

Maï LE MEUR

Guillaume LERAY GIRARDEAU

Village by CA Website

Technical Documentation

Table of Contents

- User Stories and Mockups
 - User Stories ----- 3
 - Mockups ----- 6
- System Architecture: High-level diagram ----- 7
- Entity Relationship Diagram ----- 8
- Database Schema ----- 9
- Sequence Diagrams
 - Creation of a Start-Up -----10
 - Creation of a User ----- 11
 - Update a Partner -----12
- API Specifications ----- 13
- SCM and QA Plans -----28
- Technical Justifications -----29
- Design Justifications -----33

USERS STORIES

Must Have:

Admin Stories

As an admin, I want to add, delete and modify start-ups information so that I can update the Village's information and make the start-up visible on the website.

As an admin, I want to create, delete and update new events on an agenda, so that I can keep start-ups and visitors informed of the upcoming events in the Village.

As an admin, I want to post, update and delete blog articles, so that I can make the website more attractive and increase the number of visits.

As an admin, I want to add, update and delete the partners lists and information, so that I can keep the website updated with new partnerships and make them visible on the website.

Start-up Stories

As a Start-up, I want to update my account information, so that I can keep my information updated for the visitors.

As a Start-up, I want to see the upcoming events in the Village agenda, so that I can plan to come to the events that interest me.

Visitor Stories

As a visitor, I want to take a rendez-vous to know more about how the Village and the Mayenne environment can help me in my current situation, so that I know who I can contact.

As a visitor, I want to know what are the upcoming events in the Village, so that I can take part in events that interest me.

As a visitor, I want to discover the start-ups present in the Mayenne department so that I can contact them for a collaboration if one of them is interesting to me.

As a visitor, I want to know what services the Village by CA is delivering so that I can know if the Village by CA can respond to my needs.

As a visitor, I want to have a partnership with the Village by Ca so that I can help them in their activities and collaborate with them.

As a visitor, I want to contact the Village headmaster, so that I can ask various questions about the events or the Village.

Should Have:

Admin story:

As an admin, I want to add, update and delete room for rent information, so that I can keep the visitors updated with available rooms.

Visitor Story:

As a visitor, I want to rent a place so that I can organize some events here.

Could Have:

Visitor and Start-up Story:

As a visitor I want to talk with a chatbot so that I have my answer quickly and don't bother the administrator with very simple questions.

Won't Have:

Start-up Story:

As a start-up, I want to have a list of the actors in the Mayenne ecosystem so that I know who can help my business.

