

Quottravel tech whitepaper

Quonext

Palma de Mallorca

March, 2020

Table of contents

Introduction	3
Work options	4
Software as a Service (Saas)	4
Managed service	4
Docker images	4
Artifacts	4
Fork source code	5
Architecture	6
Overview	6
Load balancing and routing	6
Object cache	7
Database	8
Monitoring	9
Reporting	9
Static content	10
B2C	10
Apps	11
Extension points and customization	12
Documents customization	12
Translators	12
BPM integration	12
External services	13
Reports	13
Containers	14
Images repository	14
Parametrization	14
Development	15

Technology stack	15
Overview	15
Java	15
Persistence	15
Caching	16
Clustering	16
Apis	16
User Interfaces	17
MDD	17
B2C	18
Workflow	19
Conclusions	20

Introduction

Quotravel is the new generation information system for travel agencies (DMC, Retail and Tour Operation) developed by Quonext.

This document exposes the different technical points which can be of interest for any travel agent thinking about acquiring Quotravel and guessing how Quotravel works.

This document is not about functional scope but about our technical architecture and stack.

You should have some tech knowledge for reading this document.

Work options

QuoTravel is available through different options in order to offer you the best fit. The available options are:

- Software as a service
- Managed service
- Docker images
- Artifacts
- Fork source code

Software as a Service (Saas)

In this option we care for everything that is needed to keep the system always up and performant, and you only use QuoTravel as a service. In this modality you can still create your own users, customize your documents, add your own translators, or link Quotravel events to your own Business Process Management (BPM) engine to trigger your own processes.

Managed service

In this option, besides all of the above, you also have access to a control panel which will allow you to monitor the whole system, add or remove servers to your cluster, and control deployments. We take care of all the configuration underhood and you only need to worry about adjusting your cluster size.

This option is interesting if you want to have direct control on the infrastructure cost but you do not want to do the effort of configuring and maintaining all of the infrastructure.

Please notice that servers can be both cloud or bare metal servers.

Docker images

In this option we provide you with access to our private Docker image repository, and you are free to deploy them on your own infrastructure. This option is interesting if you have your own infrastructure and you have people to manage it.

Artifacts

In this option we give you access to our maven repository, so you can add our artifacts to your own projects. This option is interesting if you plan to develop your own services on top of QuoTravel and you need to access our data model, or if you want to add your own classes (and

make them stay in the same database as Quotravel), screens and menu options to Quotravel. This mode is compatible with all of the options above.

Fork source code

In this option you can create a fork of our source code repository and build your own artifacts and images. This option allows you to develop your own application on top of Quotravel, so you can use Quotravel as a code base for building your own information system. If you choose this option you should have your own development team and some training is needed.

Please note that, though there are many unit tests in Quotravel, this way of work makes it impossible for us to guarantee that everything will work as expected and this responsibility moves to your own development team.

Architecture

During this chapter we will review Quotravel's architecture so you should be able to understand how Quotravel works and where its performance and scalability come from.

This chapter is targeted to those customers who are not interested in the SAAS option but they want to have more control on its Quotravel. Besides, having a look at Quotravel's architecture will let you understand and leverage the power of Quotravel.

Don't be scared. All of this complexity can be hidden behind simple control panels so you can manage the whole system at a breeze.

Overview

At the highest view Quotravel is based on nodes, which can work both as intranet and as service nodes. Quotravel nodes are available as Docker images and can be deployed anywhere using any container orchestration tool on top of any infrastructure (cloud, bare metal or hybrid).

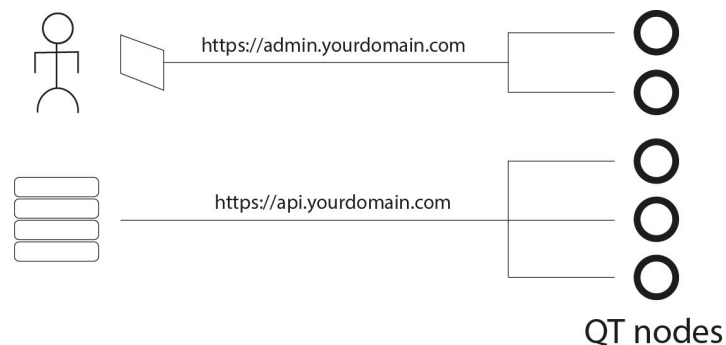


Fig. 1: Bird view of Quotravel architecture

Load balancing and routing

On top of our nodes, a service mesh tool (e.g. Istio, Consul, ...) is used to load balance your requests and redirect them to the underlying nodes. Please note that nodes register themselves so the service mesh knows where to redirect traffic.

