SaaS Web Application Development Principles to be Followed

written by

Anton Logvinenko

Web Solution Architect at MobiDev

Upwork 2016 Upwork 2017 Upwork 2018 Best Upwork Software Development Agency In Ukraine

COMPLEX SOFTWARE DEVELOPMENT

WITH A FOCUS ON INNOVATION

WEB & CLOUD INFRASTRUCTURE

AR & MOBILE APPS

IOT & HARDWARE INTEGRATION

DATA SCIENCE & MACHINE LEARNING

For Startups | For Emerging companies | For Enterprises

Guaranteed delivery
on time and on budget
No surprises

You can **adapt to evolving business needs and increase ROI** with our flexible, proven processes

Top US-level quality for **1/3 the price** to bring **3x features** to your product

350+ Products launched

100% Approval rating by Upwork

300+ English speaking professionals

Find more at **www.mobidev.biz**

info@mobidev.biz +1 888 380 0276

UNITED STATES OFFICE
Atlanta, Georgia

UNITED KINGDOM OFFICE
Sheffield

ENGINEERING OFFICES IN UKRAINE
Kharkiv, Chernivtsi, Mykolaiv

Table of Contents

If you're reading this article, it probably means that you aren't satisfied with the progress of your web development project. More often than not, it means something went wrong during the development cycle.

It could have been a problem with delivery, scalability, or cost estimation. Regardless, whether it's a new build or an implementation of new features, it can quickly become overwhelming.

So what's a product owner to do?

Those product owners I was consulting with, react in different ways. But most often, they tend to put all other business activities on hold to dive deep into managing the development process.

The process of developing sophisticated web-based software is more like a marathon than a sprint. But we have to keep moving forward. To avoid potential problems, follow the best practices introduced in this guide. The following recommendations are based on The Twelve-Factor App methodology and our expertise.

The web application development best practices discussed here can be applied to any Software-as-a-Service (SaaS) model. It covers everything from back-end development on a local machine to cloud architecture.

They are also based on the contributions of a broader community of software engineers who boast significant expertise in enterprise web development.

Technology Choice: Symfony PHP Framework

This journey into web application development best practices can be used as a guide to designing back-end architecture, and more. You can also do this by using any programming language you like.

However, for this post, we will focus on examples that leverage the Symfony PHP framework. It makes perfect sense as this framework is often used to build mid-sized cloud-based SaaS products.

According to the Roadmap, Symfony has continued to evolve since it was first released in 2005. Today, the framework follows PHP Standards Recommendations and boasts SOLID design principles for web development. So software engineers will be able to follow these best practices, seamlessly.

Principle 1: One codebase can accommodate multiple deployments during the web development cycle

If you're designing a web application and not a distributed software, I strongly recommend sticking to a single codebase. The best way to go about it is to store your code in a Git repository (or any other version controlled solution).

Web application development cycle for the new features will look (more or less) like this:

- Software engineers work with the same codebase on their local computers.
- After each feature is developed, they'll push the code to a Git repository.
- The updated codebase is then deployed and tested on a staging server by the Quality Assurance team.
- The same code is then deployed to production, but only after it's tested on a staging level to ensure that it works as intended.

Principle 2: Explicitly declare and isolate dependencies for the software

The golden rule is to first make a list of all external libraries and services that have been used. These should be separate from the code and quickly switched on/off or changed as needed. This enables you to do more with minimum influence on the overall code.

To manage libraries, for example, use a Composer. It's a great way to add and manage libraries demanded by your web application.

The Composer will create a file called composer.lock. Here, you'll find the same (exact) version of each package downloaded. These are the same packages that ran during the composer installation.

Store composer.lock in your Git repository and exclude the vendor folder from Git. This approach will ensure that each deployment (for another developer, for staging, or production servers) will get the same versions of libraries after the run “composer install” command.

If you fail to do this, you may end up with different package versions on production that will result in unstable app behavior.
Check our Case study about moving a Web product from monolith to microservices architecture and reducing dependencies to a minimum.

