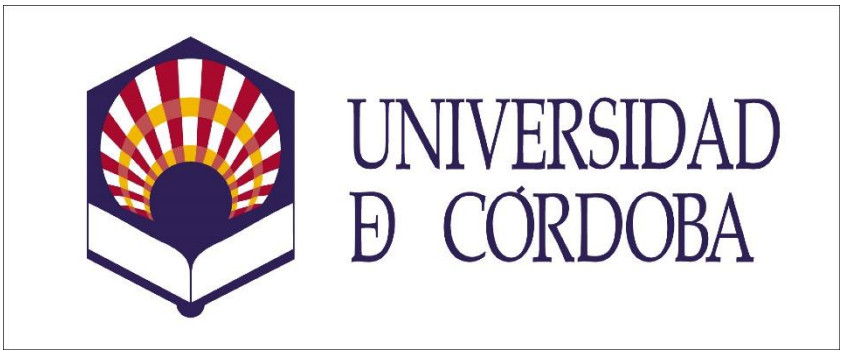


SOFTWARE ENGINEERING GROUP 5 DOCUMENTATION

INDEX

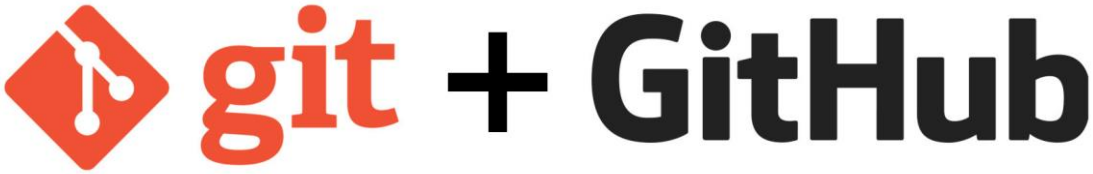
Group Members:

Francisco Javier Molina Prieto

Álvaro Prieto Barón

Antonio Muñoz Cubero

PRACTICE 1: INTRODUCTION TO "GIT & GITHUB"



In the first practice we had to do a tutorial on how to use "Git", to be able to start working with this tool, because it makes it very easy to work together with a team where each one is in a different place.

After knowing the basic use of "Git", we started to make a new tutorial this time using GitHub, this already allowed us to have a common repository where to house our code files or documentation and greatly expedited our work as a team.

We also had to make a little effort and learn how to use the documentation in "Markdown", as this will allow us later to make several files where we explain our code or different tools we have used in the development of our program.

GIT TUTORIAL

Visit the website.

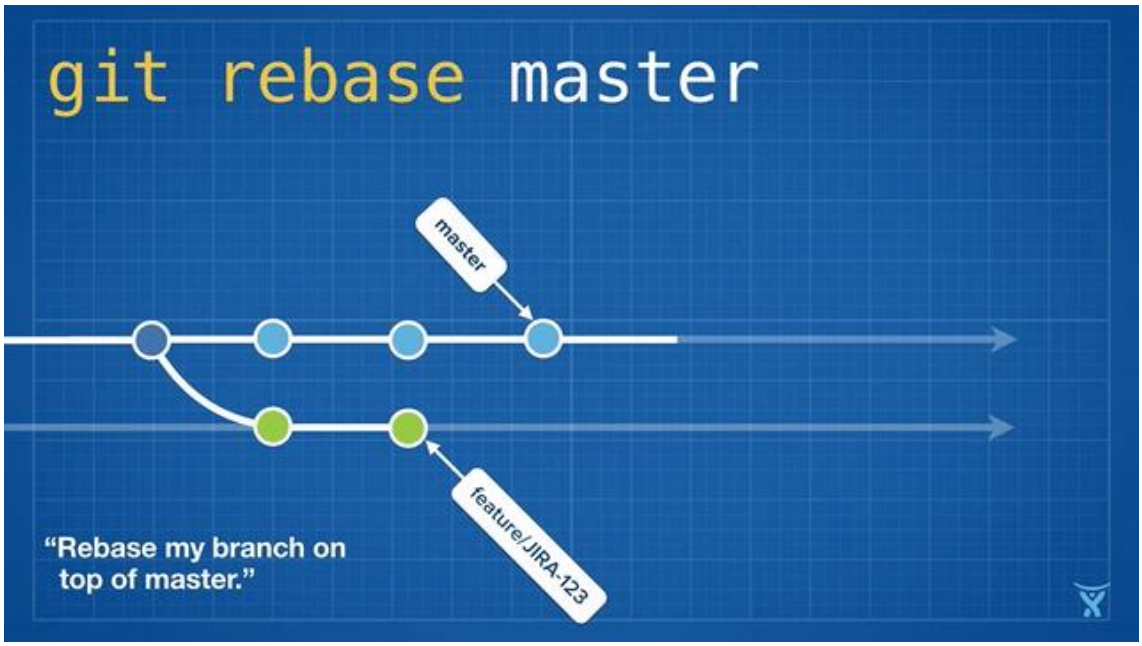
Git is a version control system for **tracking changes** in computer files and coordinating work on those files among multiple people.

In this guide, we will teach you the essential knowledge of git a how to work with it.

TABLE OF CONTENTS

Some advantages of using Git

- The ability to do and undo changes.
- History and documentations of different changes.
- Using of multiple versions of the same code at once.
- The ability to solve conflicts between version of different programmers.



How to configure Git

First, we will set the administrator name, email, core editor, interface, etc.

