Queue Controller:

Development of a Redis management platform with serverless microservices on AWS

Alberto Schiabel

September 25, 2019

UNIVERSITÀ DEGLI STUDI DI PADOVA

Bacherlor's degree in
Computer Science

2018 / 2019

DIPARTIMENTO MATEMATICA

Table of Contents

UNIVERSITÀ DEGLI STUDI DI PADOVA

Business Context

Pagination.com is a company that:

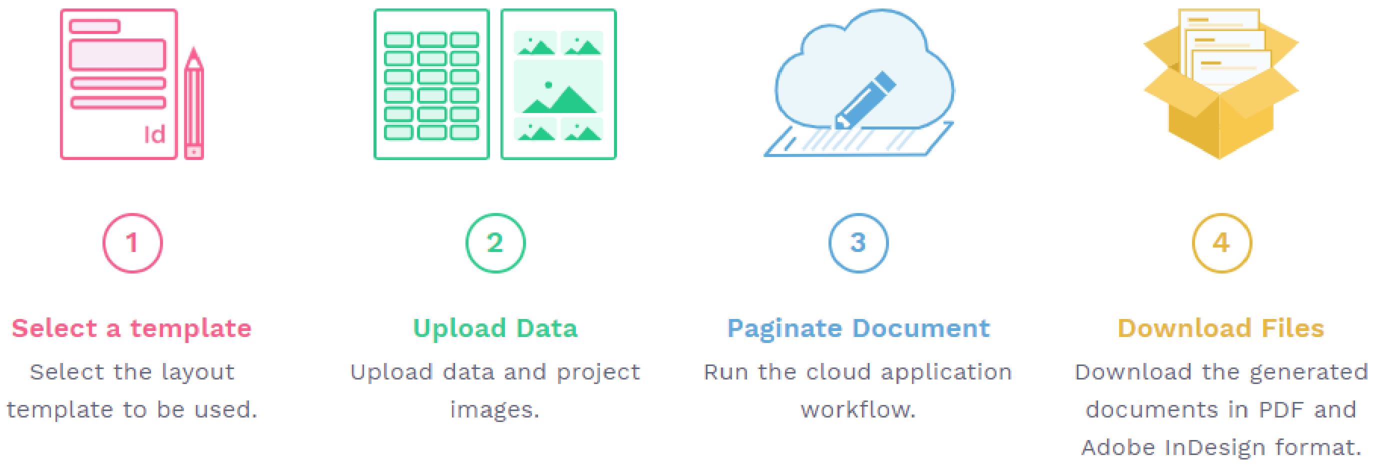
- ❖ Offers automatic layout cloud services
- ❖ Specialized in catalogs and price lists generation
- ❖ Crafts documents from a template and multiple data sources (CSV, SQL, Excel…)
- ❖ Uses cutting-edge tools like AWS and practices like Continuous Integration

It has customers all over the world:

Gardner Denver GEOX

The Project should…

- Offer a simple web UI for non tech-savvy employees
- Interact with business data stored in a Redis database
- Expose a REST API via AWS API Gateway and AWS Lambda
- Offer a dashboard to monitor and control Pagination's main software via a shared Redis database



1 **Select a template**
Select the layout template to be used.

2 **Upload Data**
Upload data and project images.

3 **Paginate Document**
Run the cloud application workflow.

4 **Download Files**
Download the generated documents in PDF and Adobe InDesign format.

Technologies and Languages

- ❖ In-memory data structure store
- ❖ Database that supports lists, hashes, sets, …
- ❖ Uses Lua scripts to perform composite atomic operations

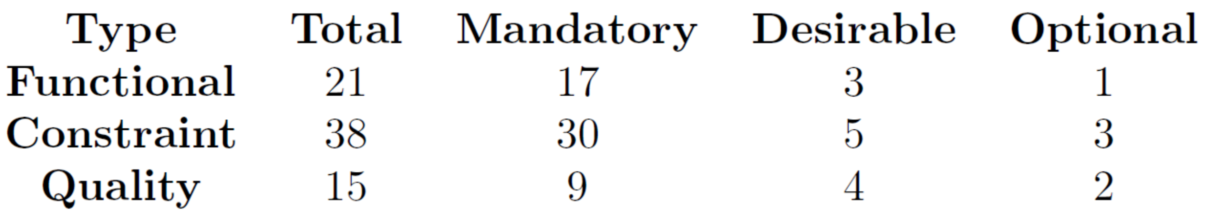
- ❖ Fast container platform used to define build environments running on every system
- ❖ Containers are virtual packages for software and their dependencies

