ENSG (National School of Geographic Sciences)

UGE (Gustave Eiffel University)

Class project - Creating a GIS system for 2D and 3D visualisation

TSI 2023

December 8, 2023

Contents

List of Figures

Glossary

GIS Geographic Information System

DEM Digital Elevation Model

Orthophotos TO DEFINE

OpenGL TO DEFINE

WMS

WMTS

WFS

UML

Introduction

The aim of this project is to create a GIS system for 2D and 3D visualisation that can also make simple treatments on geographical data in 4 weeks. To do so, all class members have taken part in this project, and Agile project management has been used to develop and manage our project. The duration of a sprint was only 1 week, because there were no more time to increase this duration. The team is presented in this report.

The application analysis uses UML language. UML language provides a standard way to visualize the design of a system, and all sorts of diagram can me made to represents database model or the application components for example.

The project client was Victor COINDET, head of the Teaching Department in Information Systems Technologies at ENSG. He asked the team to work with Lyon's data, a french city located in south-east of France, specifically on the 5^{th} district. It was the team role to download 2D and 3D data of this district. The results presented in this report are of this district.

1. Analysis

1.1 Languages

One requirement of the client was to only use C++. To visualise 2D data, external libraries can be used (officials or not), such as Nohlmann JSON library for example. To visualise 3D data, OpenGL is used. To avoid too much installation for the client, we decided to use Docker containers to run the PostGIS and 3DCityGML database. The docker container is (re)created each time that the application is opened. This way, the client does not know how the platform interact with the database.

1.2 Functionalities

1.2.1 List of functionalities

The application is a primitive Geographical Information System (GIS), so it has to handle primitives functionalities, such as:

- **2D Visualisation**:
  - Interacting with map (move, zoom etc.) ;
  - Managing layers ;
  - Display raster and vector layers ;
  - Import flow layers from IGN website *Géoservices* ;
  - Show entity attributes when clicking on them.
- **3D Visualisation**:
  - Controlling camera (move, zoom, camera rotation etc.) ;
  - Managing layers ;
  - Display raster layers (orthophotos on DEM) and vector layers (3DCityGML file for example) ;

Additionally, basic treatments and operations on layer will be implemented, such as:

- **2D Visualisation**:
  - Changing layers styles ;
  - Editing vector layers ;
  - Calculating basic treatments on vector layers (buffer, intersection, union ...) ;
  - Export layers to ShapeFile or other extensions output.
- **3D Visualisation**:
  - Calculating indicators on 3D vector layers, such as: max height, height of the lowest point of the roof...

These are not exhaustive functionalities, and they are not ordered by difficulty nor by importance.

1.2.2 Use case diagrams

Use case diagrams can be used to represent interactions between actors and the platform. The figure 1.1 shows the use case diagram of the application.

