HEADLESS ECOMMERCE

How does it work?

DIVANTE
eCOMMERCE SOFTWARE HOUSE

Introduction

Headless Web Apps, headless eCommerce - these concepts are relatively new to software development. Roughly speaking, to go headless means to separate the UI (frontend) from the application logic (backend).

This simple application of the classic "Command and Conquer" strategy opens a whole new spectrum of advantages and challenges for software projects.

Usually, a Headless eCommerce app is a set of back-end services (CMS, Order Management, Stock Management) and a single, lean front-end application. The front-end application integrates all the services and features into one coherent user environment.

First, by separating the concerns, software engagement can be simplified. The UI can be managed by a different team, utilizing different skills and technology stacks than the back-end technologies. That's one possible path to operational and organizational optimization.

Monolithic software projects - like a classical eCommerce platform - with time, become more and more challenging to maintain and develop. Nowadays, with Microservices Architecture based projects, companies are able to perform a few dozen deployments per week - even per day. Could you imagine this situation with your project five years ago?

Headless Apps are easier to maintain and test (as the UI and Backend operate on well-established service contracts).

Front-end development is hot right now. Each week you can find new JS frameworks - and there are some strong leaders (Vue.js, React, Angular) that have been with us for quite some time. Material design is the new industry standard for the UI.

Going headless means investing in the best UX and modern frontend technologies. If you already have a platform - integrated with ERP, WMS, CMS - going headless can be one way to modernize your existing platform without stepping into time-consuming and costly integration projects.

Finally, you can find best-in-class parts for your eCommerce platform. Need a flexible CMS? Why not go with Contentful? Do you want to optimize conversion rates on search pages? Algolia is ideal. The Headless and Microservices approach allows you to integrate these separate apps to provide the best possible user experience.

FASTEN YOUR SEAT BELTS, WE'RE GOING FOR A QUICK TOUR ON HOW YOU CAN BUILD HEADLESS ECOMMERCE!

SEE THE FULL VIDEO ABOUT MAGENTO MIGRATION PROCESS

Practice: Headless Fast track

Headless eCommerce doesn't have to be all about revolutionary changes. From our experience, it's more about evolutionary, fast-paced changes. Let's see how to approach it.

Migration or innovation?

Not so long ago, Magento's VP Strategy - Peter Sheldon wrote a brilliant blog post about the Business value of Progressive Web Apps. It's hard not to agree with the reasons he gave for investing in PWAs next year.

According to our last report on eCommerce Trends for 2018, Progressive Apps are a new Mobile Standard. It makes little or no sense to build, maintain and promote the install base for separate, dedicated apps. It's a no-brainer to do it once, provide virtually the same experience as native apps, and leverage the same marketing budgets you've already spent on promoting the core e-shop.

We believe this is the reason our Vue Storefront (the first open-source Progressive Web App for eCommerce) is doing well—with about 300 contributors across the globe and 12 enterprise partners. Right now, with version 1.0, we're ready for the first commercial implementations!

Many clients we've been speaking with are considering eCommerce replatforming next year. Switching from Magento 1 to Magento 2 is one of the most popular scenarios. Of course, each project is different, but usually, the replatforming process takes a long time and quite a significant budget (50–100% of initial implementation is standard).

Therefore, it's always good to do some ROI calculations after spending 50–150 thousand for having pretty much the same features and layouts but on the newer version of the platform.

In our experience, most of the implementation budget is spent on the integration part of the project (ERP, CRM, WMS…) - and this is also the riskiest part. On the other hand, **the most significant value (Conversion Rate optimization, time spent on site) can be achieved by investing in the frontend** - the User Experience.

Implementing a PWA before the migration is another thing you should consider. You can use Headless as a bridge that can work with both the legacy platform (e.g. Magento 1) and the new one (e.g., Magento 2). If you have your eCommerce up and running, integrated with all your back-end systems, maybe it's worth considering adding PWA features by simply connecting the frontend with your existing framework. By doing so, you'll be able

to update the platform however you want in the future while keeping the modern, progressive front-end layer in place. This is the value of the Headless approach.

I don't mean to convince you NOT to re-platform. It can be a significant and necessary change for many reasons, starting with new back-end features and security. What I would instead suggest is to consider phasing - **first investing in front-end and business value, and keeping the operations safe and sound.**

Quick wins

How to start with Progressive Web Apps without taking significant risks and re-designing the whole website? **Start small, do a PoC**. Here are some ideas on where to begin.

Landing Page

For those of you selling apparel or sports items, it may be an option to build a landing page for a specific category of products and make this section a Progressive Web App. Or maybe your brand is running some marketing campaigns which require a dedicated landing page? It's a good idea to start small with PWAs on a subdomain/ subdirectory.

Test it in a way that guarantees you safe returns without spending significant development budgets. After testing it live and getting higher conversion rates, substantial evidence is on the table which makes the whole redesign/migration more manageable and less risky!

Soft launch

Another way is to build the new version and run it in parallel with the current one, aka a soft launch. You can run two parallel shops - your existing one and the PWA, and even do some A/B tests. All the orders will be synchronized back to your current CMS (and then to ERP if you have it integrated), while all the products and categories are synchronized to the PWA.

Exclusive app for travelers or loyalty program

If you're an airline with a webshop with gifts for travelers, a booking engine or any other service users may want to use on the plane or under ground, PWAs have the considerable advantage of working offline. Maybe you're thinking about an exclusive shop for loyal customers ("get special prices for being with us")? Such a unique storefront with some premium goods - that otherwise, users are not willing to consider - **is an excellent way to engage customers, and maybe it's a good idea for a PWA quick win too?**

Partial redesign

The last idea is to make part of your webshop a PWA. Let's say, make the catalog (Category Page, Product Page) run smoother and be available offline. It's doable (for example via HTTP Proxy composition or by using the https://micro-frontends.org/ approach). All the checkout and pricing stuff will be resolved to your legacy app for the time being.

Community and Open Source accelerators

The good news is, if you would to build a Progressive Web App or implement general use Headless eCommerce - most of the building blocks are already there. Most of them are Open Source. The majority of the time required to put it all together and run a Headless site is spent on integrations.

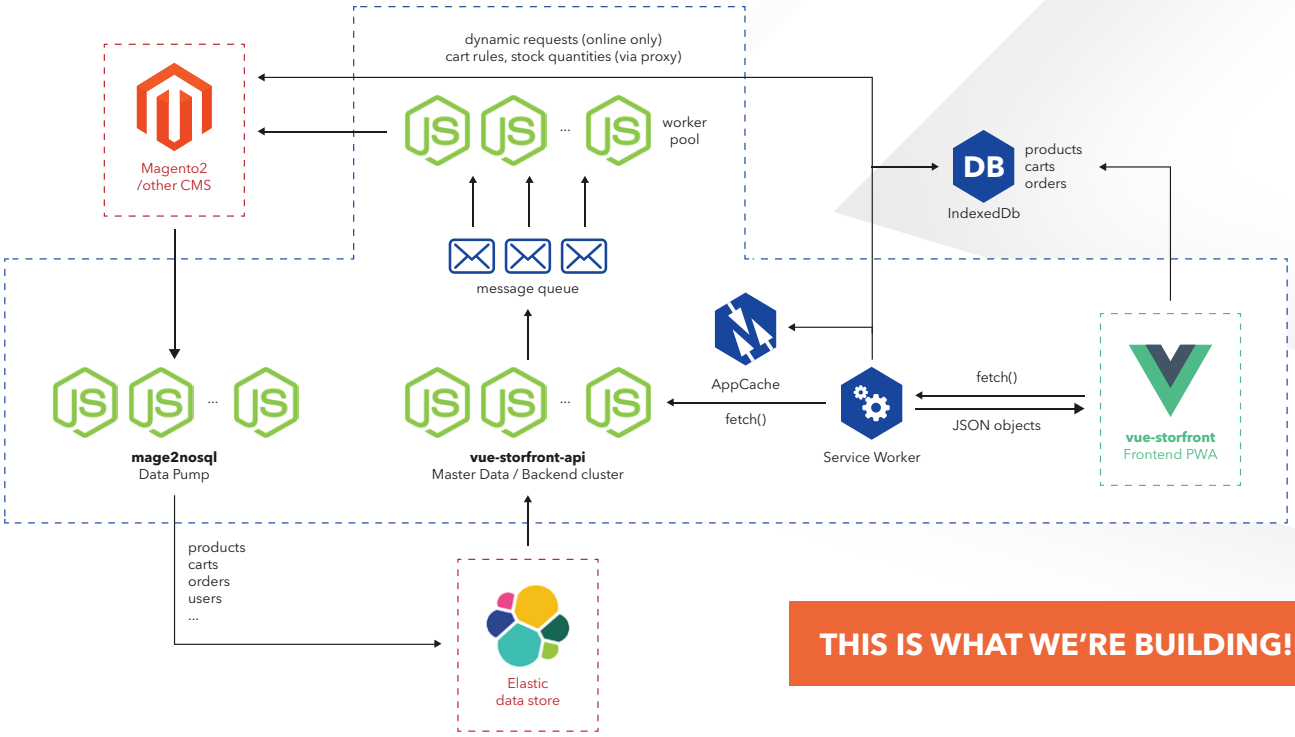
Headless apps can be built in many different ways. Usually, the team starts with some frontend application - written using modern JS frameworks (Vue.js, React, Angular). The second team (or the same team if you've got Full stack developers :)) exposes some API endpoints.

