GESTIUNEPROIECT.RO

CASE STUDY

Developed by Mibro Labs SRL – www.djangocoders.com

GESTIUNEPROIECT.RO – CASE STUDY

PROJECT DESCRIPTION

GestiuneProiect.ro is a SaaS project management and online collaboration web app. Its main target are the projects that were funded by the European Union and by the Romanian government. Traditional project management features like tasks, messaging or file sharing are tightly integrated with a human resources module which helps people involved in the project to reduce the time required by bureaucratic activities by generating most of the documents required by the EU/Romanian authorities.

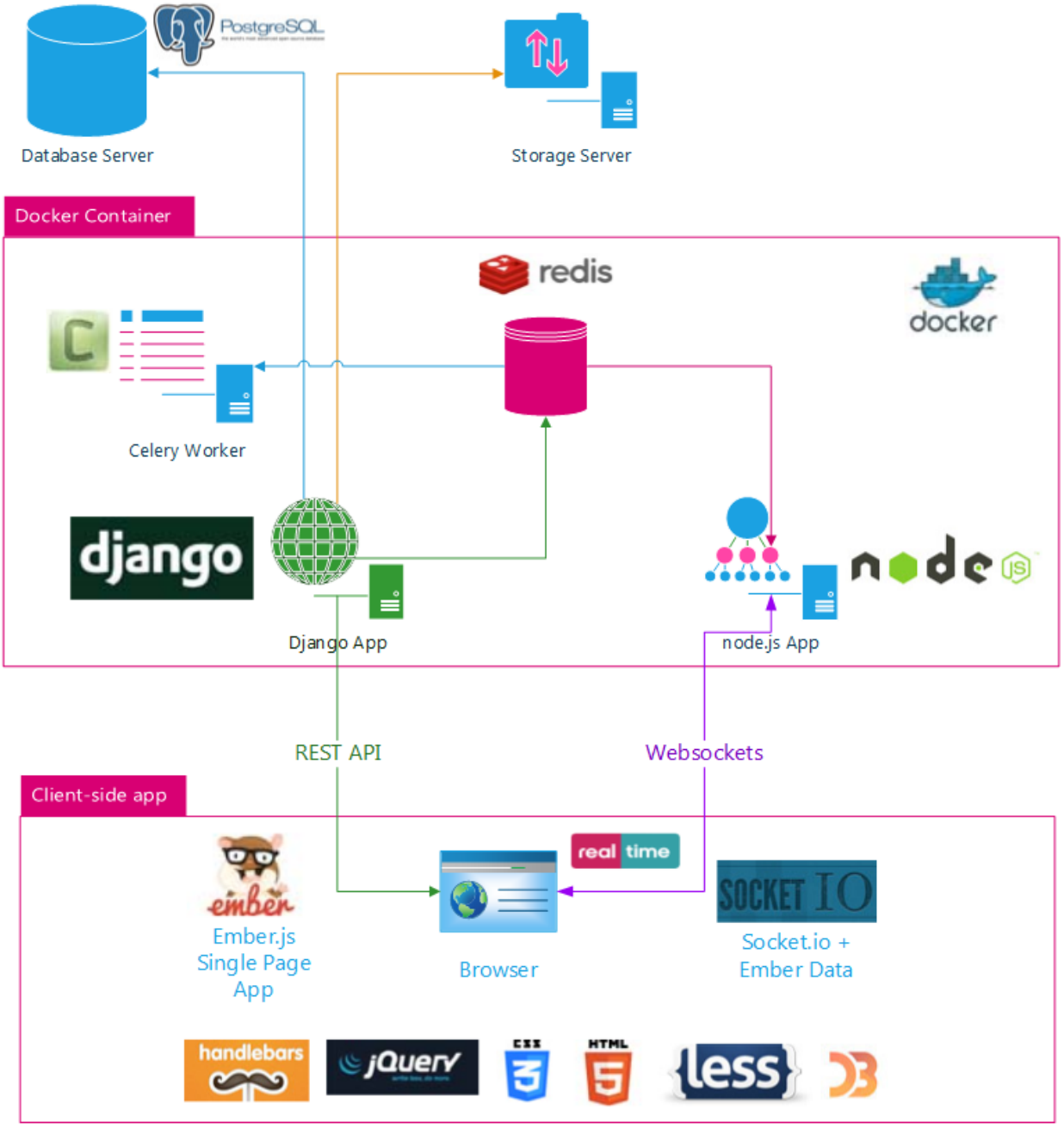
The beneficiaries are mainly private or public organizations, and exceptionally individuals, chosen by the European Commission for their capacity to implement the projects concerned. It also offers collaboration and communication tools.