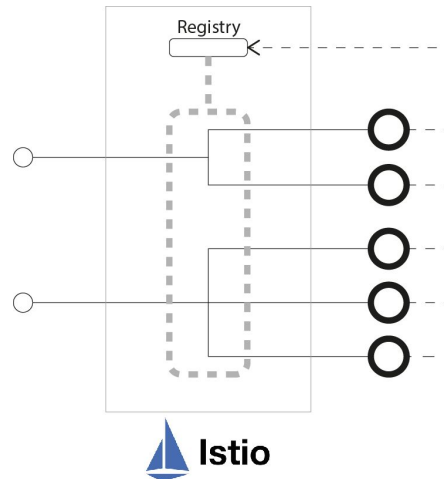


Fig. 2: Istio as a service mesh for Quotravel

Object cache

Inside each node's memory there exist your business objects (contracts, bookings, agencies, ...), which are (when needed) persisted to a database. As these objects exist in each node's memory you get the best performance and you are able to attend unlimited millions of requests with the lowest response time at the lowest infrastructure cost.

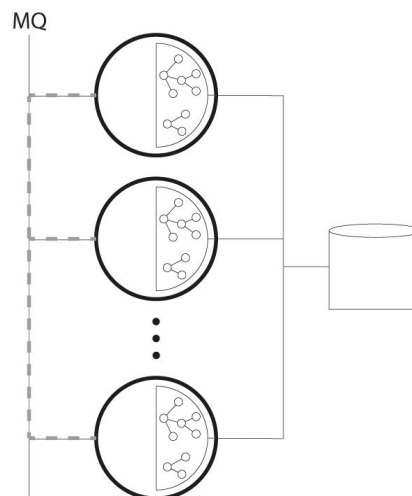


Fig. 3: Object cache

In order to find objects in memory each Quotravel node works with its own cache, and all of the Quotravel node's caches are always synchronized through a message queue. So, if any object is updated in any node, a message is sent to all of the other nodes through the message queue in order to invalidate their cached copy of the same object. At the same time there exist mechanisms to resolve any conflict in case 2 nodes try to modify the same object at the same time, which guarantees the overall consistency.

Database

Quotravel is database agnostic and works on top of the main database engines (Oracle, DB2, Microsoft SQL Server, ...) though our preferred option is PostgreSQL. PostgreSQL is an enterprise level open source database which supersedes by far all of the features that we need for Quotravel and which has demonstrated its reliability for more than 10 years in other important travel agencies with the highest requirements and performance.

One of the coolest features of PostgreSQL is the ability to have as many streamed replicas as we want, and that we use for failover, for extreme load balancing and for reporting offload but, as I have said, remember Quotravel is database agnostic and any of the major database engine is suitable.

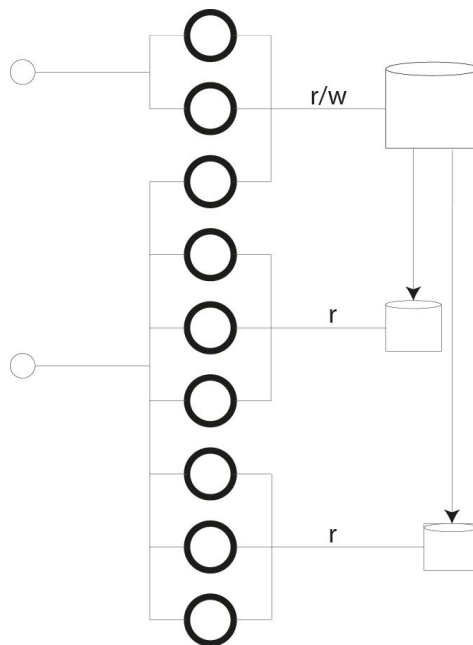


Fig. 4: Database replication for scaling

Each node works on top of a database and, depending on your needs, you can make all of your nodes work against the same database instance, or you can configure your nodes to work against different databases attending to their purpose. This is useful if you want to have different Quotravel instances federated through the apis or if you have such an extremely huge number of requests that you would like to create specific read-only nodes which would work against streamed replicated database instances, in order to reach an even higher scale order.