Principle 3: Separate configuration files and application codebase

According to the Twelve-Factor App principles, “an app’s configuration is everything likely to vary between deploys.”

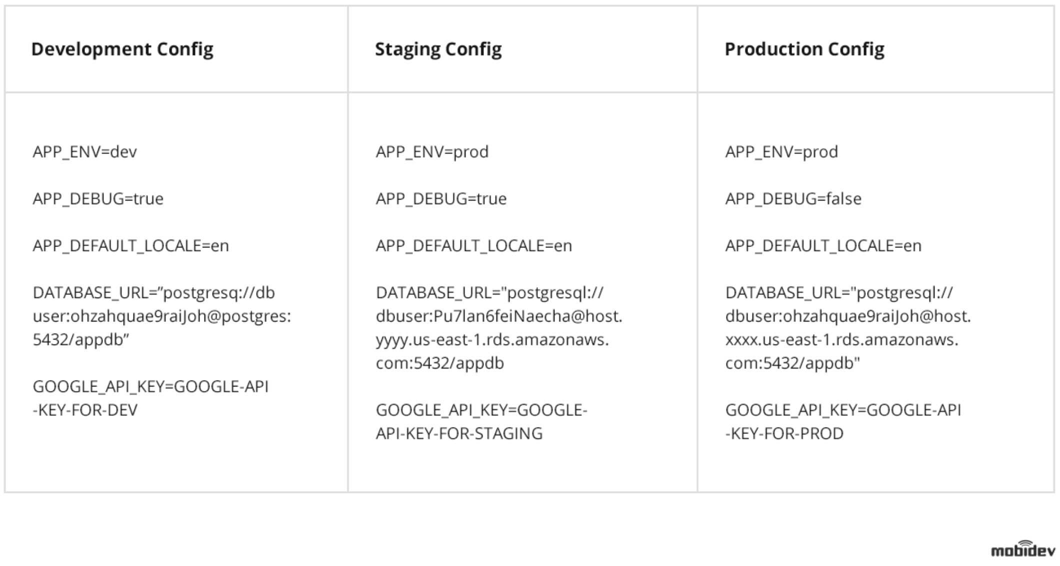
Often, software configuration settings can be changed between development, staging, and production environments. However, according to the application development principals, storing them in the codebase as configs or constants is prohibited.

The following are examples of settings that shouldn’t be stored within the code:

- Database and cache server connection settings
- Credentials to third-party services
- APIs and payment gateways
- Everything that changes depending on the deployment stage

Not following these web development principles may lead to scenarios where your app works perfectly on the developer's machine and at the QA stage, but not when it’s deployed to production. Typically, something always breaks because of the changes made to the configuration in the code.

However, Symfony web development uses the DotEnv library to work with environment variables. Here’s an example of different environment variables for development, staging, and production.



Development Config	Staging Config	Production Config

APP_ENV=dev
 	
APP_DEBUG=true
 	
APP_DEFAULT_LOCALE=en
 	
DATABASE_URL=”postgresq://db user:ohzahquae9raiJoh@postgres:5432/appdb”
 	
GOOGLE_API_KEY=GOOGLE-API-KEY-FOR-DEV
	APP_ENV=prod
	
	APP_DEBUG=true
	
	APP_DEFAULT_LOCALE=en
	
	DATABASE_URL="postgresql://dbuser:Pu7lan6feiNaecha@host.yyyy.us-east-1.rds.amazonaws.com:5432/appdb
	
	GOOGLE_API_KEY=GOOGLE-API-KEY-FOR-STAGING
		APP_ENV=prod
		
		APP_DEBUG=false
		
		APP_DEFAULT_LOCALE=en
		
		DATABASE_URL="postgresql://dbuser:ohzahquae9raiJoh@host.xxxx.us-east-1.rds.amazonaws.com:5432/appdb"
		
		GOOGLE_API_KEY=GOOGLE-API-KEY-FOR-PROD

So it doesn’t matter where the web app is deployed because there won’t be any need to change the code. Only proper environment variables have to be set. If you’re using Docker, run different Docker containers from the same image with different environment variables for different deploys.

