UNIVERSITÀ
degli STUDI
di CATANIA

Dipartimento di Ingegneria Elettrica Elettronica Informatica

CORSO DI LAUREA IN INGEGNERIA INFORMATICA

CodeMeet: a serverless web application project for a software ticketing system on Amazon Web Services

---

Anno accademico [2020-2021]

Candidato: Alessio Leone

Relatore: Simone Palazzo

Abstract

CodeMeet is a web platform created for programmers that are struggling with errors, bugs and any kind of problems that may come to exist during software development. Every user can post a ticket to report a problem, so other coworkers can solve it, or close tickets that are with status “open”. An open ticket is made of two main components: title, which briefly describes the problem, and text, in which the user deeply describes the problem and gives all the kind of information that could be helpful to solve it more quickly; A closed ticket also has a solution, which will be added by the user who solved it.

CodeMeet is built using Amazon Web Services serverless architecture, which is, as time goes by, more and more preferred to the server architecture because the server management is not handled by the developer but by AWS itself, and the application is also auto-scaled (according to the incoming traffic, AWS allocates the exact number of required resources in that moment).

Sommario

CodeMeet è un sito web per programmatori che incorrono in errori, bugs e ogni tipo di problema riscontrabile durante la fase di sviluppo di un software. Ogni utente può postare un ticket per segnalare un problema, affinchè altri colleghi possano risolverlo, o chiudere dei ticket che denotano lo status *open*. Un ticket aperto è costituito principalmente da 2 componenti: il titolo, che descrive brevemente il problema, e il testo, in cui l'utente descrive più approfonditamente il problema dando tutti i tipi di dettagli che potrebbero essere utili a risolvere il ticket più velocemente. Un ticket *closed*, invece, ha anche una soluzione che viene aggiunta dall'utente che lo risolve.

CodeMeet è costruito usando l'architettura serverless di Amazon Web Services che, con il passare del tempo, è sempre più usata rispetto all'architettura server, in quanto la gestione e manutenzione dei server è fatta non dallo sviluppatore, ma dallo stesso Amazon Web Services. Inoltre, l'applicazione ha anche la proprietà di auto-scaling, ossia il ridimensionamento automatico (secondo il traffico in entrata, Amazon Web Services alloca l'esatto numero di risorse richieste in quel momento).

Contents

Chapter 1

Introduction

During software development it is not uncommon to stumble in software bugs, defined as an error, flaw or fault in a computer program or system that causes it to produce an incorrect or unexpected result instead of the one expected. Most bugs exist because there are errors in the source code, while others can be related to the IDE (integrated developer environment) with which the programmer is writing the code.

When a programmer stumbles upon a bug there is a certain amount of time that is wasted, resulting therefore in a loss of valuable work-time for either the company or the programmer themselves.

Also, while for a programmer a specific bug could take days to solve, for another it may be a matter of few minutes (because their job has to do more with that field or because they have already stumbled upon that specific bug, or other unknown reasons). It is thus necessary for programmers to cooperate, in order to increase productivity: hence, the need of a platform

where the members of a team can communicate between each others.

This kind of platform better suits the name of Ticketing sites, a place where users can try to ask for help for something relative to their development field: a ticketing system is a management tool, or a customer service tool, that processes and catalogs customer service (the website users) requests. Tickets, also known as cases or issues, need to be properly stored alongside relevant information, as these will be customer service (users) data collections, making easier, over time, to resolve more complicated problems.

Once a company decides which ticketing system to use, it needs to be set up in order to be ready to run. This happens by choosing a hosting solution.

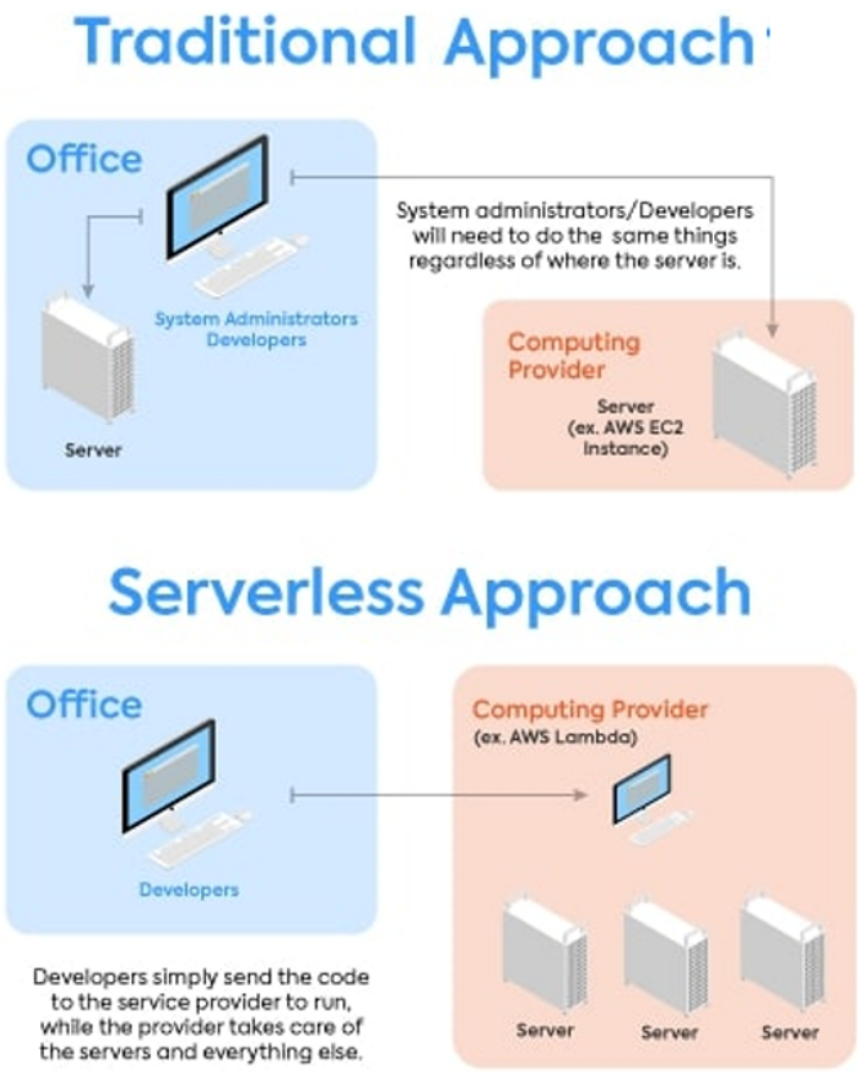
Every website, indeed, included the above mentioned ticketing systems, in order to be accessible over the internet needs to be hosted through a hosting service, which is a company that provides storage on a server and internet connectivity (usually in a data center).

For security, customization and optimization reasons, a company prefers to choose a dedicated hosting service rather than a common hosting service, with the difference being that in a dedicated hosting service the client does not share the server, in which its website is hosted, with anyone else: the server is, indeed, dedicated.

In the dedicated server architecture the web application is hosted on a server, which is by definition a computer (remote, usually, and not shared with others because it is dedicated) that provides data to other computers over Local Area Network (LAN) or Wide Area Network (WAN). The server needs to provide all the resources (such as CPU, RAM and Storage) needed for the web application: these resources must be configured according to website expected users flow (users connected to the web application at the same time), or, in more technical terms, according to the capacity planning, namely the process of determining the production capacity needed by an organization to meet changing demands for its products.

When the users flow changes (this may be intended in a positive or negative note, the site could have more or less users visiting the site than usual), the server has to be scaled according to the new expectations: if the server needs to provide data to more users than what it can handle, there will be a service malfunction due to overloading (the server cannot handle more requests than what its resources allow it to do).

The problem is that the server needs to be designed, handled and it also needs to be constantly treated (and sometimes, according to the users flow, even redesigned).



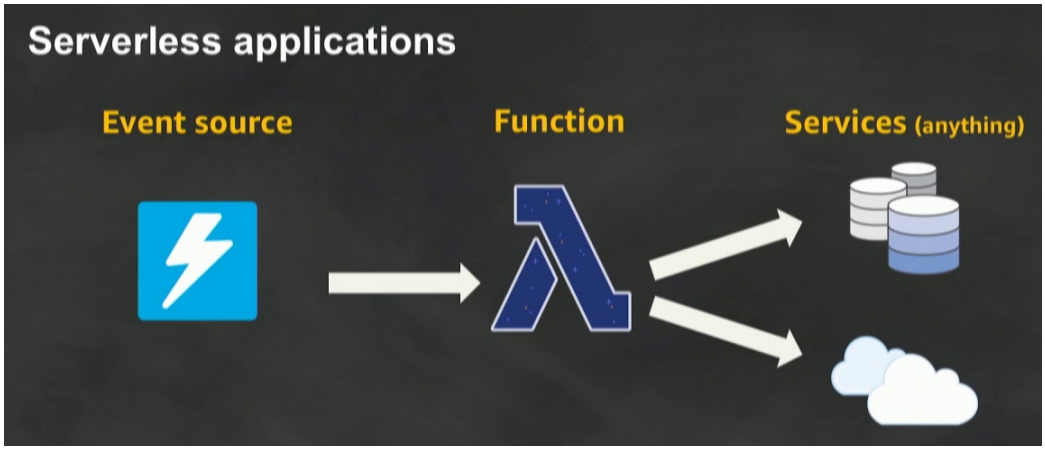
Traditional server approach versus serverless approach, goodfirms.co

Serverless computing [1] solves all the problems regarding the server handling: it is a cloud-native development model that allows developers to build and run their applications without worrying about the server management. Despite the name, serverless is not completely unbounded by servers, in fact servers still play a big role, but they are handled directly by cloud providers

(and not by developers) that handle the routine work of provisioning, mantaining and scaling the server infrastructure.

Once deployed, serverless based application respond to demand autoscaling their resources according to what is required, automatically increasing or decreasing resources as needed.

Serverless apps are deployed in containers, developed and uploaded by developers, that automatically launch on demand when called (namely triggered by an event).



Serverless model [2]

Serverless computing offerings typically fall into two groups, Backend-as-a-Service (BaaS) and Function-as-a-Service (FaaS): usually, when developers refer to "Serverless", they are referring to FaaS models, where developers

still write their custom server-side logic, but it is run in the containers fully managed by the cloud services provider.

What is FaaS? *tritekconsulting.com*

The main cloud providers rely on the FaaS model: AWS Lambda for Amazon Web Services, Azure functions for Microsoft Azure and multiple choices for Google Cloud [3].

Lastly, another reason to choose a Serverless solution is tied to the website sustaining prices: while a server infrastructure can take a lot of money because the servers must run all the time, 24 hours per day, and with all

the maximum capacity that the server was designed at first, the Serverless solution does instead pay attention to the pricing as Serverless computing does not hold resources in volatile memory; computing is rather done in short bursts with the results persisted to storage.

When an app is not in use, there are no computing resources allocated to the app: pricing is therefore based on the actual amount of resources consumed by an application [4].

While in Chapter 2 we will further explore the most famous ticketing sites, in the next Chapters we will see one by one the technologies that were used for the development of CodeMeet and how they all are tied each others.