1. Change *Administrator name*

```
git config --global user.name "Admin Name"
```

2. Change *Email*

```
git config --global user.email name@mail.com
```

3. Choose your core *text editor*.

```
git config --global core.editor "atom"
```

4. Set the interface color

```
git config --global color.ui true
```

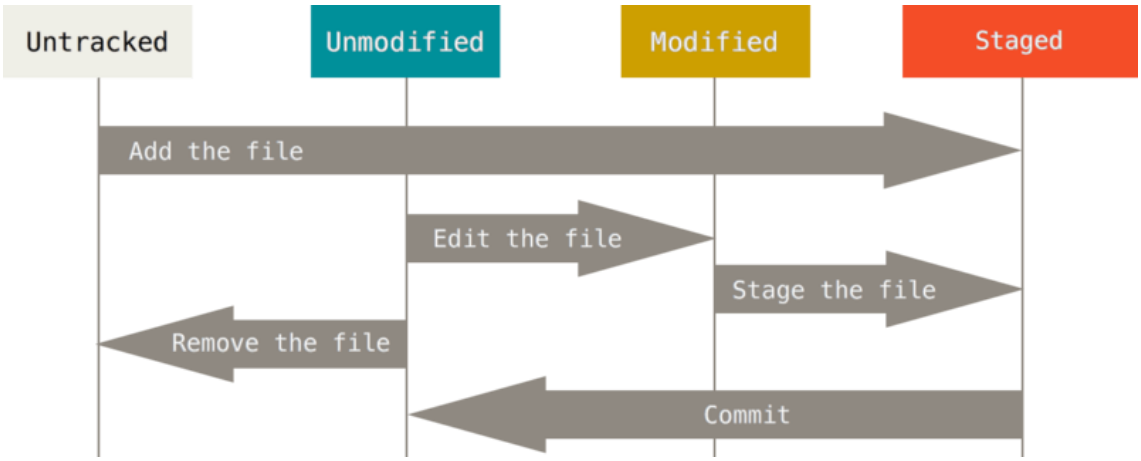
4. Show the configuration list

```
git config –list
```

The three states of Git

Typically, you'll want to start making changes and **committing** snapshots of those changes into your *repository* each time the project reaches a state you want to record.

Remember that each file in your working directory can be in one of two states: **tracked or untracked**. Tracked files are files that were in the last snapshot; they can be unmodified, modified, or staged. In short, tracked files are files that Git knows about.



Untracked files are everything else – any files in your working directory that were not in your last snapshot and are not in your staging area. When you first **clone** a repository, all your files will be tracked and unmodified because Git just checked them out and you haven't edited anything.

As you edit files, Git sees them as *modified*, because you've changed them since your last commit. As you work, you selectively stage these modified files and then commit all those **staged changes**, and the cycle repeats.

Check our Git tutorial to learn more about basic Git commands.

GIT & GITHUB

GitHub Inc. is a web-based hosting service for version control using Git. That's it, Git is the software that allows you to **track and host** versions of files on GitHub. In other words, you use commands of Git to track versions of your files. And GitHub is just a remote platform where these files are hosted.

If you want to know more about GitHub, you can visit our GitHub tutorial and give us some feedback.

GITHUB TUTORIAL

Visit the website.

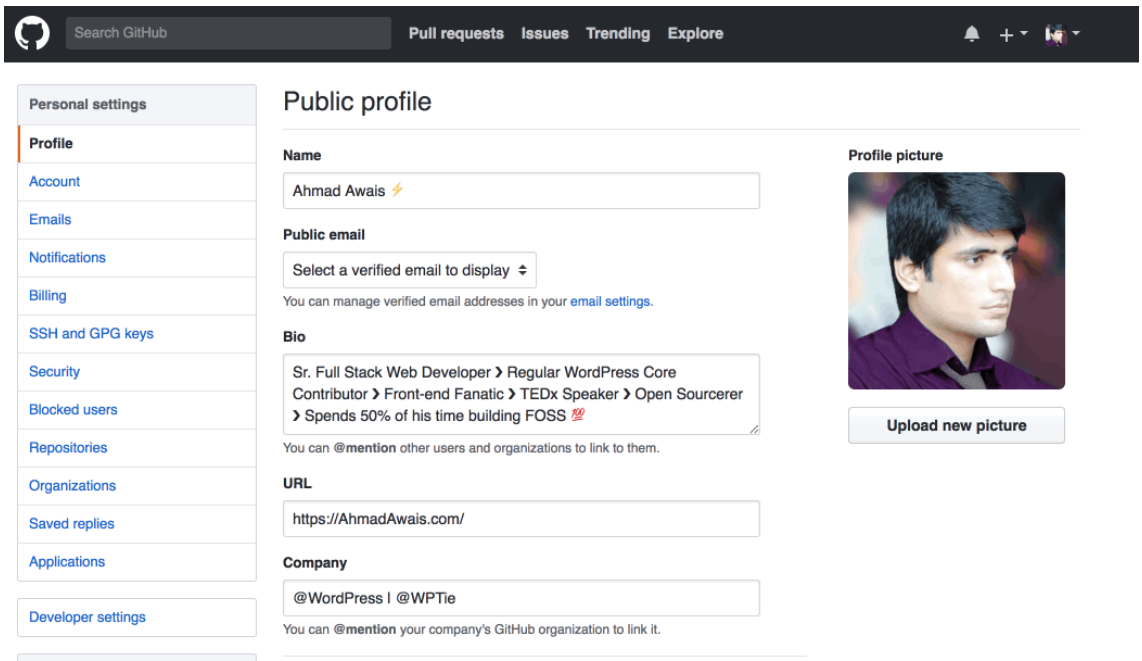
GitHub is a code hosting platform for version control and collaboration using **Git**. It lets you and others work together on projects from anywhere, so it's commonly used in **Open Source projects** and it has its own community.

This tutorial teaches you GitHub essentials like repositories, branches and commits.

STARTING IN GITHUB

To complete this tutorial, you need a GitHub.com account and Internet access. You don't need to know how to code, use the command line, or install Git (the version control software GitHub is built on). To install Git, you can follow our Git Tutorial.

When you create your first account, you should visit the **Settings** section on top of the page.