MOCKUPS

Index page



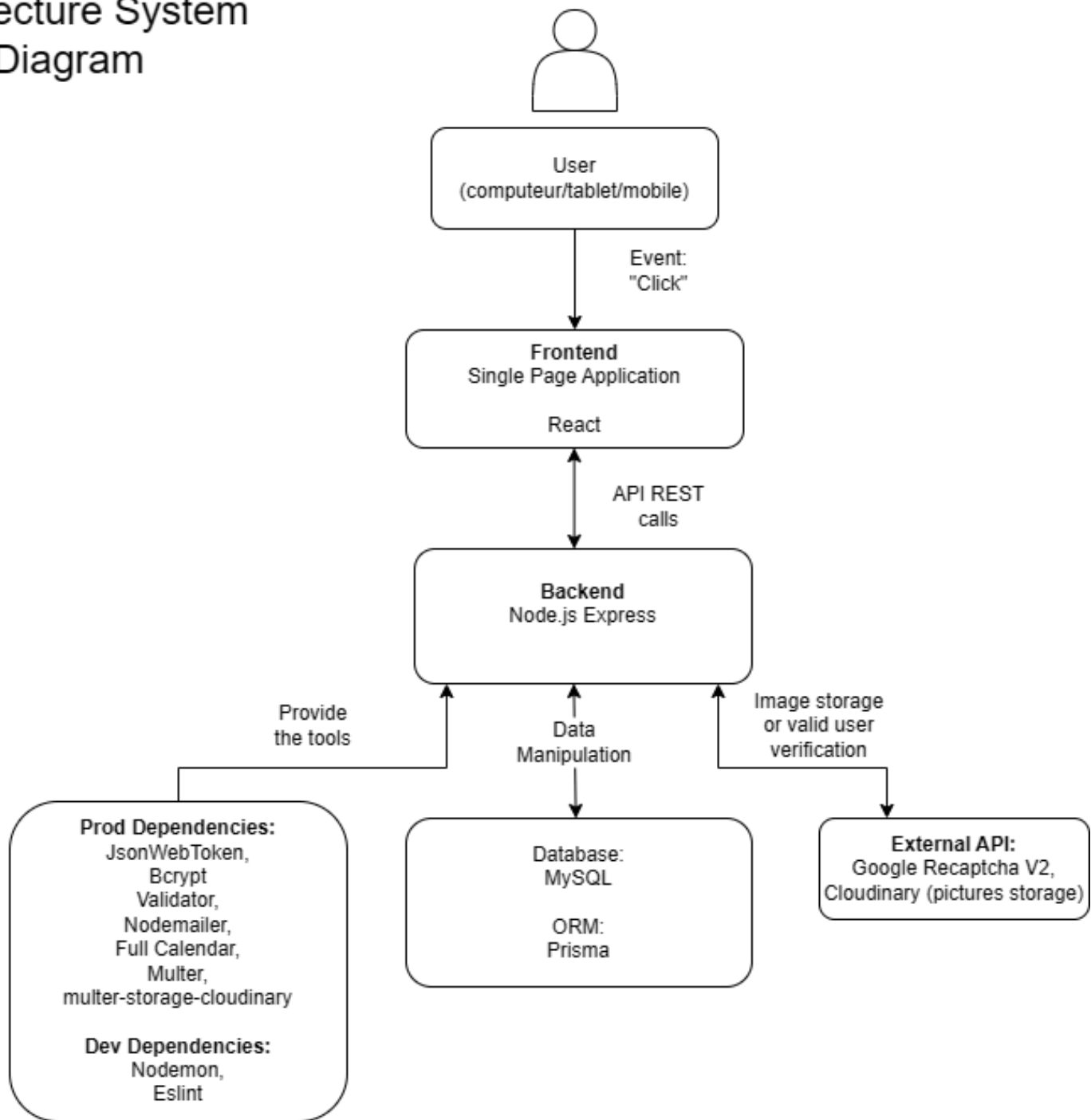
Start-Up Page



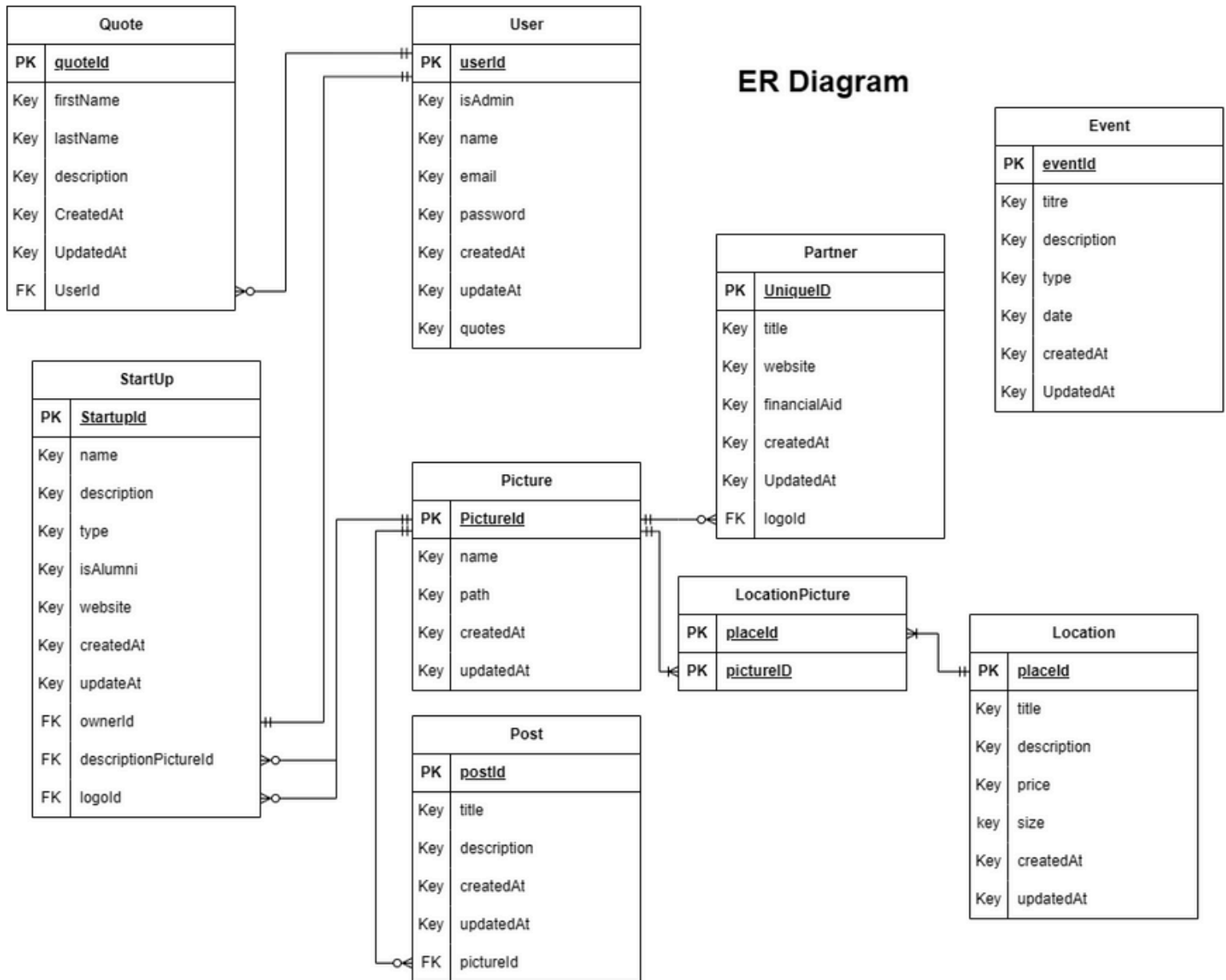
The mockup shows the 'Start-Ups' page of 'Le Village by CA'. The header is green with the logo and navigation links: Start-Ups, Partenaires, Agenda, Locaux, and Contact. The main heading is 'Start-Ups'. Below it, a paragraph describes the ecosystem's mission. A horizontal filter bar includes categories: Technologie, Environnement, Agriculture, Santé, and Ingénierie. The content area features six cards for different startups: MedyNova, NexaCloud, GreenLoop, TerraPulse, FarmLink, and SynapTech. Each card includes a small image, the startup name, a brief description, and a 'En savoir plus' button. The footer is dark blue with the logo, navigation links, social media icons (LinkedIn, Instagram), and copyright information: '© 2024 VILLAGE BY CA - TOUS DROITS RESERVÉS' and 'Mentions légales'.

This mockup shows the 'Start-Up Page' with additional content. The header is green with the logo and navigation links: Start-Ups, Partenaires, Agenda, Locaux, and Contact. The main heading is 'Acompagner les entreprises de demain'. Below it, a paragraph describes the ecosystem's mission. A horizontal filter bar includes categories: Technologie, Environnement, Agriculture, Santé, and Ingénierie. The content area features six cards for different startups: MedyNova, NexaCloud, GreenLoop, TerraPulse, FarmLink, and SynapTech. Each card includes a small image, the startup name, a brief description, and a 'En savoir plus' button. The footer is dark blue with the logo, navigation links, social media icons (LinkedIn, Instagram), and copyright information: '© 2024 VILLAGE BY CA - TOUS DROITS RESERVÉS' and 'Mentions légales'.