Chapter 2

State of the art

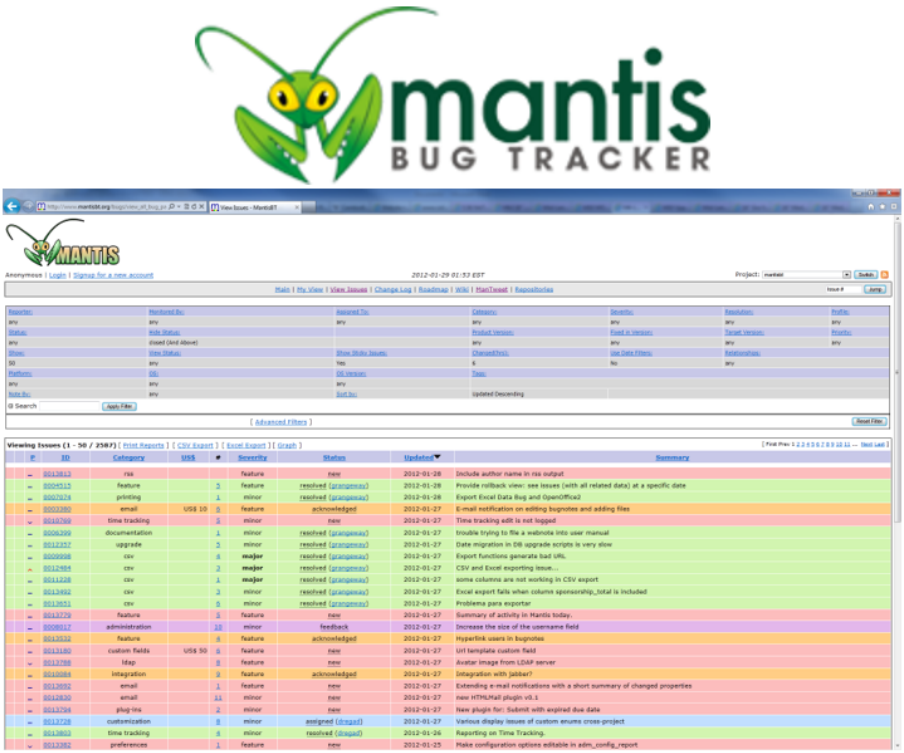
2.1 Overview

There is a vast number of web applications that are based on the same purpose of CodeMeet, namely: ticketing sites where users can try to ask for help for something relative to their development field.

A ticketing system is a management tool, or a customer service tool, that processes and catalogs customer service (the website users) requests. Tickets, also known as cases or issues, need to be properly stored alongside relevant information, as these will be customer service (users) data collections, making easier, over time, to resolve more complicated problems.

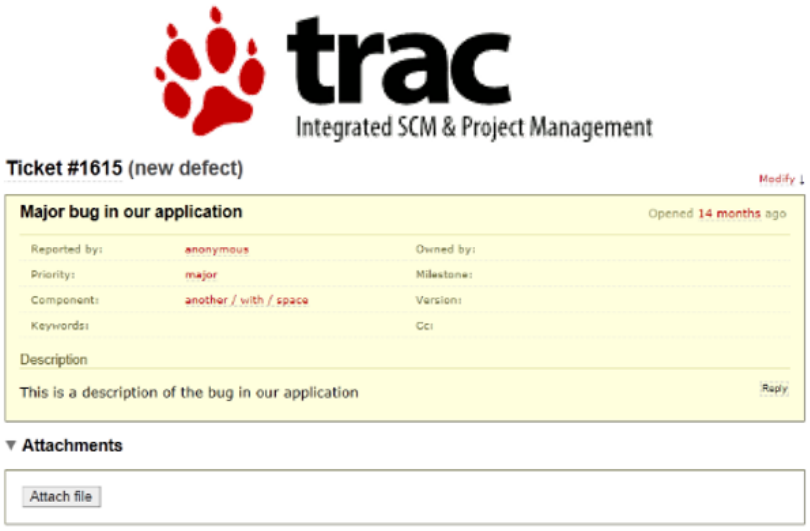
While the first two sites are perfect ticketing systems examples, next will follow more systems that are less focused on ticketing and with a lot of additional features essential to web development.

2.2 MantisBT



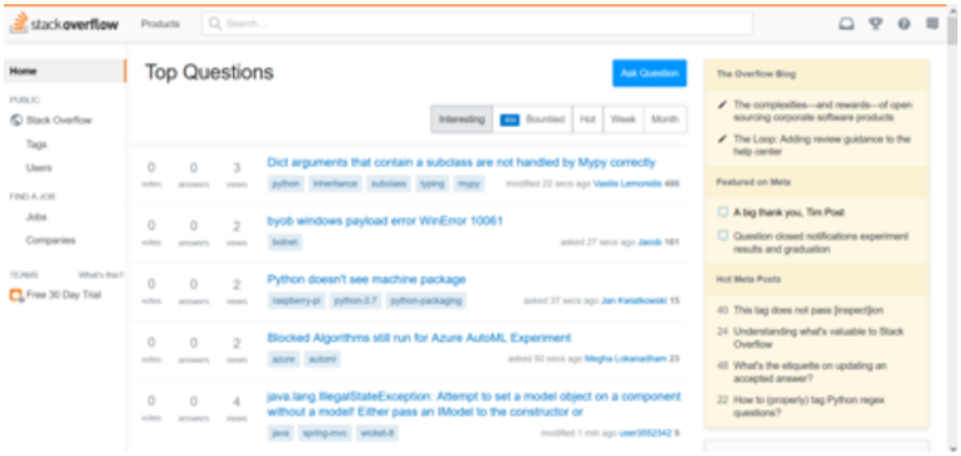
MantisBT [17], released in 2000, is a mature, open source bug tracker and a popular choice. The most common use of MantisBT is to track software defects, but it is often configured by users to serve as a more generic issue tracking system and project management tool. It has several unique features, such as mobile apps for Android, iOS, and Windows Phone and the ability to Tweet tickets in. It supports CVS, SVN, and git, and it provides a SOAP API to write integrations in as well. Also it supports a wide variety of database back ends.

2.3 TRAC



TRAC [6] is an enhanced wiki and issue tracking system for software development projects. It uses a minimalistic approach to web-based software project management and integrates with source control (git, Mercurial, and various common DVCSs), it has a limited feature set by design, and as a result it is easy to use and get accustomed to. TRAC allows wiki markup in issue descriptions and commit messages, creating links and seamless references between bugs, tasks, changesets, files and wiki pages. A timeline shows all current and past project events in order, making the acquisition of an overview of the project and tracking progress very easy, and the roadmap shows the upcoming milestones.

2.4 Stack Overflow



Stack Overflow, created in 2008 by Jeff Atwood and Joel Spolsky, is a question and answer website for programmers written in C´ using the ASP.NET Model–View–Controller framework. It was designed as a place where people from all around the world could share help, doubts and advices about computer programming topics, it does indeed features questions and answers about a several number of development fields.

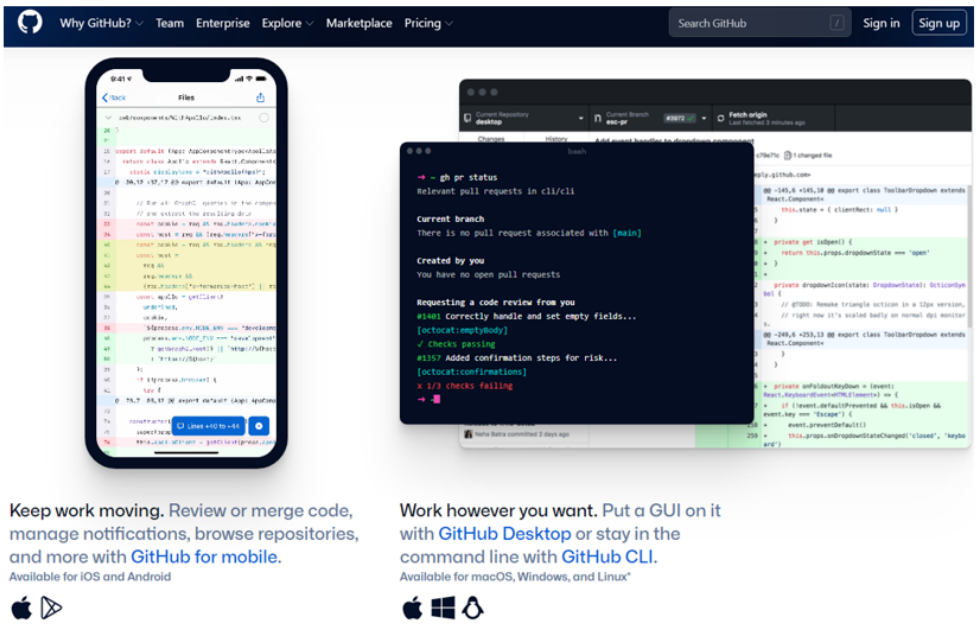
Stack Overflow was recently (2 June 2021) sold to Prosus, a Netherlands-based consumer internet conglomerate for 1.8 billion.

The website, which counts over 14 million registered users, serves as a platform for users to ask and answer questions, and, through membership and active participation, to vote questions and answers up or down and edit questions and answers. Users of Stack Overflow can earn reputation points and "badges"; for example, a person is awarded 10 reputation points for receiving an "up" vote on a question or an answer to a question, and can receive badges for their valued contributions. Besides the "social status" linked to the digital user, having a good reputation is, to date, useful when someone is looking for a job: an high reputation is evidence of knowledge in some fields, as there are different tags that help to better organize questions and topics. The most discussed topics on the site are: JavaScript, Java, PHP, Android, Python, jQuery, and HTML.

Futhermore, users unlock new privileges with an increase in reputation like the ability to vote, comment, and even edit other people's posts: a new registered users will not be able to downvote questions or edit answer, in order to avoid chaos and unfairness.

Stack Overflow also has a Jobs section to assist developers in finding their next opportunity.

2.5 GitHub



GitHub development began in 2007 and since 2018 is a Microsoft Branch. While Stack OverFlow is more "Question and Answer based", GitHub is a provider of Internet hosting for software development and version control using Git: it is estimated to have over 40 million users and more than 190 million repositories, making it the largest host of source code in the world.

It provides access control and several collaboration features such as bug tracking, feature requests, task management, continuous integration and

wikis for every project.

GitHub offers its basic services free of charge. Free GitHub accounts are commonly used to host open-source projects. The free plan allows unlimited collaborators, but restricts private repositories.

Tickets in GitHub are used to define and track issues and potential issues at the repository level. A ticket is created:

- To note any problem observed in the application
- To suggest any feature, enhancement or fix
- To define any feature or enhancement tasked for development

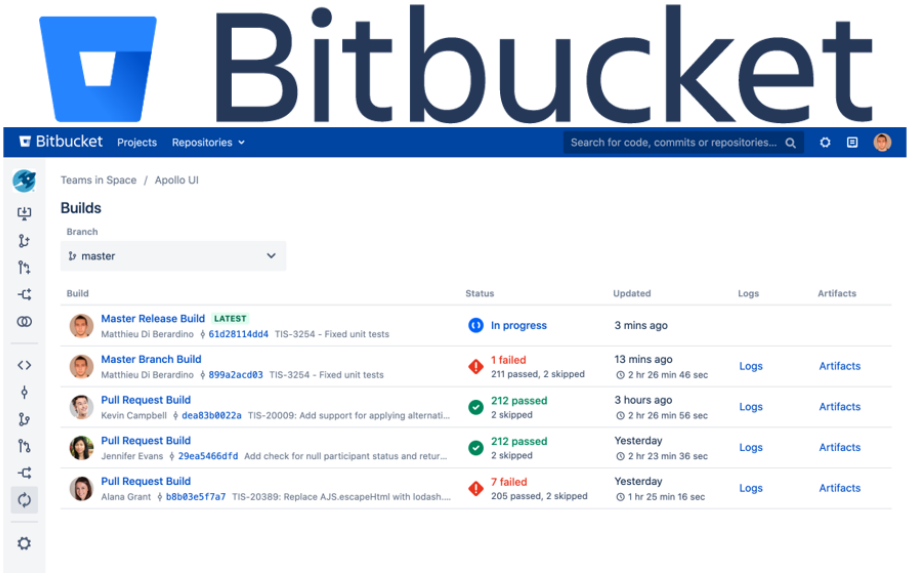
The ticket is, in the case of GitHub, restricted to the project related to the repository. A ticket is made of title and description.