The most commonly used backend systems integrated within Headless eCommerce are:

- eCommerce platforms - for products, customers and order management,
- CMS - for blocks, statics, marketing and navigation,
- PIMs and DAMs - Digital Assets Management systems (for product marketing data)

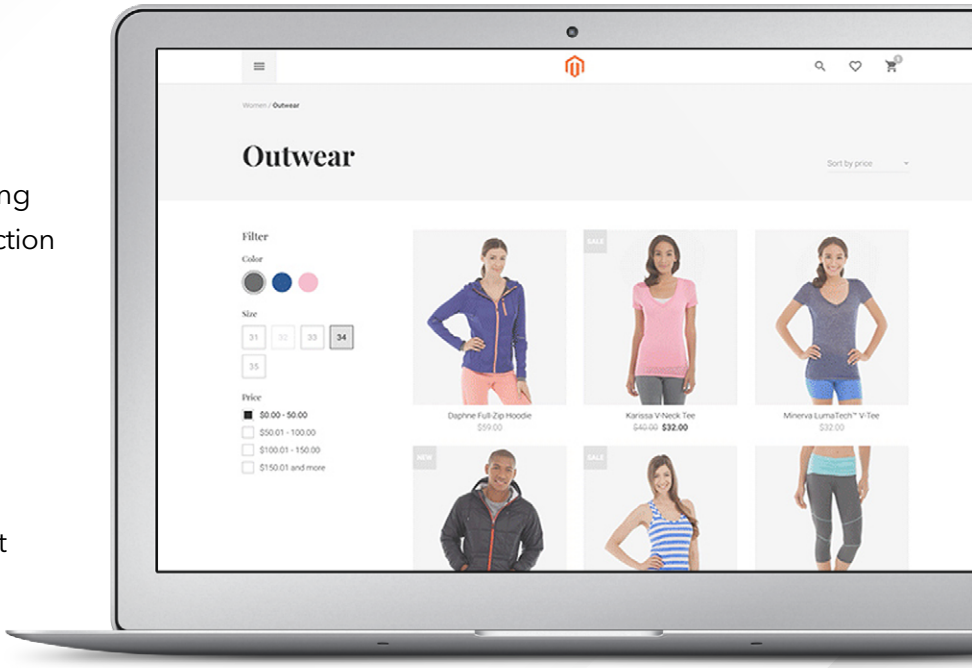
Vue Storefront - the first Progressive Web App for eCommerce

Vue Storefront is a standalone PWA storefront for your eCommerce, able to connect with any eCommerce backend (eg. Magento, Pimcore, Prestashop or Shopware) through the API.



THIS IS WHAT WE'RE BUILDING!

Vue Storefront is and always will be open source.



Anyone can use and support the project; we want it to be a tool for the improvement of the shopping experience.Theprojectisintheproduction ready phase.

Vue Storefront was designed as a framework. Developers just create their own theme and extensions to apply a 100% customizable look and feel, without modifying the core.
We've applied design system and component-based design patterns to keep the layouts (which are also open source) maintainable and reusable.

The project also has a strong community around it. Vuestorefront is started by Divante and released on an open-source licence (MIT), it is now maintained and developed by 27 active core developers across the world. 140 more developers have joined the Slack community and work on different projects based on Vue Storefront. The official partnership network includes companies from Argentina, the United States, Australia, the United Kingdom and Poland. The project has already attracted its first clients - prominent brands from fashion, jewelry and the sports industry.

Technologies used:

- Vue.js + Node.js,
- Webpack + Babel,
- Elastic Search,
- Magento2, Magento1, Pimcore bindings via API.

More information:

- Official website - https://www.vuestorefront.io/
- Official Github repository: https://github.com/DivanteLtd/vue-storefront

Algolia

Search has long surpassed the mandatory checkbox of an e-commerce site architecture. The technology has matured to power all forms of discovery: navigation, recommendations and content, as well as powerful analytics features.

Search is not simply about helping the user find what they are looking for. Today's consumers come to your site for experiences, and it is inspiring experiences that will keep them coming back for more.

Algolia is a hosted search API – also available as a Magento extension – that provides e-commerce technology teams with the building blocks necessary to create advanced search without having specialized search knowledge.

Seamless implementation is secured with standard APIs and advanced front-end libraries. A dedicated, high-security infrastructure is maintained by security experts, ensuring significant engineering time savings.

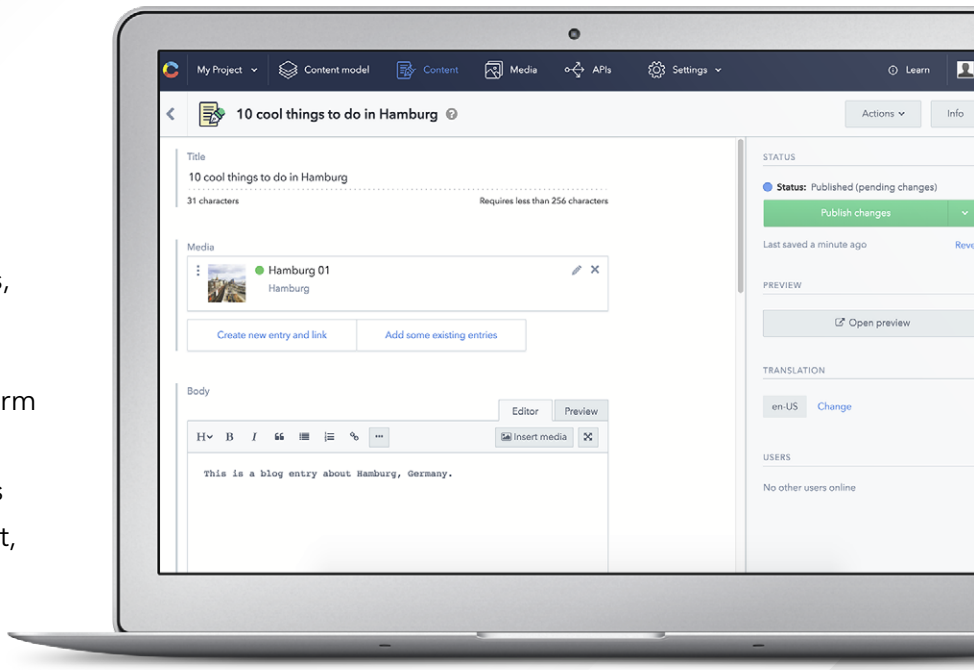
Algolia was built from the ground up for speed and relevance that exceed the expectations of users "spoiled" by Google and Amazon.

On the business side, powerful built-in search analytics are a way to understand and accelerate the ROI of search. By using specialized search metrics–like popular searches, no result rates, or click and search-to-conversion rates–you can ensure that your search implementation is optimized to meet your business goals.

More information:

- Official website: https://www.algolia.com/ecommerce
- Github repository: https://github.com/algolia
- The Architecture Of Algolia's Distributed Search Network
- "Inside the Engine" Series

Contentful



Contentful is the content engine of the modern digital factory. Contentful provides content infrastructure for digital teams to power content in websites, apps, and devices.

Unlike a traditional CMS, the platform is purpose-built to integrate with the modern software stack. It offers a central hub for structured content, powerful management and delivery APIs, and a customizable web app that enable developers and content creators to ship their products faster.