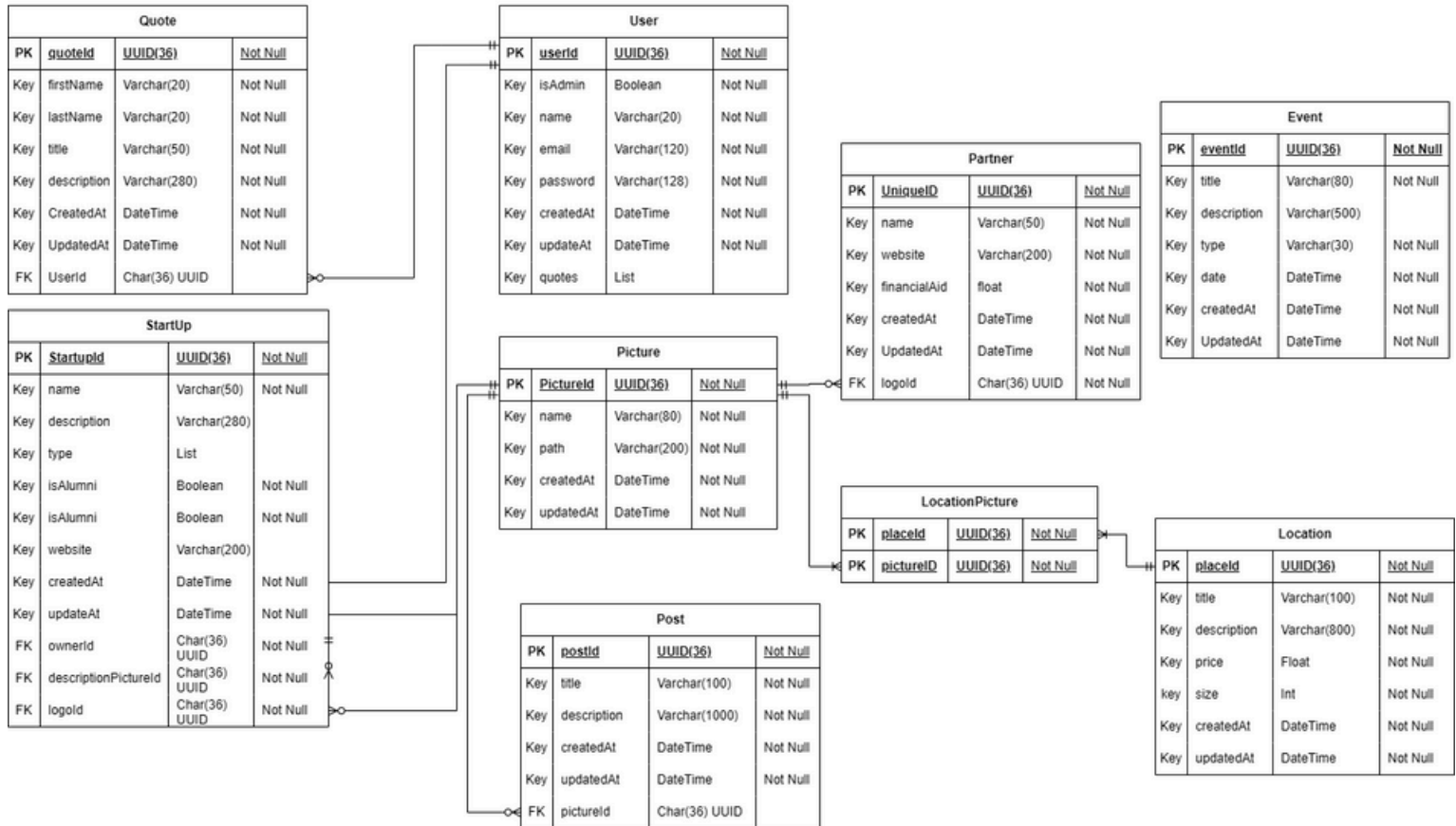
Architecture System Diagram



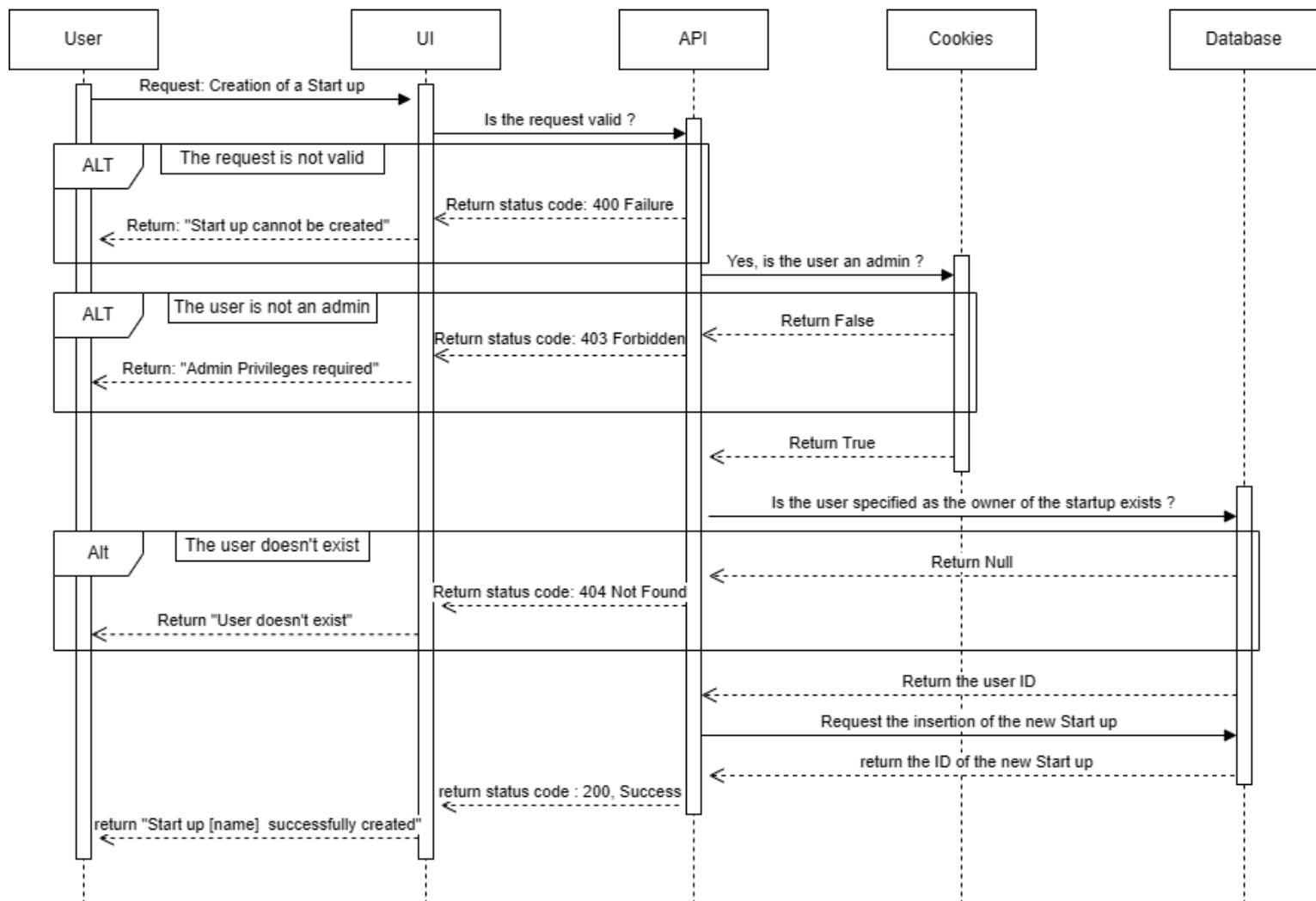
ER Diagram



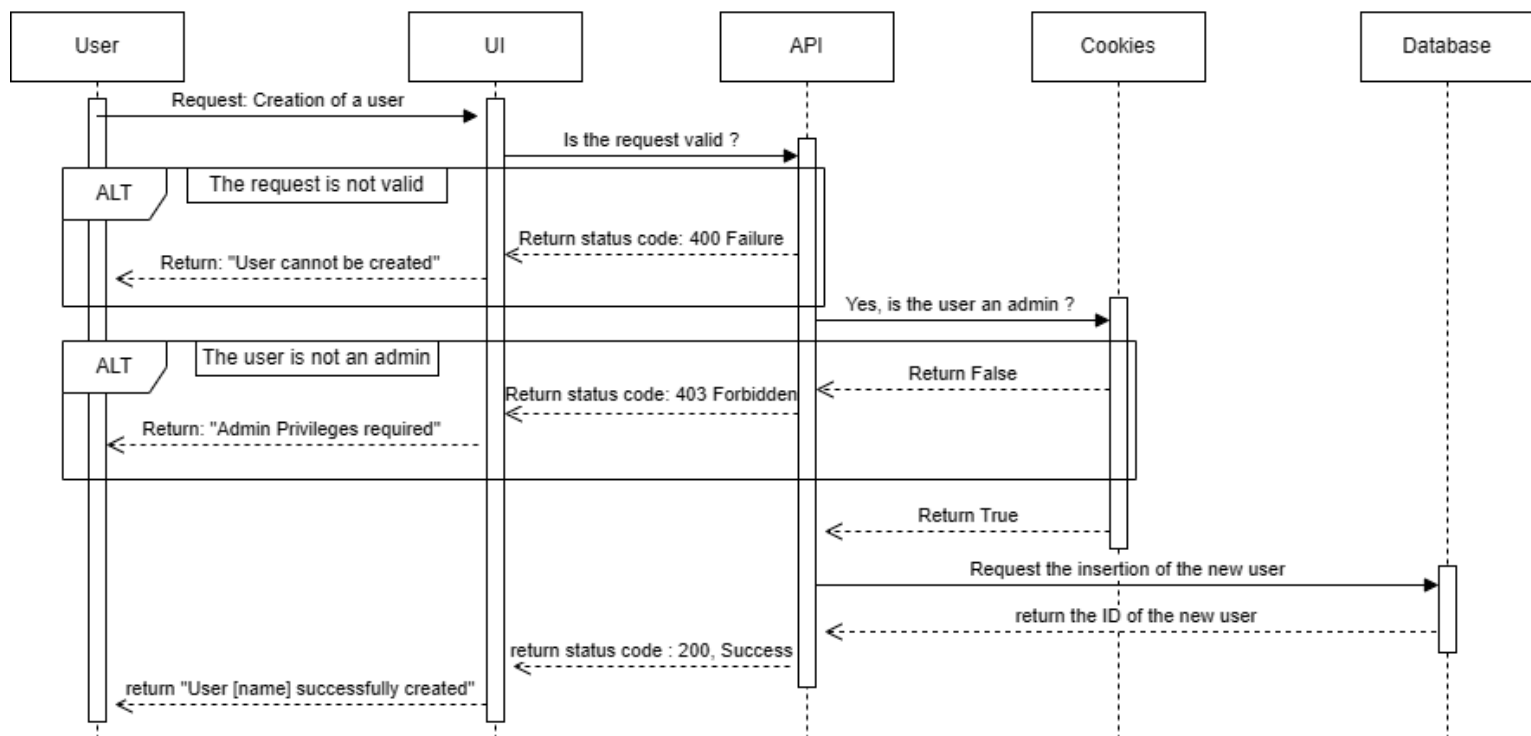
Database Schema



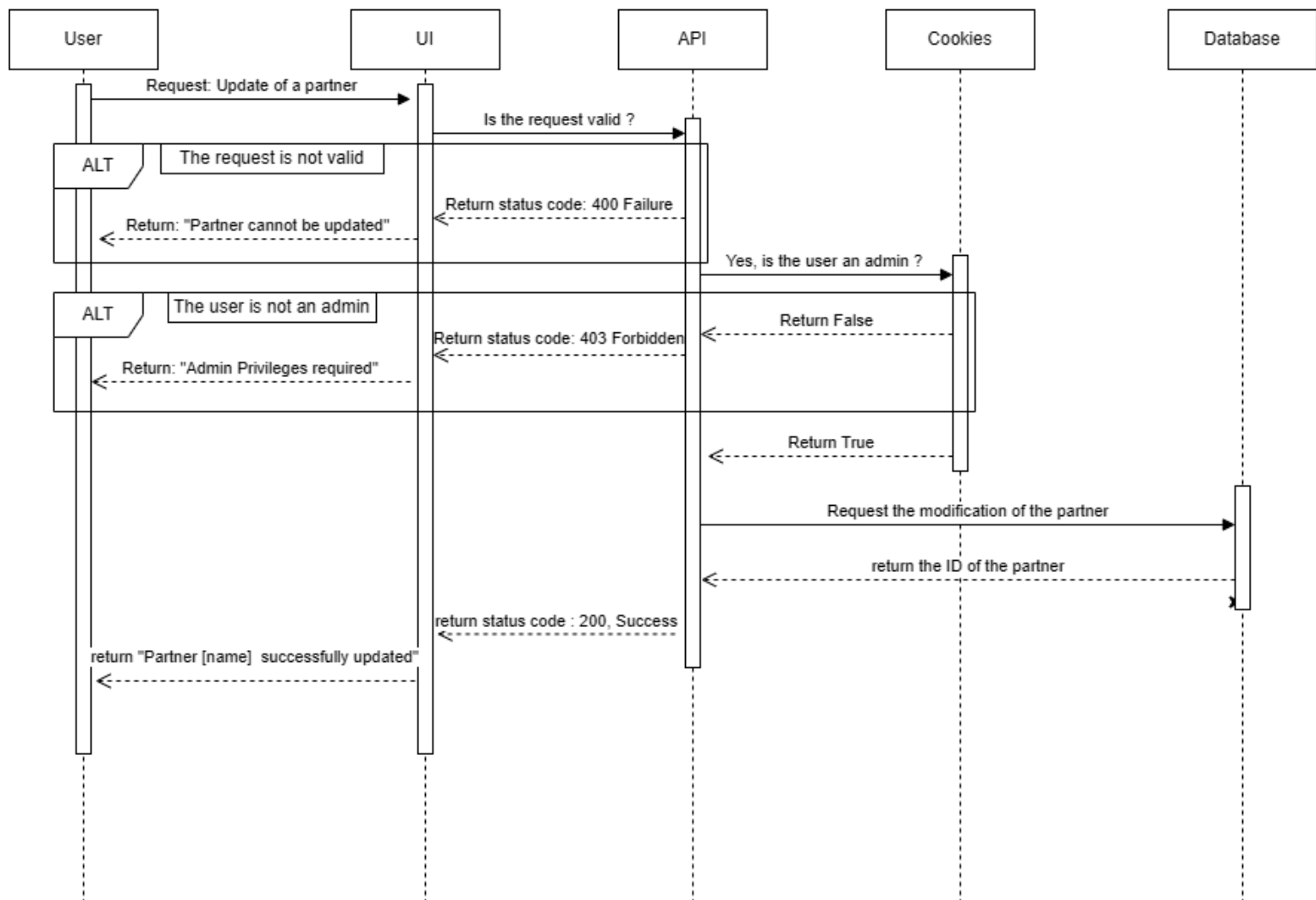
Creation of a StartUp



Creation of a User



Update a Partner



Village by CA's internal API :

For this project we use two external API, which is after the documentation of the internal API.

Below you will find all the documentation for the application's internal API.

You will find a table detailing each endpoint using the application's API:

- The route for each endpoint
- The method used for this route
- The format of the information sent to the API
- The format of the information returned by the API
- A short description of the route's purpose

Here all the API calls are used for three main purposes:

- 1.The visuals of a showcase site with data automatically updated for all visitors.
- 2.Profile pages for logged-in users.
- 3.Creation of an administrator area to manage all website informations and data easily.

There are short descriptions for each endpoint to provide details on how to access it, who will use it, how and why.

1- Endpoints for the showcase site :

Each endpoint in this category is usable by all visitors, whether logged in or not, and can only use GET methods, except for the contact form.

INDEX ENDPOINT :

The index endpoint has only one allowed method, which is a GET request, because it is only used to retrieve various pieces of information from the database in order to send them back to our front end for display.

Route	Method	Input Format	Output Format	Function
