** built on the latest UI technology.
- **Thoroughly battle tested** by some of the most demanding e-commerce sites in the Nordics, including Lyko, NA-KD, Kjell & Co, KICKS, Filippa-K and Polarn o Pyret, totaling to more than 10 000 development hours.
- **Lower cost of ownership** through shared packages that are continuously tested and updated.
- **Connects to a rich ecosystem of third-party products** for content handling, product information management, search, payment and shipping providers, customer relations and marketing.



- **A solution for both B2C and B2B**. The product ships with features for a fully functional ecommerce site, with many capabilities for supporting commerce both to consumers and to businesses such as customer specific assortment and trade agreement prices. The solution can be customized utilizing existing functionality for **D2C and B2G**.

2 Performance

Performance is at the heart of the whole stack, from the software packages included, to how we work with the search index, to how data is sent to the client and how the client is architected, to how data is loaded from data sources.

2.1 PWA

The client application is a Progressive Web App. It uses the latest browser technologies to ensure the application is resilient against slow and spotty networks. By using these technologies browsers such as Google Chrome prompts the user to add the app to the home screen, making it feel like a full-fledged mobile app.

We use different strategies to ensure that the user experience isn't tightly coupled to the state of the network connection. One example is that we use optimistic updates such as adding something to the cart and seeing it in the cart before the server has accepted it. Another example is rendering content and pages from partial data and rendering placeholders until the data comes in from the server.

If the user makes an action such as adding something to the cart or navigating to a new page and that fails because of a network error, we show a graceful error message to the user.

2.2 Scaling out with asynchronous processes

We make heavy use of asynchronous processes to ensure availability and scalability. Instead of sending a product to the search index directly when the product is changed, that change is placed in a queue which a separate background job processes and can therefore process multiple changes at the same time. Transactional emails work in the same fashion where it's put in a queue to be processed by a separate background job. These background jobs also have the ability to gracefully handle errors and retry later.

2.3 Pre-calculating and pre-processing content

Instead of calculating and processing content on the fly as the user request it, we achieve a higher throughput and faster response times by pre-processing and caching the content. Instead of doing price calculations on the fly we send the list prices to the search index and use event-based triggers to send new prices to the index as they change. For use cases where it's important to have real-time data we send one version to the index that is good enough to show during the time we asynchronously load the real-time data. The important principle is to not compromise the user experience by making the consumer wait for such operations but still make it clear that something is loading.

2.4 Optimizing the network with a CDN

In order to achieve the best possible network transport, the solution makes heavy use of a CDN that's included in Optimizely DXC. Images are cached with a far future date to ensure that images can be served directly from the CDN but still makes it possible to change an image and have that propagate over the systems directly.

The CDN ensures that the most optimal version of an image is sent to the client. Newer browsers support modern image formats such as WebP, and the CDN sends the image in the best possible format.

The CDN uses HTTP2 when communicating with browsers that supports it, and through that the solution uses HTTP2 Push to push critical assets together with the initial response.

3 Software packages

The solution comes with a set of core compiled software packages installed, and these packages are continuously maintained, and new releases created with new features and bug fixes. All packages are not applicable for every solution but it's possible to install these packages later in the project if needed.

By using packages, the code is more isolated and therefore easier to test and upgrade. Every solution that has implemented any package can chose to upgrade at their own pace.

3.1 Starter Site

Avensia Nitro has a pre-built starter site which makes it possible to conduct efficient discovery workshops with a fit/gap approach showing all the out-of-the box features and then build your project on this site.

The starter site is delivered as source code that can be customized to optimize customers business value. New releases of the starter site give you the opportunity to copy best practices to the implementation of your customer specific e-commerce site.

Our starter package consists of pre-built integrations APIs, starter site, and user stories with test criteria's. The starter site is not a demo site, it is a true project accelerator.

3.2 Epi Foundation

A foundational package that a lot of our other packages depend on. It contains a framework for writing and running migrations that is sometimes needed to enable new features or extend the system.

It contains a powerful content processing and dependency tracking framework. When a product is sent to the search index the indexed entity is composed of a data from different sources. Such as the name of the product, the prices of the variations, the names of the categories it exists. The dependency tracking system ensures that whenever any of these entities change the indexed product is updated in the search index. This enables us to only run incremental index builds and only have to run full index builds on rare occasions.