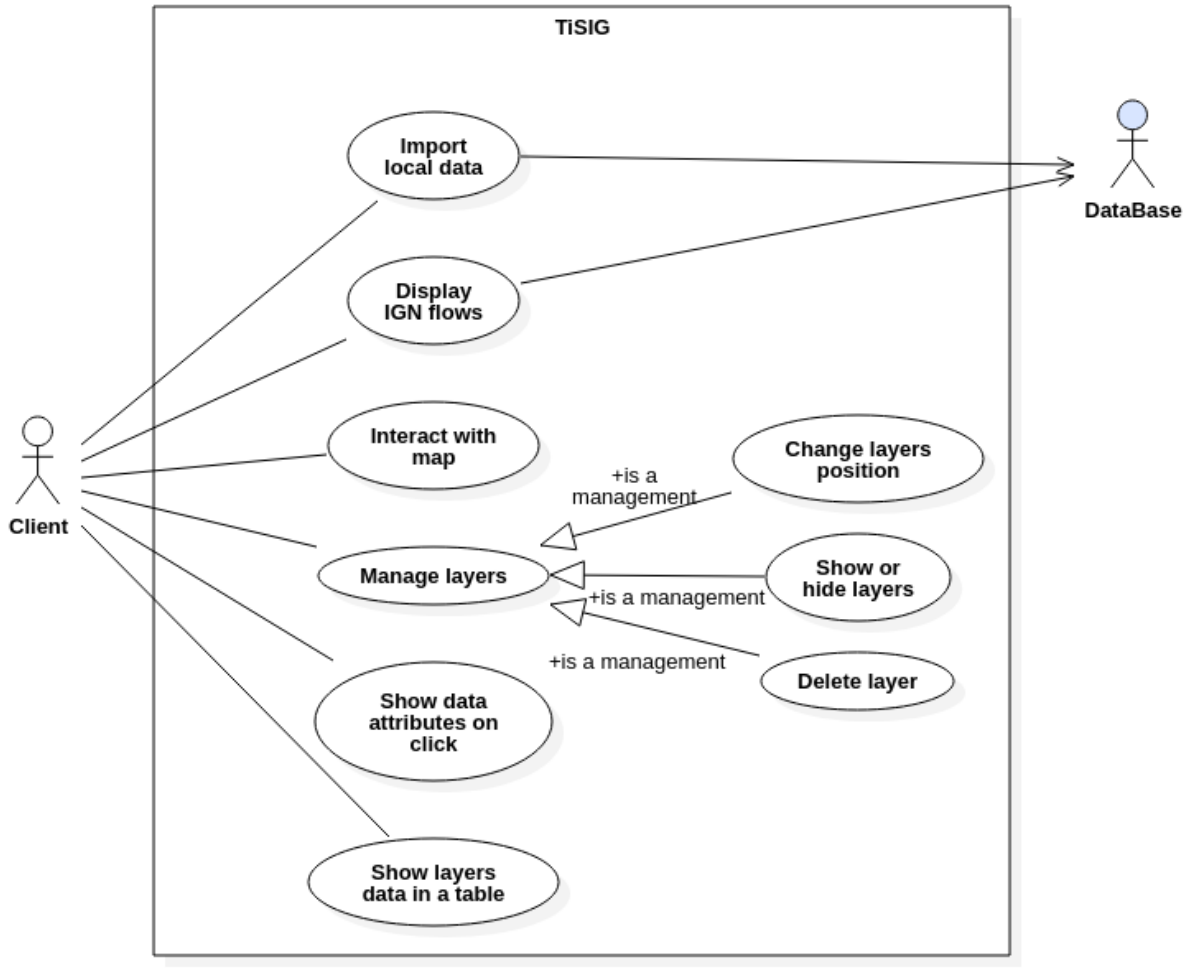

Figure 1.1: Use case Diagram

On this diagram, the client can interact with TiSIG according to the functionalities discussed before. Indeed, the client can interact with map, manage layers and visualise data information. Also, he can import data and display IGN flows. For this two actions, data are saved into the system internal database, and then used by TiSIG to be displayed.

Only first needed functionalities has been included on this diagram, but it is possible to include more actions. Even though, actors are the same: the client and the database. When it is about changing data or manipulating data, interacting with the database is necessary. The DBSM (DataBase System Manager) used is PostgreSQL, and PostGIS, a geographic extension of PostgreSQL, is used too. PostGIS can manage spatial treatments such as intersection or buffer creation. Each time a user wants to manipulate data - which means not only show data and interact with map - interactions with the database are done.

1.3 Application architecture

For the application architecture, it has been decided to work with spatial database, such as PostGIS, even though it is not necessary for the beginning of the project. Indeed, libraries in C++ exists to display ShapeFile for example without external database. However, this decision has been made to implement easily spatial treatments, as PostGIS already implements these. To difference 2D and 3D visualisation, two databases are needed. The 3D database implements 3dcitydb, the CityGML Database. To make the architecture analysis, 3 UML diagrams have been made: a class diagram, a component diagram and a deployment diagram.

1.3.1 Class diagram - Database model

To structure the database, a class diagram is adapted. Each class represents a table, and those table belong to one database (2D or 3D). 3 stereotypes are used in this diagram: *Database*, *Abstract table* and *Table*. An abstract table is a table that give a table model but can not be instantiate. Database class diagram is represented on figure 1.2.