- ❖ Garbage-collected compiled programming language
- ❖ Well adopted by the cloud community
- ❖ Advanced concurrency standard libraries

- ❖ Web UI JavaScript library built by Facebook
- ❖ Huge open-source community
- ❖ Encourages web component composition and functional programming practices

Requirements

- Offer a CRUD interface for business entities (pagination jobs, locks, requests)
- Create an intuitive and responsive client-side web app
- Provide uniform interface to retrieve paginated batches from Redis
- Limit HTTP requests with a client-side cache for data read from Redis
- Set up a Continuous Integration pipeline with Jenkins and Docker
- Set up an Automated Deployment process to create the cloud infrastructure needed to run Queue Controller

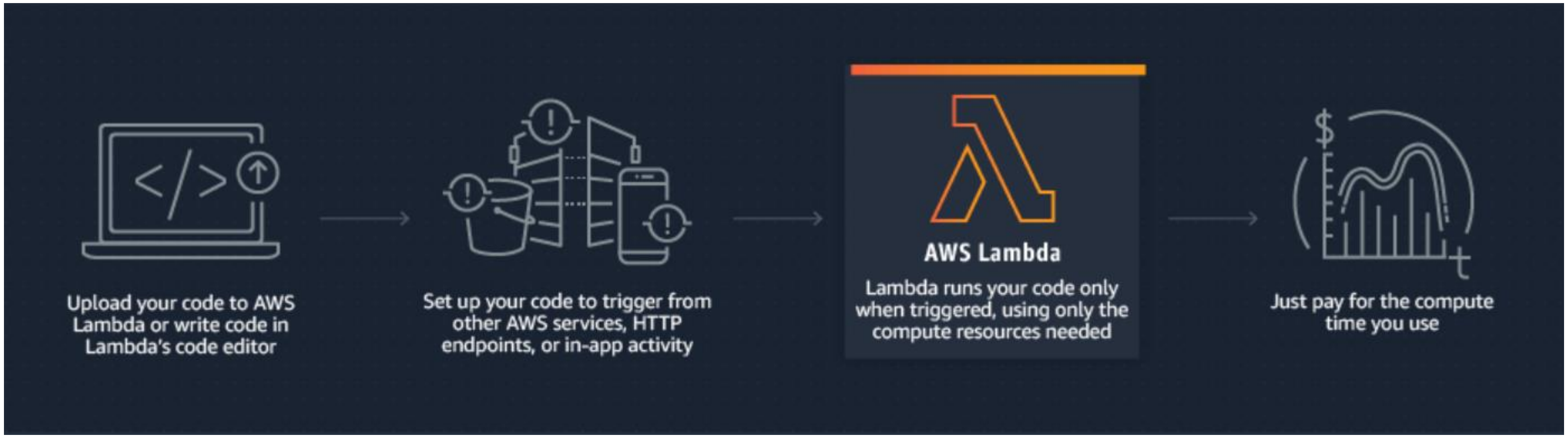


Type	Total	Mandatory	Desirable	Optional
Functional	21	17	3	1
Constraint	38	30	5	3
Quality	15	9	4	2