If the ticket is created for bugs-related issues, it should include:

- the steps to reproduce the defect, and any relevant context such as he environment in which the defect took place
- the expected result and the actual result
- the interested area of code, as in big projects it can be a problem to identify a small bug

Furthermore, the commit history is public and accessible, with even frequency contribution and members; there are even code review and comments available.

2.6 BitBucket



Bitbucket is a Git-based source code repository hosting service owned by Atlassian. Bitbucket offers both commercial plans and free accounts with an unlimited number of private repositories. Bitbucket is mostly used for code and code review, it supports the following features:

- pull requests with code reviews and comments
- a continuous delivery service
- 2 step verification

- code search (alpha version)
- Git Large File Storage (LFS)
- documentation, including automatically rendered README files in a variety of Markdown-like file formats
- issue tracking.

While GitHub is free for public repositories and paid for private repositories, BitBucket has free private repositories.

2.7 GitLab

GitLab is a web-based DevOps life-cycle tool created by Ukrainian developers Dmitriy Zaporozhets and Valery Sizov that provides a Git repository manager providing most of the GitHub features.

Its purpose is to automate the entire DevOps life-cycle, from planning through to creation, build, verification, security testing, deployment and monitoring.

GitLab is highly scalable and can be hosted on-premises or on cloud storage, it also allows all repositories to be up to 10 gigabytes in size. GitLab

currently does not have any limits on how large a single file can be, as long as it stays under the 10 gigabyte limit.

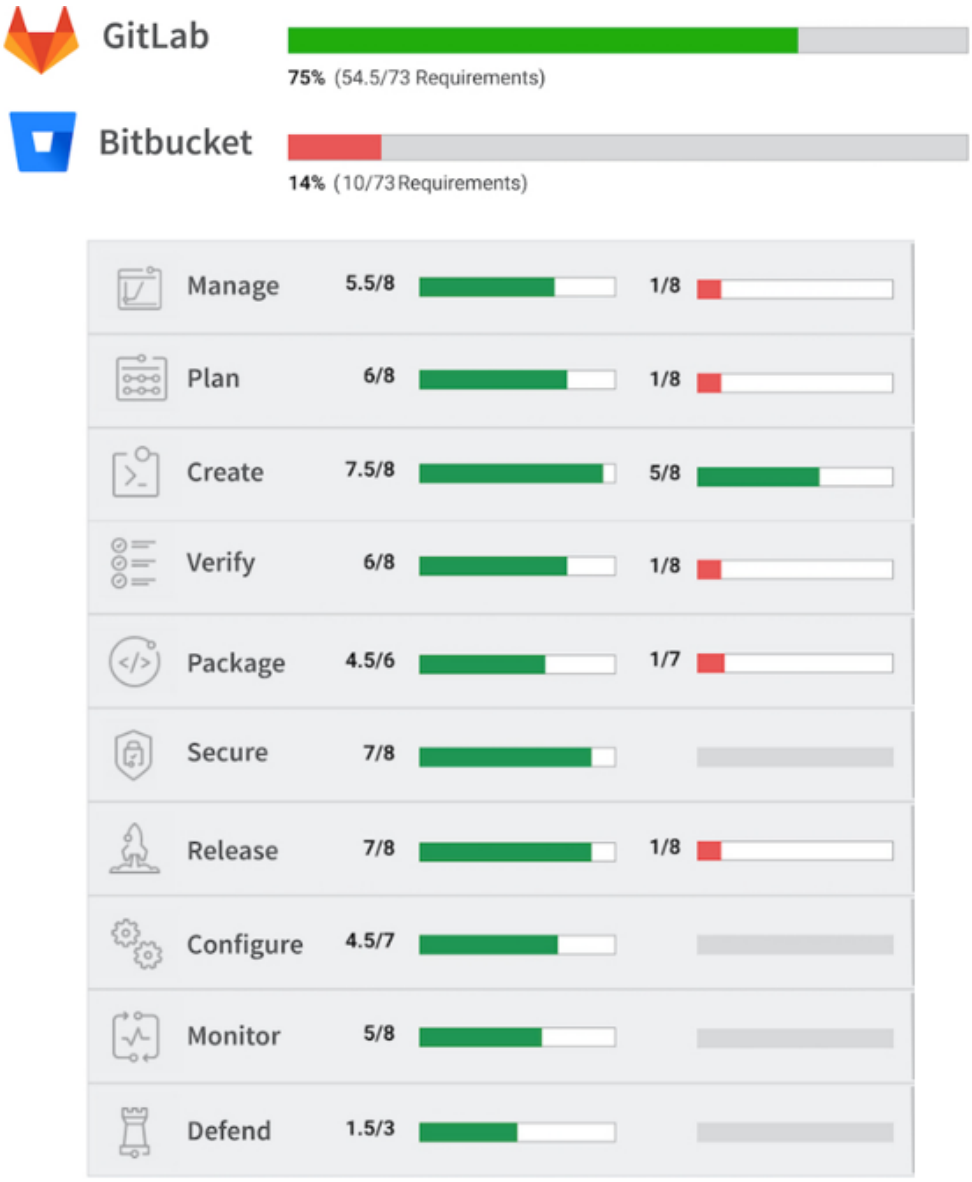
The main difference from GitHub, just like BitBucket, is that for private repositories it is free, while GitHub its not: GitLab is used by companies that want their code to be private, completely free.

The difference between GitLab and BitBucket is that GitLab has a slower interface than BitBucket's one, and its less user friendly: BitBucket is more used by companies because of these 2 factors, but looking at their features, GitLab is better than BitBucket in almost every aspect.

The first plus point for GitLab is that it is completely free, while BitBucket is free but only with a limit of 5 people who can access the company codebase [7].

In conclusion, GitLab and Bitbucket are the best solution if the repositories have to be private (considering a free account).

Furthermore, GitLab is better than BitBucket for almost any feature (Requirements are intended as features that the services in comparison should offer):



GitLab versus Bitbucket comparison based on features [7]

2.8 CodeProject

Code Project is a community for computer programmers with articles on different topics and programming languages such as web development and software development.Every registered user can gain reputation and so unlock different privileges such as the ability to store personal files in the user's account area, have live hyperlinks in their profile biography, and more. Members can also write and upload their own articles and code for other visitors to view.

Articles can be related to general programming, GUI design, algorithms or collaboration. Most of them are uploaded by visitors and do not come from an external source. Nearly every article is accompanied with source code and examples which can be downloaded independently. Most articles and sources are released under the Code Project Open License (CPOL).

Code Project employs a rating and comment system that helps to filter

the good articles from the poor. It also has forums, and is a resource for resolving difficult software development issues.

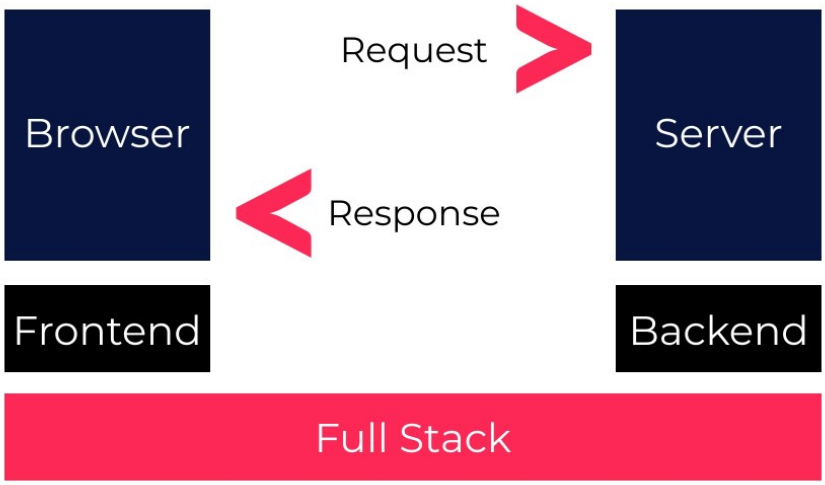
Rather than being just a collection of samples, contributors are encouraged to explain concepts and ideas, and discuss design decisions. There is also the section "Tips and Tricks" where are posted short code snippets that don't fit the requirements for an article.

Code Project strives to be a wealth of information and a valuable resource. The site encourages users to share which source code or knowledge they can in order to help the community grow. Code Project also conducts interviews with notable developers.

Chapter 3

Technology

3.1 Defining FrontEnd and BackEnd



How the web works *intagleo.com*

When a user visits a certain domain using the browser, it sends requests to the server, and the server, in return, sends back responses which consist of files that the browser uses to render properly the website required.

Frontend development (also called client-side development, which consists of HTML, CSS and JavaScript) is the website's backbone and it is related to elements that clients actually see in their browsers, and it is responsible for the User Interface and also the entire user-facing experience of a website.

Backend development has to deal with server-side business logic, database interactions (accessing, storing, fetching), performance optimization, and security. There are a lot of languages that are used in backend.

In our web application, based on serverless architecture, we use Amazon Web Service S3 (Simple Storage Servive) as a service hosting provider to host frontend related files such as HTML/CSS/JS.

With Javascript, via fetches, the Frontend is connected to the Back-End, through AWS API Gateway (that defines methods, paramethers, query strings and other requirements needed for the call) used as an intermediary to AWS Lambda functions.

Lastly, the AWS Lambda functions are the backend's heart, where each function has an own purpose and it is connected to the Database, for which AWS RDS (Relational Database Service based on MySQL Engine) is used.

3.2 HTML

HTML (HyperText Markup Language) [8] was created in late 1980 by Tim Berners-Lee and it is the standard markup-language for documents designed to be displayed in a web browser. Used to describe the content and the structure of a webpage, it also supports CSS and JS resources via the *link* tag.

When the user visits a website, their browser sends a request to the server which replies with an HTML file, linked to eventual CSS (that defines look and the content's layout) and JS (scripts that affects the behaviour and the content itself) files: HTML describes the structure of a webpage semantically, with HTML elements as the building blocks of these webpages. Each HTML element is delineated by tags (used by browsers to correctly display the page) such as *img*, *a*,*input*, *p*, ..., and it also has attributes and values (such as, for example, `images : src="x.png"`).

An HTML Document is made of two parts: a head, where metadata (data that does not appear in the rendered webpage), and a body, which

contains the elements viewed in the webpage, are stored. The body consists of elements used to organise the webpage contents such as `p` (paragraph), `h1...h6` (headers), `article` (it usually follows right after the body), `section` (generic document section), `header` (introductory document section), `footer` (conclusive document section).

HTML, from its birth to today, had 5 versions:

- HTML 2, released in 1995
- HTML 3, released in 1997, is the first edition standardized exclusively by W3C (World Wide Web Consortium)
- HTML 4, released in the same period of HTML 3, differs from it because of 3 possible variants
- HTML 5, released in 2014, is, as of today, the latest version.

3.3 CSS

CSS (Cascade Style Sheets) [9], created by Hakon Wium Lie in 1994 (at the time he was a coworker of Tim Berners-Lee, HTML's creator) and shortly after accepted by W3C, is a style sheet language used to describe the presentation of an HTML document: it defines the webpage's layout and how its content is organized. It is designed to enable the separation of presentation and content, including layouts, colors, fonts; this results in improved content accessibility, more flexibility and it allows more webpages to share a single css file.