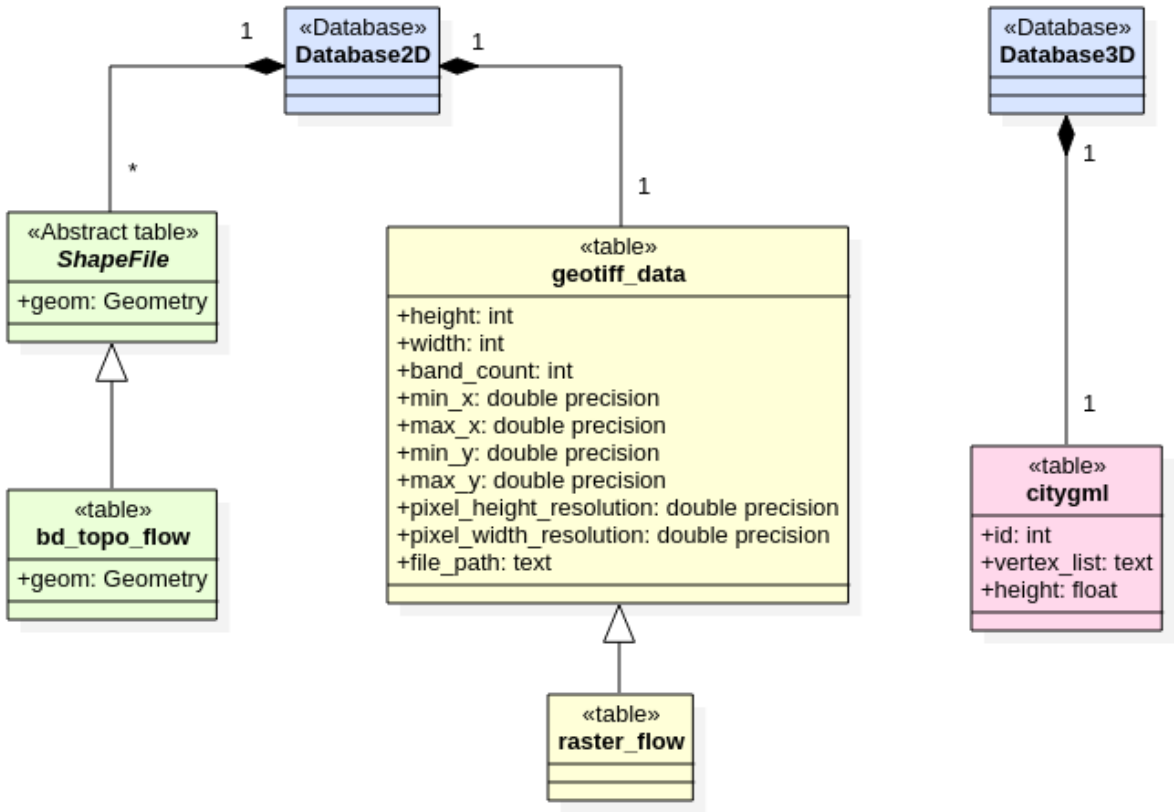

Figure 1.2: Class diagram

ShapeFile table is an abstract table, which means that there are no tables name "ShapeFile" in the database. However, this table gives a model for all shapefiles transformed into the database: they all have at least a geometry and an id. Each layer imported in the system is saved into the database in the form of a table. The table name is extracted from the file name, where all conflicts are resolved - for example, the system transforms points and space into underscores. For each vector flows, a table exists too. In the diagram, only Topographic database - *BD TOPO* in french - is represented.

For raster data, it works differently. Only one table - **geotiff_data** - is created, and all information

about every raster files will be inserted here. This choice have been made because it is easier to read directly a geotiff than to parse it into a database and then read information and display it on the application. Metadata about the file, like width, height or band_count are also saved into the database. This way, it is possible to show metadata about this by doing a request in the database. Like vector flows, raster flows are inherited from the raster table. It allows the platform to request the good table when displaying flows.

Finally, a different database is used for 3D visualisation. This database is named *Database3D*, and contains a lots of table, but only the one requested by the application is represented here: **citygml** table. Each feature of this table represents a building containing in the 3DCityGML file. A building has an id, a list of vertexes used to display 3D data with OpenGL, an height and a category like school, church... The other tables are not represented here because they are used by the application to treat 3DCityGML file, but they are never requested.

1.3.2 Component diagram

Each database only represents one system component, but there are more than two components in reality. To show how system components interacts with each other, it is possible to use a component diagram. The figure 1.3 represents the system component diagram.