Principle 4: All the variety of services your web app runs, have to stay detached

The software community verdict is clear. It doesn’t matter if the application uses some third-party back-end services, APIs, or built-in components like a MySQL database. The best practice in this scenario will be to treat them all as attached resources.

Enable software access via a URL or use any of the other methods stored in the configuration. This approach will allow the development team to manage components without making changes to the code.

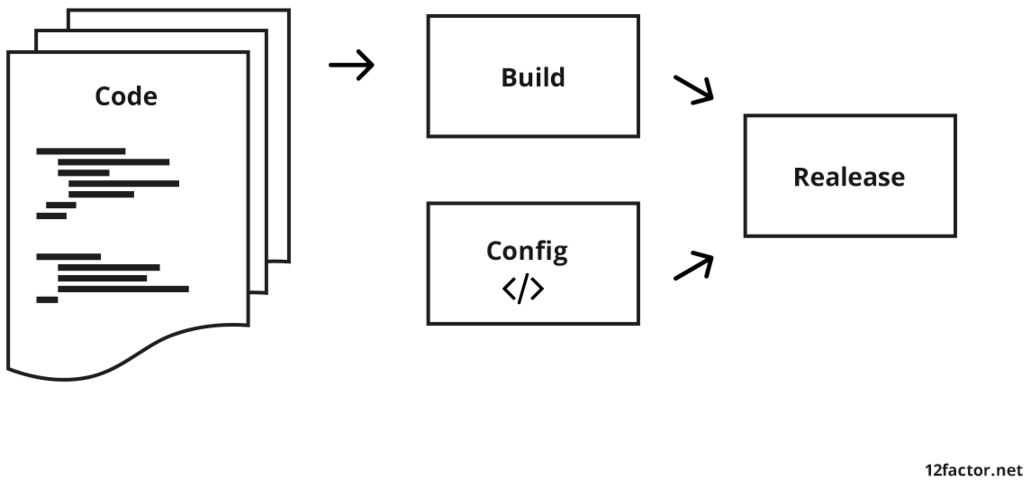
Symfony development is all about following SOLID principles. For example, to change database storage from MySQL hosted on EC2 instance to Amazon RDS, you just need to change the DSN style URL in the environment configuration without making any changes to the code.

You can even change static storage from Amazon S3 to any other Cloud Storage just by changing the environment variables, without changing the code.

Some libraries support various types of storage with an implemented abstraction layer. Flysystem, for example, supports different types of cloud storage. Doctrine is another example, but it's abstracted around database storage. So you can switch from MySQL to PostgreSQL after making minor changes to the code.

Principle 5: Build, release, and run each software development stage separately

To ensure stability during all stages of the build, you can leverage automation tools like Gitlab CI and Jenkins.



Software development pipeline stages:

During the **build stage**– the code must be set, depending on the PHP libraries and CSS/JS assets that need to be prepared. Finally, the Docker container with a version tag has to be built and pushed into the Docker registry. You'll also have the option of executing unit tests and code sniffer.

At the **release stage**– combine the Docker container (produced during the build stage) with the configuration for the environment (staging, and/or production) where you will run your build.

At the run stage– execute the app in the selected environment with proper environment configuration. This will be based on the release stage.

Principle 6: Make processes, stateless, and store the data outside the Web application

Leading web development practices recommend the storage of persistent data in separate services. It can be any relational database, for example, like Redis, Amazon S3, and so on.

However, if you're working with Docker, you don't have to store all the data inside the container. This is because your app has to be stateless.

Going forward, this approach will be critical to scalability. If you need authorization in your API, you can use stateless JSON Web Tokens, or you can use sessions, but store them in Redis.