Step 1. Create a repository

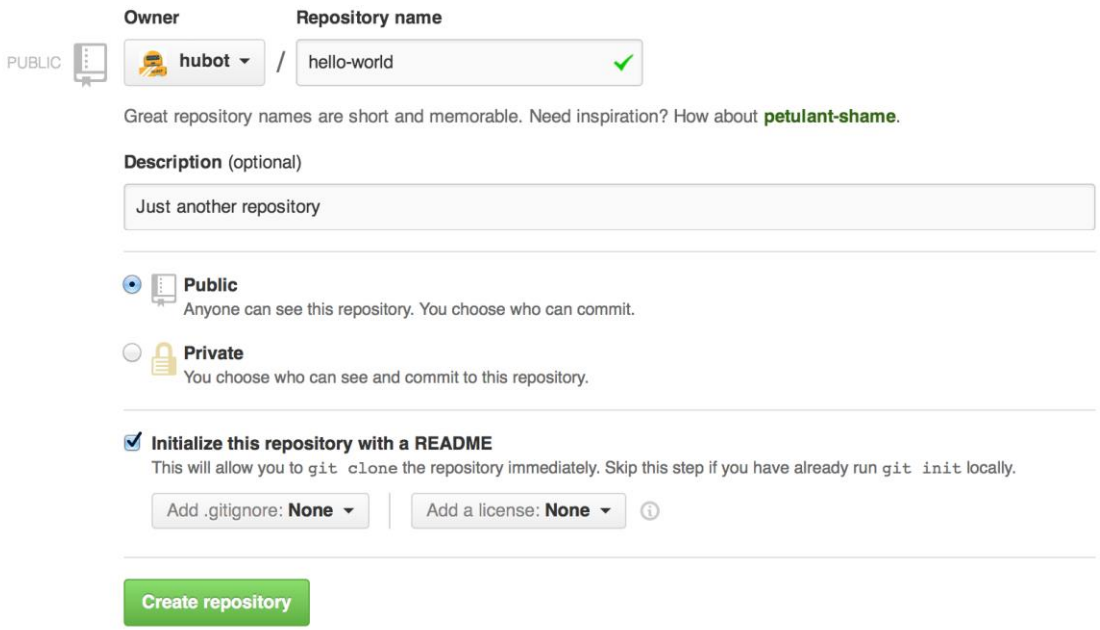
A repository is usually used to organize a single project. Repositories can contain **folders and files**, images, videos, spreadsheets, and data sets – anything your project needs. You can also ignore some files to be in your repository if you want to keep them private. Take a look at our Git Tutorial and look for the *.gitignore* command.

We recommend including a **README.md**, or a file with information about your project, as we did here. GitHub makes it easy to add one at the same time you create your new repository, but you can also create it using your source code editor. You can also add a **license file** in your repository to make it easier for other people to contribute.

Your repository can be a place where you store ideas, resources, or even share and discuss things with others and cowork with them.

To create a new repository:

- In the upper right corner, next to your avatar or identicon, click and then select **New repository**.
- Name your repository.
- **(OPTIONAL)** Write a short description.
- Select **Initialize this repository with a README**.
- Click **Create repository**.



Step 2. Create a branch

Branching is the way to work on different versions of a repository at one time.

By default, your repository has one branch named master which is the definitive branch. We use branches to experiment and make edits before committing them to master.

If you want to know more about branches, we recommend you to visit our branches tutorial

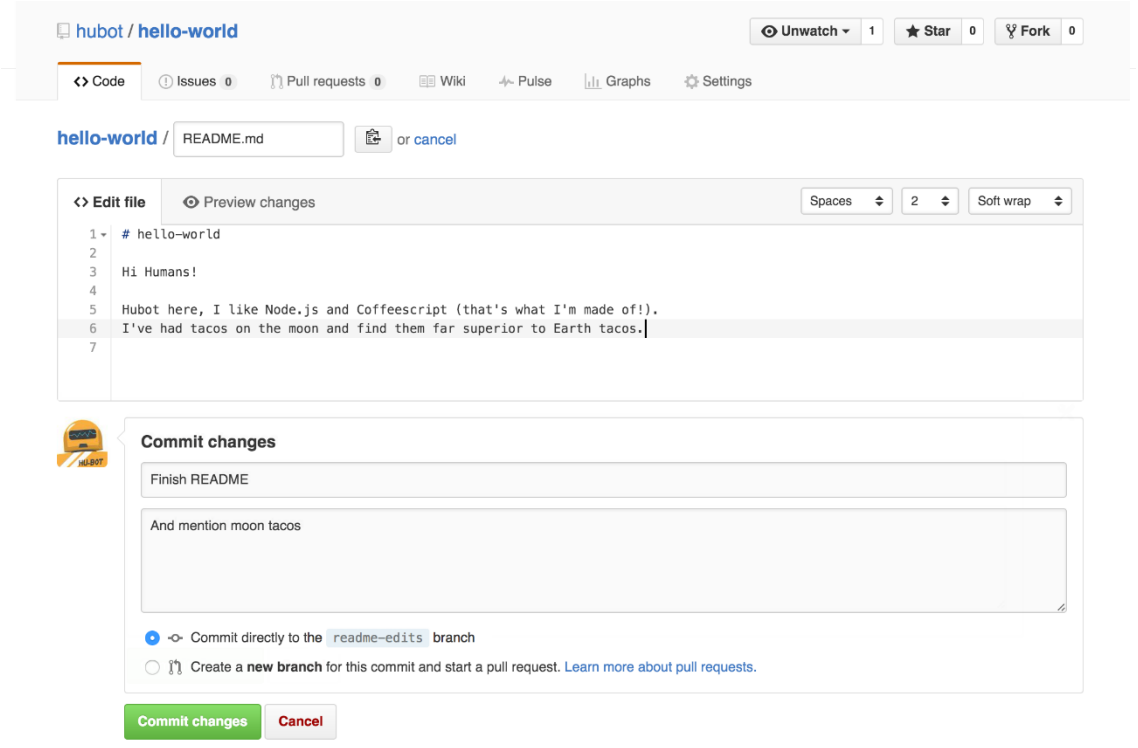
Step 3. Make and commit changes

On GitHub, saved changes are called **commits**. Each commit has an associated *commit message*, which is a description explaining why a change was made. Commit messages capture the history of your changes, so other contributors can understand what you've done and why.