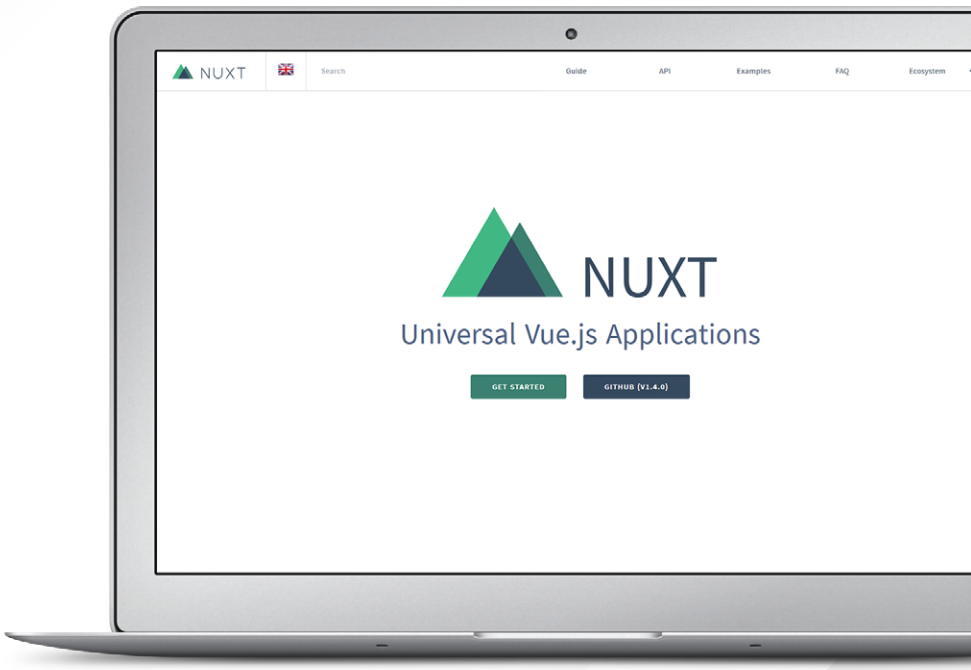
Contentful supports the team's development style and workflow. Developers can easily integrate it with existing websites and apps using their favorite languages and frameworks – or quickly spin up a new project.

Flexible content modeling and an intuitive UI give editors the structure and autonomy they require to create and iterate content as often as needed.

More information:

- Official website: https://www.contentful.com/
- Github repository: https://github.com/contentful

Nuxt.js framework



Nuxt.js is a framework for creating Universal Vue.js Applications.

Its main scope is UI rendering while abstracting away the client/server distribution. Nuxt.js presets all the configuration needed to make your development of a Vue.js Application Server Rendered more enjoyable.

In addition, the authors provide another deployment option called: nuxt generate. It will build a Static Generated Vue.js Application. We believe that option could be the next big step in the development of Web Applications with microservices.

As a framework, Nuxt.js comes with a lot of features to help you in your development between the client side and the server side, such as Asynchronous Data, Middleware, Layouts, etc.

Nuxt.js can be used to build the site frontend - and integrate with other tools mentioned here - like getting data from a Headless CMS or including some components of a PWA eCommerce platform.

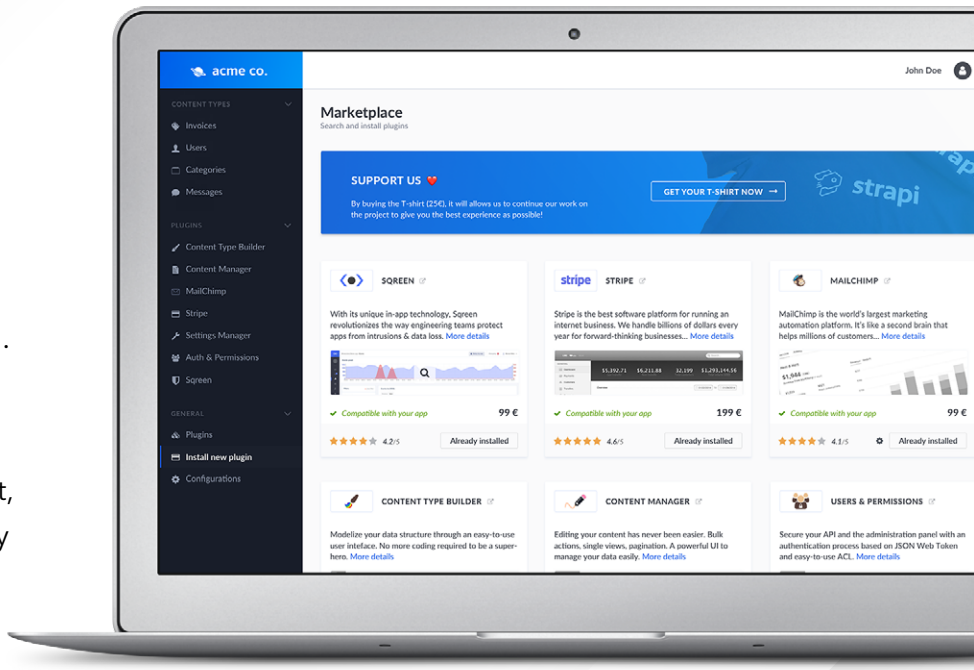
Technologies used:

- Node.js,
- Vue.js 2

More information:

- Official website: https://nuxtjs.org/
- Github repository: https://github.com/nuxt/nuxt.js

Strapi Headless CMS



Strapi is a Content Management Framework (CMF) that can be easily connected to Vue Storefront, a Nuxt.js project or any other custom eCommerce frontend via API. You can manage the Content classes in the backend.

Key features:

- Modern Admin Panel: Elegant, entirely customizable and fully extensible admin panel.
- Secure by default: Reusable policies, CSRF, CORS, P3P, Xframe, XSS, and more.
- Plugin Oriented: Install authorization system, content management, custom plugins, and more, in seconds.
- Blazing Fast: Built on top of Node.js, Strapi delivers amazing performance.
- Front-end Agnostic: Use any front-end frameworks (React, Vue, Angular, etc.), mobile apps, or even IoT.
- Powerful CLI: Scaffold projects and APIs on the fly.
- SQL & NoSQL databases: Works with Mongo as a main database, also supports Postgres, MySQL, etc.

Development starts with the CLI tool used to generate the data models and API's for them. It's very similar to the Ruby on Rails way of scaffolding. The models are then accessible via the admin panel and can be modified without any development skills.

Then, all data objects are just exposed via generated API's and can be used within the eCommerce frontend - for example, for managing Sliders, banners or navigation items.

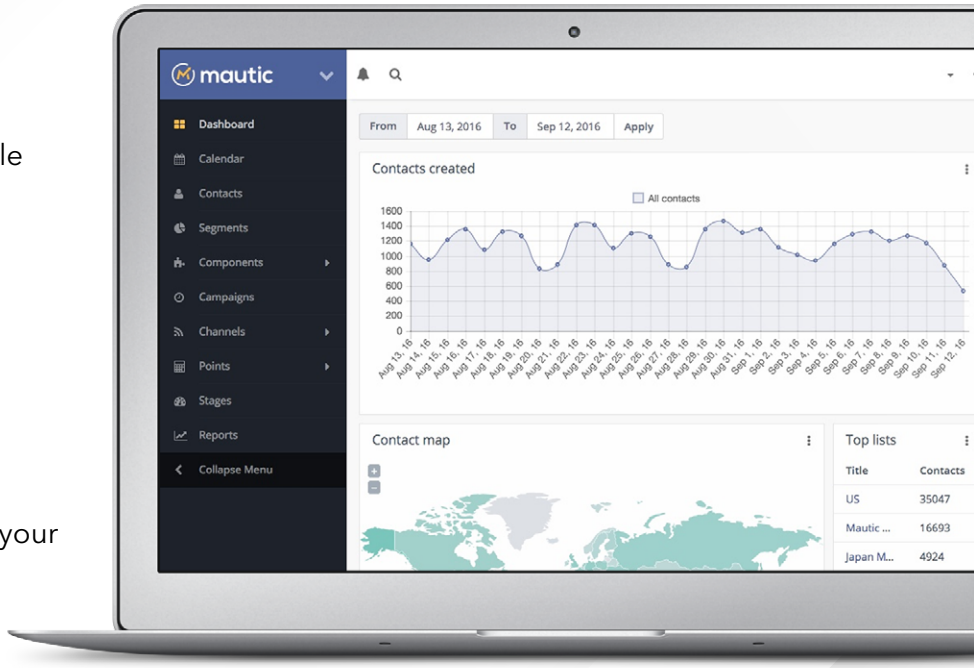
Technologies used:

- Node.js

More information:

- Official website: http://strapi.io
- Github repository: https://github.com/strapi

Mautic



This software has been written to run on most standard hosting environments and requires very little in the way of time to install.