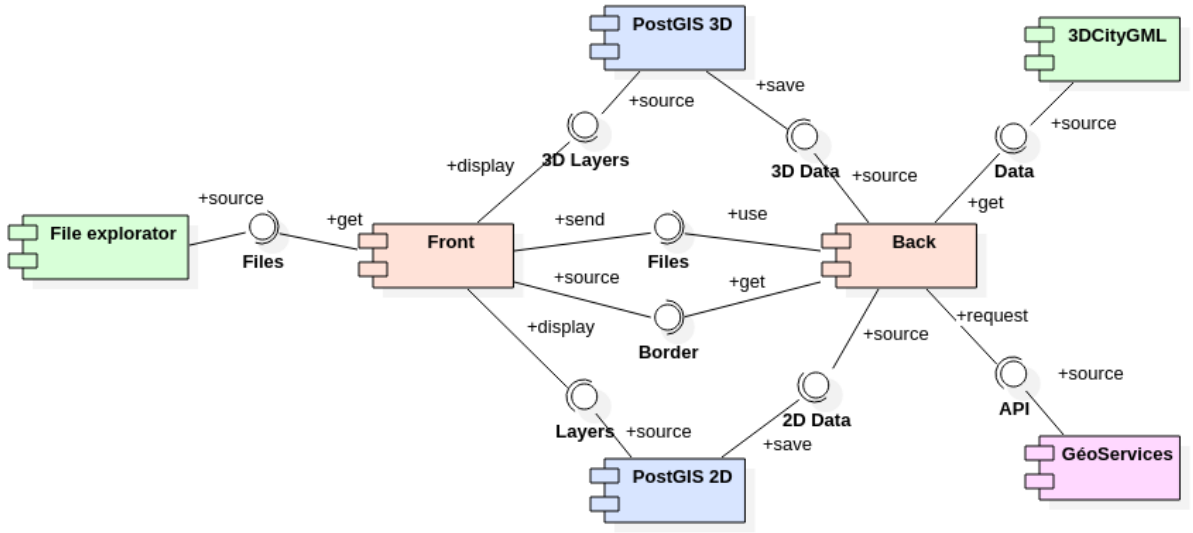

Figure 1.3: Component diagram

One category can be assigned to each component. There are four categories in the system: Database, external services, application and internal files / storage. They are respectively coloured in blue, pink, orange and green. These categories will then be used by the system deployment diagram, represented in figure 1.4. It has been decided to divide the application between front-end and back-end in the component diagram for two reasons. The first reason was for the understanding of the diagram. By dividing front and back, the interactions with other components and between them - i.e. interactions inside the application - are more visible. Secondly, this choice has been made to know how to divide work between front and back teams. By dividing them in the analysis, the task planning has been easier to make.

To display 2D or 3D data, the user has to import his own files from the file explorer. Those files are get by the front, and send to the back-end. Then, treatments on data is made, and they are saved into their respective database: 3dCityGML and DEM into PostGIS 3D; ShapeFiles, GeoTIFF and GeoJSON into PostGIS 2D. Once they are saved, front-end can request these databases to display

layers in the application. 2D and 3D data can not be visualised in the same time.

For raster and vector flows, it works differently. First, front-end send the border of the screen (coordinates in Lambert 93 or WGS 84) to back-end. With borders, it is possible to request Géo-Plateforme's API to get only interesting features to display in the application. Those features are then treated like ShapeFiles or GeoTIFF, saved into the database and displayed after. As long as a flow is displayed on the application, this process is repeated.

1.3.3 Deployment diagram

System components are known now, but it is also important to know on which devices each component will be. To do so, a deployment diagram can be useful. The figure 1.4 represents the system deployment diagram.