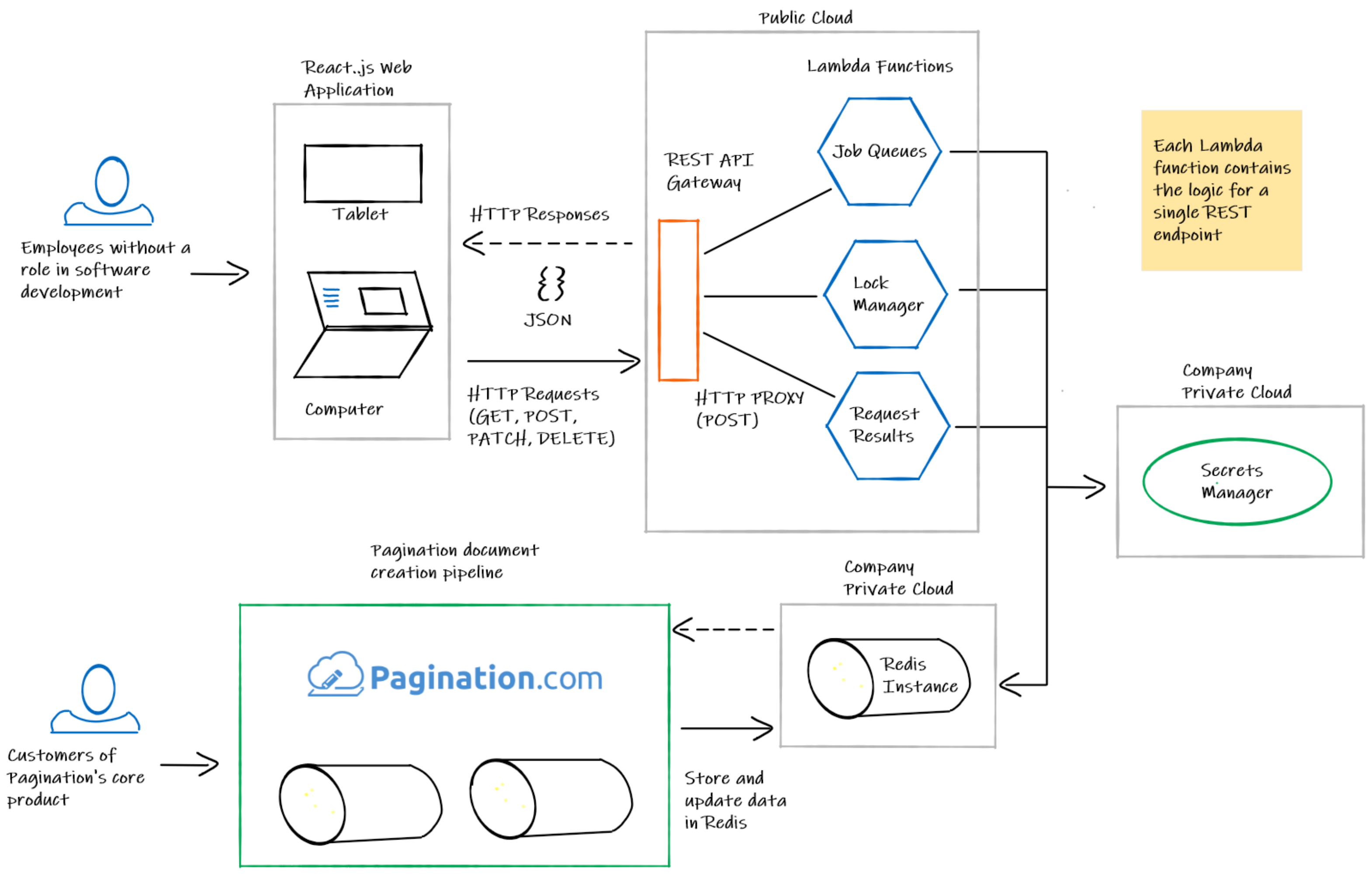
Table 4.4: Summary of the project requirements.