Monitoring

Quotravel nodes statuses and logs are monitored in order to verify that everything is working and alerts are raised if something is broken. Statistic info is also recorded and available for forensic analysis if needed.

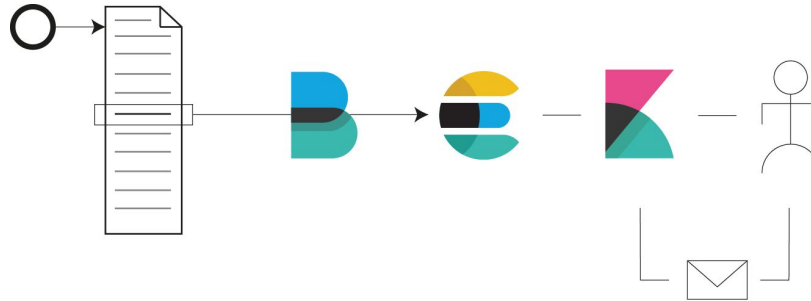


Fig. 5: Monitoring with the ELK

We are currently using the ELK stack for monitoring, APM, log and alerts though we can easily integrate Quotravel with any of the major monitoring platforms (Dynatrace, AppDynamics, New Relic, ...).

Reporting

Quotravel database is also available for reporting through a streamed replicated instance or daily backups, so you can safely use your preferred BI and reporting tool (Qlik, Microsoft Power BI, ...) to run and schedule your own reports against a separate database.

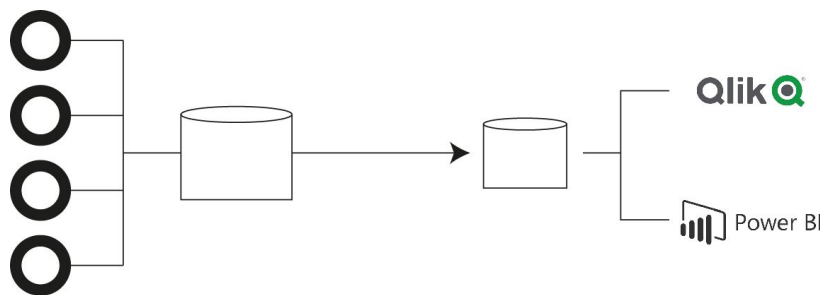


Fig. 6: Reporting

The most commonly used reports are included in Quotravel, but for a more complete reporting solution we expect you to use any of the BI tools available.

Static content

Images and other resources can be stored in a different database than the operational one. We use our own Content Management System (CMS) which can be integrated with a Content Distribution Network (CDN) for making those contents available worldwide without consuming your infrastructure resources.



Fig. 7: CMS and CDN for static content

B2C

For public websites we use a Static Site Generator (SSG) for creating static sites from our content, and web components created with Vue and distributed through npm for the dynamic part of our sites (the booking engines) which can be embedded in any website.