We think commits are one of the most important things in a GitHub repository timeline because you can always look at its commits and see where exactly something was modified. That allows us to prevent our project to fail.

Make and commit changes

Click the **README.md** file. Click the pencil icon in the upper right corner of the file view to edit. You can write whatever you want in the editor. Write a commit message that describes your changes. Click **Commit changes** button.



These changes will be made to just the **README.md** file on your branch, so now this branch contains content that's different from master.

CONGRATULATIONS!

By completing this tutorial, you've learned to **create a project** and make a pull request on GitHub!

Here's what you accomplished in this tutorial:

- Looked at your GitHub profile
- Created an open source repository
- Started and managed a new branch
- Changed a file and committed those changes to GitHub

Finally, you might visit GitHub Explore, look at a bunch of Open Source projects and... maybe get involved in one of them.

PRACTICE 2: REQUIREMENTS ANALYSIS:

In our second practice, we had to create a software that manages a student database. This software would be used by the teacher to manage the student's teams in the practice classes.

EXTRACTION OF REQUIREMENTS

Functional

- *Application that manages the student data, and allows the professor to:*

➔ Insert a student: ID 000

Required Data:

- ID
- Name
- Surname

Optional Data:

- Group Number
- Is leader of group?
- Phone Number
- Corporative Email
- Address
- Highest Registered Course
- Birth Date

Constraints:

- The number of students is limited to 150.
- A group can't have more than one leader.
- If any of this would happen, the system throws an error and doesn't insert the student.

➔ Search for a student: ID 001

- By surname (ask for ID if there are more than one)
- By ID
- If the student doesn't exist, the system asks for another one or to exit.

➔ Delete a student: ID 002a

- Search for a student (like in 001) and delete it.

➔ Delete all students: ID 002b

- Clean the database completely

➔ Edit a student's data. ID 003

- Search for a student (001), ask for the new data, and override the old data. A group's leader can't be changed this way.

➔ Edit a group's leader. ID 004

➔ Show all students: ID 005

- The system will generate a Markdown/HTML file that contains the data of all the students.
- This file can be ordered alphabetically by name or surname, by ID or by highest course, both ascending and descending.

➔ Import data from backup: ID 006

- The system will load the data from a binary file, like the ones created with the back up.

➔ Create a backup: ID 007

- Only the coordinator can do it.
- Creates a binary file containing the data from all the students.

Non-Functional

- Written in C++
- Works in Linux-based systems.
- Simplicity
- Optional GUI

USER CASES:

Insert Student

ID: 000

Brief description: The function lets you introduce a student in the list.

Main Actor: The user.

Secondary Actor: The student.

Preconditions:

1. You must insert the *DNI, NAME, SURNAME, ETC.*
2. Optionally, I want to insert: *USER'S TEAM NUMBER, IF IS THE LEADER OR NOT, PHONE NUMBER, E-MAIL, POSTAL CODE, HIGHEST COURSE THE USER IS REGISTERED, BIRTH DATE.*

Main flow:

1. The user case starts when the User want to insert an student.
2. The teacher introduces the student's data.
3. The system checks if the student exists.

Postconditions: * The student is inserted in the list.

Alternative flow:

a. If there are already 150 students, an error message is shown and the case finishes.
a. If you don't fill all the obligatory fields, the student will not be saved in the list.
b. If the student is the leader of a group that already has a leader, the system asks the user who will remain as leader.
a. If the student already existed (checking by ID), it is not inserted.

Student Search

ID: 001

Brief description: The function let you search a student in the list.

Main Actor: The user.

Secondary Actor: The student.

Preconditions:

1. You must introduce the *SURNAME or the ID*.

Main flow:

1. The user case starts when the user wants to search an student.
2. The teacher introduces the student's data.
3. The system shows the student.

Alternative flow:

1. If the student doesn't exist, the system shows an error

Remove a student

ID: 002

Brief description: The function deletes a student.

Main actors: The user

Secondary actors: The student

Preconditions: 1. The user must exist.

Main flow: 1. The case starts when the User needs to delete a student. 2. The system asks if the user wants to delete from the database. 3. The system ask for a Surname (ID in case of conflict). 4. The system remove the student from the database.

Postconditions: * The student is deleted. * If the student was the leader of the team, the group stays with no leader.

Alternative flow:

a. If the student doesn't exist, the system shows an error message and asks for the student's ID/Surname again.

Remove all students

ID: 002

Brief description: The function deletes all students.

Main actors: The user

Secondary actors: The student

Preconditions: * None

Main flow: 1. The case starts when the User wants to empty the database. 2. The system asks for confirmation. 3. The system deletes all the students from the database.

Postconditions: * The database is empty.

Alternative flow:

1. If the user doesn't confirm the action, the case ends.
2. If the database is already empty shows an error.

Modify student

ID: 003

Brief description: The function modifies a student.

Main actors: The user

Secondary actors: The student

Preconditions: 1. The user must exist.

Main flow: 1. The study case starts when the User needs to modify a student. 2. The system asks for a Surname (ID in case of conflict). 3. The system asks the user the parameters to change. 4. The system saves the data.

Postconditions: * The student is modified.

Alternative flow:

a. If the student doesn't exist, the system shows an error message and asks for the student DNI/Surname again.
a. If any parameter is not valid, the system shows an error message and asks for the parameter again.
b. If the student is the leader of a group that already has a leader, the system asks the user who will remain as leader.

Modify Team Leader

ID: 004

Brief description: The function modifies a team's leader.

Main actors: The user

Secondary actors: The student

Preconditions:

1. The team must exist.

Main flow:

1. The study case starts when the system needs to modify a student.
2. The system asks for the team's number.
3. The system modifies the role of the team's leader.
4. The system asks the user who will be the team's leader of the team.
5. The system modifies the new team's leader role.
6. The system saves the data.