/index	GET		<pre>{ "startUps" : [{"logo" : "logo.path"}], "partners" : [{"logo" : "logo.path"}], "quotes" : [{"firstName" : "firstName", "lastName" : "lastName", "logo" : "logo.path", "description" : "description"}], "events" : [{"title" : "title", "date" : dateTime, "type" : "type", "description"}], "posts" : [{"title" : "title", "description" : "description"}]}</pre>	Display multiple pieces of information related to the association, to present what they're doing.

PARTNERS ENDPOINTS :

The partner endpoints only have one allowed method, which is a get, because they are only used to retrieve partner information with more or less detail in order to send it back to our Front for display.

/partners	GET		<pre>{ "name" : "name", "logo": "<logo.path>", "financial aid" : 0} </pre>	Display all the partners
/partners/:id	GET		<pre>{ "name" : "name", "website" : "website", "logo" : "<logo.path>" "financial aid" : 0} </pre>	Display the details of a partner

STARTUPS ENDPOINTS :

Startup endpoints only allow one method, which is a GET request, because they are only used to retrieve startup information with varying levels of detail in order to send it back to our front end for display.

Route	Method	Input Format	Output Format	Function
/startups	GET		<pre>{ "logo": "<logoPicture.path>", "name" : "name", "type" : ["type"]} </pre>	Display all the start-ups
/startups/:id	GET		<pre>{ "logo": "<logoPicture.path>", "name" : "name", "type" : ["type"], "isAlumni" : False, "website" : "website", "owner" : "owner.id", "description" : "description", "descriptionPicture" : "<descriptionPicture.path>" }</pre>	Get the details of a specific start-up

LOCATIONS ENDPOINTS :

The endpoint locations only have one allowed method, which is a get, because they are only used to retrieve location information with more or less detail in order to send it back to our Front for display.

/locations	GET		<pre>{ "title" : "title", "price" : "price", "size" : "size" }</pre>	Display all the locations