Mautic is open-source software, customizable to a user's particular needs and situation.

You can finally use a marketing automation platform the way your business runs instead of changing your business to fit a piece of software.

There is a service available, Mautic.com, which gives you a free hosted Mautic platform in short time.

This free, hosted solution offers a few different options compared to the version you download and configure on our own server, but gives you a version of Mautic instantly available with no set-up or servers to configure.

Technologies used:

- Symfony

More information:

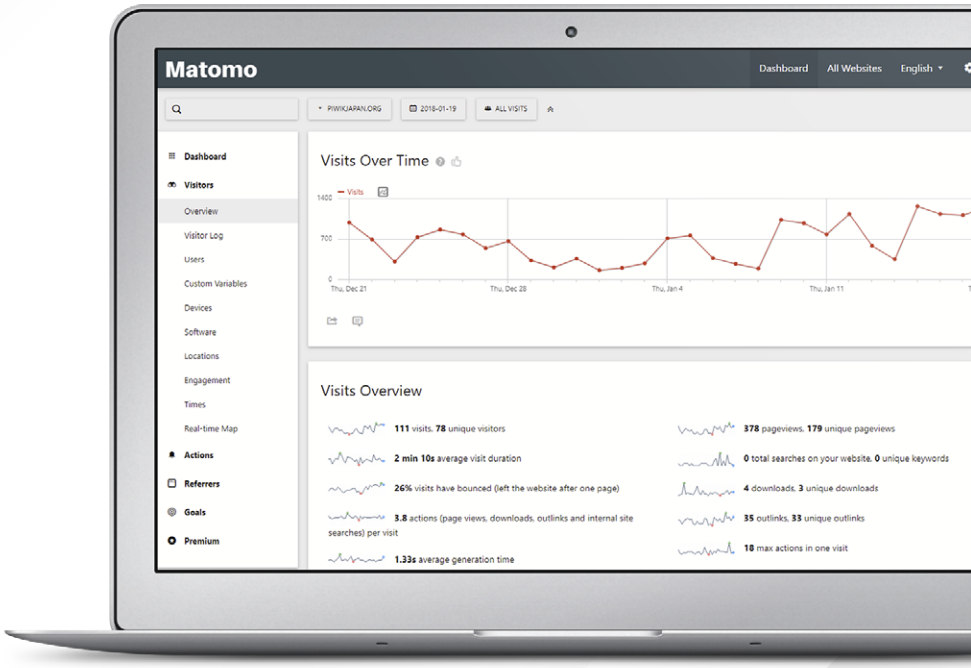
- Official website: https://www.mautic.org/
- Github repository: https://github.com/mautic/mautic

Matomo

Matomo (formerly Piwik) is an open analytics platform and can be used by individuals, companies and governments.

Matomo helps gather and analyze important information about users. Administrators can track Key Performance Indicators such as visits, goal conversion rates, downloads, keywords, and many more.

Matomo also enables importing and analysing your web server logs.



Matomo enables the analysis of visit details, including IP address, URL, user agent, referrer URL, search keywords, campaign information, and more.

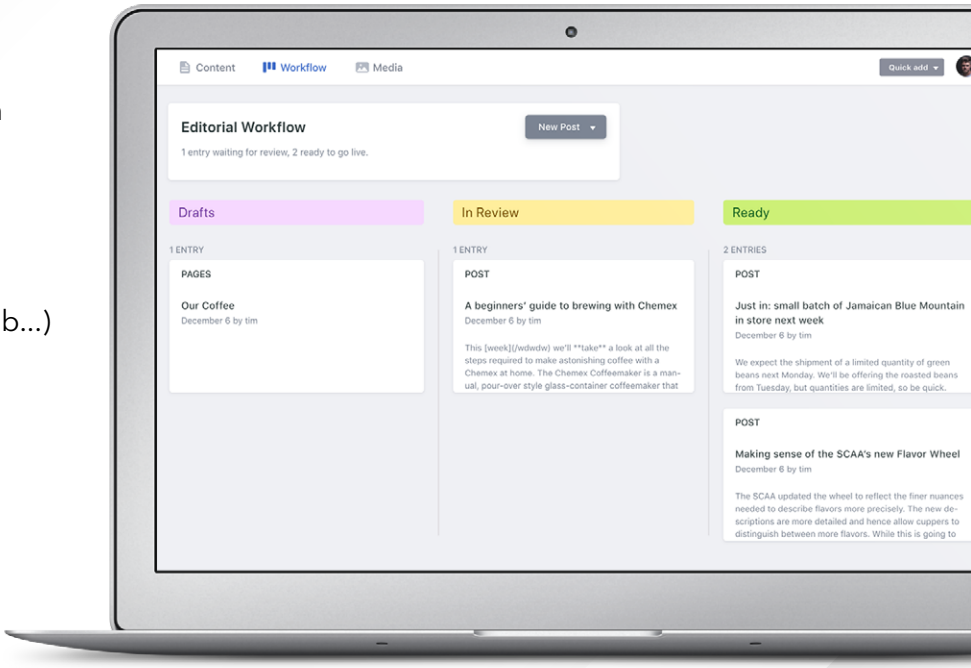
Technologies used:

- PHP/MySQL

More information:

- Official website: https://matomo.org/
- Github repository: https://github.com/matomo-org

Netflify CMS



Netflify CMS is another "Headless CMS" but with a different approach to Strap.

Here you've got content management tools that use Gitflow (based on Github, Gitlab...) to manage the content, which can be easily merged / diffed / versioned or deployed.

Netlify CMS is adaptable to a wide variety of projects. The only hard requirement is that your site content must be written in markdown, JSON, YAML, or TOML files, and stored in a repo on GitHub. (If you're partial to another Git hosting service, check out the PRs in progress for GitLab and Bitbucket support.)

Data structures are modeled using YAML config files, and rich content is managed via Markdown files (like on Github).

The data backend is git itself - so you just configure your Github repository and no other database or data storage is required. It's really useful because it can be easily integrated with the deployment workflow.

Technologies used:

- Node.js

More information:

- Official website: https://www.netlifycms.org/
- Github repository: https://github.com/netlify/netlify-cms

Pimcore PIM and DAM

Pimcore is an Enterprise Grade Open-Source, PHP-based CMS, PIM and DAM solution.

With it's very elastic data architecture (everything based on Object - whose structure can be visually modeled) it's the vgo-to solution for any application that requires data management.

Most of the operations related to data modelling and management are done via the useful Web Panel.

Data Modelling

Flexible data modeling is key to Pimcore PIM/MDM. It includes a web-based data modeling engine to create a new product data model within minutes.

Data Management

The user-friendly and consistent organization, aggregation, classification, and translation of rich product information based on a flexible and agile data model is key to Pimcore PIM/MDM.

Data Integration & Delivery

Pimcore enables you to take charge of data integration through the easy import and export of data between Pimcore and external systems.

Asset Portal

Pimcore Asset Portal is an image library for marketing teams.
It's a commercial extension to Pimcore.

Digital Asset Management

Pimcore provides a powerful central repository for any type of digital asset and its meta-data.

Digital Asset Delivery

Easy integration with other systems make Pimcore the ideal enterprise content hub.