A style sheet consists of a list of rules, which is made up of one or more selectors and a declaration block where properties (name of a stylistic property for an element) and their values are listed. A selector, always described descending, declares to which part of the markup the style will be applied, according to tags and attributes in the markup, and they are generally applied on

- all elements of a specific type (for example `a, p, h1`)

- elements specified by an attribute (id for a unique element, class for multiple elements)
- elements depending on their positions in the Document Object Model (DOM).

Selectors are the key to flexibility, as they can be rearranged in multiple ways and combinations to have different results.

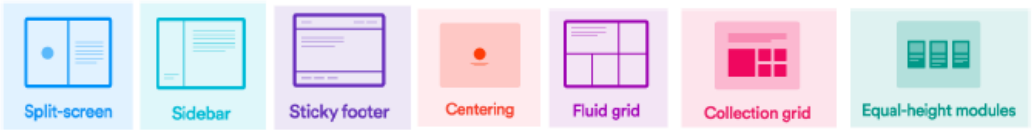
Close relatives to classes are pseudo-classes, used to permit formatting based on information that is not contained in the DOM (for example, `:active, :hover, :visited, :link`), while a pseudo-element makes a selection that may consist of partial elements.

Every css element falls within a category:

- Block: content with customizable height and width, its contents are arranged for the entire webpage width.
- Inline: its content are small, without height or width, they are arranged in the needed space and an inline element cannot be located directly.
- Inline-Block: inline contents but with height and width, also they takes only the needed space so it is possible to align "block" elements such as images.

Regarding block and inline elements, div and span tags are respectively used to represent a generic block element and a generic inline element.

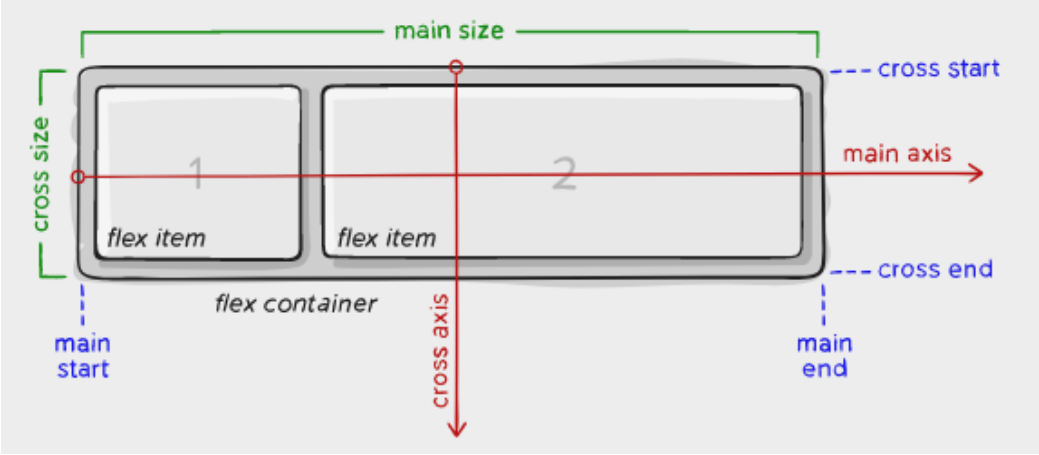
FlexBox is another CSS fundamental, it is heavily used in web development to better organize the disposition of elements in the webpage.



Several possible layouts made using flexbox

The main idea behind the flex layout is to give the container the ability to alter its items' width/height (and order) to best fill the available space. A flex container expands items to fill available free space, shrinks them to prevent overflow or just organize them filling all the available space in a designed area (flex-grow, flex-shrink and flax-wrap properties). Most importantly, it is direction-agnostic (properties flex-direction with possible values column or row) as opposed to the regular layouts (block which is vertically-based and inline which is horizontally-based).

The parent element, also called flex-container, contains the flex-items.



Flex container and flex items, css-tricks.com

3.4 JAVASCRIPT

JavaScript [10] is a scripting or programming language, created in 1995 by Brendan Eich (from NetScape), that allows to create dynamically updating content, control multimedia, animate images and pretty much everything else. It is the third layer of the frontend architecture, below HTML and CSS which were just discussed.

Besides creating content dinamically, JS is based on event-driven paradigm, it can be in fact used to run code in response to certain events on the web-page, such as click or submit: this is usually used to make a static website communicate with the APIs (Application Programming Interfaces), ready-made sets of code building-blocks that allows the programmer to implement other functionalities in its web application.

JavaScript is indeed often referred to as "MiddleWare", as it is the key to make frontend (client-side) and backend (server-side) communicate: Server-side code is run on the server, its results are downloaded and displayed in the browser in response to a client-side request. Examples of popular server-side web languages include PHP, Python, Ruby, ASP.NET and JavaScript itself can also be used as a server-side language, for example in the popular Node.js environment which we will discuss forward, as it is used as runtime programming language in AWS Lambda.

3.4.1 JS and requests

A static website, through JavaScript, can become dynamic (it can update the display of a web page/app to show different things in different circumstances, generating new content as required).

Amongst the possibilities to make calls to the server there are XML-HTTPRequest, Fetch, Axios and JQuery.

Fetch allows the client to make an HTTP request with a straightforward interface by using promises. It is not supported by old browsers, but very well supported among the modern ones.

Below follows an example call made using a fetch.

```
fetch(ENDPOINT).then(onResponse).then(onJson);
```

3.4.2 Interpreted and Compiled Code

JavaScript is an interpreted programming language, as its compilation is handled at run-time. The difference between interpreted and compiled languages stands in how the code is treated: while in interpreted ones the code is run from top to bottom and the result is returned directly, in those compiled the code is compiled (transformed) before being executed by the machine.

3.5 Amazon Web Services

Amazon Web Services (AWS) [11] is a subsidiary of Amazon providing on-demand cloud computing platforms and APIs to individuals, companies, and governments, on a metered pay-as-you-go basis. These cloud computing web services provide a variety of basic abstract technical infrastructures and distributed computing building blocks and tools.

As of 2021, AWS comprises over 200 products and services including computing, storage, networking, database, analytics, application services, deployment, management, machine learning, mobile, developer tools, and tools for the Internet of Things. The most popular include Amazon Elastic Compute Cloud (EC2), Amazon Simple Storage Service (Amazon S3), Amazon Connect and AWS Lambda.

3.6 AWS S3

As previously said, the files regarding the client-side application (they define the website as static, while the server-side makes it dynamic) are hosted in a cloud storage service, in our case AWS S3: it is used to store application data that are not suitable for a database.

Amazon Simple Storage Service (S3) is an object storage service that aims to provide, through a web service interface, industry-leading scalability, data availability, security, and performance. This means customers of all sizes and industries can use it to store and protect any amount of data for a range of use cases, such as data lakes, websites, mobile applications, backup and restore, archive, enterprise applications, IoT devices, and big data analytics [12].

It provides easy-to-use management features to make easier to meet specific requirements.

The basic storage units of Amazon S3 are objects organized into buckets, where each object is identified by a unique user-assigned key.

Requests are authorized using an access control list associated with each object bucket and support versioning which is disabled by default; Buckets also have the option to enable Bucket policies to allow or deny access, read/write or other kind of permissions so that users cannot access/modify/read files stored in the buckets affected by this policy.

```
{
    "Version": "2012-10-17",
    "Id": "Policy_Number",
    "Statement": [
        {
            "Sid": "Sid_Number",
            "Effect": "Allow",
            "Principal": "*",
            "Action": "*",
            "Resource": "arn"
        }
    ]
}
```

Bucket policy blueprint

Bucket names and keys are chosen so that objects are addressable using HTTP URLs, of the kind: *https://s3.region.amazonaws.com/bucket/key.*

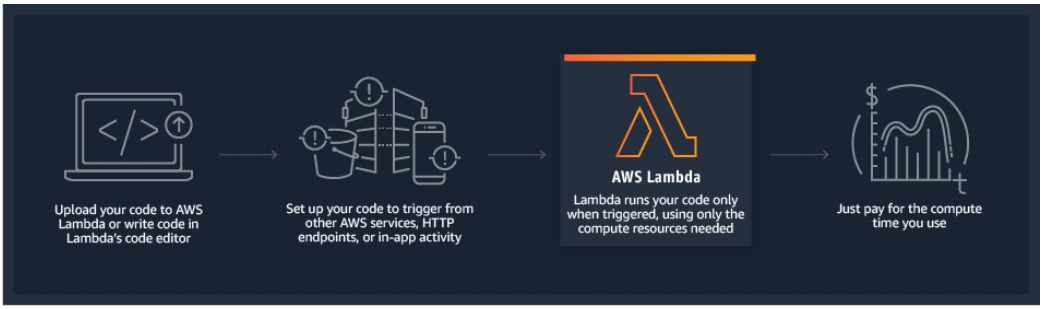
Amazon S3 can be used to replace significant existing (static) web-hosting infrastructure with HTTP client accessible objects. The Amazon AWS authentication mechanism allows the bucket owner to create an authenticated URL which is valid for a specified amount of time.

3.7 AWS Lambda

Such as the FaaS on which AWS serverless architecture relies, AWS Lambda is an event-driven computing service that allows developers to program functions without having to provide storage or compute resources to support them: it abstracts server management away from the IT professionals, as servers are handled by Amazon itself.

AWS Lambda is made up of functions working as server-side handler for web and API requests, designed to carry out specific tasks, each of them running in their isolated computing environment with their own resources.

The functions code is uploaded through a zip file or a container image that contains both the code and all the package configuration files needed, such as the package data, modules, libraries and general settings (for example the credentials needed to the connect with the Database).



How does Lambda work? [13]

The first time a function is invoked through some triggering (functions are ready to run as soon as they are triggered), AWS Lambda creates an instance of the function and runs its handler method to process the event. When the function returns a response, it stays active and waits to process additional events.

A serverless solution allows the website to respond faster to change: a scalable website can in fact resize itself to scale from zero to peak demands, adapting to the variable concurrency without any problem (of the kind a server-hosted website could loom over) [14]. This is possible because Lambda functions are stateless, namely there is no data stored about the underlying infrastructure (each instance execution is run in a new environment so access to the execution context of previous and subsequent runs is not possible), AWS can rapidly launch as many copies of the function as needed to scale to the rate of incoming events [15].

If the function is invoked again while the first event is being processed, Lambda initializes another instance, and the function processes the two events concurrently. As more events come in, Lambda routes them to available instances and creates new instances as needed. When the number of requests decreases, Lambda stops unused instances to free up scaling capac-

ity for other functions.

A function's concurrency is the number of instances that serves requests at a given time [16].

As previously said, each lambda function performs a different task, and in order to do so AWS Lambda relies on Run-time languages: among the ones supported by Lambda there are NodeJS, Python, Java, Go, Ruby, .NET.

At last, talking about bills, using Lambda as FaaS makes the web application sustaining way cheaper than the dedicated server alternative: Amazon charges users based on the number of requests served and the compute time needed to run the code, if a function is never called there is no charge.

3.8 Node.js

Node.js [17], built by Ryan Dahl in 2009, is an open-source, server-side cross-platform , it deals with server-side business logic, database interactions (accessing, storing, fetching), performance optimization, and security. In CodeMeet it is also used as AWS Lambda's runtime language. NodeJS applications are written in JS and can be run within the NodeJS runtime on

the various OS. It also provides a rich library of various JS modules which semplifies the development of web applications.

Its most remarkable features are:

- Asynchronous and Event Driven: All the APIs of Node.js library are asynchronous, non-blocking. It essentially means a Node.js based server never waits for an API to return data. The server moves to the next API after calling it and a notification mechanism of Events of Node.js helps the server to get a response from the previous API call.