Postconditions:

- The team's members are modified.

Alternative flow:

3. a. If the team doesn't have a leader the system does nothing.
4. a. If the student doesn't exist, the system shows an error message and ask for the student DNI/Surname again.

Show Students

ID: 005

Brief description: Generates a file with a sorted list of all the students in the system.

Main actor: User.

Secondary actor: Students.

Preconditions:

1. There must be at least 1 student in the system.

Main flow:

1. The case starts when the user wants to generate a list of all the students.
2. The system asks whether the user wants a markdown or a HTML file.
3. The system asks the type of sort.
4. The system gathers all data from the students and sorts it.

Postconditions:

- A markdown/HTML file is generated.

Import Data

ID: 006

Brief description: Imports a binary file as a list of students.

Main actor: User.

Secondary actor: Students.

Preconditions: None.

Main flow:

1. The case starts when the user wants to import a list of students.
2. The system deletes all the students it has been using.
3. The system asks the user for the file name, check if it exists.

Postconditions:

- The system reads the file and generates a list of students.

Alternative flow: 3. a. If the file doesn't exist, the system prints an error message.

Save Data

ID: 007

Brief description: Generates a binary file with all the students in the system.

Main actor: User (Only coordinator).

Secondary actor: Students.

Preconditions:

1. There must be at least 1 student in the system.

Main flow:

1. The case starts when the user wants to save the data they have.
2. The system asks the name of the file, checks if it exists.
3. The system gathers all data from the students.

Postconditions:

- If the file exists, it is overwritten with the new data.

Alternative flow:

- 2. a. If the file doesn't exist, the system creates it before writing.

Add Teacher

ID: 008

Brief description: Adds a new user to the system.

Main actor: User (Only coordinator).

Secondary actor: Teacher.

Preconditions:

- None

Main flow:

1. The case starts when the user wants to create another user.
2. The system asks the username and password.
3. The system encrypts the password.
4. The system saves the data.

Postconditions:

- A new user is added to the system.

Alternative flow:

- 2.a. If the username already exists, the system prints an error message.

Delete Teacher

ID: 009

Brief description: Removes a user from the system.

Main actor: User (Only coordinator).

Secondary actor: Teacher.

Preconditions:

1. There must be at least 1 non-coordinator user in the system.

Main flow:

1. The case starts when the user wants to remove another user.
2. The system asks the name of the user.
3. The system deletes the data from the user database.

Postconditions:

- A user is removed.

Alternative flow:

- 2.a. If the user doesn't exist, the system prints an error message.

Login

ID: 010

Brief description: Asks for and checks credentials before letting a user access the system.

Main Actor: The user.

Secondary Actor: None.

Preconditions: None.

Main flow:

1. The user case starts when the user wants to use the system.
2. The user introduces username and password.
3. The system looks for the username in the user database.
4. The system encrypts the given password and compares it with the stored one.
5. The system grants access to the user.

Alternative flow:

- 3.a. If the username doesn't exist, the system prints an 'incorrect data' message.
- 4.a. If the password doesn't match, the system prints an 'incorrect data' message.

USER STORIES

ID: 000 Insert Student

As a user I will have to be able to insert students into a database of students that I will manage later.

Priority: 1

- I want to insert the following user data:
  - *ID*
  - *Name*
  - *Surname*
  - *Phone number*
  - *e-Mail*
  - *Postal Code*
  - *Highest course the user is registered*
  - *Birth date*
- Optionally, I want to know:
  - *The user's team number*
  - *If the user is the leader of the team*

ID: 001 Student Search

I will be able to search for any student registered in the student database.

Priority: 1

- I want to look for a student by his:
  - *Surname* (In case of conflict, ask for the *ID*)
  - *ID*
  - If no user matches the search, ask for the parameters again

---

ID: 002.A Remove A Student

As a teacher, I want to delete a student from my *list*.

Priority: 2

- I want to search a student for *ID*.
- I want to search a student for *surname*, in case of conflict with the *ID*.
- If you *remove the leader* of a team, this team stay without leader until the teacher asign another one.
- I want to *delete all* the student.

ID: 002.B Remove All Student

As a teacher, I want to delete all students from my *list*.

Priority: 2

- I want to *delete all* the students from the database.

ID: 003 Modify A Student

As a teacher, I want to modify the data of a student from my *list*.

Priority: 2

- I want to search a student for *ID*.
- I want to search a student for *surname*, in case of conflict with the *ID*.
- I want to *modify any date* of a student

ID: 004 Modify The Leader Of A Team

As a teacher, I want to modify the leader of a Team from my *list*.

Priority: 1

- I want to search a student by *ID*.
- I want to search a student by *surname*, in case of conflict with the *ID*.
- I want to modify the leader of a Team, *asign the leadership* to another student of the Team.

ID: 005 Show Students

The user should be able to visualize all the students and their info.

Priority: 2

- It will generate an HTML or Markdown file for all the students.
- It should offer the possibility to sort by Name, Surname, ID or highest year, ascending and descending.

Id: 006 Import Data

The user wants to be able to import the data from a binary file.

Priority: 2

- All the data that has been in use before will disappear.
- The file name will be asked and introduced by keyboard.

ID: 007 Save Data

The user wants to be able to save the data from a binary file.

Priority: 2

- The file will be created if it doesn't exist.
- The file will be overwritten if it already exists.
- The file name will be asked and introduced by keyboard.

ID: 008 Add Teacher

The coordinator must be able to add other users.

Priority: 3

- The system will ask for the data and credentials of the new user.
- The password won't be saved, instead, an encrypted version will be stored.

ID: 009 Delete Teacher

The coordinator wants to be able to delete another teacher from the system.

Priority: 2

- The teacher will be chosen by ID.
- If it doesn't exist, the system prints an error message.