The Symfony framework can work both ways, and such techniques enable scalability benefits. For example, you can run one container or hundreds of containers that use the same code and work with the same services.

You can restart your cluster (or a bunch of containers) or run a new one with a new version--without data loss.

At some point during the growth cycle, you'll need more than one server. At this juncture, the software will start running on several servers using a load balancer.

It's the best approach to orchestrate the execution of functions and various requests. This is the exact moment when all the processes have to be stateless.

The users will be authorized on one server, but their requests will be pointed to another server to be executed. Unless the authorization service is stateless (and can support both servers), the user won't be able to continue running the app due to lack of authorization on the second server.

Principle 7: Keep software self-contained and export services via port binding

All web apps have to be made available without dependencies. If an app uses several services, each service has to be made available via separate ports.

Once you run Docker Compose on the developer’s local machine, for example, the application will become available on http://localhost:8085.

At the same time, a separate GUI for Database management services will be available on http://localhost:8084.

Such an approach allows us to run many Docker containers and use load balancers (like HAProxy and Amazon Elastic Load Balancing.)

If the development team refuses to follow this approach, it'll make the launch process more complicated. All the unnecessary (extra) steps may create additional bugs in the system.

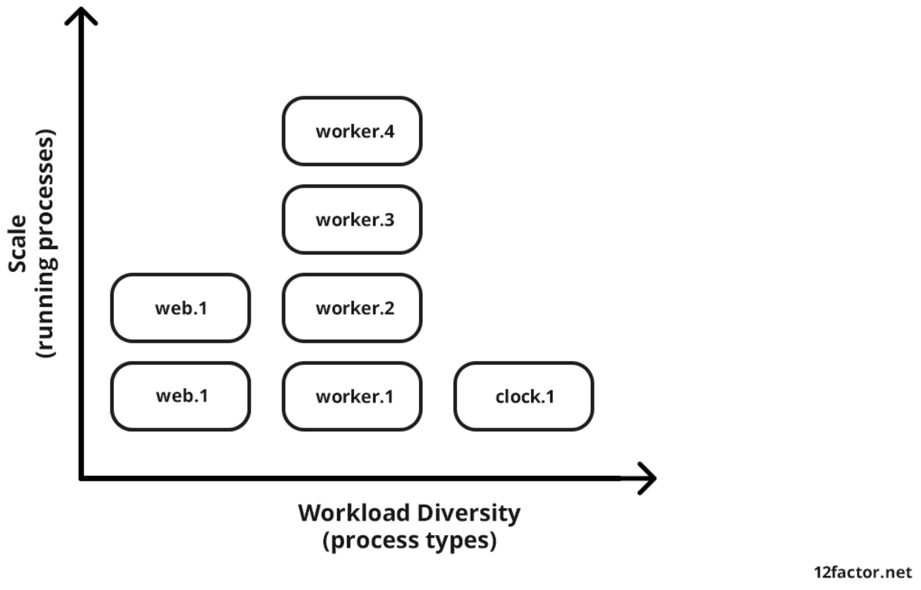
Principle 8: Apply the process model with NO sharing

As the web application is developed, there will be a variety of processes running within the app. However, it's important to keep all these processes separate. You can even divide these processes into separate ones if some of them are getting "too heavy" to manage.

So nothing has to be shared between several different processes. It's the best way to keep the application simple and stable. What's great about this approach to web application architecture is the fact that it makes it easy to scale.

Below, you'll find these recommendations illustrated with a PHP developed multi-tenant application:

- The app can run as many PHP containers as needed to scale (up or down).
- It can work for as many clients as needed.
- If we need to handle more HTTP requests, we can just add more running PHP containers.
- If we need to handle more background tasks, we can add more containers with background workers to handle these needs.



Principle 9: Stay sustainable with fast launching and shutting down processes

All web applications have to be developed in a manner that enables scaling (up or down) at will. It should be able to run 100 containers, for example, during business hours (when the app users are most active), and run ten containers at night (to save money allocated for the infrastructure).