- Fast and No buffering: NodeJS applications never buffer any data. These apps simply output the data in chunks.

- Single Threaded but Highly Scalable: Node.js uses a single threaded model with event looping. Event mechanism helps the server to respond in a non-blocking way and makes the server highly scalable as opposed to traditional. The single threaded program and the same program can provide service to a bigger number of requests than traditional servers like Apache HTTP Server.

3.8.1 Callbacks

Node heavily relies on callbacks, so all the Node APIs are written in such a way to support them. A callback function is called at the completion of a given task: a function made to complete a task will return the control to the execution environment once it starts to work on the task, only to recall control when the task is completed, passing the task output as a parameter when the function says it is time to regain control.

There is so no blocking or wait while completing tasks. This makes NodeJS highly scalable, as it can process a high number of requests without waiting for any function to return results or instancing multiple threads.

3.8.2 Pools

Node.js runs JS code in the Event Loop (initialization and callbacks, main thread) and offers a Worker Pool (threads pool) to handle expensive tasks. NodeJS scales well as it uses a small number of threads to handle many clients, consequently having more system memory and time free for clients rather than having it "wasted" for threads (as more thread implies more memory, context-switching). As a direct consequence of having few threads, they have to be used wisely.

Why avoid blocking the Event Loop and the Worker Pool is important

There are 2 kinds of thread: Event loop (the main loop, main thread, event thread) and Worker pool (also known as a thread pool). If a thread is taking too much time to execute a callback (Event Loop) or a task (Worker), it is blocked. While a thread is blocked, it cannot handle requests from any other clients: this provides two reasons to never block neither the event loop nor the worker pool

- Performance: if heavyweight tasks are regularly performed, the server throughput (requests satisfied for second) will suffer.
- Security: blocking some threads may be used to perform a DDOS (Distribuited Denial of Service) on the server.

NodeJS has an Event Loop for orchestration (callback) and a Worker Pool (tasks) for expensive works [18]. Both Worker Pool and Event Loop maintain "queues" (while worker pool really uses queue, event loop uses collections of file descriptors) for pending events and pending tasks, respectively.

Event Loop

Nodejs applications are initialized, as they require modules and to register callbacks for events, and then enter the Event Loop responding to incoming client requests by executing the appropriate callback. This callback executes

synchronously and may register asynchronous requests (also executed in the Event Loop) to continue processing after it completes.

Worker Pool

The Worker Pool is used to handle expensive tasks such as I/O for which the OS does not provide a non-blocking version, especially if CPU-intensive.

To handle complex calculations without blocking the Event Loop there are 2 options to follow: Partitioning and Offloading. While the first is about dividing the calculations in small parts in order to make each run on the Event Loop, that regularly yields (the yield keyword is used to pause or end an execution to give back control, as in JS is easy to store the state of an ongoing task) other pending events, Offloading is more recommendable as complexity increases because Partitioning uses just the Event Loop, that should orchestrate and not fulfill the requests itself. It is good as long as requests are not that complex, but when they are it is better to use multiple cores (Worker Pool) and that is what Offloading does. In conclusion, while Partitioning is good enough for simple tasks, Offloading is better when the tasks are not so simple, as it uses the multiple cores. The downside of using the worker pool is that the communication is more heavy-weighted: if the tasks are too complex maybe NodeJS is not the best solution as it better

supports I/O-bound work.

When working with Worker Pool the best choice is to optimize the task throughput of the pool: to do this, the variation in task times is minimized by using task partitioning, namely divide splitting a complex task into several simpler parts.

If an application makes blocking callbacks or tasks, this can lead to degraded throughput (clients/second) at best, and complete denial of service at worst. To write a high-throughput, more DoS-proof web server, neither Event Loop nor Worker Pool must block.

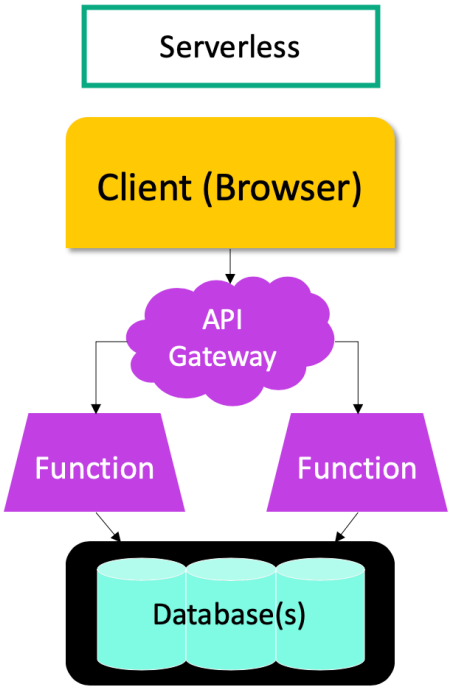
3.9 AWS API Gateway

AWS API Gateway [19] is needed to make Lambda functions communicate with the outer world, such as - in CodeMeet case - users that try to reach our functions through the browser requests.

It is a fully managed service that makes it easy for developers to create, publish, maintain, monitor, and secure APIs at any scale. APIs act as the

"front door" for applications to access data, business logic, or functionality from backend services.

An API [20] can be seen as a mediator between the users or clients and the resources or web services they want to get. It is also a way for an organization to share resources and information while maintaining security, control, and authentication—determining who gets access to what. It is a set of definitions and protocols for building and integrating application software. It is sometimes referred to as a contract between an information provider and an information user—establishing the content required from the consumer (the call) and the content required by the producer (the response).



https://github.com/ryanrashid/fassml

APIs usually fall within 2 families:

- HTTP APIs: An HTTP (HyperText Transfer Protocol) API uses HTTP as communication protocol between two systems.
- RESTful APIs: A REST (representational state transfer) API follows REST ruleset that defines best practices to share data between 2 systems. REST Applications uses HTTP Methods (such as GET, POST)

Not all the HTTP APIs are REST APIs, as the API needs to meet several requirements to be defined REST, among whom there are stateless and cacheable. CodeMeet makes use of REST APIs.

API Gateway handles all the tasks involved in accepting and processing up to hundreds of thousands of concurrent API calls, including traffic management, CORS support, authorization and access control, throttling, monitoring and API version management.

3.10 AWS RDS

Amazon Relational Database Service (RDS) is a distributed relational database service by Amazon Web Services (AWS). It is a cloud service designed to simplify the setup, operation, and scaling of a relational database for the use in applications. Administration processes like patching the database software, backing up databases and enabling point-in-time recovery are managed automatically.

While RDS was released supporting only MySQL databases, year after year Oracle Database, Microsoft SQL Server, PostgreSQL, MariaDB and many others were added as supported.

The most remarkable features [21] are:

- Backup and Recovery (Snapshots): RDS creates and saves automated backups of RDS DB instances. The first snapshot of a DB instance contains the data for the full DB instance and subsequent snapshots are incremental, maximum retention period is 35 days. Elevated latencies can be experienced for a few minutes during backups.

- Pricing: RDS instances are priced very similarly to Amazon Elastic Compute Cloud (EC2). RDS is charged per hour, only for what is used, and comes in two packages: On-Demand DB Instances and Reserved DB Instances. Besides the hourly cost of running the RDS instance, users are charged for the amount of storage provisioned, data transfers and input and output operations performed.

- High Availability and Read Replicas: RDS will scale out beyond the capacity constraints of a single database instance for heavy workloads.

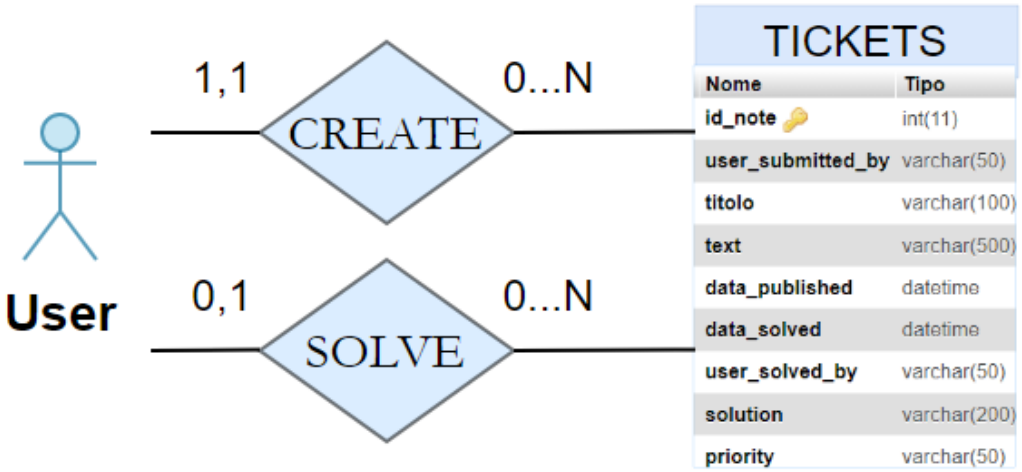
In CodeMeet the used DB Engine is MySQL, which is the most famous open source relational database. RDS makes it easy to set it up and also scale its deployment in the cloud.

Chapter 4

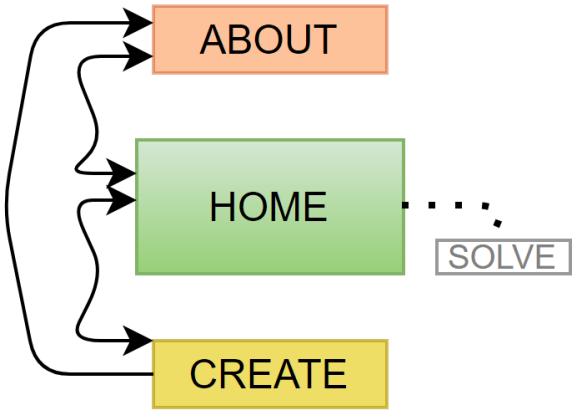
Architecture

4.1 Entity–Relationship Model

CodeMeet is a ticketing platform where users can create and solve tickets made up of several attributes used to better describe the problem and the possible solution, such as who posted the ticket, the data-time, title, text priority, and at last the eventual user that solved the problem with the right solution (and the data-time).



