

| | |
|---|----------|
| Nahova developer documentation..... | 1 |
| Laravel Backend..... | 1 |
| VueJs frontend..... | 3 |
| Project setup..... | 4 |
| Setting up the project on Windows:..... | 4 |
| Setting up the project on Linux and Mac:..... | 5 |
| Database..... | 6 |
| users table..... | 6 |
| roles table..... | 7 |
| carTypes table..... | 7 |
| types table..... | 7 |
| products table..... | 8 |
| chats table..... | 10 |
| mechanic table..... | 11 |
| messages table..... | 12 |
| Rest API..... | 13 |
| User:..... | 13 |
| Roles:..... | 14 |
| Products:..... | 15 |
| Types:..... | 16 |
| CarTypes:..... | 16 |
| Mechanics:..... | 16 |
| Chats:..... | 18 |
| Messages:..... | 18 |

Nahova developer documentation

Laravel Backend

- The whole backend structure is in the 'WebApplication' directory. The paths I will mention for the backend are all inside this directory.
- The API controllers are located in the 'app/Http/Controllers' folder. Here you can see what our API endpoints are capable of and how they would respond to a request.
- The API requests are in the 'app/Http/Requests' directory used to validate all the requests.
- API resources take their places in the 'app/Http/Resources' folder. Containing the endpoints of different Http requests.
- The 'app/Models' directory contains our Models and connections between each of them.
- The api.php file is located in the 'routes/' directory. This file contains all the routes separated to different groups based on our database.
- In the 'database/' folder you can find the 'migrations' directory which contains our database structure. And the "seeders" directory which contains all the permanent data we have in our database.
- The 'public/images' contains all the images uploaded into the database.

VueJs frontend

- The frontend part of the project is located in the 'FrontendApplication' directory. The paths I will mention for the frontend are all inside this directory.
- In the 'public/img' folder we have our images that are connected directly to the frontend part, they are not from the database.
- Inside the 'assets' folder there is the app.scss file containing all the styles we use on the page.
- The 'src/components' and 'src/views' folders contain the base of our pages. They have all the visual stuff that the user sees.
- In the 'src/lang' directory we have a .mjs file for each language we use or will plan on using on the page. Currently the 'en.mjs' is in use. But you can extend the application to any language just by creating a new "language".mjs' file containing the translations.
- The 'router/index.js' file contains how each view is accessible.
- The 'story' directory contains .js files. Each file contains all the functions we use for a group of data. Different groups like 'products, mechanics, types, users' etc.
- Inside the 'utils' folder there is the http.mjs file. This contains our own axios instance. We can communicate with the backend through the link inside this file.

Project setup

Setting up the project on Windows:

First you need to clone the git repository. Open your terminal in the destination folder you want to clone the project into. And run this command:

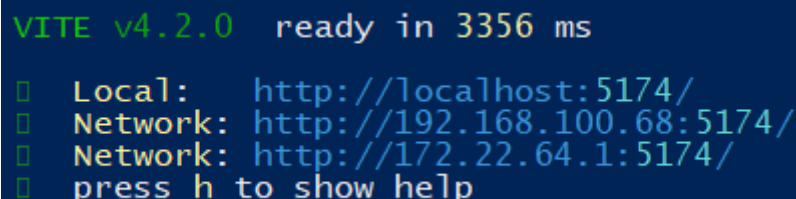
'git clone https://github.com/ImAdamNagy/Project_Nahoba.git'

After that, start your docker. If you don't have docker installed you can download it from here: <https://docs.docker.com/engine/install/>

Then run the 'sh start.sh' command in the shell script opened from the root directory of the WebApplication folder.

If the script fails for any reason. Install git bash (<https://git-scm.com/downloads>) and follow the instructions below. We will start by setting up the server.