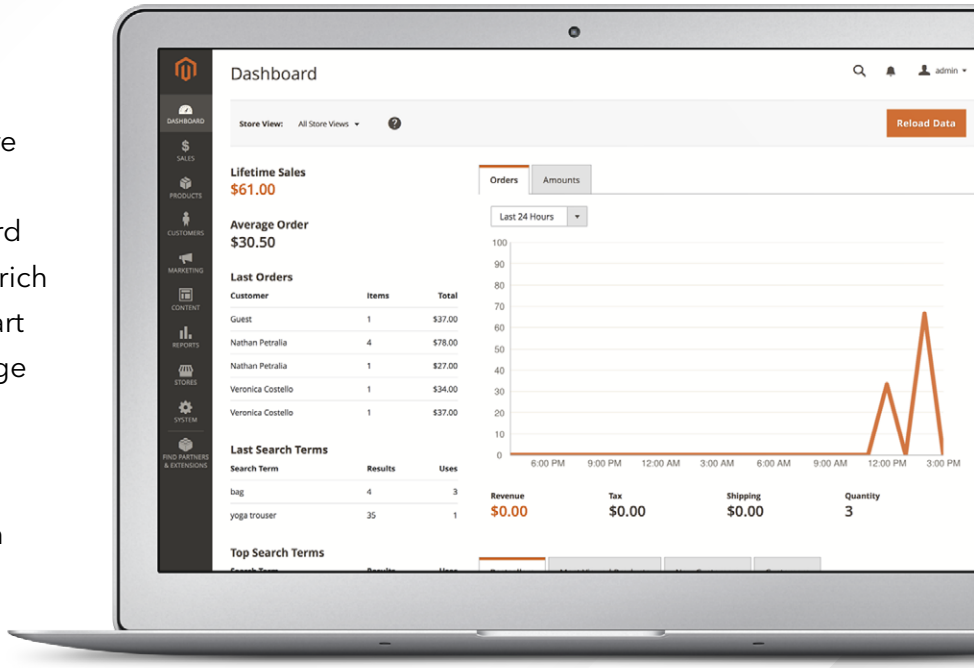
Among other features, Pimcore supports very useful features regarding eCommerce:

- data versioning,
- live editing,
- eCommerce framework - payments, order management.
- multimedia and video file management (DAM),
- multichannel integrations,
- workflow management,
- extensible REST API.

More information:

- Official website: http://pimcore.org
- Github repository: https://github.com/pimcore/pimcore
- Dev docs: https://pimcore.com/docs/5.x/index.html
- PimOnCloud: Divante is offering the http://PimOnCloud.com service which allows you to start a Pimcore instance in the cloud without any major upfront costs.

Magento



Magento is kind of the Industry Standard for eCommerce with more than 25% of total eCommerces running on it. To be honest, it is hard to find a more flexible and feature-rich platform. However, the frontend part is currently no longer bleeding edge (phtml templates and CSS).

Magento is taking some serious steps to improve the frontend, with Magneto PWA Studio planned for later this year.

The Magento platform is great choice for going Headless with it's powerful API. Vue Storefront supports Magento 2 and it's going to support Magento 1 in the coming months.

More information:

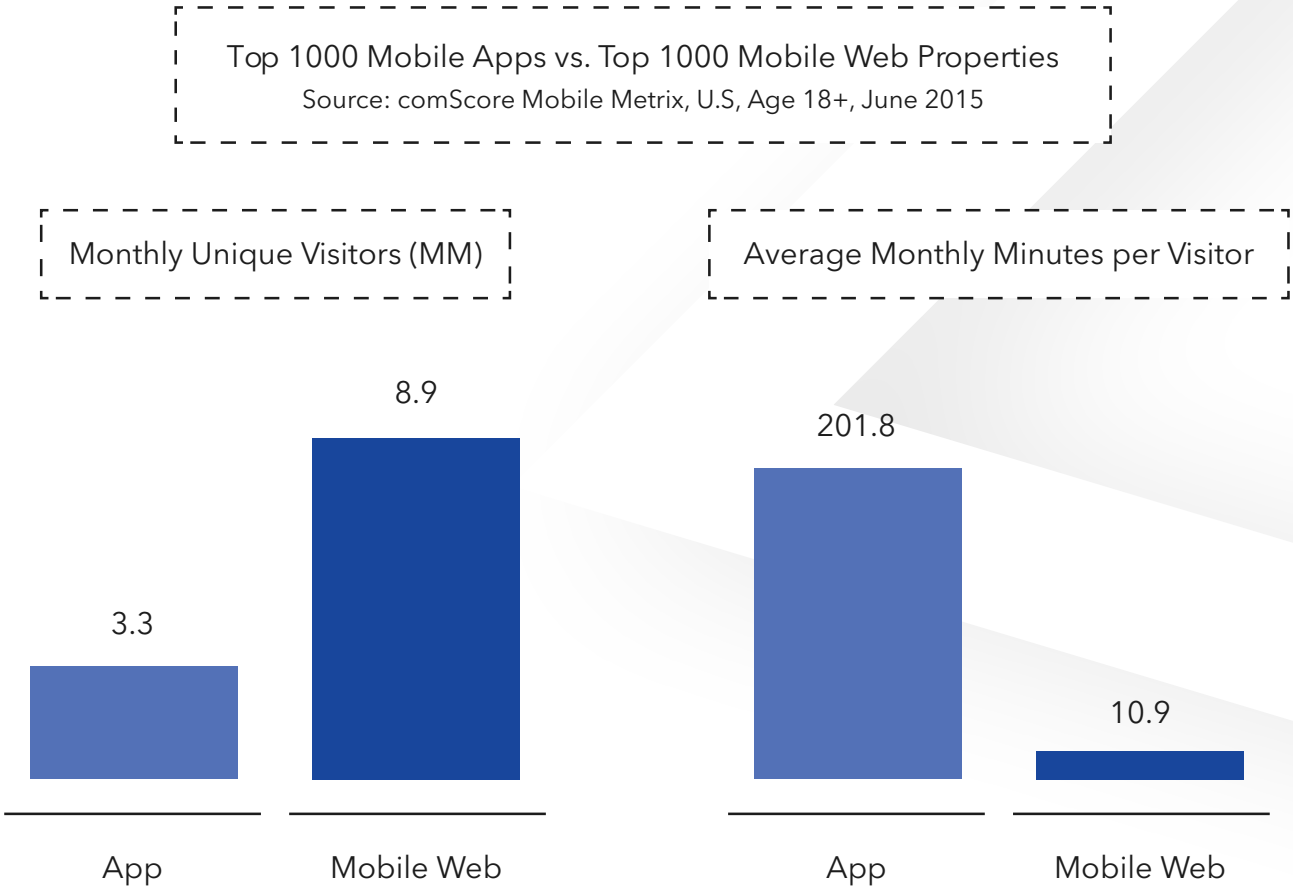
- Official website: https://magento.com/
- Github repository: https://github.com/magento/
- https://www.yireo.com/blog/1851-headless-with-magento-2
- https://github.com/sitewards/headless-magento2-resources

Progressive Web Apps

PWA - Progressive Web Application - is becoming a promising new concept for web development that can rapidly improve the web experience. Although a headless app doesn't automatically mean progressive, as both techniques are gaining momentum, it's good to understand what Progressive Web Apps bring to your eCommerce project.

All the signs show us that eCommerce will be about mobile in the foreseeable future. It's surprising, but when you take a look at the data, mobile websites are roughly 3 times more popular than Mobile Apps. The distribution model - without all these app stores and installations - is much more natural. By contrast, the mobile app experience is much better than current mobile websites; users spend more time and money in apps.

It would be great to get the benefits of both platforms and the **Progressive Web Apps idea is all about that.**



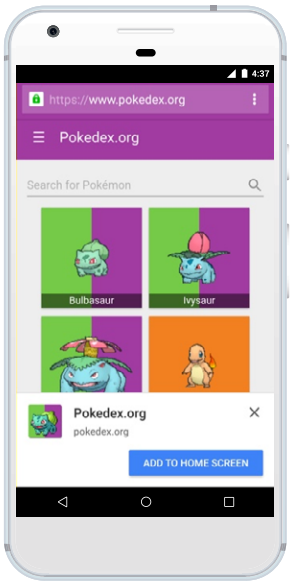
PWA for shopping experience

Let's start with the explanation of how exactly a PWA can improve the eCommerce shopping experience. On our list of key features you can find:

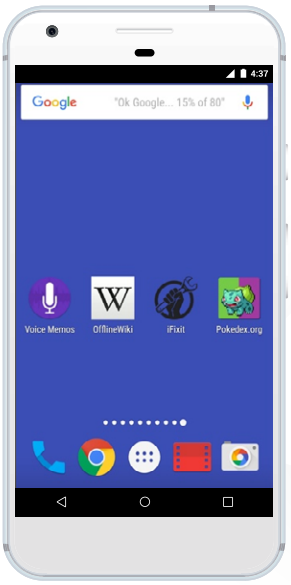
- **Super fast frontend** where the customer has a feeling of immediate interaction, similar to that on well-built mobile applications;
- **Overloads safety** for front-end, which is crucial for every business, especially if you have seasonal traffic peaks;
- **Off-line readiness**, where the online store can be browsed by customers with a weak or even no network connection;
- **Mobile-first experience**, with no reloads of pages and animations.

