# Human Resources Management System (HRMS)

## Application Introduction

*Human Resources Management System (HRMS)* will be responsible for entities:

* Employees
* Salaries
* Vacations
* Payments
* Bonuses
* Promotions

## System Requirements

Functional:

* Web-based application.
* Performs CRUD operations on the employees.
* Manages salaries:
  + Allows the manager to ask for an employee’s salary change.
  + Allows the HR manager to approve or reject a request.
* Manages vacation days.
* Manages promotions & bonuses.

Non-functional:

* Classic data-driven application.

Open questions:

* *“How many concurrent users do you expect?”*
* *“How many employees are you going to manage?”*
* *“What do we know about the external payment system?”*

Answers we got from the clients:

* 10 concurrent users.
* 250 employees

Let’s start with the data volume. We are going to estimate the total size of the data the system should be able to store.

We asked our client and calculated a bit. Each employee holds around 1 MB of data. On top of that, the company stores around 10 scanned documents for every hired person – contracts, reviews, etc. One document is around five megabytes, so the total data for one employee is:

*5 MB x 10 + 1 MB = 51 MB / Employee*

The company has 250 employees currently, but we need to think long term and ask about their planned growth. Expects 500 employees in five years. The total storage required is:

*51 MB x 500 Employees = 25.5 GB*

In summary, we have the following non-functional requirements:

* 10 concurrent users.
* Manages 500 employees.
* 25.5 GB data volume forecast.

o Relational & unstructured.

* Not a mission-critical system.

## Identifying Modules

Based on our requirements, we have these initial entities – *Employee*, *Vacation*, *Salary*. The latter will include bonuses and promotions.

Based on our entities, we would define the following modules to serve our tasks:

* *Employees Service* – manages the full CRUD operations on the Employee entity.
* *Salary Service* – manages the salary approval workflow.
* *Vacation Service* – manages the employee’s vacations.

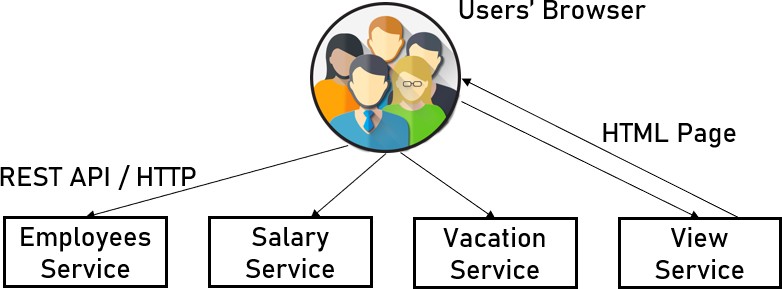
Additionally, since our application is web-based, we would need a *View Service* to return static files to the browser (*HTML*, *CSS*, and *JavaScript*).

Here is our architecture diagram so far:

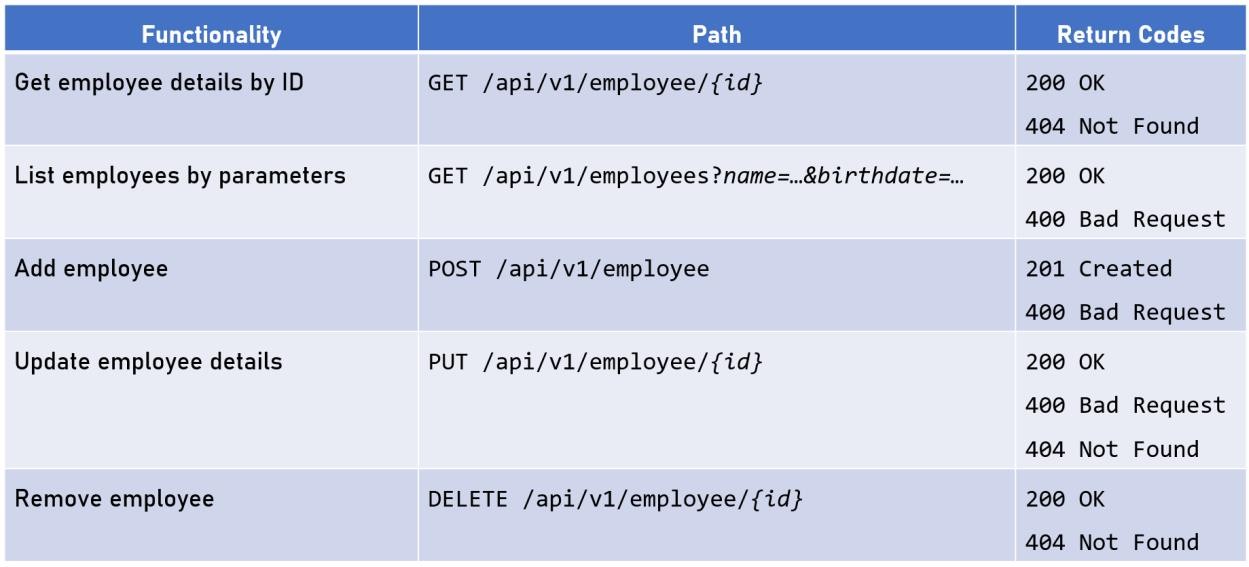


The standards for the above scenario is *HTTP* and *REST API*

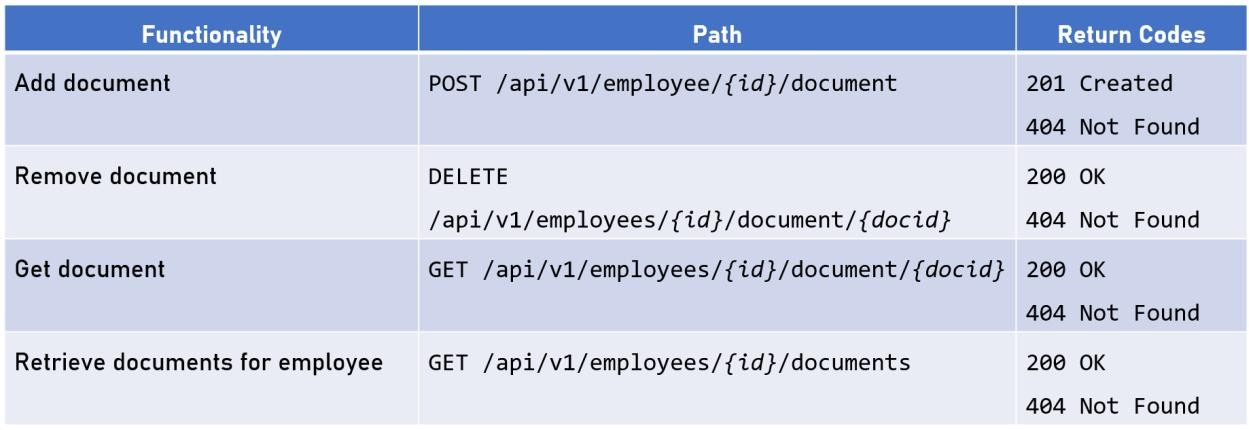
Our diagram so far:



*Employees API*:



And for the *Documents API*:



## Designing The Salary Service

Responsibilities:

* Allows managers to ask for an employee’s salary change.
* Allows HR representatives to approve or reject the request.

*REST API* functionalities:

* Add salary request.
* Remove salary request.
* Get salary requests.
* Approve salary request.
* Reject salary request

## Designing The Vacation Service

* Allows employees to manage their vacation days.
* Allows HR representatives to set available vacation days for employees.

TODO…

## View Service

Create a website directly from a repository on GitHub.com.

GitHub Pages are public webpages hosted and published through GitHub.