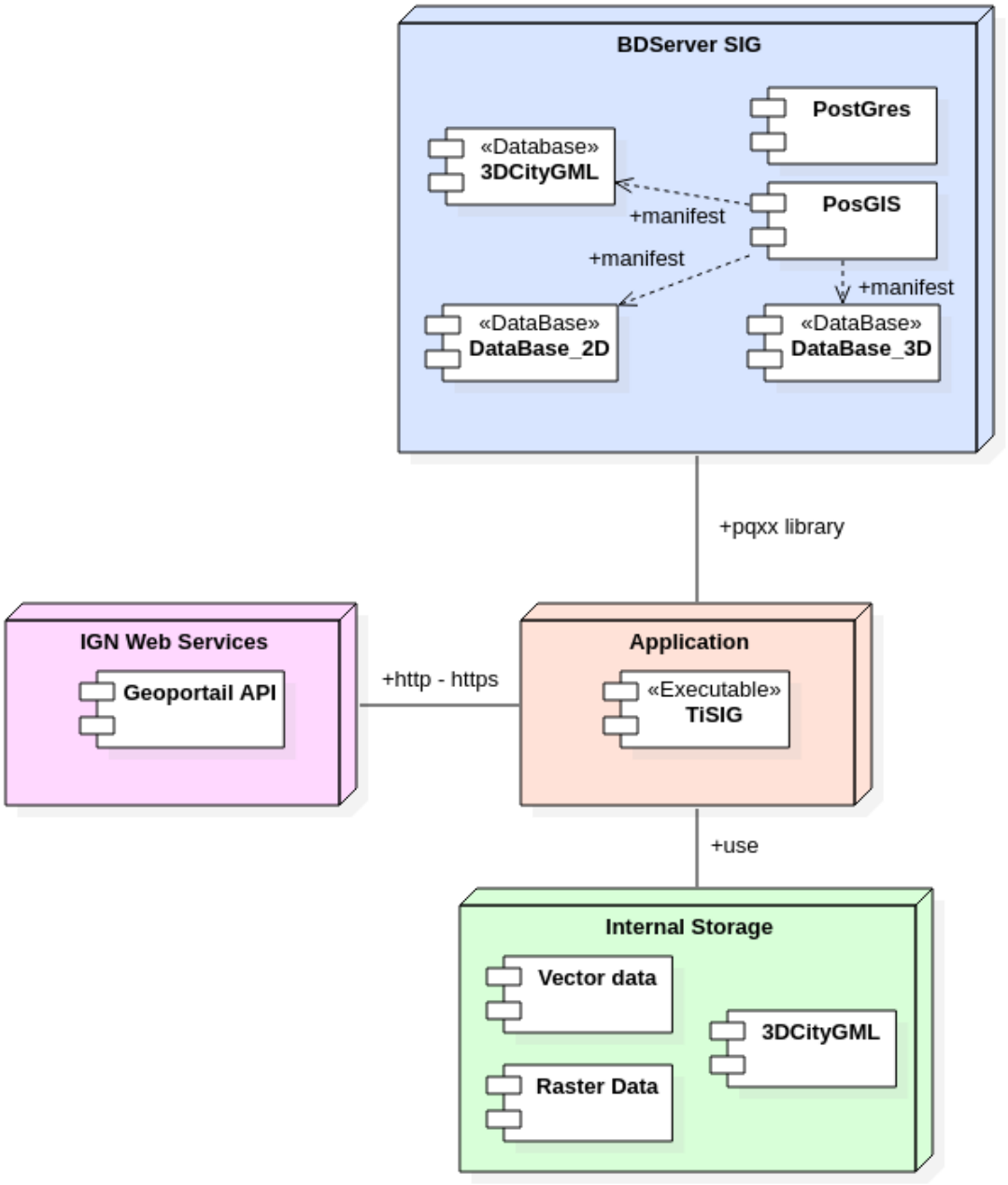

Figure 1.4: Deployment diagram

A deployment diagram represents interactions between devices and protocols they use to communicate. As described with the component diagram, they are 4 devices: Application, Internal Storage, BDServer GIS and IGN Web services. Names are different, but they represent the same thing than before and same colours have been used. Front-end and back-end have been put into TiSIG application, because dividing them here would not have been useful.

The application use files contained in internal storage to display them. Once they are transformed, they are saved into the database. All interactions between database and the application is made with the pqxx library, a C++ library used to manage PostGreSQL servers. For flows, GéoPlateforme API is requested using http protocol. As it does with internal files, data are then saved into the database.

It is not represented in the deployment diagram, but a Docker Container, based on a 3dcitydb image, is used to contains databases. It does not change the interactions between devices, only where the database is in reality. By using a docker, the user does not have to launch and install PostGreSQL nor PgAdmin 4, and it ease TiSIG's installation. Except for IGN webservices, all others nodes are in the client's computer.

1.4 TiSIG's sketch

Right after analysing client's needs, our UX Designers, Frederic YE and Lisa ZARATIN, have drawn what would they imagine TiSIG would look like. After client's approval, this sketch has been used to develop TiSIG. The sketch is represented on figure 1.5.