Serverless Cloud

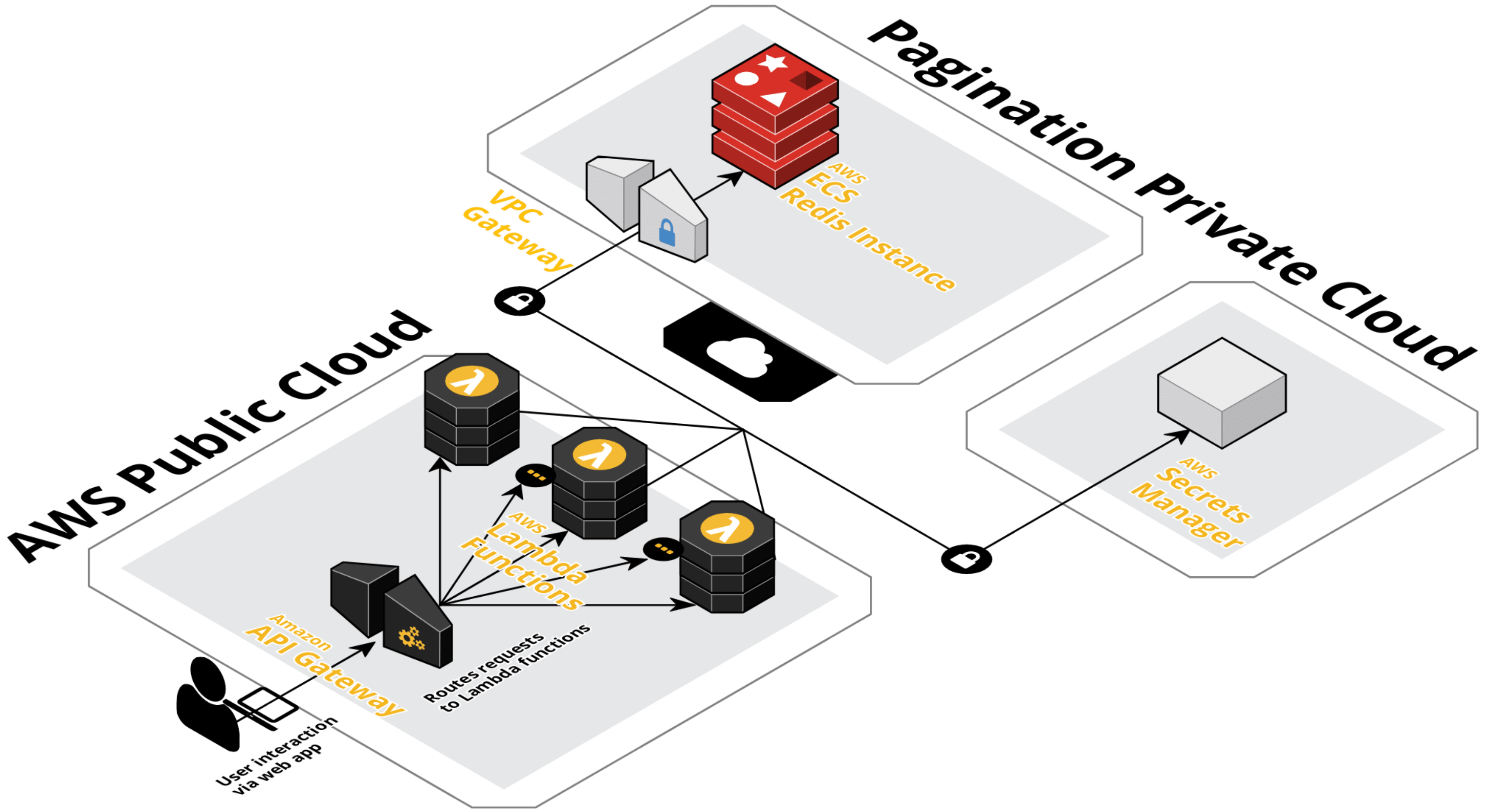
- Relieves the user from manually managing and scaling a server
- Charges based on the execution time, not the rent of a server
- Fully managed by a cloud vendor (AWS, Google Cloud, Microsoft Azure, …)
- Suitable for simple short-running operations
- Stateless, independent server-side functions (Functions as a Service)


Upload your code to AWS Lambda or write code in Lambda's code editor
Set up your code to trigger from other AWS services, HTTP endpoints, or in-app activity
AWS Lambda
Lambda runs your code only when triggered, using only the compute resources needed
Just pay for the compute time you use

System Design


Public Cloud
Lambda Functions
React.js Web Application
Tablet
Computer
Employees without a role in software development
HTTP Responses
JSON
HTTP Requests (GET, POST, PATCH, DELETE)
REST API Gateway
HTTP PROXY (POST)
Job Queues
Lock Manager
Request Results
Each Lambda function contains the logic for a single REST endpoint
Company Private Cloud
Secrets Manager
Pagination document creation pipeline
Pagination.com
Customers of Pagination's core product
Company Private Cloud
Redis Instance
Store and update data in Redis