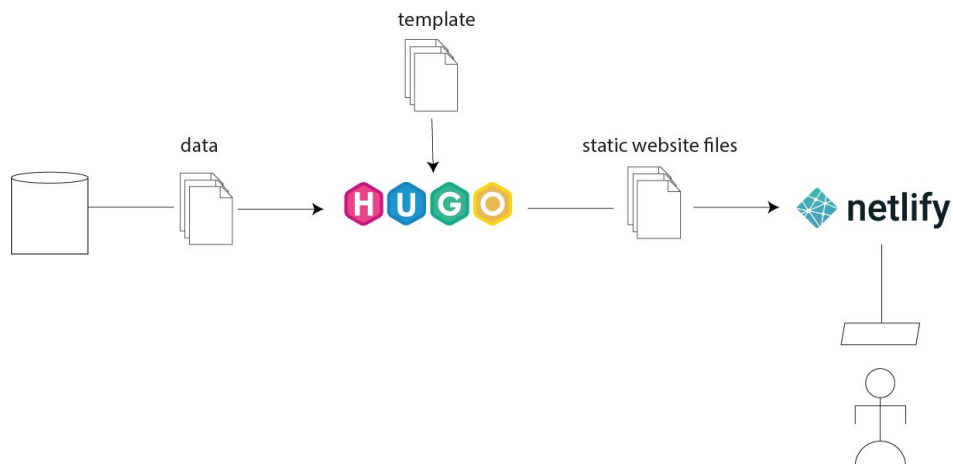


Fig. 8: B2C website generation pipeline

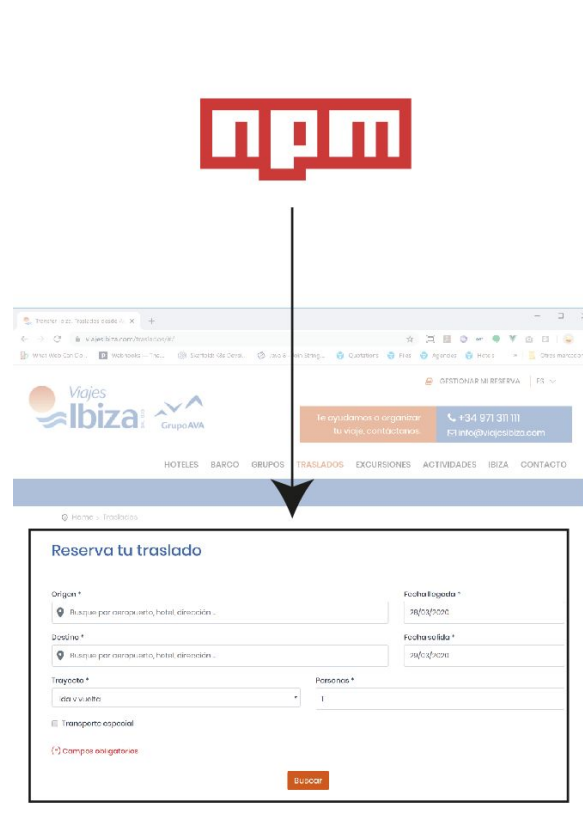


Fig. 9: Web components

Apps

For creating our mobile apps (e.g. the rep selling and ticket validation apps) we develop Progressive Web Applications (PWA) which allow us to effortlessly develop compatible mobile apps for any operating system and quickly distribute them without the annoying app stores. The resultant user experience is the same as with native apps (the app can be installed as any other native app) and we can still access the mobile devices like GPS, camera or NFC so we have the best of both worlds (web and native).

Extension points and customization

Quotravel provides out of the box several extension points and customization options, and they are:

- Documents customization
- Translators
- BPM integration
- External services
- Reports

Documents customization

All documents generated from Quotravel are customizable. Quotravel uses xsl-fo for pdf generation and freemark for email templates, and they all are available and modifiable at the intranet.

Translators

All third party integrations are done according to the Easy Travel Api specification. When you want to add a new provider to Quotravel we only need to develop a translator which is, indeed, nothing more than a translator from Easy Travel Api to the proprietary specification of the provider. Developing new translators is a highly mechanical task and can be done by Quonext, by your own development team or by a third party.

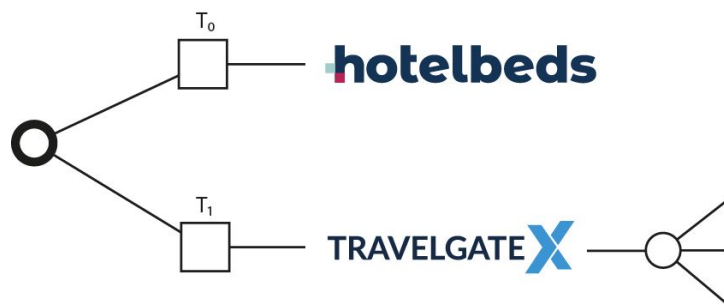


Fig. 10: Easy Travel Api translators

Obviously any Easy Travel Api compliant provider can be directly connected without any translator (e.g. for Quotravel instances federation).