ID: 010 Login

The users must enter their credentials before being able to use the system.

Priority: 0

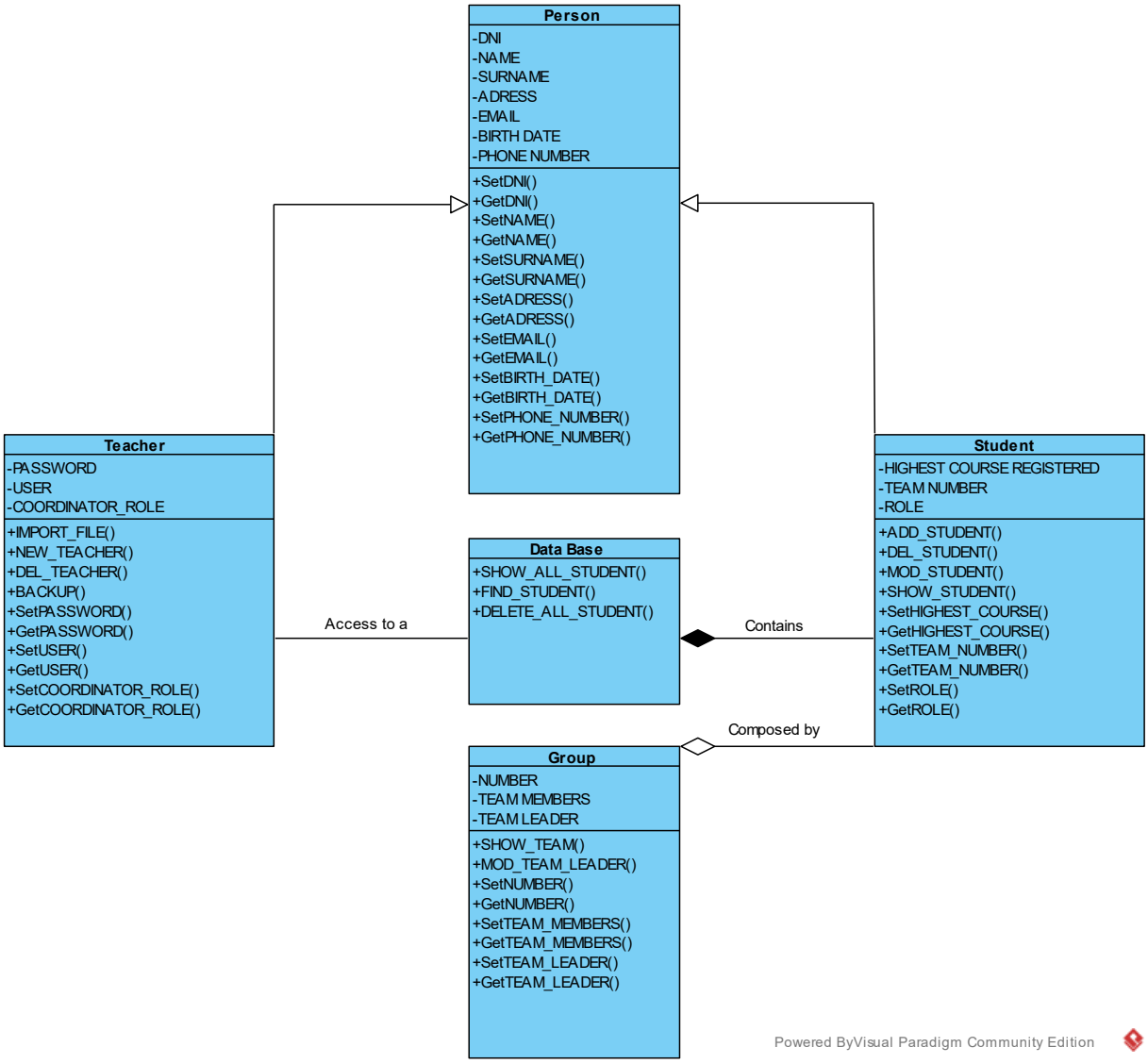
- The system will ask for the credentials of the user.
- The system will check if they are correct.

PRACTICE 3: SYSTEM DESIGN

In our third practice, we learned about Class Diagrams and Sequence Diagrams, two types of UML structure diagram. We had to make our own class diagram and a sequence diagrams for each user case. Also, we did a brief explanation of these diagrams in Markdown.