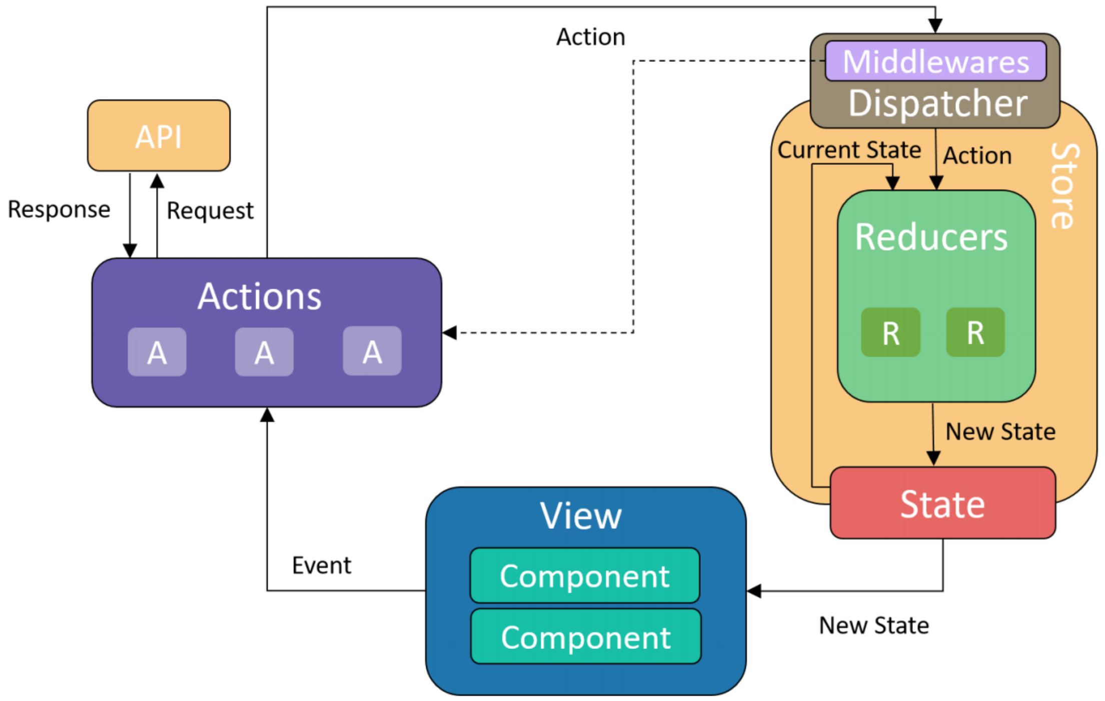
Back-end Design



Front-end Design

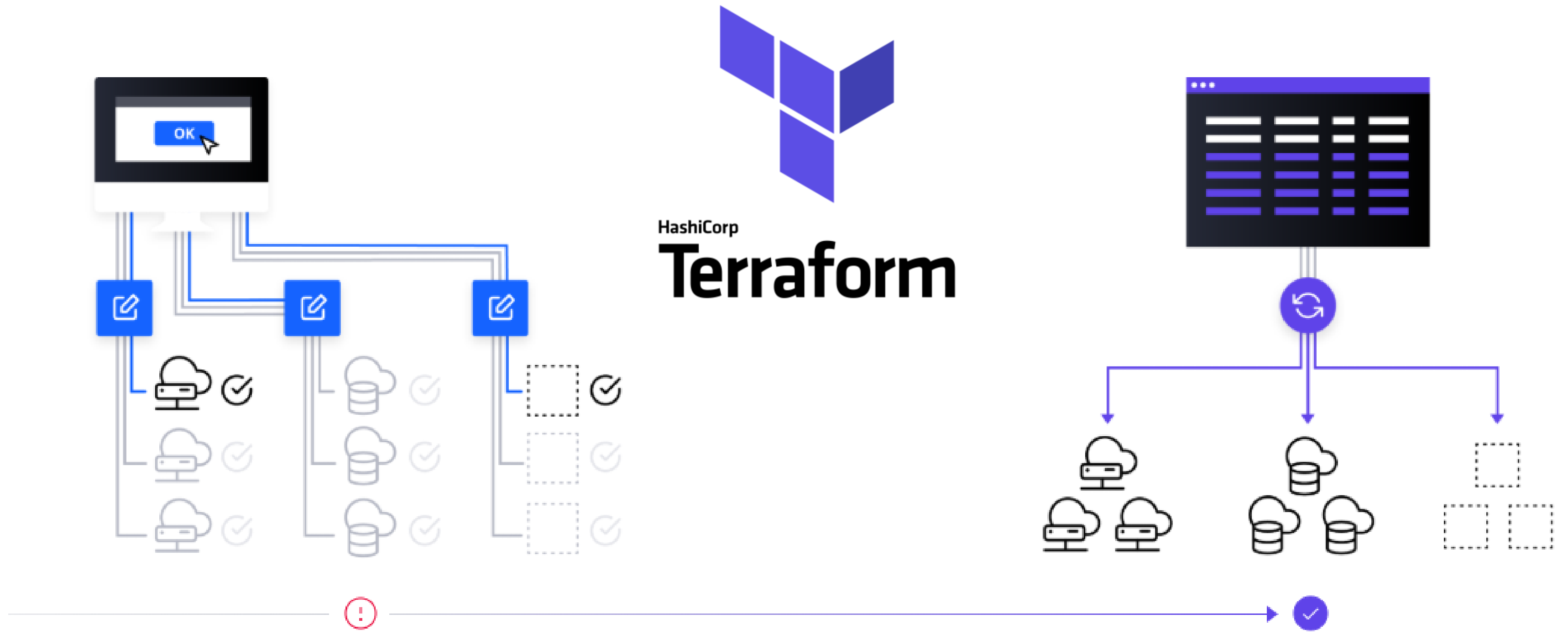
- React.js web application written in TypeScript
- Redux as global state manager
- React Router to handle client-side routing
- SCSS as style preprocessor