MAIN FEATURES

- **Human Resources Management**. Strictly follows the European Union imposed structure. Offers management for Organizations, Users, Work Contracts, Permissions, Project Activities, Time tracking, Actions, Meetings, Groups, Tasks. The app automates and simplifies the process and generates reports required by the European and Romanian authorities.
- **Custom dashboards**. All components can be widgetized and used in customizable dashboards.
- **Messaging, Group chat & Comments**. Supports real-time updates using websockets, file attachments, embedded media, history, full text search, rich text formatting and integrates features from the HR module.
- **Notifications** (real-time in app rich notifications, emails, SMS)
- **Tasks.** Supports real-time updates, advanced filters with history support, integrates features from the HR (Similar to Asana)
- **Reports**. Highly customizable reporting engine that exports to Excel, Word, PDF, CSV and custom formats.
- **File sharing / File manager** with complex (per file/per folder) permissions. (Similar to Google Drive)

- **Complex role and permission configurator** at user, group or work contract level.

Technology Stack used



BACK-END

This is our bread-and-butter. It's Django and PostgreSQL running with Gunicorn behind Nginx. The biggest difference between this and your typical Django site is that it mainly serves as a REST API consumed by the front-end. We are using traditional Django views in a few places like login, logout, and some custom cases, but the rest of the views are handled by the front-end.

Using Django, we were able to quickly build the nuts-and-bolts parts of our site that didn't require any special real-time functionality. A few of the important libraries we're using:

- djangorestframework for building the REST API
- haystack backed by solr for search
- celery backed by Redis for task queueing
- misaka and pygments for text formatting
- easy-thumbnails and PIL (pillow) for image thumbnailing
- django-storages for remote server storage
- django-compressor with a custom filter to handle Handlebars templates

FRONT-END

The application is designed as a SPA (single-page application).

Our front-end uses Ember.js, Ember Data and jQuery to handle all the routing and interactions.

Ember.js is designed to build ambitiously large web applications that are competitive with native apps.

We use Handlebars templates that update automatically when the underlying data changes. HTML5, CSS3, D3.js, LESS (Less is a CSS pre-processor, meaning that it extends the CSS language, adding features that allow variables, mixins, functions, etc.)

All the data received from the APIs is stored on the client side in a central repository of records by following the SSOT principle (Single Source Of Truth). You can think of the store as a cache of all of the records available in the app. The application's controllers and routes have access to this shared store; when they need to display or modify a record, they will first ask the store for it.

Ember Data is a library that integrates tightly with Ember.js to make it easy to retrieve records from a server, cache them for performance, save updates back to the server, and create new records on the client. (similar to an ORM library)

REALTIME PUSH/PULL

We decided to get started with Node.js and socket.io. It seemed to have the most development activity and was keeping up well with the changing browser implementations. The socket.io/websocket API is a breeze and a joy to work with.

Our Node.js server is very small, weighing in at less than 200 lines of code. Its primary purpose is to fan data out to all the connected clients. Almost all writes/pushes from the client go directly to our Django HTTP API. This lets us keep our logic where our best skills are (Python) while getting the benefits of socket.io.

Redis serves as a Pub/Sub transport from Django to Node.js. As Django gets requests over the HTTP API, it places events on Redis queues which are immediately grabbed by Node.js and distributed to the clients.

As Ember-data caches records on client-side, we must send notifications when any of the cached objects changes on the server. Django puts serialized objects in Redis as one of the last things before returning the HTTP response to the client (after performing all the logic, hitting the database and making sure the transaction is not rolled back). Despite this, the clients consistently receive the event over the websocket before receiving an HTTP response. Redis to Node.js to client is blazingly fast.

DEPLOYMENT

We use docker to deploy our application in isolated, lightweight containers. Based on shipyard, a docker management application we created an internal tool to ease monitoring, deployment and upgrades of the containers.

MONITORING

With multiple client instances, it's important to keep on top of what's going on across the board. We use Munin to graph metrics across all of our system, and also alert us if anything is outside of its normal range. We use StatusCake (similar to Pingdom) for external monitoring of the services

For Python & Javascript error reporting, we use Sentry. At any given time, we can sign-on and see what errors are happening across our system, in real time.

OTHER BITS

- Jenkins for continuous integration testing with plugins for code coverage and lint checking (pylint/jshint)
- Selenium for functional testing
- Mandrillapp for email
- Custom built app as a SMS Gateway
- Chef for server configuration management
- Supervisord for process management inside docker containers
- Fabric for deployment-related tasks
- Private GitLab for repository hosting
- Phabricator for code review, tasks, issues

SC MIBRO LABS SRL

http://djangocoders.com