It contains a framework for payments to streamline using different payment provider such as Klarna or Paypal. The framework ensures that the customer project can just hand over the cart to the payment framework and the payment framework knows which provider to communicate with to run authorization, capture and credit.

3.3 Payment Connectors

Contains a set of fully featured connectors against common payment providers such as Klarna, PayPal, Paytrail, Bambora, Adyen, Qliro and Collector (see Functional Specification for full list). When a new payment provider is implemented in a Nitro project, that is also packaged and made available to our customers.

3.4 Checkout Framework

Our experience is that the checkout flow is the most complex part of an e-commerce solution and it's a challenge to make it robust. The possibility to combine different payment and delivery options, gift cards, promotion codes, service fees and discounts create a plethora of scenarios that must be covered. If we do not get this right, we risk losing not only conversion but also customer trust.

To cope with this, we have created a shared package that abstracts checkout complexity. It enables the development team to focus on customer experience rather than technical edge cases.

The checkout package is agnostic in presentation so it's possible to implement any design or checkout flow.

Another example is getting a fast and consistent experience when the user rapidly clicks Add to cart or quickly switches between payment or shipping methods.

3.5 Scope Performance Framework

This package is what enables us to tie together Optimizely and ASP.NET MVC with Single Page Application principles and React. It does so without compromising what makes Optimizely and ASP.NET MVC great.

This package contains a lot of features that enables more efficient loading of data, such as deferring data loading until it's needed or grouping multiple requests to external systems (such as the search index) to a single query.

3.6 Typescript Type Generator – Ensures Quality

This package takes a model defined in C# and generates a TypeScript definition for it. This means that any JSON sent out from the API is expressed with a type safe contract and that contract is automatically updated when a C# model is updated.

Having this in place gives confidence in making solution wide changes since the type checker gives compile time errors if the client code does not follow the generated contract.

After we introduced this in our solutions, we have seen that a whole category of bugs we were used to seeing in all projects simply disappeared.

3.7 URL Redirector – Accelerates SEO

Keeping track of redirects is crucial to not losing business from users coming to the website from old URLs. This package monitors renaming of content and keeps track of old URLs and ensures that we redirect the user to the new URL.

It also enables you to add custom rules that resembles regular expressions. You can set up a rule to redirect "/en-us/*" to "/en/$0" which would redirect a request from "/en-us/about" to "/en/about". It also contains the same powerful matching for query parameters and makes it possible to pass through query parameters when we redirect.

3.8 Search & Relevance

A fundamental part of any site is to serve relevant content to the visitor and that the conversion of the usage is directly related to the performance of the site. In Nitro almost all content (category listings, brand pages, campaign pages, blog articles, editorial content etc.) is served by a powerful search engine.

Apptus eSales and Optimizely Find

As part of the standard implementation, Nitro offers either **Optimizely Find** or **Apptus eSales** as the internal search and relevance engine.

In Optimizely Find you get a capable search engine that is optimized to be used in conjunction with the Optimizely platform, whereas in Apptus eSales you get a market leading relevance engine that not only returns the search or list result with high performance, but also sorts the result to prioritise

the most relevant result for the current visitor. Behaviour by the current and other visitors are collected and used to determine whet products are relevant at what time and context.

Filter facets

Both search engines enable the user to filter the product listing on product attributes and with the Apptus eSales implementation, the relevance of the filters and filter options are automatically displayed and sorted to minimize the need for manual administration while constantly ensuring optimal conversion.

Search

All editorial content such as blog articles, how-to articles, stores etc. and the product catalogue such as categories, product and variants are indexed by the search engine and can be retrieved by the visitor by normal search, auto-complete and supports fuzzy search, synonyms and search suggestions.

Recommendations

Apptus eSales also adds recommendation panels to Avensia Nitro. This could be products lists added to the product detail page or the cart or checkout and can be configured in Apptus eSales management app to show e.g. people who view item A also purchased B..E, top-sellers for one or more categories etc. The recommendation panels also use the same automatically optimized relevance algorithm as the search results to ensure optimal conversion.

3.9 Relevance Search Engine Optimizer

Built on top of the dependency tracking and content processing system from EpiFoundation. It allows us to populate the Apptus eSales or Optimizely Find search index with data from the Optimizely catalog.