TypeScript



Automated Deployment

Shift from manual, error-prone provisioning to **automated** provisioning at scale



Reduce risks and discover errors before they happen

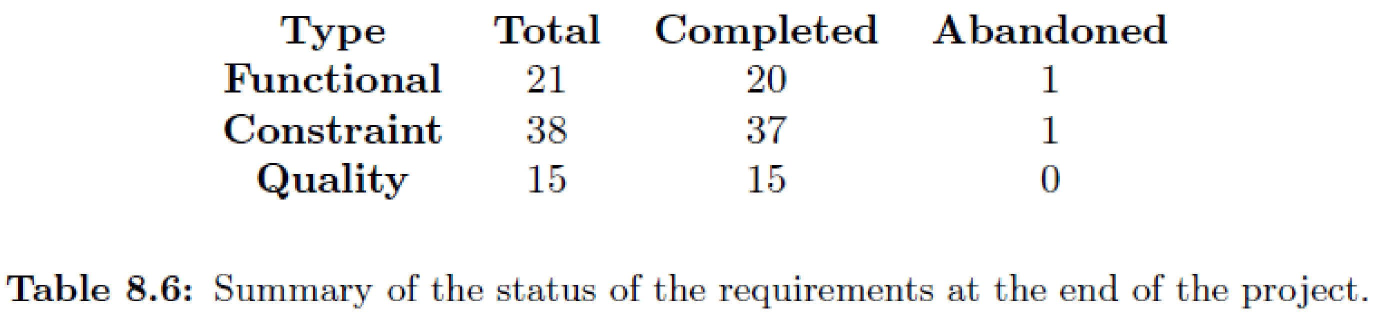
Software Testing

Ginkgo
A Golang BDD Testing Framework

Jest

- Statically analyzed conformity to code standards
- Tried using **TDD**
- **Integration Testing** required more time than Unit Testing
- **100%** code coverage on some software modules

Conclusions



Type	Total	Completed	Abandoned
Functional	21	20	1
Constraint	38	37	1
Quality	15	15	0

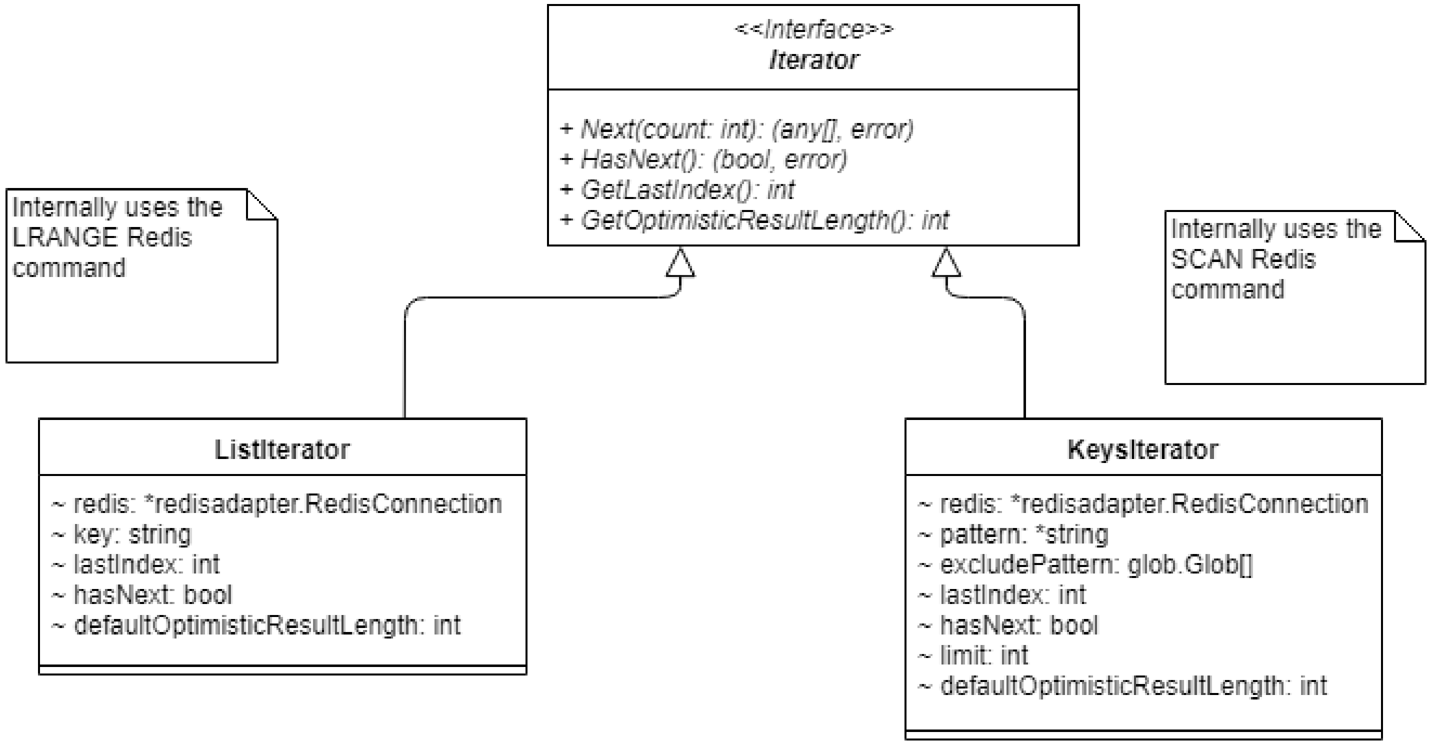
Table 8.6: Summary of the status of the requirements at the end of the project.