4.2 Pages Interaction Diagram

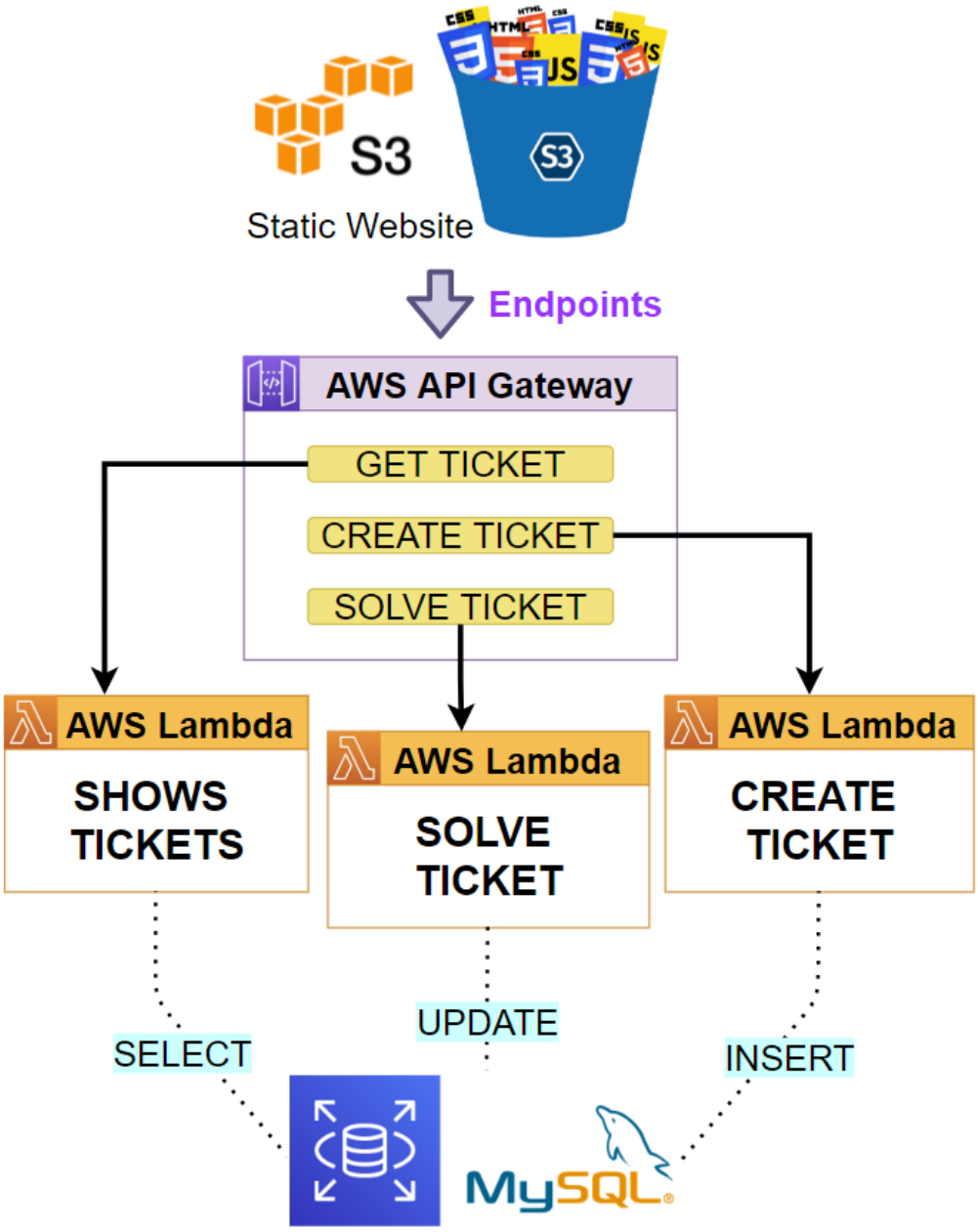


A new user visiting the website is welcomed by the About page, where the site is briefly explained: as the page states, CodeMeet can be either used to create and solve tickets. From About the only possible route is Home, where the user can see the open and closed tickets. Lastly, from Home both Create and About are accessible, and the same goes for Create.

4.3 Amazon Web Services interactions Diagram

The services discussed in Chapter 3, namely S3, Lambda, API Gateway and RDS MySQL, interacts among themselves. The Static Website is linked to the server-side through JS and Endpoints (API Gateway) where each of them (the endpoints) communicate with a Lambda function that works in a specific way with the Database. Specifically, there are three lambda functions (and

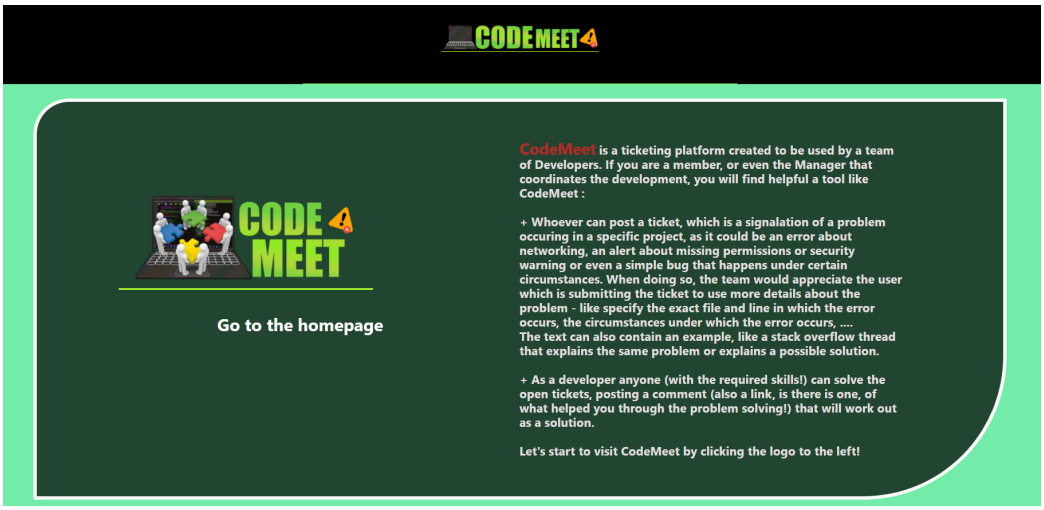
therefore three API Gateway Endpoints): the first is just a select all the tickets from the database for the home, the second is relative to the create section, where the data filled are inserted in the database and at last, the third is an update that is triggered when a ticket is solved.



All the three Lambda functions are connected via API Gateway as three resources with a GET Method.

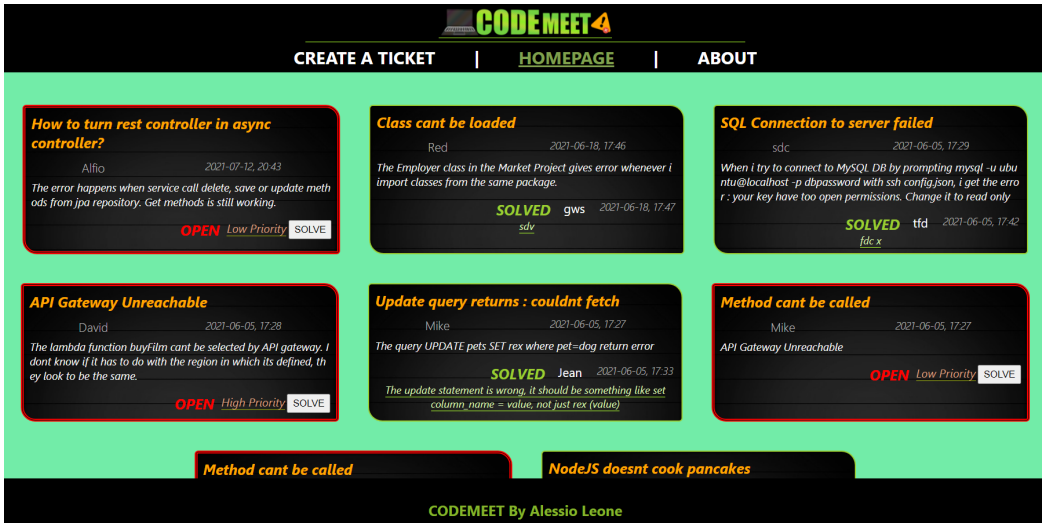
4.4 CodeMeet Structure

4.4.1 ABOUT



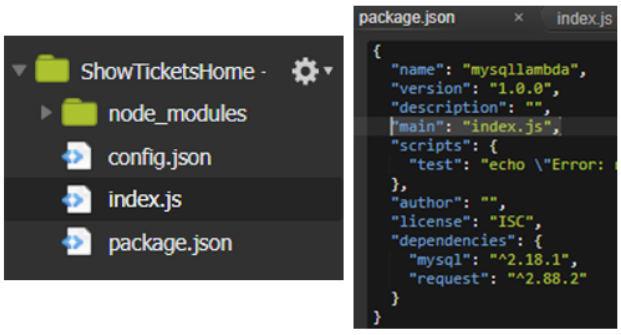
As said before, the first page a new user is greeted with is About, where the website and its functionalities are briefly explained. The CSS responsible for the webpage layout is made up of one flex column with inside two flex div. The only possible route from About is the Homepage.

4.4.2 HOMEPAGE



The home is filled with tickets displayed through the Lambda function "Show Tickets" — this is made through a JavaScript fetch to the API Gateway Endpoint, connected to the above mentioned function.

The lambda function relies on a .zip or image container which content is made up of node modules, index.js (the actual Node.js code that fulfill the work) and config and package .json files that have settings related data, such as the ones needed to connect to the Database.



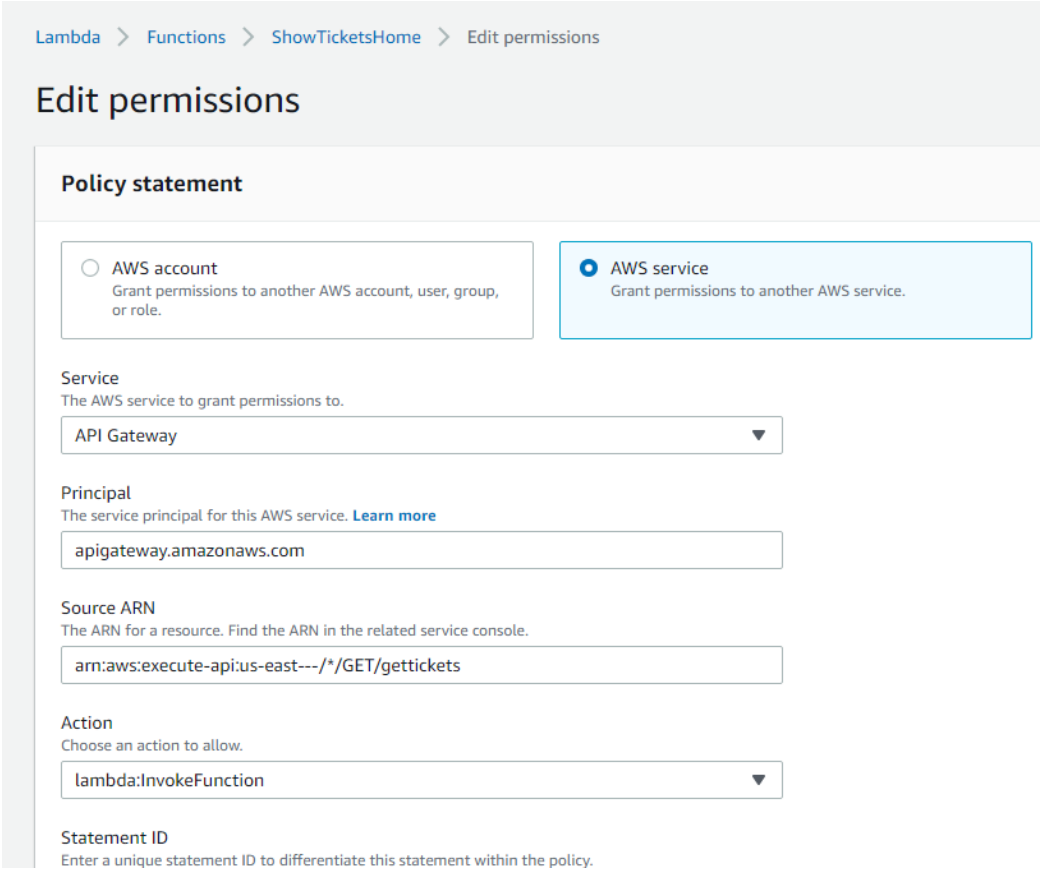
The actual Node.js code for the Show Tickets Lambda function is Index.js:

```
var mysql  = require('mysql');
var config = require('./config.json');

var pool = mysql.createPool(    );

exports.handler =  (event, context, callback) => {
  //prevent timeout from waiting event loop
  context.callbackWaitsForEmptyEventLoop = false;
      pool.getConnection(function(err,connection){
        var sql="SELECT * FROM tickets ORDER BY id_note DESC";
        connection.query(sql, function (error, result, fields) {
          connection.release();
          if (error) {
            callback(error);
          }else {
            callback(null,result);
          }
        }); //ends connection.query
      }); //ends pool.getConnection
}
```

To connect with the database we use pools, as with them all the queries are run in parallel; when pools are used there is no need to connect with the DB for each request, as the pool can be queried directly: MySQL module will search for the next free connection to execute the query.