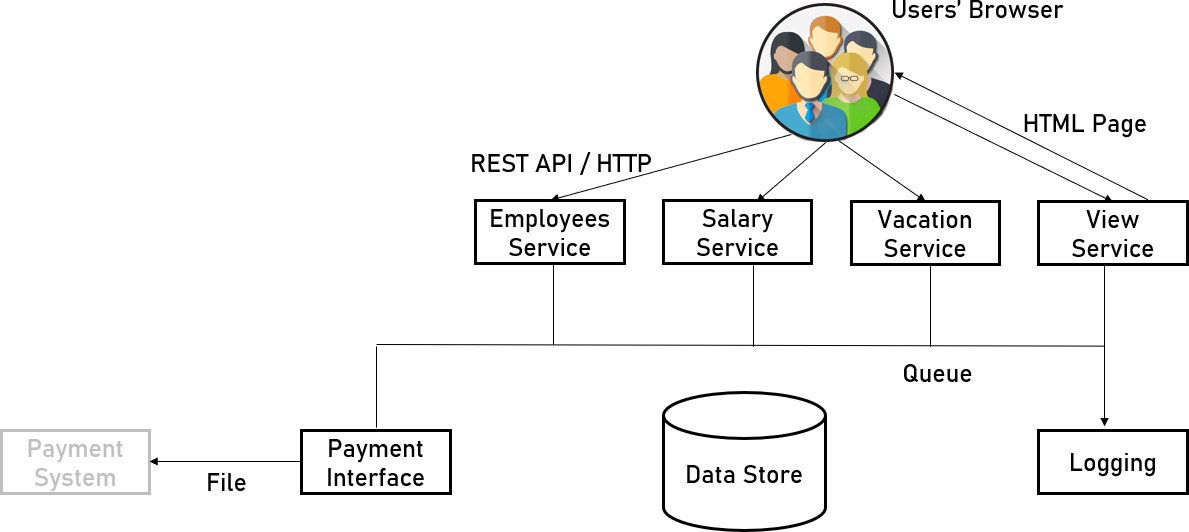
## Infrastructure Considerations

We need to consider the infrastructure and deployment configurations like:

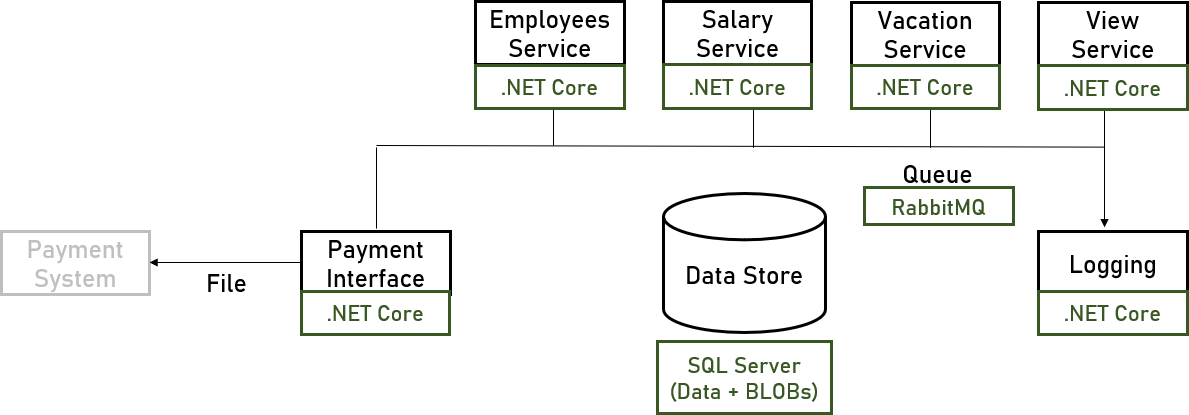
* *CI/CD* – automated integration and delivery. *GitHub pipelines*
* *Containerization* – packaging and virtualization for all application parts. *Docker* is a popular technology choice.
* *Orchestration* – automatic managing, scaling and updating for all the different modules. *Kubernetes* and *Docker Swarm* are perfect for the job.
* *Environment Infrastructure* – data centers or cloud providers.

## Final Architecture Diagrams

*Logical Diagram - describes the different modules we designed and how they communicate with each other*



*Technical Diagram - shows what the technology stack is for each service.*



*Physical Diagram - It depicts the redundancy of each module and how to develop and deploy it on the hardware*