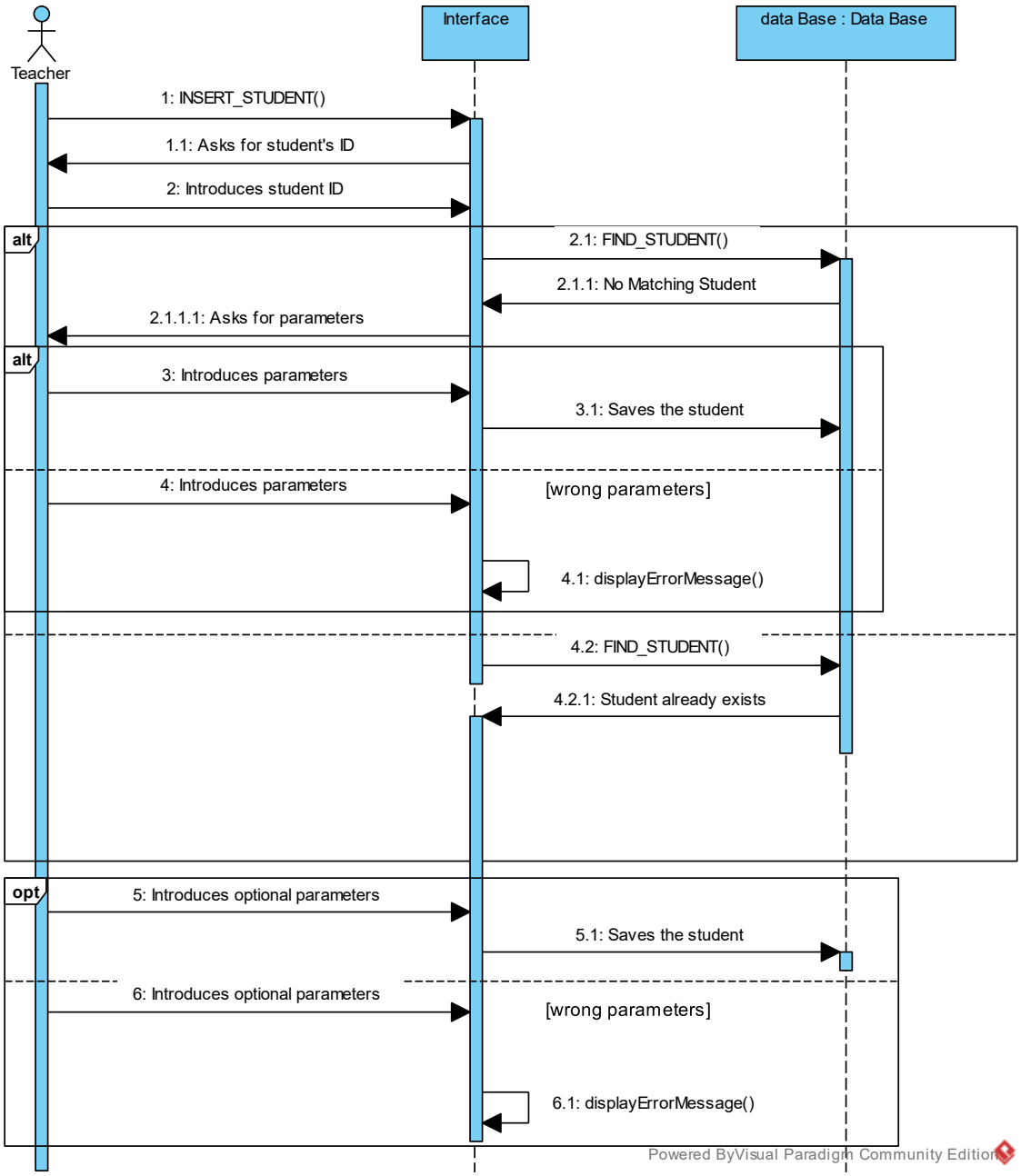
TABLE OF CONTENTS

CLASS DIAGRAM

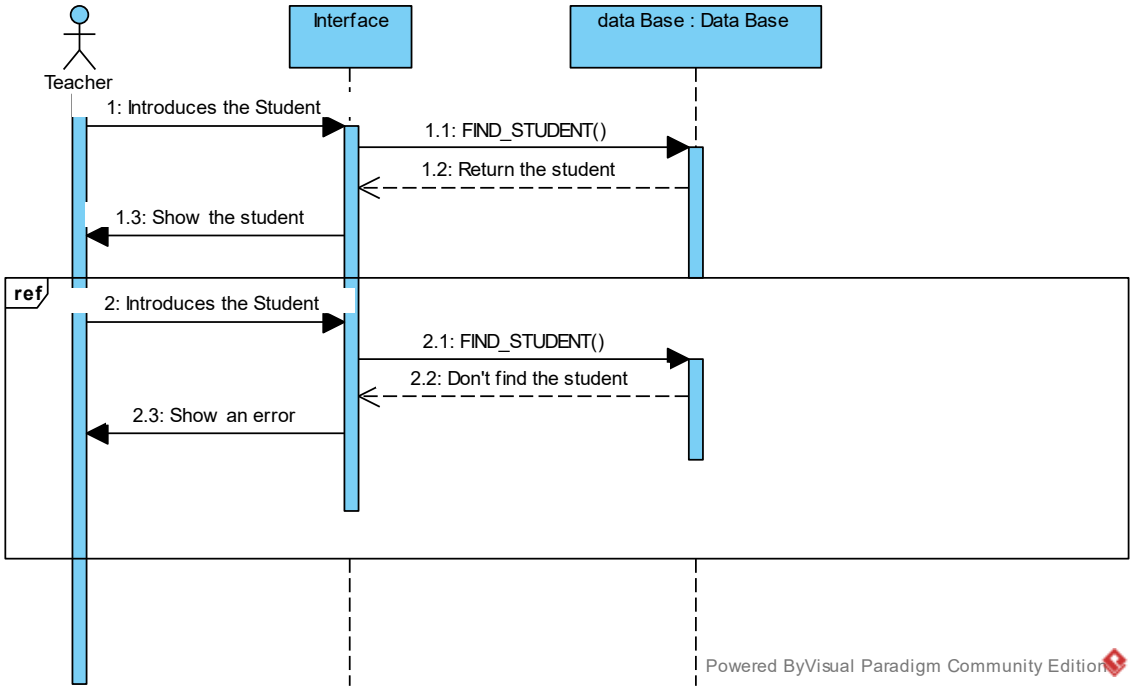


SEQUENCE DIAGRAMS