/locations/:id	GET		<pre>{ "title" : "title", "description" : "description", "price" : "price", "size" : 0, "picture" : ["<picture.path>"] }</pre>	Display the details of the location

CONTACT ENDPOINT :

The only endpoint open to all visitors with a post request, the API retrieves the data sent from the Front and processes it in order to send an email to the admin with the request information.

/contact	POST	<pre>{ "email": "email", "name": "name", "subject": "subject", "text": "text" }</pre>	<pre>{ "message" : "Demande de contact envoyée" }</pre>	Sending a contact request
----------	------	-----------------------------------------------------------------------------------------------	-----------------------------------------------------------	---------------------------

2- Endpoints for logged users :

Apart from the login route, here are all the endpoints available to all logged-in users so they can view and modify their information.

AUTH ENDPOINTS :

To access all the auth endpoints, the user has to login to their account via the route /auth/login. Once the API validates that the information are correct it creates a session token stored in a cookie on the client side, that permits the access of all the other auth routes. As you can see you must have a valid token in each route.

Route	Method	Input Format	Output Format	Function
/auth/login	POST	<code>{"email": "email", "password": "password"}</code>	<code>{"message": "Welcome back"}</code> + Creation of a token on a cookie.	Login to the application.
/auth/profile	GET	<code>{ Authorization : Bearer <Token>}</code>	<code>{"name": "name", "email": "email", "quotes": [{"firstName": "firstName", "lastName": "lastName", "title": "title"}] "startUp": "name", "logo": "<logoPicture.path>", "type": ["type"], "website": "website", "descriptionPicture": "<descriptionPicture.path>", "description": "description"}</code>	See current user details and information.