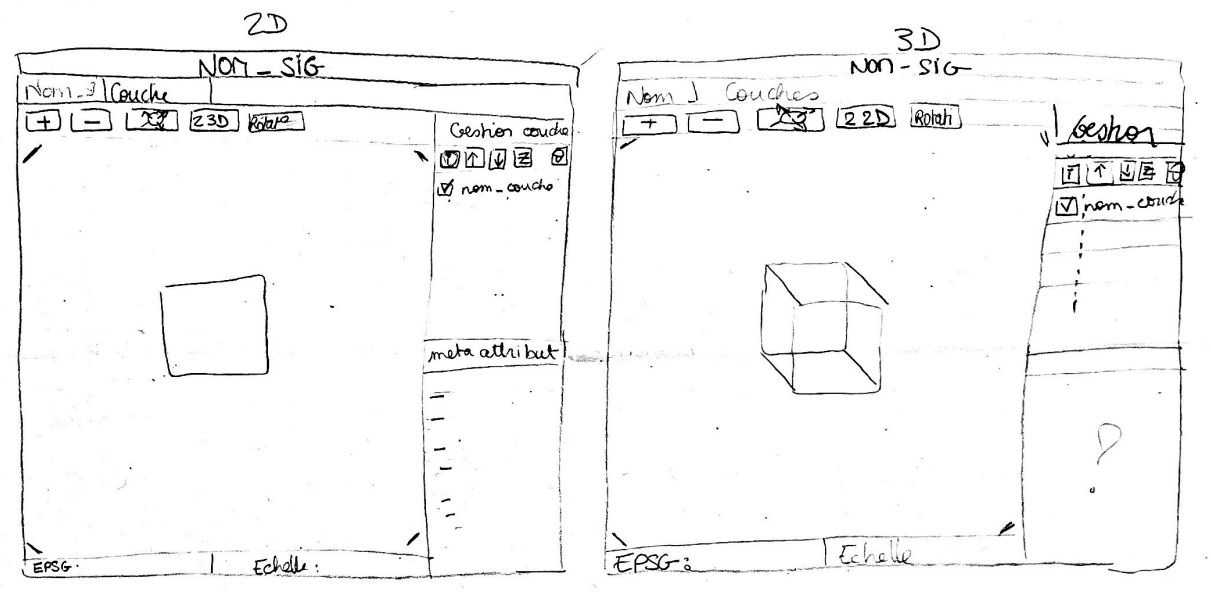

Figure 1.5: TiSIG's sketch

2. Project Management

This project was our first project in a team of 15 using agile methodology. To ensure that everything goes well, different tools have been used.

2.1 Team members and agile roles

The team was composed of 15 students. We divided it in two subteams: one for the front-end (front team) and one for the back-end (back team). This way, it was easy to split the work to do for the project life. As agile methodology was used, a scrum master and a product owner had to be chosen for each team. During the 4 weeks of project, scrum masters did not change, but product owners did sometimes. This decision was made naturally, as we discussed it between us, and let people take the role they wanted simply by saying it. A "Scrum of Scrum", supposed to make scrum masters agree when they were conflicts, was also nominate at project beginning. However, we took the decision to erase this role because it was useless, as the two scrum masters could reach on an agreement when it was necessary.

In the following list, when someone is tagged with Product Owner for one or several weeks, it means that this person was developer for the other weeks. Because it was a small project, scrum masters also was developer, only they had to make the product backlog or other things, it let enough time to also develop. It is the same for product owners. The front team was composed of:

- Vittorio TOFFOLUTTI: **Scrum Master**;
- Lisa ZARATIN: **Product Owner from week 2 to 4**;
- Victorien OLLIVER: **Product Owner for week 1**;
- Romain COURET: **Developer**;
- Vincent GIUDICELLI: **Developer**;
- Claire GUERRINI: **Developer**;
- Frédéric YE: **Developer**;

The back team was composed of:

- Mathis ROUILLARD: **Scrum Master**;
- Claire GIRARDIN: **Product Owner from week 3 to 4**;
- Clovis BERGERET: **Product Owner for week 2**;
- Axel DUMONT: **Product Owner for week 1**;
- Hannick ABDUL-KUTHOOS: **Developer**;
- Mathéo MARÉCHAL: **Developer**;
- Hicham OUTMRHOUST: **Developer**;
- Cécile TALEC: **Developer**;

2.2 Agile methodology during the project

During the whole project, as mentioned before, each person had a unique role per week. Additionally, we used Trello, an online project management tool. With Trello, it is possible to create boards. These boards can be used as Product Backlogs, Kan ban or for other things too. For example, to easily see who was the product owner of a week or who work on a specific user story, we had a board named *Roles* where each week, for front-end and back-end, agile roles and main user story were assigned to one person.

2.2.1 Product Backlog and Sprint Backlog

To create the product backlog, we first chose to do a team brainstorming to define what was important for the client. It is important to mention that we did not have a project subject during the first two days, so we had to create user stories without knowing if it would be relevant. Each member had a user story and we decided that everyone had to write their own user stories, and not only scrum masters and product owners. That was probably a good idea because the first week of project was a sprint of barely two days, but every member was involved. We divided them into front and back, and created a product backlog for the two subteams. An example of the front's product backlog is represented on figure 2.1.