A lambda function, in order to be linked through AWS API Gateway and thus be invoked (following events that trigger it), needs to grant permissions to services. This happens by configuring policy permissions, with which lambda grants permission to a service (which ARN, Amazon Resource Name, is required) regarding a specific action (there is an enormous amount of them, with over 200 services). In our case lambda gives permission to invoke the

function to AWS API Gateway:



Once the results are received by the client, JS handles the JSON by adding the most suited elements to the DOM; then we rely on the attribute data solved to see if the ticket is already solved: if it is, data solved will be a datetime value different from the "null" equivalent to datetime, namely '0000-00-00 00:00:00' as seen in the picture below, where the datetime values are the output of the "NOW()" MySQL function.

data_published	data_solved
2021-06-05 14:46:33	0000-00-00 00:00:00
2021-06-05 15:31:28	2021-06-05 15:38:01
2021-06-05 15:31:32	0000-00-00 00:00:00
NULL	NULL

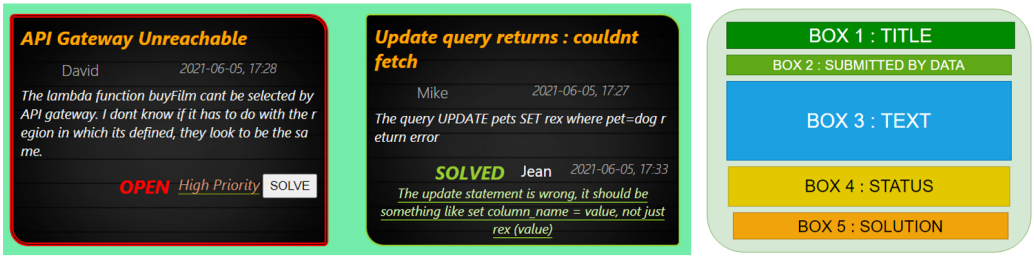
To work it around and have it more readable, we use the *substr* function to later join the chopped parts of the original datetime string.

```
post_data_solved.textContent = evento.data_solved;
var yymmdd = post_data_solved.textContent.substr(0, 10);
var res = post_data_solved.textContent.substr(11, 15);
var hhmm = res.substr(0,5);
post_data_solved.textContent=(yymmdd + ', '+hhmm);
```

Then JS will establish if the ticket is solved or open, and therefore will create one kind of element or one other.

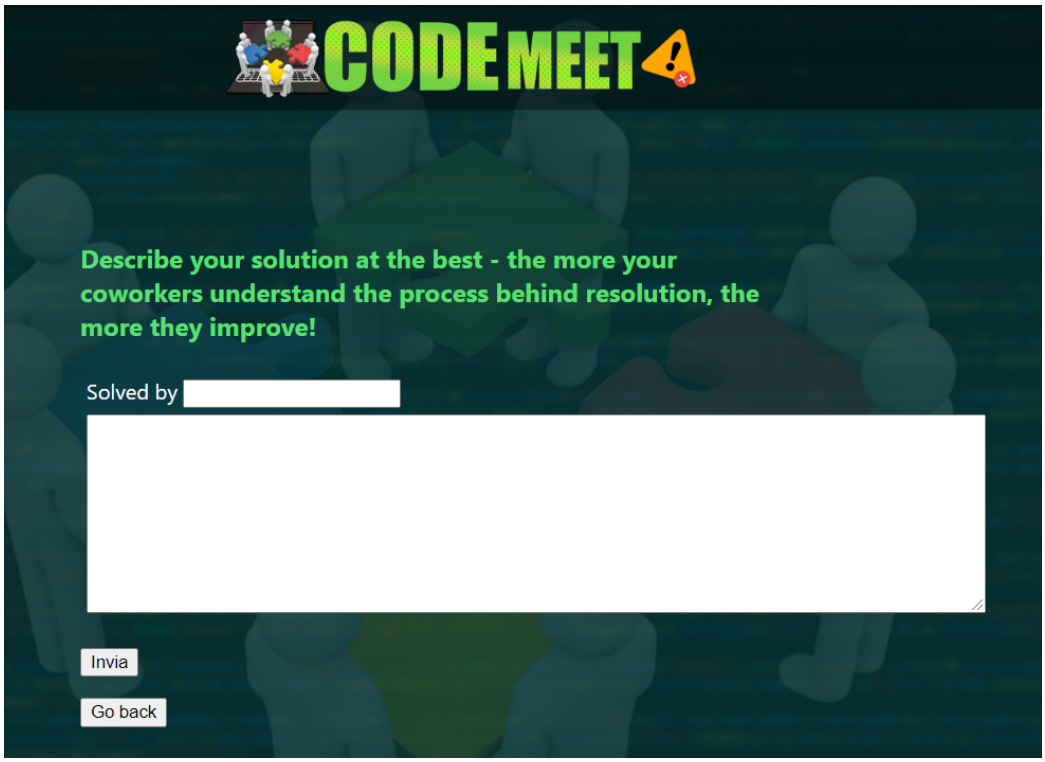
```
if(post_data_solved.textContent != '0000-00-00, 00:00'){

    post_opened.textContent='SOLVED';     // ...
}else{
    post_opened.textContent='OPEN';     // ...
```

Regarding the tickets layout on the homepage (CSS), the multiple tickets are disposed according to flex-wrap:1, which let the elements arrange in the available space without shrinking or spreading them. The single ticket is instead organized in several parallel (rows) boxes, each one of them is sub-split in smaller boxes (columns), all outlined with flex rules.



The remaining two sections, Create and Solve, are similar to Home (and Show Tickets Lambda function) under many aspects, such as the policy permission and the node.js pool connection. We will treat only the unique features.

4.4.3 SOLVE



From home open tickets it is possible to open, via modal view, the solve page, where through the Solve Ticket Lambda function is executed an update query on the Database. The Node.js code is the same of Show Tickets, but with a different query:

```
var sql="UPDATE tickets SET data_solved = NOW(), user_solved_by= '"+solved_by+"'
, solution = '"+solution+"' WHERE id_note='"+box_solution_var+"';";
```

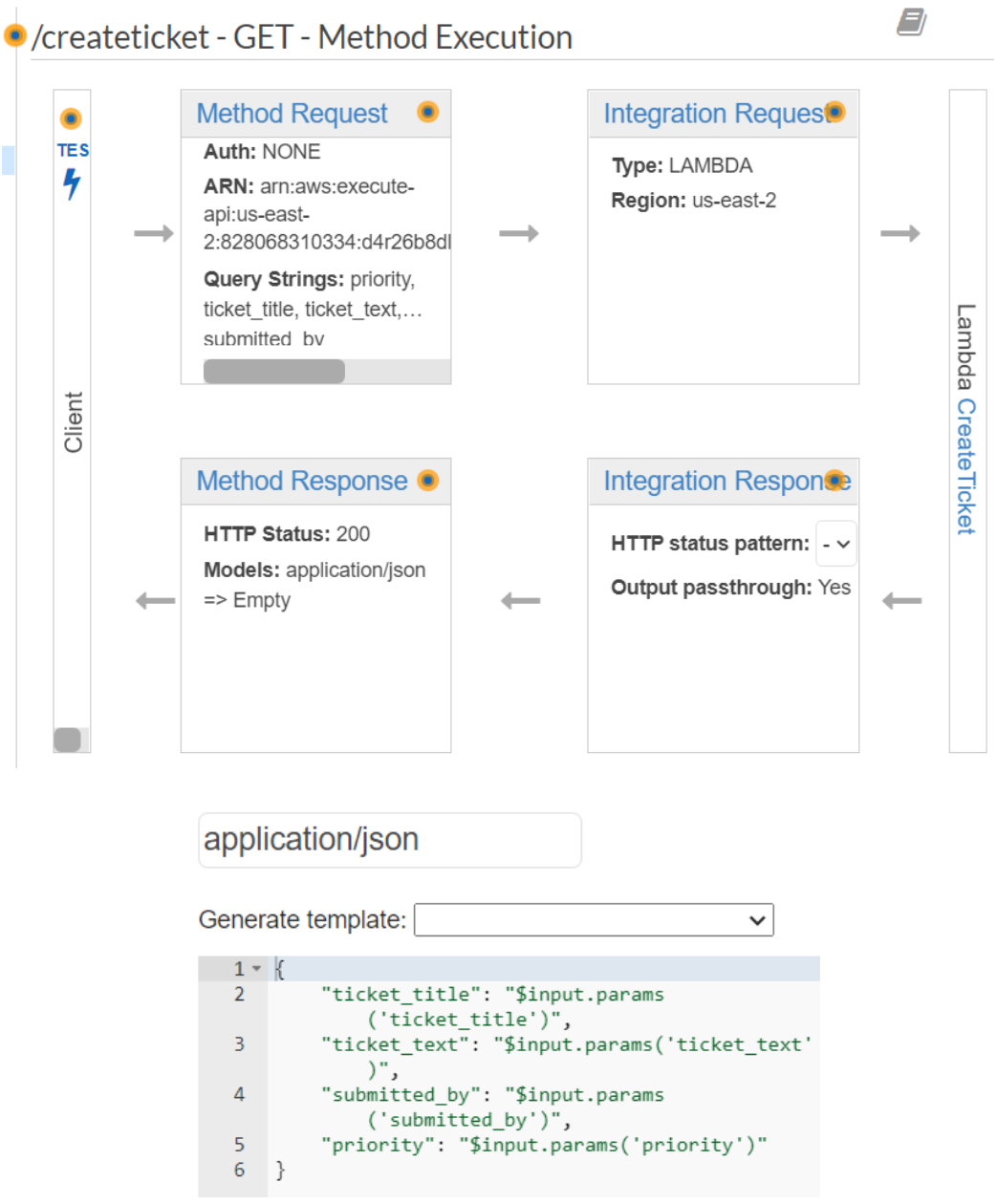
The field values are passed by JS through the form which requires a username (the username who is solving the problem) and a solution;

```
function post(event){
    event.preventDefault();
    const solved_by=form_modal.post_solved_by.value;
    const solution=form_modal.post_solution.value;

    fetch("https://-.execute-api.us-east-2.amazonaws.com/Dev/solvetickets?solved_by="
    +solved_by+"&solution="+solution+"&box_solution_var="+box_solution_var).then(onResponseP);
}
```

MySQL's *NOW()* function will handle the datetime in which the ticket solution was submitted, while the variable *box solution var* is required for the update query to understand which row it will affect, as tickets' primary key (unique identifier) is the ticket id which box solution var retrieves through *parentNode.querySelector*. The CSS responsible for the solve page is a simple flex row.