/auth/profile/update	PATCH	<pre>{ Authorization : Bearer <Token>} {"name" : "name", "email" : "email", "startUp" : "name", "logo" : "<logoPicture.path>", "type" : ["type"], "website" : "website", "descriptionPicture" : "<descriptionPicture.pat h>", "descriptpion" : "description"}</pre>	<pre>{"message " ; "Profile updated successfully !"} </pre>	<p>Updating the current user profile.</p> <p>The user can change almost all the information he sees in the GET request of auth/profile.</p> <p>He just cannot change his role and his password.</p>
/auth/profile/:id	GET	<pre>{Authorization : Bearer <Token>}</pre>	<pre>{"firstName" : "fistName", "lastName" : "lastName", "title" : "title", "description" : "description"}</pre>	Get a quote details that the current have created
/auth/profile/:id	PATCH	<pre>{Authorization : Bearer <Token>} {"firstName" : "fistName", "lastName" : "lastName", "title" : "title", "description" : "description"}</pre>	<pre>{"message" : "Quote successfully updated !"} </pre>	Update a quote that the current user have created
/auth/profile/:id	DELETE	<pre>{Authorization : Bearer <Token>}</pre>	<pre>{"message" : "Quote successfully deleted"} </pre>	Delete a quote the current user have created

3- Endpoint of the admin interface :

Here are the endpoints for the administrator interface. To access them, you must be a logged-in user with the admin role and a valid session token. The API will verify the token's validity with each request and identify whether the current user is an admin.

ADMIN ENDPOINT :

Here are the user management options. The administrator can list all existing users in the database, view their detailed information, and modify it if necessary or delete a user. Only the administrator can create new users.

Route	Method	Input Format	Output Format	Function
/admin/users	GET	{ Authorization : Bearer <Token>}	{ "name" : "name", "startUp" : "<logoPicture.path>" }	Display all the users
/admin/users	POST	{ Authorization : Bearer <Token> { "name" : "name" , "email" : "email@email.com", "password" : "password" }	{ "name" : "name", "email" : "email", "message" : " User.<name> created." }	Creation of a user.
/admin/users/:id	GET	{ Authorization : Bearer <Token>}	{ "name" : "name", "email" : "email", "isAdmin" : False, "startUp" : "<logoPicture.path>" }	Get the details of a User
/admin/users/:id	PATCH	{ Authorization : Bearer <Token> { "name" : "newName" , "isAdmin " : False, "email" : "newEmail@email.com", "password" : "newPassword" }	{ "message" : "User<name> updated" }	Modification of a user.
/admin/users/:id	DELETE	{ Authorization : Bearer <Token>}	{ "message" "User successfully deleted" }	Deleting a user.

Here are the routes for managing startups. The administrator can list all existing startups in the database, view their detailed information, and modify or delete them as needed. The administrator can only assign one user as the startup owner and cannot assign a user who is already associated with another startup. Only an administrator can create a new startup.

/admin/startups	GET	{Authorization : Bearer <Token>}	{ "logo": "<logoPicture.path>", "name" : "name", "type" : ["type"] }	Display all the start-ups.
/admin/statups	POST	{Authorization : Bearer <Token>}	{ "logo": "<logoPicture.path>", "name" : "name", "type" : ["type"], "isAlumni" : False, "website" : "website, "owner" : "owner.id" }	Creation of a new start_up Some fields like "logo", "type" or "website" are not mandatory at the creation.
/admin/statups/:id	GET	{Authorization : Bearer <Token>}	{ "logo": "<logoPicture.path>", "name" : "name", "type" : ["type"], "isAlumni" : False, "website" : "website, "owner" : "owner.id", "description" : "description", "descriptionPicture" : "<descriptionPicture.path>" }	Get the details of a start-up.
/admin/statups/:id	PATCH	{Authorization : Bearer <Token>}	{ "logo": "<logoPicture.path>", "name" : "name", "type" : ["type"], "isAlumni" : False, "website" : "website, "owner" : "owner.id", "description" : "description", "descriptionPicture" : "<descriptionPicture.path>" }	Updating a start-up
/admin/startups/:id	DELETE	{Authorization : Bearer <Token>}	{ "message" : "Start-up deleted successfully !" }	Deleting a start-up

Here are the routes for managing partners. The administrator can list all existing partners in the database, view their detailed information, and modify or delete them as needed.

Only an administrator can create a new partner.

/admin/partners	GET	{Authorization : Bearer <Token>}	[{"name" : "name", "logo": "<logo.path>"}]	Display all the partners
/admin/partners	POST	{Authorization : Bearer <Token> {"name" : "name", "website" : "website", "logo" : "<logo.path>" "financial aid" : 0}	{"message" : "<partner.name> created successfully !"}]	Creation of a new partner
/admin/partners/: id	GET	{Authorization : Bearer <Token>}	{"name" : "name", "website" : "website", "logo" : "<logo.path>" "financial aid" : 0}	Getting the details of a partner
/admin/partners/: id	PATCH	{Authorization : Bearer <Token> {"name" : "name", "website" : "website", "logo" : "<logo.path>" "financial aid" : 0}	{"message" : "Partner successfully updated !"}]	Updating a partner
/admin/partners/: id	DELETE	{Authorization : Bearer <Token>}	{"message" : "Partner successfully deleted !"}]	Deleting a partner

Here are the routes for managing citations.

The administrator can list all existing citations in the database, view their detailed information, and delete them if necessary. All logged-in users can create a new citation.

/admin/quotes	GET	{Authorization : Bearer <Token>}	{{"firstName": "firstName", "lastName": "lastName", "title" : "title"}}	Display all the quotes
/admin/quotes/:id	GET	{Authorization : Bearer <Token>}	{"logo" : "<startUpLogo.path>", "firstName" : "fistName", "lastName" : "lastName", "title" : "title", "description" : "description"}	Display the details of a quote
/admin/quotes/:id	DELETE	{Authorization : Bearer <Token>}	{"message" : "Quote successfully deleted !!"}	Delete a quote

Here are the routes for managing events. The administrator can list all existing events in the database, view their detailed information, and modify or delete them as needed. Only an administrator can create a new event.

/admin/events	GET	{Authorization : Bearer <Token>}	{{"title" : "title", "date" : dateTime, "type" : "type"}}	Display all the events for the actual month and the next 2 months
/admin/events	POST	{Authorization : Bearer <Token>}	{{"title" : "title", "date" : dateTime, "type" : "type", "description" : "description"}}	Creation of a new event
/admin/events/:id	GET	{Authorization : Bearer <Token>}	{{"title" : "title", "date" : dateTime, "type" : "type", "description" : "description"}}	Display the details of an event
/admin/events/:id	PATCH	{Authorization : Bearer <Token>}	{{"title" : "title", "date" : dateTime, "type" : "type", "description" : "description"}}	Updating an event
/admin/event/:id	DELETE	{Authorization : Bearer <Token>}	{"message" : "Event successfully deleted"}	Deleting an event

Here are the routes for managing posts. The administrator can list all existing posts in the database, view their detailed information, and modify or delete them as needed. Only an administrator can create a new post.