While working on product design, the most important thing for us is to ship faster browsing experiences; all the other improvements seem to be the result of speed improvements.

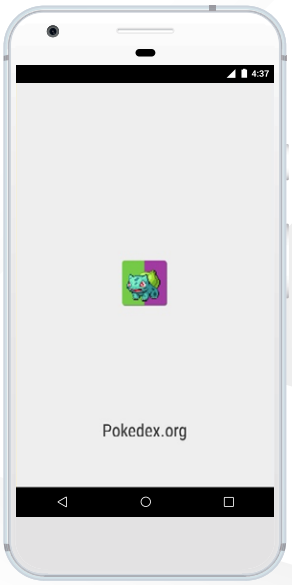
Here are several case studies showing great results after implementing PWAs.



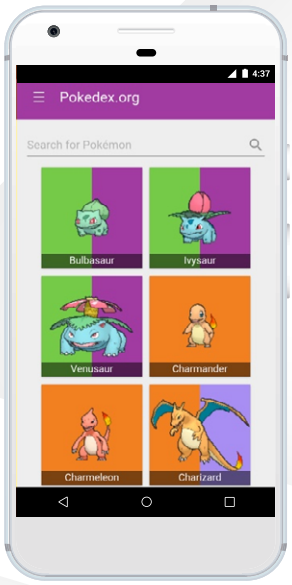
Web App install banner for engagement



Launch from user's home screen



Splash screen (Chrome for Android 47+)



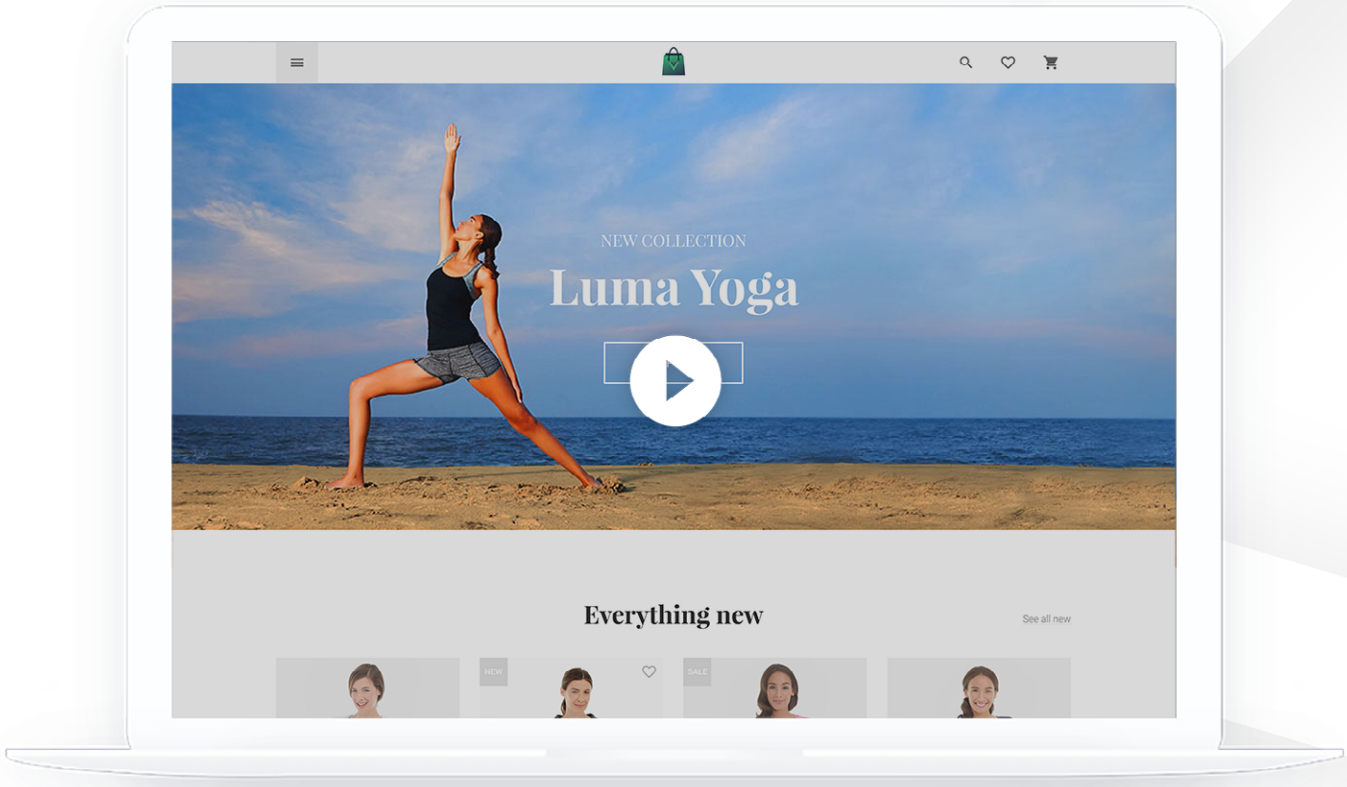
Works offline with Service Worker

By creating Progressive Web Apps, business clients pay just once, not having to build and maintain a dedicated mobile application for each platform (iOS, Android, Windows) separately. All major mobile platforms are now supporting the standard, and we believe it's the future of app development. This is what users want, and it's something we can leverage to improve eCommerce conversion rate.

DIVANTE
eCOMMERCE SOFTWARE HOUSE

VUE STOREFRONT

PWA for your eCommerce



Know more

Theory

To get the whole picture of how headless apps are designed and integrated, let's take a quick tour of the Microservices architecture and modern deployment techniques!

Microservices

Microservice architecture structures the application as a set of loosely coupled, collaborating services. Each service implements a set of related functions. For example, an application might consist of services such as an order management service, an inventory management service, etc.

Services communicate using protocols such as HTTP/REST or (a less popular approach) using an asynchronous approach like AMQP. Services can be developed as separate applications and deployed independently. Data consistency is maintained using an event-driven architecture because each service should have its own database in order to be decoupled from other services.

The most common forces dictating the Microservice approach: (NOTE: According to: http://microservices.io/patterns/microservices.html)

- Multiple teams of developers working on a single application.
- System must be easy to understand and maintain/modify, no matter the number of changes deployed.
- Urgency for new team members to be productive.
- Need for continuous deployment (although possible to achieve with monolith design, microservices include some features of devops approach by design).
- Scalability requirements that require running your application across server clusters.
- Desire to adopt emerging technologies (new programming languages, etc.) without major risks.

The assumptions of the orthogonal architecture followed by microservices architects implies the following benefits:

- Each microservice could be deployed separately and without shutting down the whole system.
- Each microservice can be developed using different technologies while allowing them to publish HTTP end-points (golang-based services can interoperate with PHP, Java …).
- By defining strict protocols (API), services are easy to test and extend into the future.
- Microservices can be easily hosted in the cloud, Docker environments, or any other server platform, and can be very easily scaled as each service can live on its own server(s), VPS(es), etc.
- The services are easy to replace.
- Services are organized around capabilities, e.g., UI, front-end, recommendation, logistics, billing, etc.

The scalability and deployment processes of microservice-based systems can be much easier to automate compared to monolithic architectures. The Devops approach to infrastructure, along with Cloud services, is commonly in use.

The examples of Spotify and Netflix (NOTE: https://www.nginx.com/blog/microservices-at-netflix-architectural-best-practices/) inspire IT engineers to implement continuous delivery and monitoring.

Dockerization of IT environments, monitoring tools and DevOps tools (Ansible, Chef, Puppet and others) can take your development team to the the next level of effectiveness.