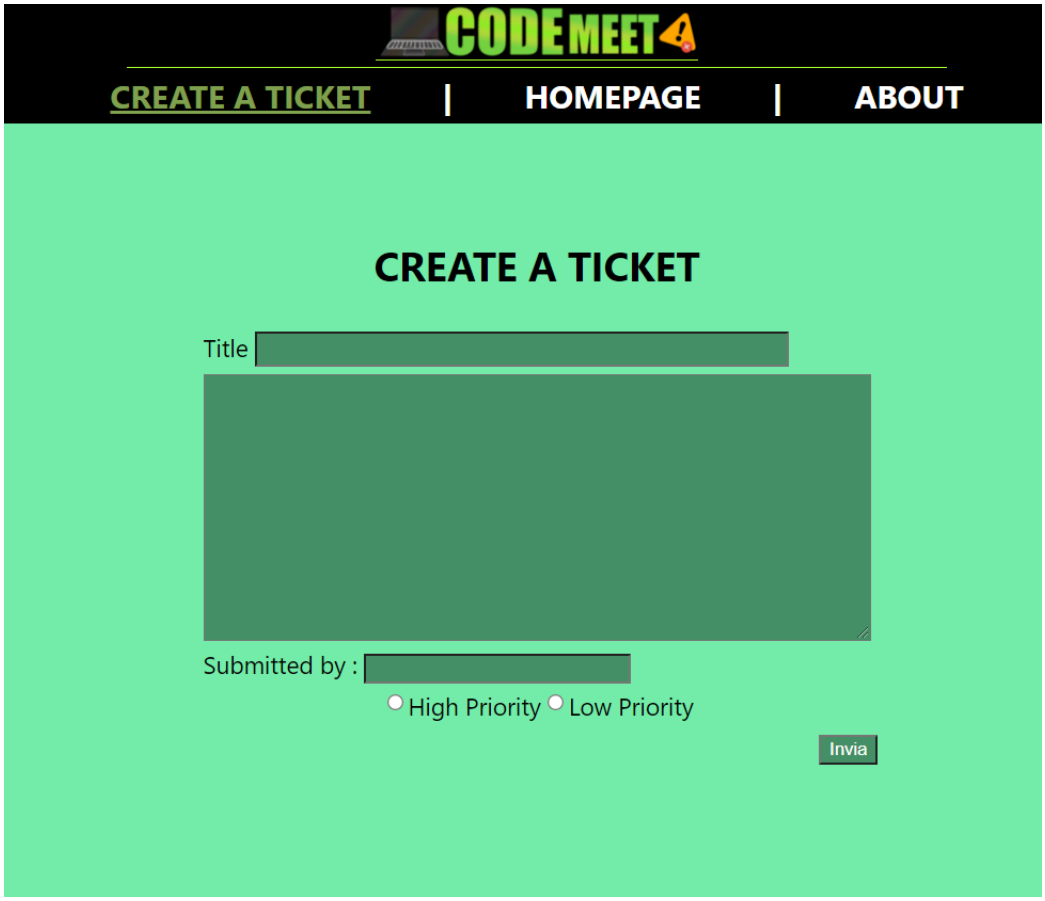
Once the solution is submitted, an alert will be prompted and the data will be inserted in the database. In the next refresh, that ticket will be solved, thus even its layout will be different from before.

Regarding API Gateway, the three elements (get tickets, solve and create) are very similar between them, but solve and create differ from get-tickets because they have to handle form's data.



Form's data are handled adding *query strings* to the method requests, which are then handled with the mapping templates that is used to convert form's data into json elements.

4.4.4 CREATE



Back to the home, the create page allows users to create and post tickets. This happens thanks to Create Ticket Lambda function, which has an insert query:

```
var sql="INSERT INTO tickets VALUES ('','"+submitted_by+"','"+title+"','"+text+"',NOW(),'','','','"+priority+"')";

connection.query(sql, function (error, result, fields) {
```

The required fields are a username, the ticket priority (implemented with a radio button), the title and the text where the problem is described at the best.

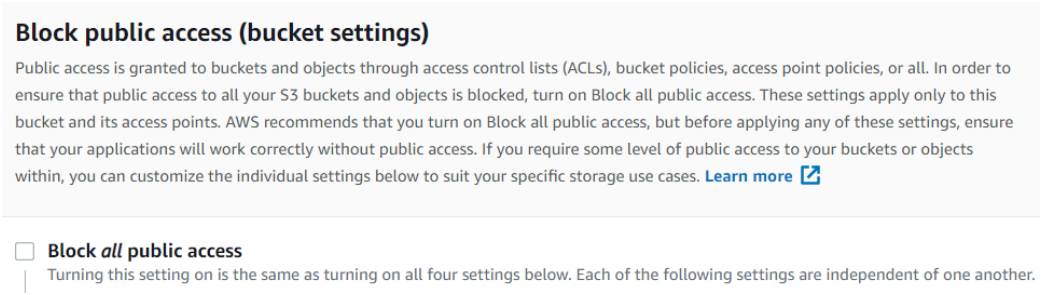
```
function create(event){
    event.preventDefault();
    const priority = document.createElement("em");
    if(form.priority.value=="HIGH"){ priority.textContent='High Priority';
    }else  { priority.textContent='Low Priority'; }

    fetch("https://-execute-api.us-east-2.amazonaws.com/Dev/createticket?ticket_title="
    +form.ticket_title.value+"&ticket_text="+form.ticket_text.value+"&submitted_by="+
    form.submitted_by.value+"&priority="+priority.textContent).then(onResponse);

    alert("Your ticket has been published!");
}
```

After the ticket is posted, an alert will appear confirming that the operation was successful: the ticket is now ready to be solved!

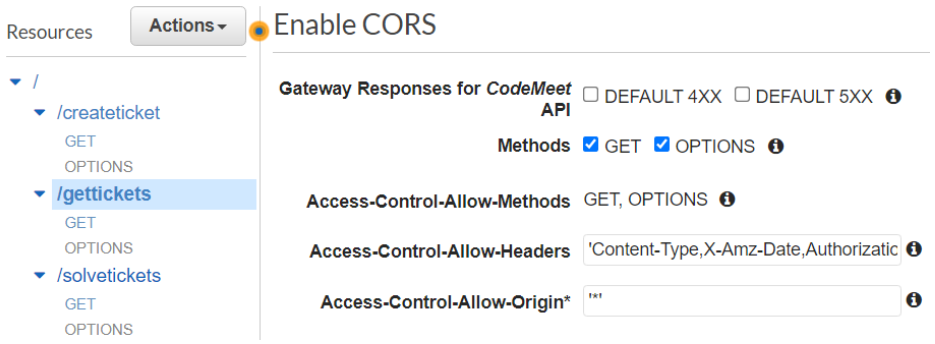
The websites' files are hosted, as said in chapter 3, in Amazon S3.

Name	Type	Last modified	Size	Storage class
create.css	css	June 23, 2021, 15:23:13 (UTC+02:00)	1.2 KB	Standard
create.html	html	June 23, 2021, 16:12:59 (UTC+02:00)	2.5 KB	Standard
create.js	js	June 23, 2021, 15:23:15 (UTC+02:00)	902.0 B	Standard
create2.png	png	June 23, 2021, 15:23:16 (UTC+02:00)	920.4 KB	Standard
home.css	css	June 23, 2021, 15:23:17 (UTC+02:00)	5.6 KB	Standard
home.html	html	June 23, 2021, 16:12:58 (UTC+02:00)	2.1 KB	Standard
home.js	js	June 23, 2021, 15:23:18 (UTC+02:00)	10.1 KB	Standard
index.html	html	June 23, 2021, 16:12:59 (UTC+02:00)	4.7 KB	Standard
logo0.png	png	June 23, 2021, 15:23:20 (UTC+02:00)	347.5 KB	Standard
logo0b.png	png	June 23, 2021, 15:23:22 (UTC+02:00)	562.5 KB	Standard
note.jpg	jpg	June 23, 2021, 15:23:22 (UTC+02:00)	54.8 KB	Standard

In order to be publicly accessible, S3 must be configured: the *block public access* option must be disabled, as it is, by default, enabled



Another step to make the site public is to Enable Cross-Origin Resource Sharing (CORS) in API Gateway, as this is by default disabled for security reasons.



The headers should also contain an additional field, which is

`'Content-Type,X-Amz-Date,X-Amz-Security-Token, ..., Origin'`

Chapter 5

Conclusions

Nowadays both companies and programmers benefit from cooperation: as we have seen in the Chapter 1, team-work is essential in programming related fields due to the high chance, for programmers, to loom over in bugs and others development related problems.

CodeMeet is a web platform developed as a ticketing system that tries to be a gathering place for groups of developers that needs to communicate and ask help, share solutions and recommend advice: this happens with the creation and the solving of tickets.

The web application is built using a serverless solution, as in a world where technology is exponentially growing and bringing new challenges for maintaining the dependencies, serverless is and will become more and more crucial in the coming decades. The majority believes that it is the future for applications since developers will focus more on the logic of customer interaction rather than on the infrastructure itself: serverless architecture,

contrary to the server alternative, allows developers to have a software which is easier to build and maintain. What is more, it is cheaper since serverless is usually defined as pay-as-you-go; lastly, it is natively highly-available and scalable, it saves time needed for maintenance and it disrupts distinction between developers and administrators.

As far as CodeMeet is concerned, the website presents several possible points that could be improved:

- currently CodeMeet does not prevent users to create duplicates of the same ticket, so an alert, that links to the already existing ticket and warns to check that out before proceeding with the creation of the ticket, could be implemented;
- the created tickets could have an additional field which describes the topic (such as SQL, Java, Python,...) and also every ticket could belong to a specific project, where only a restricted group of users can access (with a password known only by the team working on the project);
- the homepage content should be customizable: it should be possible to group and to order the tickets;
- following stack overflow example, the "solution space" could be re-designed in order to house more comments (the solution's follow-up) and not just the solution;

Bibliography

[1] "Serverless Architectures"

(https://aws.amazon.com/lambda/serverless-architectures-learn-more/?nc1=h_ls).

[2] "Introduction to AWS Lambda"

(https://www.youtube.com/c/amazonwebservices/).

[3] "What is serverless"

(https://www.redhat.com/en/topics/cloud-native-apps/what-is-serverless).

[4] "Serverless computing"

(https://www.cloudflare.com/it-it/learning/serverless/what-is-serverless/).

[5] "MantisBT" (https://www.mantisbt.org/).

[6] "TRAC" (https://trac.edgewall.org/).

[7] "Bitbucket vs. GitLab" (https://about.gitlab.com/devops-tools/bitbucket-vs-gitlab).

[8] "HTML" (https://html.spec.whatwg.org/toc-introduction).

[9] "CSS " (https://developer.mozilla.org/en-US/docs/Web/CSS).

[10] "What is JavaScript? MDN" (https://developer.mozilla.org/en-US/docs/Learn/JavaScript/First_steps/What_is_JavaScript).

[11] "Amazon Web Services" (https://aws.amazon.com/what-is-aws/).

[12] "Cloud Object Storage — Store Retrieve Data Anywhere — Amazon Simple Storage Service" (https://aws.amazon.com/s3).

[13] "AWS Lambda – Elaborazione serverless – Amazon Web Services" (https://aws.amazon.com/it/lambda).

[14] "Serverless Computing – Amazon Web Services" (https://aws.amazon.com/serverless).

[15] "AWS Lambda – Product Features" (https://aws.amazon.com/lambda/features).

[16] "AWS Lambda function scaling - AWS Lambda" (https://docs.aws.amazon.com/lambda/latest/dg/invocation-scaling.html).

[17] "Node.js" (https://nodejs.org/en/about/).

[18] "Don't Block the Event Loop (or the Worker Pool) — Node.js" (https://nodejs.org/en/docs/guides/dont-block-the-event-loop/).

[19] "Amazon API Gateway — API Management — Amazon Web Services" (https://aws.amazon.com/api-gateway).

[20] "What is a REST API" (https://www.redhat.com/en/topics/api/what-is-a-rest-api).

[21] "Amazon RDS for MySQL – Amazon Web Services (AWS)" (https://aws.amazon.com/rds/mysql).