BPM integration

Quotravel has an internal Business Process Management (BPM) Engine for handling tasks which are triggered when some events happen (e.g. a new booking is confirmed, a booking is

modified or cancelled, ...). Instead of using BPMN to state our workflows we hard code our logic inside Quotravel, as this logic is always the same for all the customers we have worked with and as, for us, it is the most effective way for doing this.

Anyway, if you need to launch some custom logic when something happens in Quotravel, you can also configure Quotravel to notify your own BPM and trigger your own logic.

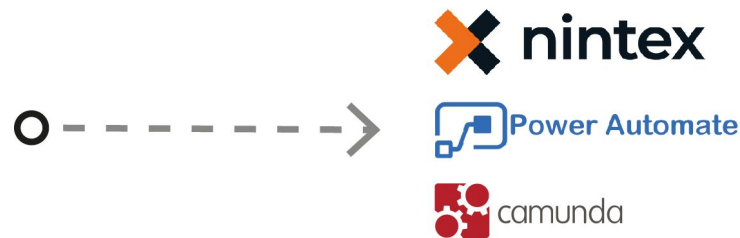


Fig. 11: BPM integration

External services

Quotravel uses some external services for several tasks (e.g. Clickatell for SMS, DeepL for translation, Flightstats for flight info, Google and Microsoft for OAuth, Sermepa, Webpay or Paypal for payments, ...). There is not much to explain here, just be aware that we can easily integrate any external service so you can use it from Quotravel.

Reports

As we have seen before you can access a backed copy of your production database for reporting. This database allows you to create your own reports, schedule them, and create whichever authorization rules with your favourite reporting tool (e.g. Power BI from Microsoft).

Containers

As we have seen above Quottravel nodes are available in the form of Docker images. This way you are able to deploy Quottravel nodes inside any Docker container and use any container orchestration tool (Kubernetes, Openshift, ...) to automate the creation of containers from our images.

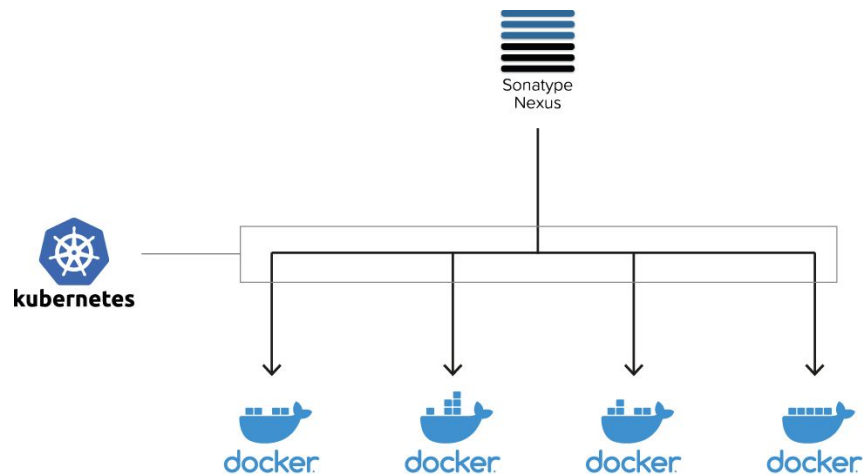


Fig. 11: Containers deployment

Please note that our container orchestrator is also responsible for handling failover, rollover updates and auto scaling.

Images repository

Quottravel images are available from our own private image repository. If you choose to deploy our images on top of your own infrastructure then we would provide you with our repository address and credentials. We can also configure webhooks if you want to implement any kind of continuous deployment mechanism or similar.

Parametrization

Container parametrization (e.g. database url, mq endpoint,...) is done with ENV variables. So, if we want a container to work with a concrete database instance, we only need to pass the database url and credentials to the container by setting the correspondent env variables. The same happens with the JVM heap size and many other parameters.