1. Open git bash in the root directory of the WebApplication folder.
2. Run **'copy .env.example .env'**
3. Run **'docker compose build'**
4. Run **'docker compose -f docker-compose.yml -f docker-compose.dev.yml up -d'**
 - a. Now that the server is up, we set up the laravel project. In the bash stay where you finished Step 4.
5. Run **'docker compose exec nahoba-app fish'**
6. Run **'composer install'**
7. Run **'php artisan key:generate'**
8. Run **'php artisan migrate:fresh --seed'**
9. Now exit the container by running **'exit'** and go to the FrontendApplication folder.
10. Install node to your computer (<https://nodejs.org/en/download>)
11. In the FrontendApplication folder run **'npm install'**
12. Run **'npm run setup:win'**
13. Once you are done with these, you will be able to start the live server by running this command: **'npm run dev'**.



```
VITE v4.2.0 ready in 3356 ms
□ Local:    http://localhost:5174/
□ Network:  http://192.168.100.68:5174/
□ Network:  http://172.22.64.1:5174/
□ press h to show help
```

If you did everything right, you should see something like this. Now open the link signed with 'Local:' and you are good to go.

Setting up the project on Linux and Mac:

First you need to clone the git repository. Open your terminal in the destination folder you want to clone the project into. And run this command:

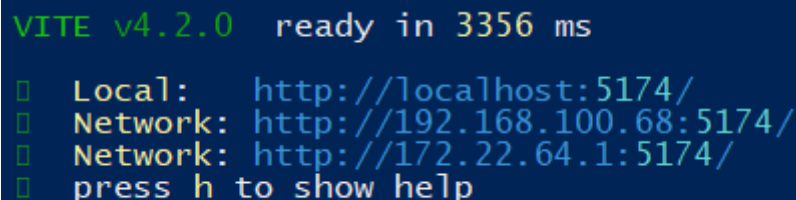
'git clone https://github.com/lmAdamNagy/Project_Nahoba.git'

After that, start your docker. If you don't have docker installed you can download it from here: <https://docs.docker.com/engine/install/>

Then run the 'sh start.sh' command in bash, opened from the root directory of the WebApplication folder.

If the script fails for any reason. Follow the instructions below. We will start by setting up the server.

1. Open bash in the root directory of the WebApplication folder.
2. Run **'cp .env.example .env'**
3. Run **'docker-compose build'**
4. Run **'docker-compose -f docker-compose.yml -f docker-compose.dev.yml up -d'**
 - a. Now that the server is up, we set up the laravel project. In the bash stay where you finished Step 4.
5. Run **'docker-compose exec nahoba-app fish'**
6. Run **'composer install'**
7. Run **'php artisan key:generate'**
8. Run **'php artisan migrate:fresh --seed'**
9. Now exit the container by running **'exit'** and go to the FrontendApplication folder.
10. Install node to your computer (<https://nodejs.org/en/download>)
11. In the FrontendApplication folder run **'npm install'**
12. Run **'npm run setup'**
13. Once you are done with these, you will be able to start the live server by running this command: **'npm run dev'**.



```
VITE v4.2.0 ready in 3356 ms
␣ Local:    http://localhost:5174/
␣ Network:  http://192.168.100.68:5174/
␣ Network:  http://172.22.64.1:5174/
␣ press h to show help
```

If you did everything right, you should see something like this. Now open the link signed with 'Local:' and you are good to go.

Database

users table

| Key | Column Name | Data Type | Description |
|---------|-------------|------------|--|
| primary | id | unique key | User Id |
| | firstname | string | Users Firstname |
| | lastname | string | Users Lastname |
| | email | string | Users email |
| | tel | string | User phone number |
| | username | string | Users username |
| | password | string | Users password |
| | role_id | integer | Users role id. Connects to the roles table |

```
Schema::create('users', function (Blueprint $table) {
    $table->id();
    $table->string('firstname');
    $table->string('lastname');
    $table->string('email')->unique();
    $table->string('tel');
    $table->string('username')->unique();
    $table->string('password');
    $table->rememberToken();
    $table->timestamps();
    $table->foreignId('role_id')->
        constrained("roles","id");
    $table->timestamp('last_used_at')->nullable();
});
```

roles table

| Key | Column Name | Data Type | Description |
|---------|-------------|------------|-------------|
| primary | id | unique key | Role Id |
| | role_name | string | Role Name |

```
Schema::create('roles', function (Blueprint $table) {  
    $table->id();  
    $table->string('role_name',25);  
});
```