/admin/posts	GET	{Authorization : Bearer <Token>}	[{"title" : "title", "description" : "description"}]	Display all the posts
/admin/posts	POST	{Authorization : Bearer <Token> {"title" : "title", "description" : "description", "picture" : "picture.path" }	{"message" : "Post created successfully"}	Creation of a post
/admin/posts/:id	GET	{Authorization : Bearer <Token>}	{"title" : "title", "description" : "description", "picture" : "picture.path" }	Display the details of a post
/admin/posts/:id	PATCH	{Authorization : Bearer <Token> {"title" : "title", "description" : "description", "picture" : "picture.path" }	{"message" : "Post updated successfully !"}	Updating a post
/admin/posts/:id	DELETE	{Authorization : Bearer <Token>}	{"message" : "Post successfully deleted !"}	Deleting a post

Here are the routes for managing rentals. The administrator can list all existing rentals in the database, view their detailed information, and modify or delete them as needed. Only an administrator can create a new location.

/admin/location	GET	{Authorization : Bearer <Token>}	{ "title" : "title", "price" : "price", "size" : "size" }	Displaying all the locations
/admin/location	POST	{Authorization : Bearer <Token> { "title" : "title", "description" : "description", "price" : "price", "size" : 0 "picture" : ["<picture.path">] }	{ "message" : "place successfully created !" }	Creating a new location
/admin/location/:id	GET	{Authorization : Bearer <Token>}	{ "title" : "title", "description" : "description", "price" : "price", "size" : 0 "picture" : ["<picture.path">] }	Display the details of a place
/admin/location/:id	PATCH	{Authorization : Bearer <Token> { "title" : "title", "description" : "description", "price" : "price", "size" : 0, "picture" : ["<picture.path">] }	{ "message" : "place successfully created !" }	Updating an existing place
/admin/location/:id	DELETE	{Authorization : Bearer <Token>}	{ "message" : "place successfully deleted !" }	Deleting a place

Village by CA's external API :

For this project, we use only two external API: Google reCAPTCHA v2 and Cloudinary.

Google reCAPTCHA :

We will use it on the endpoint /contact and for security reasons:

Google reCAPTCHA is used to verify that an action on our site—such as submitting a contact form—is performed by a human and not an automated bot.

The user's browser generates a token after passing a visual check, and our API then sends this token to Google for validation. Using our secret key, Google confirms that the request originates from our site and analyzes the behavior associated with the token to determine if it is from a human. If the verification is successful, our API proceeds and sends the email if everything is correct. This effectively protects our site against spam, automated attacks, and abuse while remaining transparent to real users.

We will therefore call the Google API from this endpoint:

`https://www.google.com/recaptcha/api/siteverify?secret=<secret_key>&response=<response_key>`

The `secret_key` is the secret key provided by Google reCAPTCHA to identify our site, which we will store in the backend.

The `response_key` is the token returned by our frontend, representing the user's action on the CAPTCHA that the Google API will analyze.

Route	Method	Input Format	Output Format
<code>https://www.google.com/recaptcha/api/siteverify?secret=<secret_key>&response=<response_key></code> In /contact	POST		<pre>{ "success": true, "challenge_ts": "2025-01-10T12:34:56Z", "hostname": "our_site.com", "action": "contact_form" }</pre>

Cloudinary :

We will use it on all the endpoints/routes that need to upload a picture.

Indeed, we will use Cloudinary to store our images on an external storage server, not in our database, as this would negatively impact our application's performance and quickly overload our database. We will only store the path to each image located in Cloudinary within our database.

Therefore, each time we want to save images in our application, we will make calls to the Cloudinary API to save them.

The images stored there are primarily logos of startups and partners, photos of posts, and photos of rooms available for rent from the administrator.

Each time a user uploads an image, our Multer dependency, coupled with multer-storage-cloudinary, facilitates the file's upload to Cloudinary.

Multer intercepts the file, verifies that it complies with the defined rules (such as allowed formats), and then forwards the file to the Cloudinary SDK.

The Cloudinary SDK automatically uses our API keys (stored in environment variables) to authenticate the request and send the image to our Cloudinary storage.

The API call generated by multer-storage-cloudinary looks like this:

```
new CloudinaryStorage({
  cloudinary,
  params: {
    folder: 'portfolio',
    allowed_formats: ['jpg', 'png', 'jpeg', 'webp']
  }
});
```

If we were to make the request ourselves using the Cloudinary Software Development Kit, it would be:

```
cloudinary.uploader.upload(filePath, {
  folder: 'portfolio',
  resource_type: 'image'});
```

Cloudinary then stores the image in the specified folder and returns a public URL. This URL is saved in our database so we can retrieve and display the image in the application.

Route	Method	Input Format	Output Format
<a href="https://api.cloudinary.com/v1_1/<cloud_name>/image/upload">https://api.cloudinary.com/v1_1/<cloud_name>/image/upload In all routes that can upload a picture.	POST	Content-Type: multipart/form-data Picture: <binary File>	{ "url": "https://res.cloudinary.com/.../image/upload/v12345/myPicture.jpg", "public_id": "portfolio/abc123" }

SCM and QA Plans

SCM Processes :

Version control : Git

Branching strategy : one branch per person, one branch for dev implementation and a production branch for the deliverable application.

Pull Request and merges : the commits of this project will follow the following template:

[changes : add, fix, remove] : [description of the new implementation].

Each merge on the development and production branch will have to be approved by the other team member through a pull request.

QA :

Code clarity : ESLint for backend

Api testing : API will be tested using Postman

Code testing: Unittest

Technical Justifications

Prod Dependencies :

REACT

First, we chose the React library for the front end, simply because it is widely used and very modern. React has a huge community with extensive documentation, and since we wanted to learn about other front-end technologies, it's the perfect tool for learning thanks to its many resources.

Furthermore, because many developers use it, and because the Village by CA association will want to update its interface in the long term, many developers will be able to easily improve it. In addition, we really like the fact that every element of the interface can be reused, as it facilitates website optimization.

NODE.JS and EXPRESS

We want to work with Node.js and Express, which allow the use of JavaScript as a backend language and faster application development with a REST API. We want to work with JavaScript because we find Python too verbose, and with Express because we also dislike the file structure imposed by Flask.

Furthermore, we have worked with Node.js and Express in class, so we are familiar with these technologies, which gives us time to learn others like React. Finally, Node.js and Express are widely used in application development and are therefore very well documented with numerous resources.