Development

In order to offer a complete picture of Quotravel technology we must have a look at how it is developed. Please note that this chapter is only relevant if you plan to use Quotravel as a code base for your own development.

We will now focus on two parts:

- Technology stack
- Workflow

Technology stack

All of our development process is guided by one simple principle: write the least lines of code. Less lines of code mean less errors and more reliable and maintainable code, and this is one of our main targets when we write our code.

Overview

In the simplest way, we use Object Oriented Programming (OOP) for creating and handling the business objects (bookings, contracts, hotels, agencies, ...) which are at some moment persisted in a database.

Java

Our development language is Java as it guarantees that we have the best technology, tools and ecosystem available for our development. Please notice that Java Virtual Machine's Just In Time compiler compiles, and optimizes, our bytecode to native compiled code as Quotravel is running so, in the short term, we have the same performance order times as if it was written in C. In fact, we have better times as programming in Java allows us to write better code than if we would have written the same logic in C++, and all from the top productive Java development ecosystem.

Persistence

For persisting our objects we use standard Java Persistence Api (JPA) which, among other things, enables Quotravel to be persistence provider independent. Today we use Eclipselink as our persistence provider, but we could use OpenJPA, Hibernate, or any other JPA compliant engine. This fits our line of looking after our invested survival over time.

From the persistence engine we get some other functionalities: caching and clustering.

Caching

Eclipselink (as any of the other persistence providers) provides us with an intelligent 2 level cache which allows us to find our business objects already in the JVM heap so we do not need to go to the database and, in this way, we avoid contention and get the best performance possible. This performance, at the end, means lower response times and less infrastructure costs, as we need less resources to handle the same number of requests.

Clustering


Eclipselink (as many of the other persistence providers) also provides us with several cache synchronization mechanisms which allows us to deploy an unlimited number of nodes while keeping all of them perfectly synchronized. In our case we use a message topic for cache synchronization but we could choose several other synchronization mechanisms.

At the end what we have is a top performant and unlimitedly scalable system out of the box, without any programming restriction. Going from 1 node system able to handle 10M availability requests per day to a 100000 nodes system able to handle 1B availability requests per day is just a matter of configuration.

Apis

Our business objects and logic is made available to our partners through a Rest API and through several web interfaces (aka intranet, extranet, backoffice).

For publishing our REST apis we use standard JAX-RS endpoints (again, we work with standards which allow us to be implementor independent) and we use the Jersey Rest framework for running them.

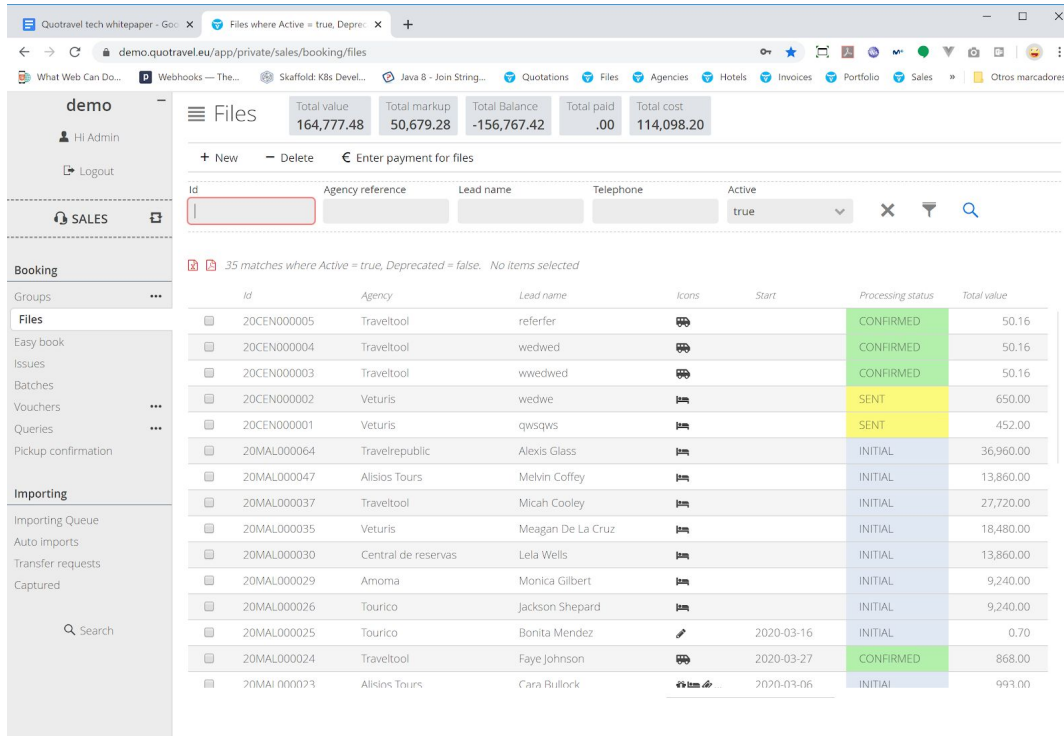


```
1 package com.quonext.quotravel.services;
2
3 import ...
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19 @Path("/pickupconfirmation")
20 public class PickupConfirmationService {
21
22     @Path("q")
23     @GET
24     @Produces(MediaType.APPLICATION_JSON)
25     public Map<String, Object> q(@QueryParam("p") String p) throws Throwable {
26         Map<String, Object> d = new HashMap<>();
27         Helper.transact(em -> {
28             List<Service> l = em.createQuery( s: "select x from " + Service.class.getName() +
```