Enables the customer project to customize what data should be sent to the index and in which format to optimize performance and the search and filter experience for end customers.

3.10 Storefront ERP Integration Framework

Avensia Storefront Integration Framework is a connector that imports product catalog information from an ERP system into Optimizely Commerce Catalog. ERP systems supported by Avensia Storefront Integration Framework are:

- **Dynamics 365 for Finance and Operations**. The integration framework connects to an Online Retail Channel defined in Dynamics 365 for Finance and Operations and fetches any updates made to the Catalog. The integration framework also sets the online retail channel in status *Published*.
- **Dynamics 365 for Retail**. The integration with Dynamics 365 for Retail is identical with the integration with Dynamics 365 for Finance and Operations.
- **Microsoft NAV 2018**. The integration framework uses LS Omni service for the catalog import. LS Omni is a part of the LS NAV Suite distributed by LS Retail and their partners. LS NAV requires a separate license from LS Retail or their partners.

Information imported into Optimizely Commerce is by default:

- **Product Master**. Product masters are templates used to create variants. The product master defines the dimensions used for variants, such as color, size and type.
- **Variant**. Variants are created by the ERP based on the dimensions defined in the product masters.

- **Retail product Kit**. Retail Product Kits are defined in the ERP system as a collection of products where each component may have substitutions that the visitor may select. A product retail kit has no unique article number and have no unique inventory.
- **Product Kit**. A product kit is a product defined in ERP that bundles other products into a unique article number. A product kit has its own article number and inventory.
- **Navigation Hierarchy**. An online retail channel requires a navigation hierarchy in the Dynamics ERP systems. The navigation hierarchy is imported into Optimizely Commerce Catalog and used when products are imported into the catalog. The web editor may use the hierarchy for navigation.
- **Media**. Images and other media are copied into Optimizely Asset Manager for SEO and performance.

3.11 PIM Connector

Connectors between Optimizely and inRiver PIM and Riversand MDM with an event-based architecture which ensures that we only send the data needed to make data flow from inRiver or Riversand to Optimizely as fast as possible.

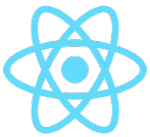
Our experience has taught us that you don't want a one-to-one mapping of the model in the PIM to the model in Optimizely. You want to exclude some fields that aren't relevant to Optimizely and ensure that changes to those fields doesn't trigger data to be sent. It might make sense to express the product hierarchy as two levels in PIM but three levels in Optimizely.

This experience made us design the connectors between these systems in a way that makes it possible to re-map the model in a layer between these systems.

4 Technology stack

The framework is built on battle tested components from companies such as Microsoft and Facebook. The value proposition of Avensia Nitro is that we've taken these components and glued them together so that they work as one coherent solution.

Microsoft .NET / C# / ASP.NET MVC. The industry standard web framework from Microsoft. These are the tools that both Optimizely and the Avensia Nitro backend (with business logic) and all integration components are built with.



React. Developed and maintained by Facebook, React is the world's most popular JavaScript library for building user interfaces. Not only does React improve performance with its virtual DOM, it also allows us to structure the frontend into components with a clear interfaces and responsibility, making the solution more maintainable.

Redux. Redux is a development library that brings a standardized way of managing state in a Avensia Nitro application.

TypeScript. TypeScript is central to the way we handle data models in Avensia Nitro: all data that is returned from the server is a typed model in .NET, and corresponding models are generated automatically for the frontend to use. This streamlines the work for the frontend developer by making the data model available through intellisense in the development environment. If a data contract is broken, a compile time error will be issued.

V8. V8 is a JavaScript engine that is central in order to serve content to user agents that cannot execute JavaScript. This is central for SEO but it also improves the first load experience such that you can see content before the client application has been bootstrapped.

Webpack. Webpack bundles JavaScript, CSS and assets in modules to make the loading of browser resources as efficient as possible.

NuGet / npm. These are package managers (for backend and frontend respectively) that are used to enrich the functionality by reusing existing components/implementations. Avensia Nitro uses a mix of publicly available packages (like Optimizely) and a handful of internal packages.