carTypes table

| Key | Column Name | Data Type | Description |
|---------|-------------|------------|---------------|
| primary | id | unique key | Car Type Id |
| | name | string | Car Type Name |
| | year | integer | Car Type year |

```
Schema::create('carTypes', function (Blueprint $table) {  
    $table->id();  
    $table->string('name');  
    $table->integer('year')->nullable();  
});
```

types table

| Key | Column Name | Data Type | Description |
|---------|-------------|------------|-------------|
| primary | id | unique key | Type Id |
| | type | string | Type Name |

```
Schema::create('types', function (Blueprint $table) {  
    $table->id();  
    $table->string("type",20);  
});
```

products table

| Key | Column Name | Data Type | Description |
|---------|---------------------|------------|--|
| primary | id | unique key | Product Id |
| | seller_id | integer | Sellers Id |
| | product_name | string | Name of the product |
| | product_price | integer | Price of the product |
| | types_id | integer | Shows the products type Id. Connects to the types table |
| | car_typeId | integer | Show the products carType Id. Connects to the carTypes table |
| | product_img | string | Products image name |
| | product_description | string | Products description |
| | product_location | string | Location of the product |
| | product_enable | boolean | Shows if the product is enable (displayed on the page or not). Only admin can enable products. |


```
Schema::create('products', function (Blueprint $table) {
    $table->id();

    $table->foreignId('seller_id')->constrained("users","id");
    $table->string('product_name');
    $table->integer('product_price');

    $table->foreignId('types_id')->constrained("types","id");

    $table->foreignId('car_typeId')->constrained("carTypes","id");
    $table->string('product_img');
    $table->string('product_description',150);
    $table->string('product_location', 120);
    $table->boolean('product_enable');
});
```

chats table

| Key | Column Name | Data Type | Description |
|---------|-------------|-------------|--|
| primary | id | unique key | Chats Id |
| | from | foreign key | The chat senders user Id. Connects to the users table |
| | to | foreign key | The chat targets user Id. Connects to the users table. |

```
Schema::create('chats', function (Blueprint $table) {  
    $table->id();  
  
    $table->foreignId("from")->constrained("users","id");  
  
    $table->foreignId("to")->constrained("users","id");  
});
```

mechanic table

| Key | Column Name | Data Type | Description |
|---------|--------------|-------------|---|
| primary | id | unique key | Mechanic Id |
| | introduction | text | A short description of the mechanic. |
| | country | string | The country where the mechanic lives. |
| | postal_code | integer | Postal code of the mechanic |
| | city | string | City of the mechanic |
| | address | string | Detailed address of the mechanics living place |
| | profile_pic | string | Name of the mechanics profile picture. |
| | profession | string | The mechanics profession |
| | user_id | foreign key | The user who is the mechanic. Connected to the users table. |

```
Schema::create('mechanic', function (Blueprint $table) {
    $table->id();
    $table->text("introduction");
    $table->string("country");
    $table->integer("postal_code");
    $table->string("city");
    $table->string("address");
    $table->string("profile_pic");
    $table->string("profession");

    $table->foreignId("user_id")->constrained("users","id");
});
```

messages table

| Key | Column Name | Data Type | Description |
|---------|-------------|-------------|--|
| primary | id | unique key | Message Id |
| | message | string | The message itself |
| | sender_id | foreign key | The user id of the sender. Connects tot the users table. |
| | chat_id | foreign key | The chat id of the chat. Connects to the chats table. |

```
Schema::create('messages', function (Blueprint $table) {
    $table->id();
    $table->string("message");

    $table->foreignId("sender_id")->constrained("users", "id");

    $table->foreignId("chat_id")->constrained("chats", "id");
    $table->timestamps();
});
```

Rest API

Base URL: <http://localhost:8881/api>

User:

Controller: UserController

Model: UserModel

| Method | Endpoint | Action |
|--------|----------------|-----------------------|
| GET | /users/{user} | show |
| GET | /users/current | getCurrentUserDetails |
| GET | /users/ | index |
| DELETE | /users/{user} | destroy |
| PATCH | /users/{user} | update |
| POST | /login | authenticate |
| POST | /register | store |
| GET | /logout | logout |