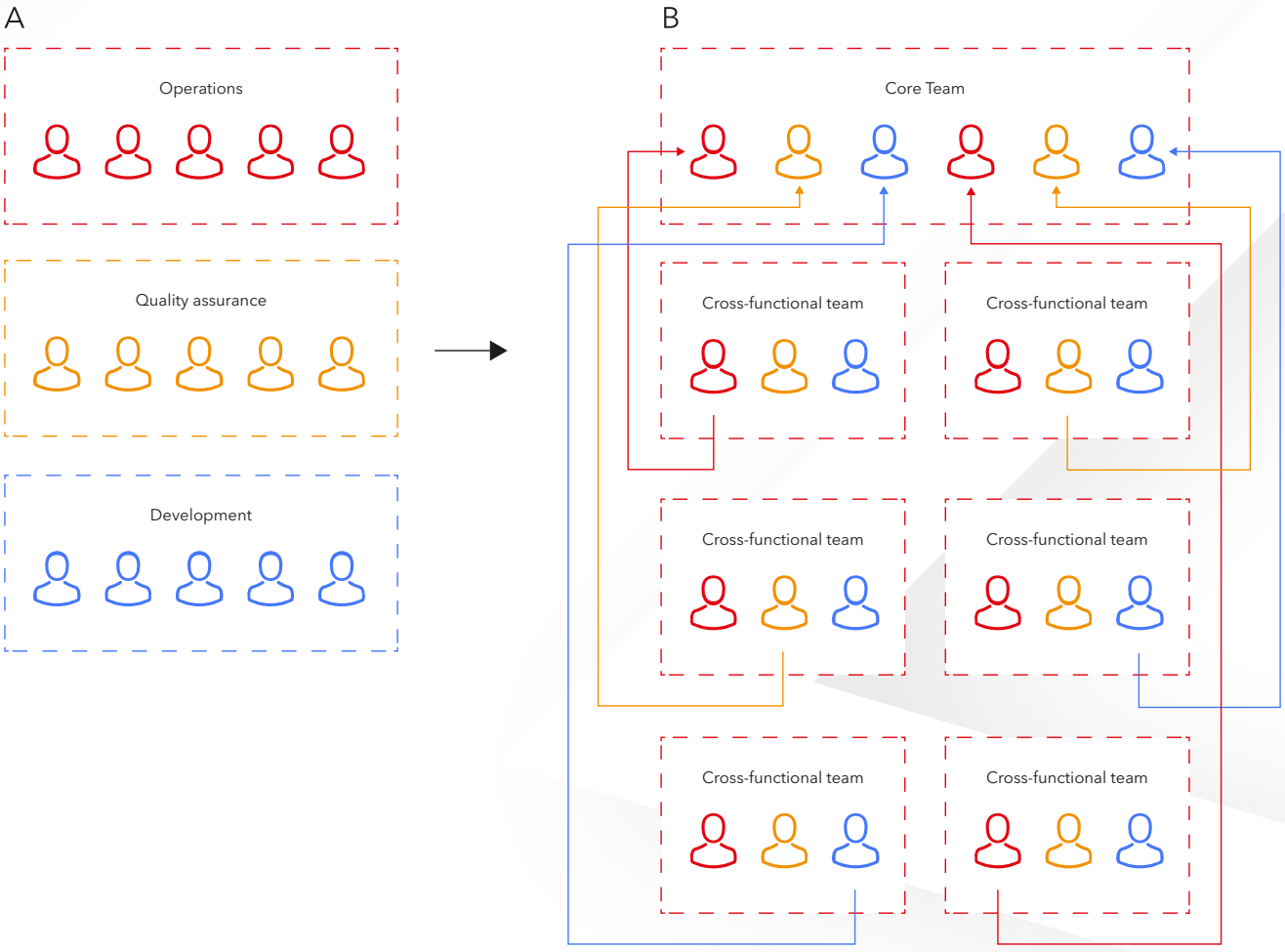


Fig. 2: A microservice approach encourages enterprises to become more agile, with cross-functional teams responsible for each service. Implementing such a company structure, as in Spotify or Netflix, can allow you to adopt and test new ideas quickly, and build strong ownership feelings across the teams.

The criticism

The microservice approach is subject to criticism for a number of issues:

- The architecture introduces additional complexity and new problems to deal with, such as network latency, message formats, load balancing, fault tolerance and monitoring. Ignoring one of these belongs to the „fallacies of distributed computing".
- Automation is possible but in the simplest cases, tests and deployments may be more complicated than with the monolithic approach.
- Moving responsibilities between services is difficult. It may involve communication between different teams, rewriting the functionality in another language or fitting it into a different infrastructure. On the other hand, it's easy to test contracts between services after such changes.
- Starting with the microservices approach from the beginning can lead to too many services, whereas the alternative of internal modularization may lead to a simpler design.

API Gateways

With the microservices approach, it's quite easy to make internal network communication very talkative. Nowadays, when 10G network connections are standard in data-centers, there may be nothing wrong with that. But when it comes to communication between your mobile app and backend services, you might want to compress as much information as possible into one request.

The second reason to criticise microservices might be a challenge with additional sub-service calls like authorization, filtering, etc.

To overcome the mentioned obstacles, we can use the API Gateway approach. It means you can compile several microservices using one facade. It combines multiple responses from internal sub-services into a single response. With almost no business logic included, gateways are an easy and safe choice to optimize communication between frontend and backend or between different backend systems.

In that way, your Frontend App treats the API Gateway as a source of truth and as a go-to data market to aggregate all the microservice calls. It simplifies the data flow and frontend application itself. There are some ready made solutions that don't require any additional coding - like Google's ApiGee or Amazon's API Gateway services.

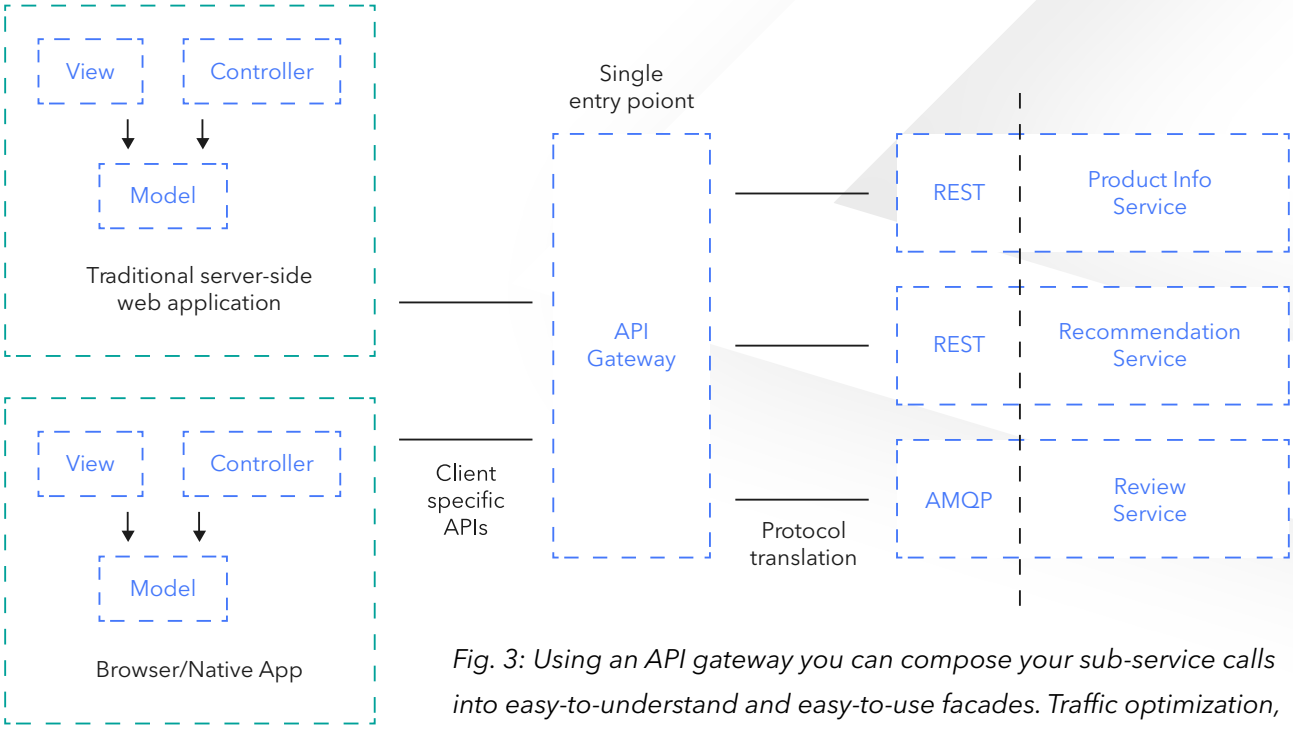


Fig. 3: Using an API gateway you can compose your sub-service calls into easy-to-understand and easy-to-use facades. Traffic optimization, caching and authorization are additional benefits of such an approach.

The API Gateway - which is an implementation of classic Proxy patterns - can provide a caching mechanism as well (even using a vanilla-Varnish cache layer without additional development effort). With this feature alone, using cloud approaches (like Amazon solutions), can scale APIs and services very easily.

Additionally, you can provide common authorization layers for all services behind the gateway. For example - that's how Amazon API Gateway Service (NOTE: https://aws.amazon.com/api-gateway/) + Amazon Cogito (NOTE: http://docs.aws.amazon.com/cognito/latest/developerguide/authentication-flow.html) work.