4.1 Single Page Application

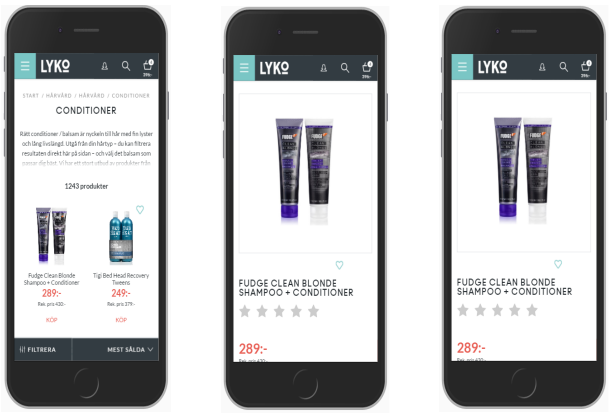
The website(s) built with Avensia Nitro becomes a Single Page Application, where all internal site navigation is done by loading only the content needed for that click. Since this is driven from the client application it enables us to efficiently use cached data that we have loaded before.

From an operational excellence perspective, the most interesting aspect of having a website as a Single Page Application is that it becomes much easier to scale and to further enhance the UI interactions and UI complexity. Since the client becomes a real application, we can use industry standard patterns and practices for managing complexity.

4.2 API first

All data that is visible on the website – such as texts and product data – are available through an JSON API that the client application uses. This means that any data presented on any page can be utilized by any client and not just the website.

4.3 Caching



A typical example of how we use client caching is going back and forth between pages such as a product listing page and a product detail page. When you navigate back from a detail page to a listing page, we already have all the data needed to render the listing page, so that rendering can happen without having to load anything from the server. This makes that experience very fast regardless of how fast your network is or the current load on the server. The image to the right shows an example of going from a listing page to a detail page. We can render the detail page immediately from the data we already have from the listing page and in parallel request the full details from the server. The consumer isn't hindered by this and can quickly and safely navigate back even if we haven't gotten the full details yet.

When we launch a new site with a customer, we typically see that page views per session increase drastically, and this is where Avensia Nitro shines. Avensia Nitro removes a lot of pain points for creating smooth navigation and makes it frictionless to browse the site.

5 Content

Avensia Nitro contains a variety of features to create and display content on a site, ranging from plain article pages (with media and text) to specific landing pages and start page with a more banner- focused layout. Optimizely CMS Blocks are components used by the web editor to create content and design on the web pages in the site.

Blocks are used to build up the structure of a page, and blocks are shared between pages. The Nitro layout blocks adapt the web page according to the users' devices - mobile devices or desktop.

6 Operations

The solution is hosted in Optimizely DXC, which is a cloud service on top of Azure. It utilizes Azure Web Apps to ensure availability and scalability.

6.1 Environments

The solution is deployed to three different environments. Integration is the first environment and should be seen as a pure testing environment where all changes are continuously deployed. As soon as a verified change gets merged to the main branch, it is built and deployed to this environment.

The next environment is the pre-production environment which is used as a testing and verification environment for changes before they are deployed to production.

The last environment is the production environment which is monitored by Optimizely to ensure availability and has auto-scaling set up to scale out as more traffic comes in.

External systems such as ERP, PIM, WMS, PSP, etc need to have at least one test environment as well as the production environment.

6.2 Deployment

Deployment is done through Optimizely DXC which uses Azure to deploy new versions of the solution. A new deployment slot is set up on the public instances and quick verification can be done on the deployment slot before going live. When going live Azure will swap the slots at the same time to achieve zero-downtime deployment.

In some cases, deployment needs a service window to run migrations or other major system changes and in those cases a maintenance page is displayed for the end users until the deployment is done. The goal is always to have as few planned downtime deploys as possible.

It is possible to deploy at any time and the time it takes from a developer making a change to that being live in production is around one hour.

6.3 Monitoring and logging

The solution is monitored using Azures Application Insights, Pingdom, Raygun and Azure Storage.

Application Insight gives insights into hardware utilization and provides triggers for auto-scaling new instances. It is possible to profile a live application through Application Insights to fine-tune the code for better performance.

Pingdom is used to monitor that the website is accessible and pings the site from different geographical regions to ensure that all parts of the world can access it. Alerts from Pingdom are sent both to Optimizelys Managed Services and their incident team as well as the project development team.

Raygun is used to monitor for application errors. Such errors and all details about them are sent to Raygun which notifies the development team.

Azure Storage is used to store application logs in a durable and searchable fashion. This is crucial for troubleshooting application issues.