Request: UserRequest

```
return [  
    'username' => ['required', 'min:4', 'max:15'],  
    'firstname' => ['required', 'min:1', 'max:15'],  
    'lastname' => ['required', 'min:1', 'max:15'],  
    'tel' => ['required', 'min:6', 'max:18'],  
    'email' => ['required', 'email'],  
];
```

Roles:

Controller: RoleController

Model: Role

| Method | Endpoint | Action |
|--------|---------------|-------------------|
| GET | /roles/ | indexwithoutadmin |
| GET | /roles/{role} | show |

Products:

Controller: ProductController

Model: Product

| Method | Endpoint | Action |
|--------|------------------------------|---------------------|
| GET | /products/ | index |
| GET | /products/{product} | show |
| GET | /products/enable | enable |
| GET | /products/userproducts/{id} | userproducts |
| POST | /products/ | store |
| DELETE | /products/{product} | destroy |
| DELETE | /products/deleteAll/{userid} | deleteUsersProducts |
| PATCH | /products/{product} | update |
| PATCH | /products/enable/{product} | updateEnable |
| GET | /products/disable | disable |

Request: ProductRequest

```
return [  
    'product_name' => ['required', 'min:5'],  
    'product_price' => ['required', 'numeric'],  
    'types_id' => ['required', "exists:types,id"],  
    'car_typeId' => ['required',  
"exists:carTypes,id"],  
    'product_img' => ['required', 'max:1536'],  
    'product_description' => ['required', 'max:150'],  
    'product_location' => ['required', 'max:120'],  
];
```

Types:

Controller: TypesController

Model: Type

| Method | Endpoint | Action |
|--------|---------------|---------|
| GET | /types/ | index |
| GET | /types/{type} | show |
| POST | /types/ | store |
| DELETE | /type/{type} | destroy |

Request: TypeRequest

```
return [  
  "type" => ['required', 'min:1']  
];
```

CarTypes:

Controller: CarTypeController

Model: CarType

| Method | Endpoint | Action |
|--------|---------------------|---------|
| GET | /cartypes/ | index |
| GET | /cartypes/{cartype} | show |
| POST | /cartypes/ | store |
| DELETE | /cartypes/{cartype} | destroy |

Request: CarTypeRequest

```
return [  
  "name" => ['required'],  
  "year" =>  
    ['required', 'numeric', 'min:1886', 'max:2023']  
];
```

Mechanics:

Controller: MechanicController

Model: Mechanic

| Method | Endpoint | Action |
|--------|-----------------------|------------------------|
| GET | /mechanics/ | index |
| GET | /mechanics/{id} | show |
| GET | /mechanics/hasmec | currentUserHasMechanic |
| GET | /mechanics/current | currentMechanic |
| POST | /mechanics/ | store |
| PATCH | /mechanics/{mechanic} | update |
| DELETE | /mechanics/{mechanic} | destroy |

Request: MechanicRequest

```
return [  
    'introduction'=> ["required","min:5"],  
    'country'=> ["required","min:3","max:100"],  
    'postal_code'=> ["required","numeric"],  
    'city'=> ["required","min:3","max:100"],  
    'address'=> ["required","min:3","max:150"],  
    'profession'=> ["required","min:3","max:30"]  
];
```

Chats:

Controller: ChatController

Model: Chat

| Method | Endpoint | Action |
|--------|-----------------|----------|
| GET | /chats/ | findchat |
| GET | /chats/{chat} | show |
| POST | /chats/ | store |
| DELETE | /chats/{userid} | destroy |

Request: ChatRequest

```
return [  
    "from" => ['required', 'exists:users,id'],  
    "to" => ['required', 'exists:users,id']  
];
```

Messages:

Controller: MessageController

Model: Message

| Method | Endpoint | Action |
|--------|---------------------|---------|
| GET | /messages/{chat} | index |
| GET | /messages/{message} | show |
| POST | /messages/ | store |
| DELETE | /messages/{userid} | destroy |

Request: MessageRequest

```
return [  
    'message' => ['required'],  
    'sender_id' => ['required'],  
    'chat_id' => ["required"]  
];
```