Fig. 12: JAX-RS service sample

User Interfaces

All of our user interfaces are pure html so no plugin neither installation is needed, and for creating them we use a Model Driven Development (MDD) approach. This means that we can generate our user interface directly from our business classes, and let the framework do the magic and create the user interface for us.



Id	Agency	Lead name	Telephone	Active	Total value
20CEN000005	Traveltool	referfer		CONFIRMED	50.16
20CEN000004	Traveltool	wedwed		CONFIRMED	50.16
20CEN000003	Traveltool	wwedwed		CONFIRMED	50.16
20CEN000002	Veturis	wedwe		SENT	650.00
20CEN000001	Veturis	qwsqws		SENT	452.00
20MAL000064	Travelrepublic	Alexis Glass		INITIAL	36,960.00
20MAL000047	Alisios Tours	Melvin Coffey		INITIAL	13,860.00
20MAL000037	Traveltool	Micah Cooley		INITIAL	27,720.00
20MAL000035	Veturis	Meagan De La Cruz		INITIAL	18,480.00
20MAL000030	Central de reservas	Lela Wells		INITIAL	13,860.00
20MAL000029	Amoma	Monica Gilbert		INITIAL	9,240.00
20MAL000026	Tourico	Jackson Shepard		INITIAL	9,240.00
20MAL000025	Tourico	Bonita Mendez		INITIAL	0.70
20MAL000024	Traveltool	Faye Johnson		CONFIRMED	868.00
20MAL000023	Alisios Tours	Cara Bullock		INITIAL	993.00

Fig. 13: Quotravel UI

MDD

For dealing with MDD we use a framework developed by ourselves which has been built of top of Vaadin and which, among other things, allows us to:

- Write UI technology agnostic code, preserving our investment over time
- Develop at lightning speed, as we just need to write the strictly necessary code. No boilerplate code is written and we can focus on business logic
- Write error prone apps. No code written, no errors
- Build mobile first apps from free, with the same code
- When we need to add special components to our screens we can develop them with Vaadin and plain html and javascript, and they are seamlessly integrated into our apps