MySQL

We chose MySQL for our database because it's a very popular technology, and therefore well-documented and easy to maintain. It's an excellent tool for relational databases. MySQL is performant and more than sufficient for the anticipated data volume. Furthermore, MySQL is free and very stable, which is ideal for a non-profit organization. Combined with the Prisma ORM, which is widely used with Node.js, it makes for a powerful and easily scalable tool because if we ever need a larger database, such as PostgreSQL, Prisma makes that migration easy.

PRISMA

We specifically want to use Prisma because it offers a cleaner structure compared to other ORMs that rely on decorators, classes, and other elements scattered throughout the code. Prisma uses only a single file that describes the entire database.

Prisma allows for easy switching between database management systems (for example, moving from MySQL to PostgreSQL) without rewriting all the application logic, providing significant scalability and flexibility for the project's future.

Finally, Prisma boasts exemplary documentation, an active community, and seamless integration with Express and Node.js, making it a natural choice for a modern project seeking to combine productivity, robustness, and scalability.

Google ReCAPTCHA V2

We use Google reCAPTCHA to protect the application's forms from bots and automated attacks. This solution verifies that a user is human before processing a request, preventing spam, fraudulent account creation, and abuse of sensitive endpoints.

reCAPTCHA v2 works seamlessly for the user and provides a score to filter suspicious requests without disrupting the user experience. Free, widely used, and easy to integrate with Express, reCAPTCHA provides an essential security layer perfectly suited to the needs of Village by CA.

Cloudinary

We chose Cloudinary for image management because it's a specialized, high-performance solution perfectly suited to the needs of a modern application. Cloudinary allows us to fully delegate image storage, optimization, and distribution, significantly reducing the load on our backend and improving overall website performance.

Images are automatically compressed, resized, and served via a CDN, ensuring fast loading and an excellent user experience. Cloudinary also offers a high level of security for uploads, as well as an API that's easy to integrate with Node.js, simplifying media management in our application.

Finally, the service is free for small projects and remains easily scalable, making it an ideal solution for an organization like Village by CA, which can evolve its platform without technical constraints.

JsonWebToken

We use the JSON Web Token (JWT) module for authentication because it's a modern, secure solution perfectly suited to REST APIs. JWT enables stateless authentication, meaning no server-side session storage, which simplifies the architecture and improves application scalability.

Each token contains essential user information (ID, role, expiration date) and is signed, guaranteeing its integrity and preventing fraudulent modification. This approach also simplifies role and permission management in our application. JWT is easy to integrate with Express, which is widely used in the industry and compatible with all platforms, making it a natural choice for securing access to our API.

Nodemailer

We use Nodemailer to manage email sending from our Express backend. It's a simple, reliable module perfectly suited to the Node.js ecosystem. Nodemailer makes it easy to send confirmation, contact, or notification emails without having to manually manage SMTP protocols, which greatly simplifies implementation.

The module is compatible with all email services (Gmail, Outlook, OVH, Mailgun, etc.), supports HTML emails, and offers secure configuration via TLS. Free, open-source, and widely used, Nodemailer is a robust and accessible solution, ideal for a non-profit project like Village by CA. We prefer it to express-mailer, which is more outdated and no longer maintained.

Validator

We use the validator module to ensure the quality and security of data sent to our API. Validator makes it easy to check sensitive formats (emails, passwords, field lengths, etc.) and prevent the recording of incorrect or potentially dangerous data.

It also offers sanitization features to clean user input (removing spaces, normalizing emails, escaping special characters), which improves the consistency of the data stored in the database. Lightweight, reliable, and widely used in the Node.js ecosystem, validator provides an essential security layer to protect the application and ensure clean and controlled processing of user information.

BCrypt

We use BCrypt to secure user passwords. BCrypt is a hashing algorithm specifically designed for password protection. Passwords are therefore never stored in plain text in the database. Upon login, BCrypt compares the provided password with the stored hash, ensuring secure authentication. Its robustness, ease of integration, and widespread adoption in the Node.js ecosystem make it a reliable and suitable solution for our project.

Multer and Multer-storage-cloudinary

We use Multer to manage file uploads in our Express API. Multer is a middleware specializing in processing forms containing files, which allows us to cleanly retrieve images sent by the frontend, control their size and type, and prevent upload errors.

To avoid storing files on our server, we also use multer-storage-cloudinary, which replaces local storage with automatic uploads to Cloudinary. As soon as a file is received, it is directly transferred to Cloudinary, which handles hosting, optimization, and distribution via the Content Delivery Network.

This combination results in a simple, secure, and efficient upload system, while keeping our backend lightweight and easy to maintain. Only the final URL of the image is stored in the database via Prisma, which significantly simplifies media management.

FullCalendar

We chose FullCalendar to manage event display because it's a comprehensive, robust solution perfectly suited to an association management application. FullCalendar offers a ready-to-use interactive calendar with advanced features such as view navigation, event display, drag-and-drop functionality, and interface customization.

The library has official React integration, allowing it to be used as a modern, easily configurable component compatible with our architecture. FullCalendar is also very flexible: it allows customization of views, colors, behaviors, and the management of different types of events (single, recurring, multi-day, etc.). Thanks to its maturity, extensive documentation, and widespread adoption, FullCalendar allows us to offer a professional interface while significantly reducing development time. It's an ideal solution for visualizing and organizing events at Village by CA.

DEV dependencies :

Nodemon

We use Nodemon to improve the comfort and speed of developing our Express API. Nodemon automatically monitors project files and restarts the server as soon as a change is detected. This avoids manually restarting the server after each change and allows for a much smoother and more efficient development cycle.

The tool is easy to configure, widely used in the Node.js ecosystem, and perfectly suited to a modern development environment. Thanks to Nodemon, we save time and reduce the risk of errors associated with manual restarts. Furthermore, we have already used it and are therefore very familiar with this tool.

ESLint

We use ESLint to ensure the quality, consistency, and maintainability of the code in our project. ESLint automatically analyzes JavaScript/TypeScript code and detects potential errors, bad practices, or style inconsistencies. This helps prevent subtle bugs and ensures clean, consistent code, even when multiple developers are collaborating.

Design Justifications :

The design choices for the website were chosen to align with the company's established visual identity, starting with a color palette directly inspired by the tones and contrasts present in the existing logo. This continuity ensures a coherent brand experience across the company's various services.

A clean and minimalist design approach was intentionally chosen to offer users a modern and intuitive navigation environment, allowing key information to stand out effortlessly. Every decision was refined through multiple meetings with the client, ensuring that the final interface reflects their expectations.