This approach can be applied to your software product by containing the processes, separately. However, you have to enable them to be launched or terminated rapidly.

That's why the minimization of the launch time for processes is a must. If you use Docker, you can run more PHP containers.

The container image should be ready for launch. Don't compile at startup, and don't run PHP Composer during the launch. The time to launch should always be as short as possible.

When you want to scale down, all you have to do is shut down some containers. It's also critical to shut down the containers without disrupting the system.

Users don’t want to see request timeout or error messages. So use proper termination signals by sending daemons, like in most queue servers.

RabbitMQ and Beanstalkd are great examples that work efficiently in the background and queue the job in case of failure. You should also apply transactions to the Relational Database.

Principle 10: Keep development, staging, and production stages as similar as possible

The development, staging, and production environments have to be as similar as possible. Following this rule will help web application development teams avoid situations like a production software failure (which often happens when the web app was developed under different conditions).

There are some cases where the developer can run an application on a local machine with Docker and use the same Docker containers for staging and production. To use something that works only in the cloud, it’s better to go with the Adapter pattern.

For example, the production Docker cluster uses the same containers that were tested on staging but with different environment variables. The PHP and PostgreSQL versions will be the same (and unchanged), but the latter will run on Amazon RDS.

Log level is different as the users don’t see details about errors like the ones seen on the developer’s machine or staging server.

Anyway, we will still receive reports about errors (with error details) in our error log aggregator server. This way, you’ll know about the error before users report it.

Principle 11: Good practice collecting application logsis is to perceive them as event streams

Software engineers should approach logs as event streams and not a steady stream of data that are headed for storage. It's the optimal way to reach high-visibility while running an application.

Storing logs as separate files for each Docker container (or separate services) should not be an option for your team.

By default, the Symfony PHP app writes logs on to log files, but you can configure the logs output to Linux stdout/stderr stream. You can do this by setting "php://stderr" string in the log path.

After that, you will be able to see all active logs from the console. Anyway, such an approach will be good for development but not good enough for production. You can see the logs while your Docker container is running but not after the container has been terminated.

For production, I would highly recommend the use of log aggregator services. Sentry, Graylog, Logstash, or Splunk are excellent examples.

Such services can aggregate logs from all running containers and enable analysis from a centralized platform. You can also leverage notifications to be informed about production issues and avoid losing data logs after a crash.

Principle 12: Manage admin activities as one-off processes

Administration and management activities play an integral part in the development and deployment of software products.

Database migrations, operations with cache, creating new system users and backups are just some examples. All these web development principles have to be applied to those activities as well.

Symfony provides a bin/console command to handle admin/management tasks. It's best to run such commands as a one-off process in the same identical environment as the regular long-running processes of the app. It means that we need to run a new container for just one task instead of using an existing container.

Here's an example of how Symfony accomplishes database migrations in Docker:

```
docker run symfonyapp_php:1.4.2 php bin/console --no-interaction doctrine:migrations:migrate
```

We use Docker run, not Docker execute. This command works with the container that's already running. After the migrations are complete, the Docker container can be exited. This will be a one-off process.

Achieve web application stability and scalability

All the web application development principles mentioned above should act as a baseline for your software development team. Such implementations should also be made wisely and aligned with business priorities.

They should be technically reasonable and seamlessly applicable to specific use cases. If you choose to ignore one or more of these best practices, there should be an excellent reason for it.

The key to a successful web development project is to create a culture of CI. It matters because it's becoming effortless to automate things like deployments, updates, and setting up infrastructure for new users. Check our Case study on creating Advisory Web Platform For Education & Mentoring with CI.

In conclusion, it’s critical to follow these steps and only divert from it when it’s absolutely necessary. When followed accordingly, these development principles ensure that your software is stable and ready to scale.

It has paid dividends time and again in my software development career and promises to do the same for you in your next web development project.

mobidev

CONTACT US

info@mobidev.biz

mobidev.biz