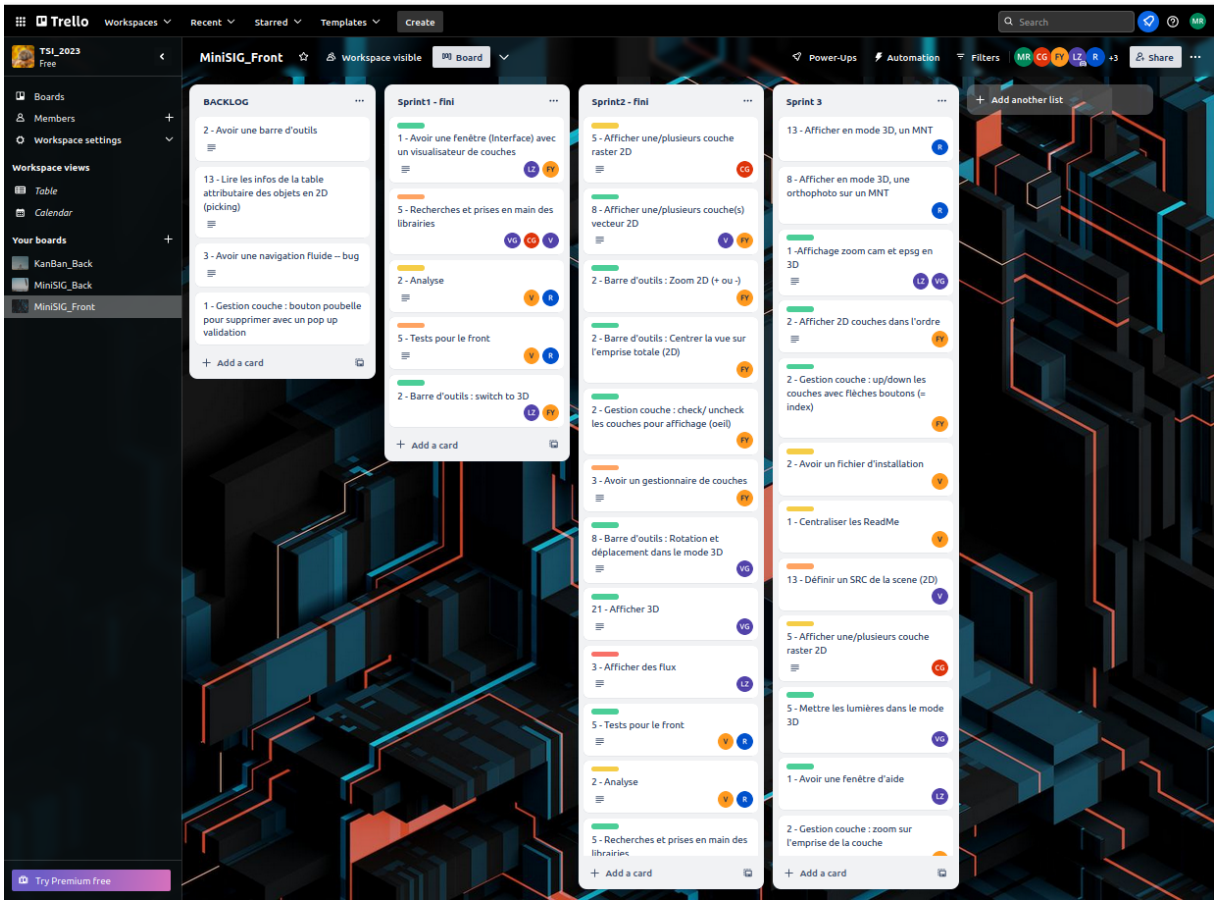

Figure 2.1: Example of the front's product backlog during week 3

Before starting the first sprint, all user stories were written and put in the *BACKLOG* list. Then, the scrum master and product owner of each subteam decide which user stories are going to be develop. The two subteams communicate to agree on a main functionality to implement. For example, if the back team decides to work on 3D visualisation, it is important that the front team

starts to work on that too. The two teams work differently: front team decided to put colour on every task to describe their progress - red when it is not started yet, green when it was finished; back team decided to estimate the progress of each user story at the end of the sprint, using Fibonacci sequence - 8/13 means that the task is almost finish.

2.2.2 Meetings with the client

One principle of agile methodology is to interact frequently with the client. To respect this, product owners had weekly meetings with the client. During these meetings, product owners shown the project progress to the client, discuss about client's needs and talked about encountered obstacles. They took place at each sprint beginning so the team could adapt the sprint depending on client's new needs. They were important to validate application progress, and to prevent misunderstandings.