Knowledge acquired:

- ❖ Cost-benefit analysis
- ❖ Amazon Web Services
- ❖ Continuous Integration with Jenkins
- ❖ Deployment Automation with Terraform

(A) Redis Iterator

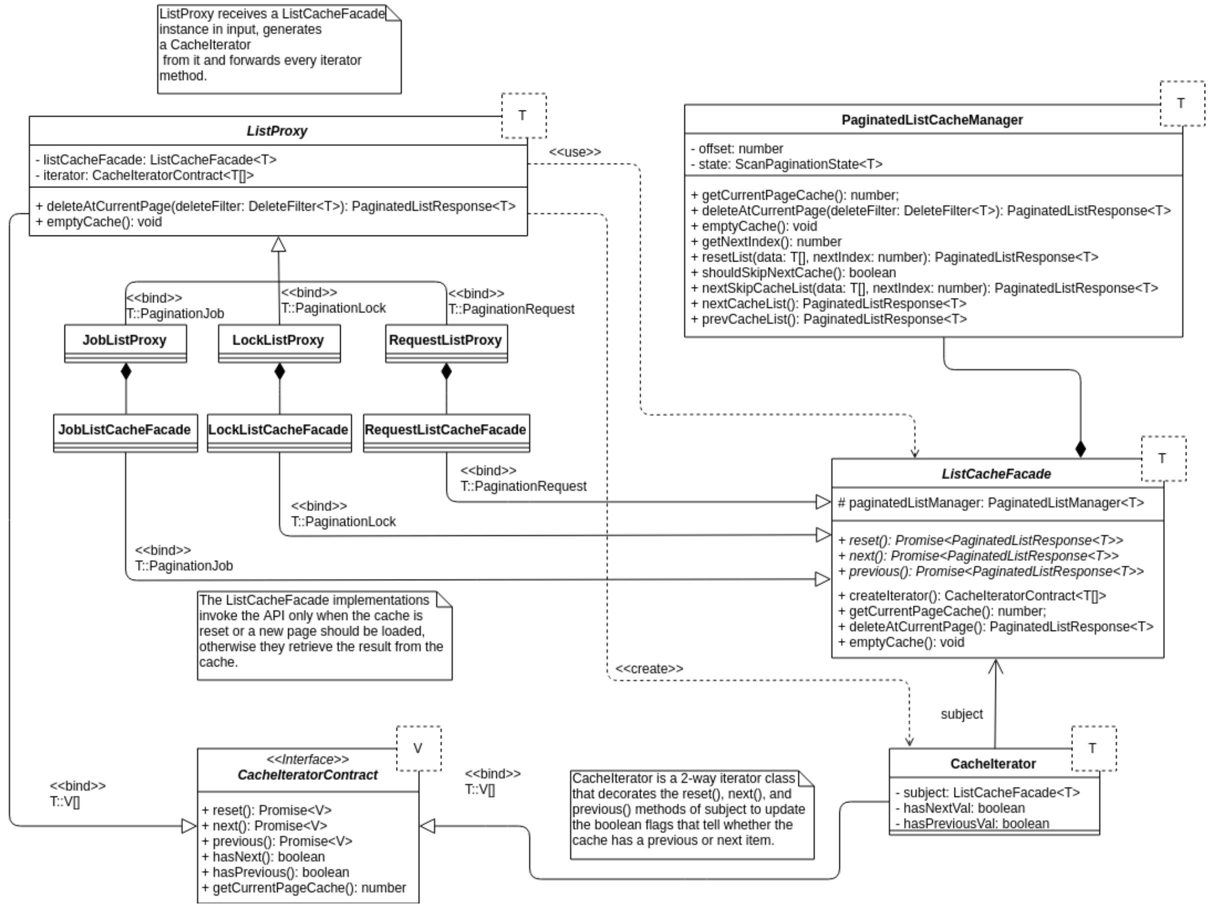
- Adapt to future changes to business data representation in Redis
- Provide uniform interface to retrieve paginated batches from Redis
- Decouple the business data from the Redis data structure to simplify the client-side cache