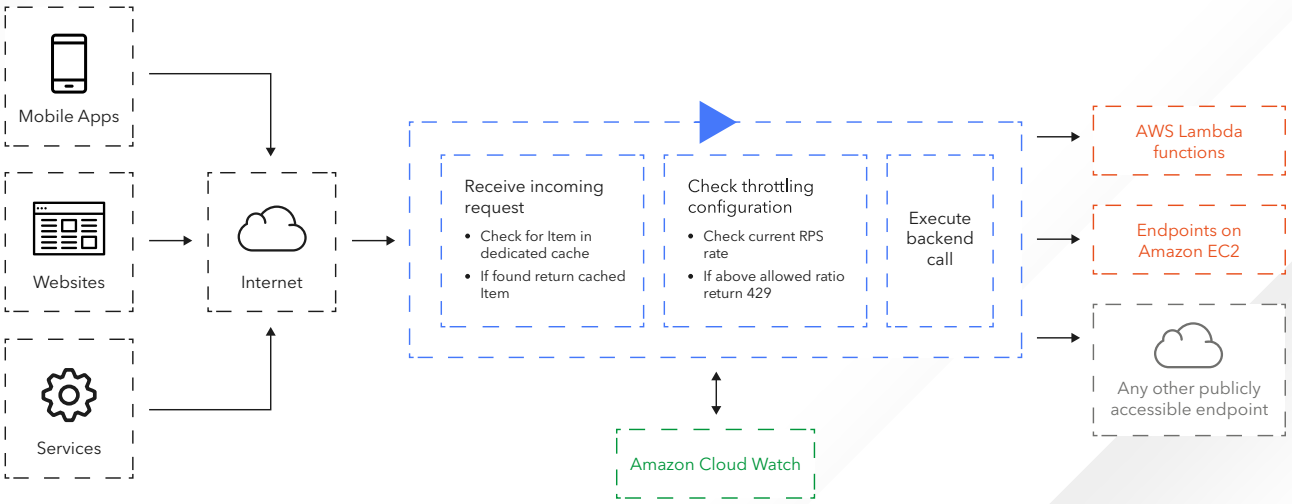


Fig. 4: Amazon API Gateway request workflow https://aws.amazon.com/api-gateway/details/). Amazon gateway supports caching and authorization features in spite of your web-service internals.

Swagger (NOTE: https://swaggerhub.com/blog/swaggerhub-feature/swagger-amazon-api-gateway-and-lambda/) can help you, once a Gateway has been built, with direct integration and support to Amazon services.

Backend for Frontends

A typical example of an API Gateway is the backend for frontends (BFF) pattern. It is about facades and compiling several microservices into optimized / device or channel-oriented API services.
Its microservice design pattern was proposed by Sam Newman of Thought Works (author of „Building Microservices"): to create single purpose edge APIs for frontends and other parties.

Creating such a facade-API brings at least two benefits to your application:

- If you manage to have some microservices behind your facade, you can avoid network latency - which is especially important on mobile devices. Using a facade, you can hide all network traffic between services executing the sub-calls in internal networks from the end-client.
- Then you can optimize your calls to be more compliant with a specific domain model. You can model the API structures by merging and distributing subsequent service calls instead of pushing this logic to the API client's code.

The diagram below shows a migration from a General Purpose API to a dedicated backend for frontends approach which integrates the sub-services into logic.

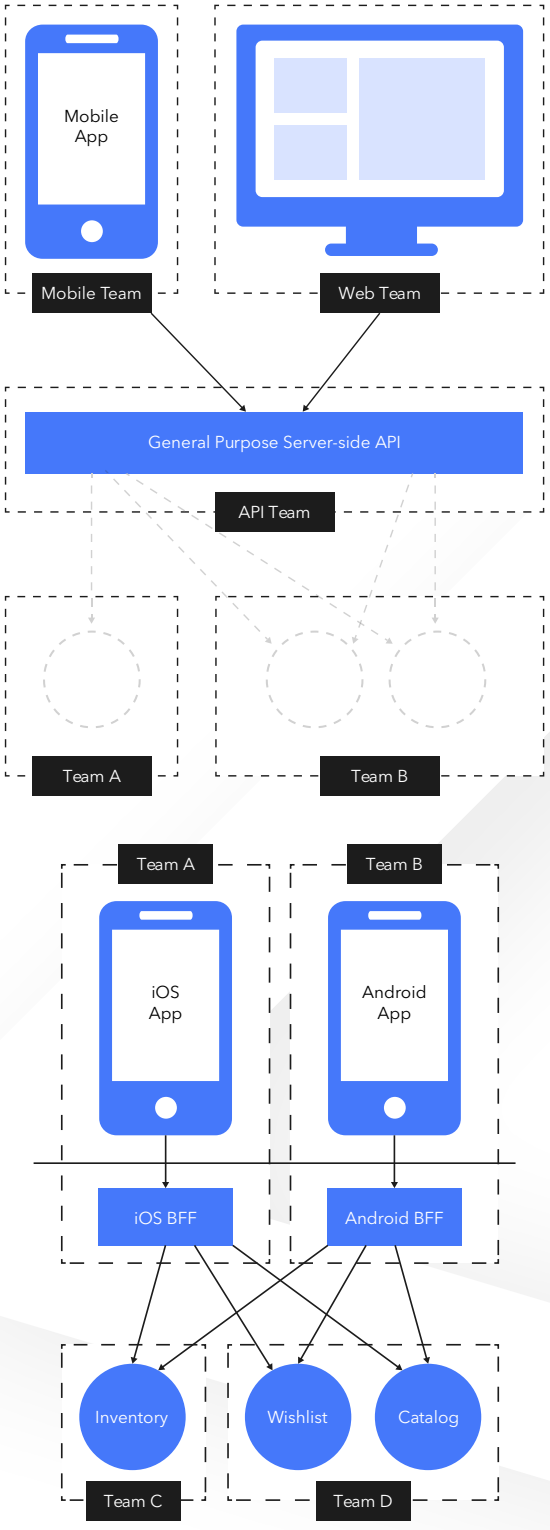


Fig. 5: The backend for frontends architecture is about minimizing the number of backend calls and optimizing the interfaces to a supported device.

There are many approaches to separate backend from frontends and roughly speaking, it always depends on the differences in data required by a specific frontend, or usage-patterns behind specific API clients. One can imagine a separate API for frontend, mobile apps - as well as separate interfaces for iOS and Android if there are any differences between these applications regarding how service calls are made or their respective data formats.

One of the concerns of having a single BFF per user interface is that you can end up with lots of code duplication between the BFFs themselves.

Pete Hodgson (ex. Thought Works) suggests that BFFs work best when organized around teams. The team structure should drive how many BFFs you have.

This is a pragmatic approach to not over-engineer your system but rather have one mobile API if you have one mobile team, etc.

It's then a common pattern to separate shared algorithms, models and code to separate the shared service or library used by frontend-related facades. Creating such duplications can be avoided.

Let me quote the conclusion on BFF presented by Sam Newman himself:

Backends For Frontends solve a pressing concern for mobile development when using microservices. In addition, they provide a compelling alternative to the general-purpose API backend, and many teams make use of them for purposes other than just mobile development. The simple act of limiting the number of consumers they support makes them much easier to work with and change, and helps teams developing customer-facing applications retain more autonomy.

(NOTE: http://samnewman.io/patterns/architectural/bff/)

Conclusion

Headless is one of the trends that is worth watching. Even if you have a large monolithic platform, give it a chance because it solves a lot of problems by breaking the system down into smaller components which are easier to maintain. Headless also gives us the possibility to use open-source tools in our architecture.

What's more, you should consider headless if you team is interested in microservices - it fits perfectly into your architecture. In one way or another, most enterprise systems support the headless approach right now.

PWA is a trend based on creating an application that gives a lot of benefits in the form of easier application distribution, lower cost of maintenance (one app on all platforms), and with a similar User Experience as mobile apps. Thanks to the headless approach, PWAs can use the latest JS frameworks like React, Vue and Angular so that it can be created faster and with higher quality.

If you want to know more
about Headless eCommerce,
contact us:

Piotr Karwatka, CTO Divante
pkarwatka@divante.co

This eBook was created by DIVANTE eCOMMERCE SOFTWARE HOUSE
in cooperation with Algolia and contentful