2.2.3 Daily meetings and Sprint Review

All project long, we had two team meetings per day, and at the end of each sprint, a Sprint Review. The first meeting, the **Daily Stand Up**, was at the beginning of the day and its goal was to say what one was planning to do for the day or what one was troubled about. It could be really quick, especially when no user stories were about to be finished. After this meeting, the day could really starts, and everyone knew what one was doing during the day.

At the end of day, an other meeting called **Retrospective** was made. During the retrospective, at the opposite of the daily stand up, one could say what they finally did during the day and if one did not progressed, they could say it out loud. To say out loud the problem encountered during the day is not always easy, but this effort was actually quite efficient, as some problems were solved after this meeting - at least ideas to solve the problem were given. These two meetings structured each day of work and gave a work routine to the team.

At the end of a sprint, we also did a **Sprint Review**. The goal of this review was to see what we succeed to do over the sprint and what we did not. Everyone participate to this meeting, and it was like a long retrospective.

3. Development and results

3.1 Development

3.1.1 Project planning

Because there was no idea of how far this project could go and how difficult some tasks would be, the project was not plan in advanced, or at least it was not formalised into a calendar. At the end of the project, it was possible to make one in order to see what we achieved. The project planning is represented on figure

3.1.2 Development retrospective

As agile methodology was used during this project, it is interesting to do a retrospective of development. Mainly, two important things need to be discussed. The first one is about tasks difficulties. Indeed, when the product backlog was created, each user story has to be assigned a grade, using Fibonacci sequence. The team did not have all necessary skills and the distance needed to assigned a "correct" grade to all user stories. For example, everyone knew that 3D visualisation would be complicated, so all tasks about 3D visualisation were important and difficult, so their weight - i.e. their grade - was high. However, to display 3D data in front-end, the task went quickly - less than 3 days - but this task had the highest weight of the two sprint backlogs - 21. This is just an example, but it was important to take that into account during the development, because when this happens, it is easy to have someone without any tasks for the last two days for example. The same thing also happened in the opposite side: a easy task turns out to be quite difficult and had to involve several person instead of one.

The second thing was the separation between front-end and back-end. At the beginning of the project, there were no problems to divide the team between front and back, with 1 person in the back team that could help the front team if needed. But when main user stories started to be finished one by one, the limit between the two teams was thin and some people started to switch teams. They did not literally change of team, but tasks they were doing was sometimes more tasks of the other team. This can lead to problems because when it starts, it is difficult to know who deals with a specific task. However, it remains useful to be able to switch teams easily when needed. One can take advantage in that by working with someone else that can bring a fresh view of the problem. Most of the time, it solves problems.

3.2 Results

3.2.1 2D visualisation

PRESENT HERE 2D VISUALISATION
INCLUDE IMAGES OF VECTOR AND RASTER DATA
COMMENT THOSE RESULTS (WHAT IS GOOD / BAD ETC.)

3.2.2 3D visualisation

SAME FOR THIS SUBSECTION

4. Personal contribution to the project

In this part, every member of the team will present how their role - in agile methodology - contribute to the project.

- **Hannick ABDUL-KUTHOOS** (Developer):
- **Clovis BERGERET** (Product Owner for week 2):
- **Romain COURET** (Developer):
- **Axel DUMONT** (Product Owner for week 1):
- **Claire GIRARDIN** (Product Owner from week 3 to 4):
- **Vincent GIUDICELLI** (Developer):
- **Claire GUERRINI** (Developer):
- **Mathéo MARÉCHAL** (Developer):
- **Victorien OLLIVER** (Product Owner for week 1):
- **Hicham OUTMRHOUST** (Developer):
- **Mathis ROUILLARD** (Scrum Master):
- **Cécile TALEC** (Developer):
- **Vittorio TOFFOLUTTI** (Scrum Master):
- **Frédéric YE** (Developer):
- **Lisa ZARATIN** (Product Owner from week 2 to 4):

Conclusion

In only 4 week, the team was able to answer the client's needs - adapted during the project - and surpassed the difficulties encountered. Agile methodology has been really useful to develop the application and to be efficient while working under good conditions. The team tried its best to apply agile methodology during all the project, but it was the first time for everyone that we worked with 15 persons. Just for that, this project made every member of the team to learn a lot.

For the application, there are a lot of things that could be done in the future if the project had been longer. For example, basic treatments on layers could have been implemented for 2D or 3D visualisation. Also, basic changes of layers symbology could be added to the application, especially for 2D visualisation. It is easy to think of tasks to implement when creating a GIS system, but the team did what they could do in 4 weeks only.