<<Interface>>
Iterator
+ Next(count: int): (any[], error)
+ HasNext(): (bool, error)
+ GetLastIndex(): int
+ GetOptimisticResultLength(): int
Internally uses the LRANGE Redis command
Internally uses the SCAN Redis command
ListIterator
~ redis: *redisadapter.RedisConnection
~ key: string
~ lastIndex: int
~ hasNext: bool
~ defaultOptimisticResultLength: int
KeysIterator
~ redis: *redisadapter.RedisConnection
~ pattern: *string
~ excludePattern: glob.Glob[]
~ lastIndex: int
~ hasNext: bool
~ limit: int
~ defaultOptimisticResultLength: int

(A) Client-side Cache Module

- Display paginated lists of data
- Reuse as much data as possible without consuming AWS resources
- Decoupled from the UI framework
- Support reset and delete operations



ListProxy receives a ListCacheFacade instance in input, generates a CacheIterator from it and forwards every iterator method.

The ListCacheFacade implementations invoke the API only when the cache is reset or a new page should be loaded, otherwise they retrieve the result from the cache.

CacheIterator is a 2-way iterator class that decorates the reset(), next(), and previous() methods of subject to update the boolean flags that tell whether the cache has a previous or next item.

ListProxy
T
- listCacheFacade: ListCacheFacade<T>
- iterator: CacheIteratorContract<T[]>
+ deleteAtCurrentPage(deleteFilter: DeleteFilter<T>): PaginatedListResponse<T>
+ emptyCache(): void
<<use>>
PaginatedListCacheManager
T
- offset: number
- state: ScanPaginationState<T>
+ getCurrentPageCache(): number;
+ deleteAtCurrentPage(deleteFilter: DeleteFilter<T>): PaginatedListResponse<T>
+ emptyCache(): void
+ getNextIndex(): number
+ resetList(data: T[], nextIndex: number): PaginatedListResponse<T>
+ shouldSkipNextCache(): boolean
+ nextSkipCacheList(data: T[], nextIndex: number): PaginatedListResponse<T>
+ nextCacheList(): PaginatedListResponse<T>
+ prevCacheList(): PaginatedListResponse<T>
<<bind>> T::PaginationJob
<<bind>> T::PaginationLock
<<bind>> T::PaginationRequest
JobListProxy
LockListProxy
RequestListProxy
JobListCacheFacade
LockListCacheFacade
RequestListCacheFacade
<<bind>> T::PaginationRequest
<<bind>> T::PaginationLock
<<bind>> T::PaginationJob
ListCacheFacade
T
# paginatedListManager: PaginatedListManager<T>
+ reset(): Promise<PaginatedListResponse<T>>
+ next(): Promise<PaginatedListResponse<T>>
+ previous(): Promise<PaginatedListResponse<T>>
+ createIterator(): CacheIteratorContract<T[]>
+ getCurrentPageCache(): number;
+ deleteAtCurrentPage(): PaginatedListResponse<T>
+ emptyCache(): void
<<create>>
subject
<<bind>> T::V[]
<<Interface>>
CacheIteratorContract
V
+ reset(): Promise<V>
+ next(): Promise<V>
+ previous(): Promise<V>
+ hasNext(): boolean
+ hasPrevious(): boolean
+ getCurrentPageCache(): number
<<bind>> T::V[]
CacheIterator
T
- subject: ListCacheFacade<T>
- hasNextVal: boolean
- hasPreviousVal: boolean