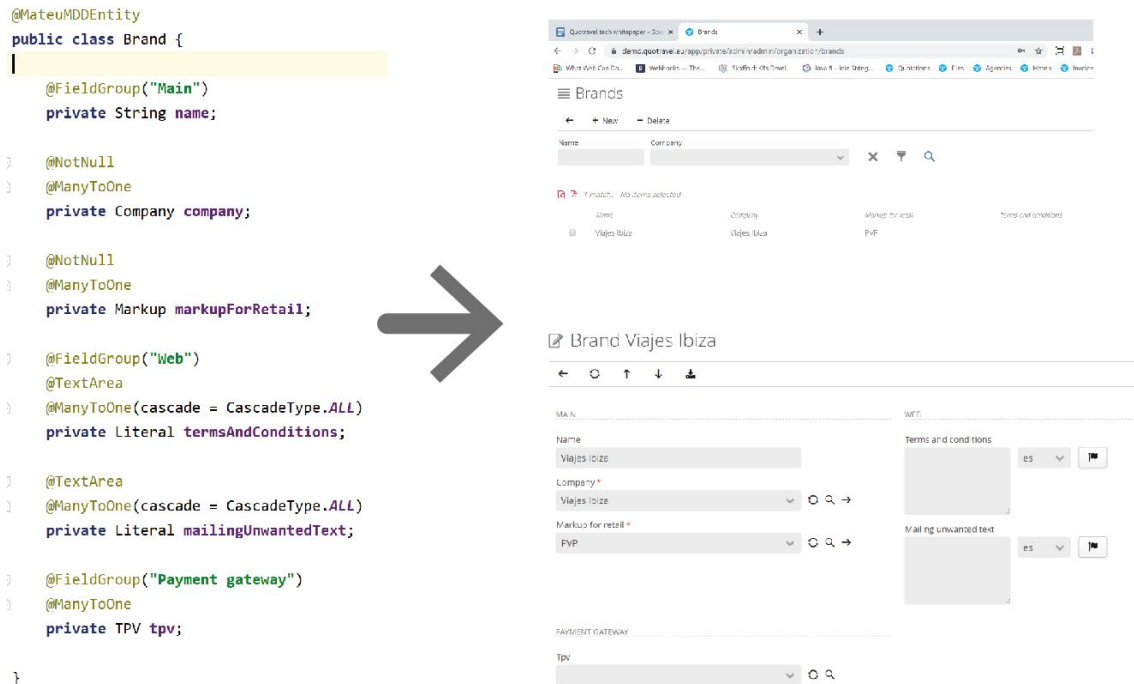


Fig. 14: MDD development

B2C

Public websites (B2C) have different requirements than intranets and are built in a different way. For B2C sites we generate static websites using a Static Site Generator (SSG) which, basically, applies a template to a set of contents that we generate from our data model so all the typical server side logic is solved ahead of time to generate static files.

As a result we have a set of pure static files (html, css and javascript) which are pushed to a Git repository. This repository is connected via a webhook to a Content Distribution Network (CDN) which deploys the web on each push to the repository, in a Continuous Deployment (CD) manner.

For the dynamic part of our websites (the booking engines and the chart) we create components (pure javascript files) with Vue and they are built and deployed using npm, and can be seamlessly incorporated to any website (e.g. a website built with wordpress or with any other Content Management System (CMS)).



Fig. 15: B2C website deployment

Workflow

Our development workflow is a pretty standard one.

All of our code is stored in a Git repository (Github) and our release code is stored in the master branch and we open as many branches as we need for any feature or epic development.

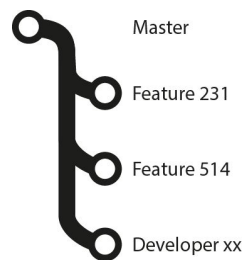


Fig. 16: Git branches

Each time we push a new commit or we merge a pull request a new build is issued and tests are passed. If all tests pass then a new artifact is built and deployed to our maven repository. For the master branch a new docker image is built and deployed to our Docker image repository.

Please note that our release repositories are protected so they can only be updated from our Continuous Integration (CI) process after all tests pass (we use Github Actions for that purpose), while our snapshot repositories can be updated by any of our developers.

Conclusions

After reading this document you should have a knowledge of Quotravel's architecture and technology stack, deep enough as to understand how it works.

Nothing in Quotravel is superfluous neither it obeys to any top trending topic nor hype. It's just the technology stuff which best fits a top performance and scalable system requirements as needed in the travel industry, and which has already proved to be rock solid for many years in many travel agents.

We are at your disposal for any clarification or for a deeper view on any of the points of our tech stack and architecture.