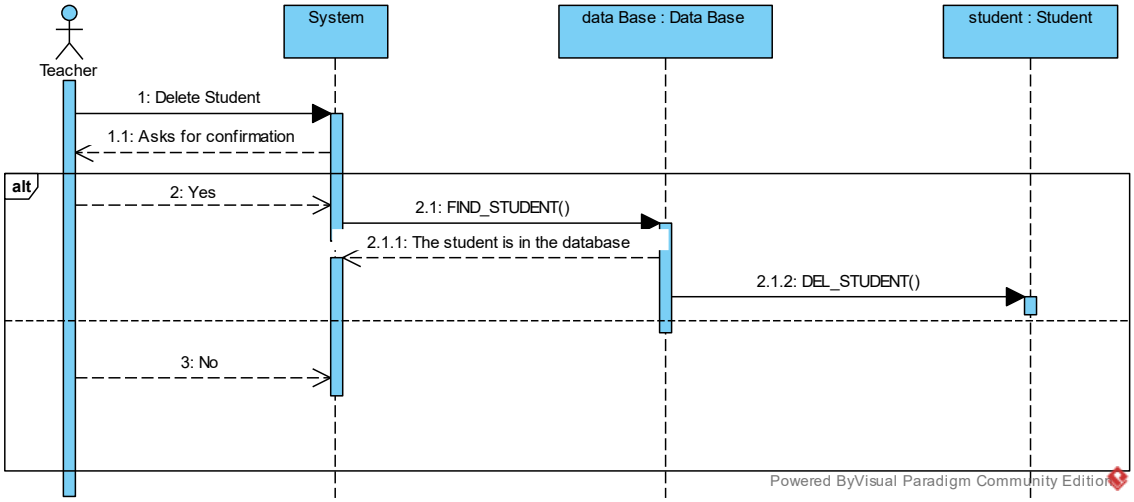
Insert Student



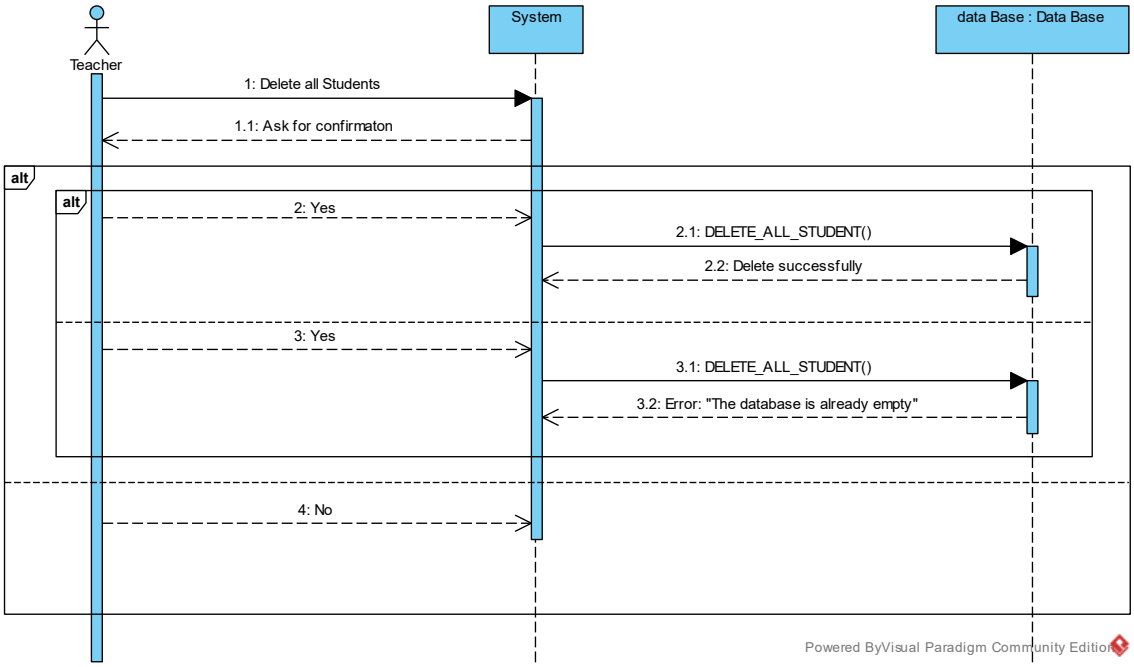
Student Search



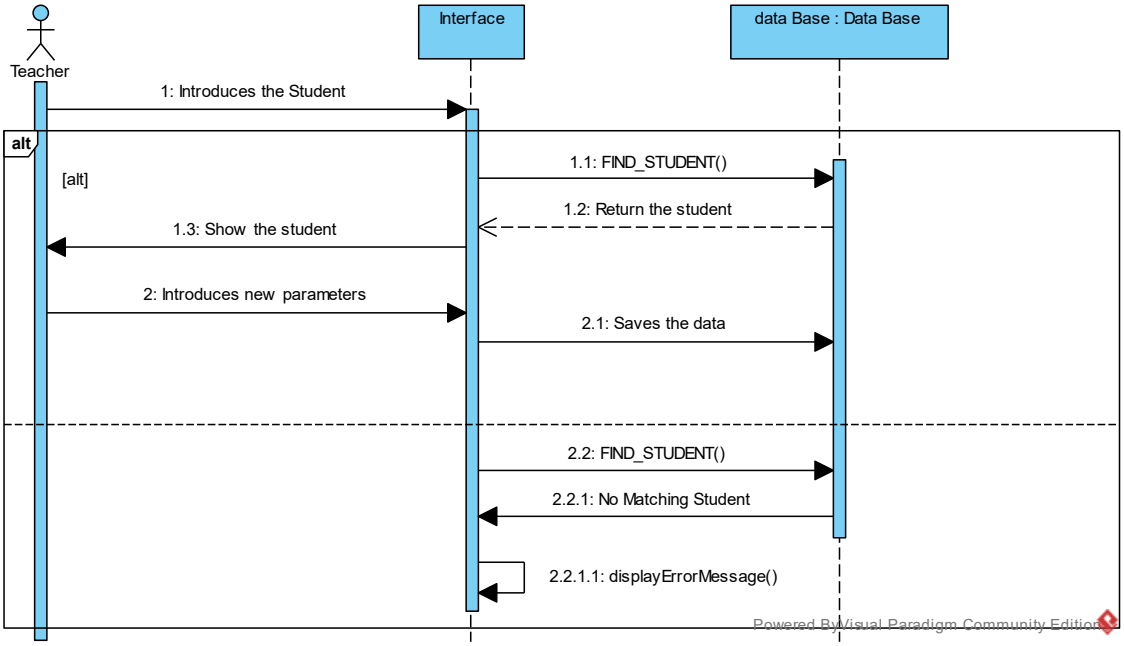
Remove Student



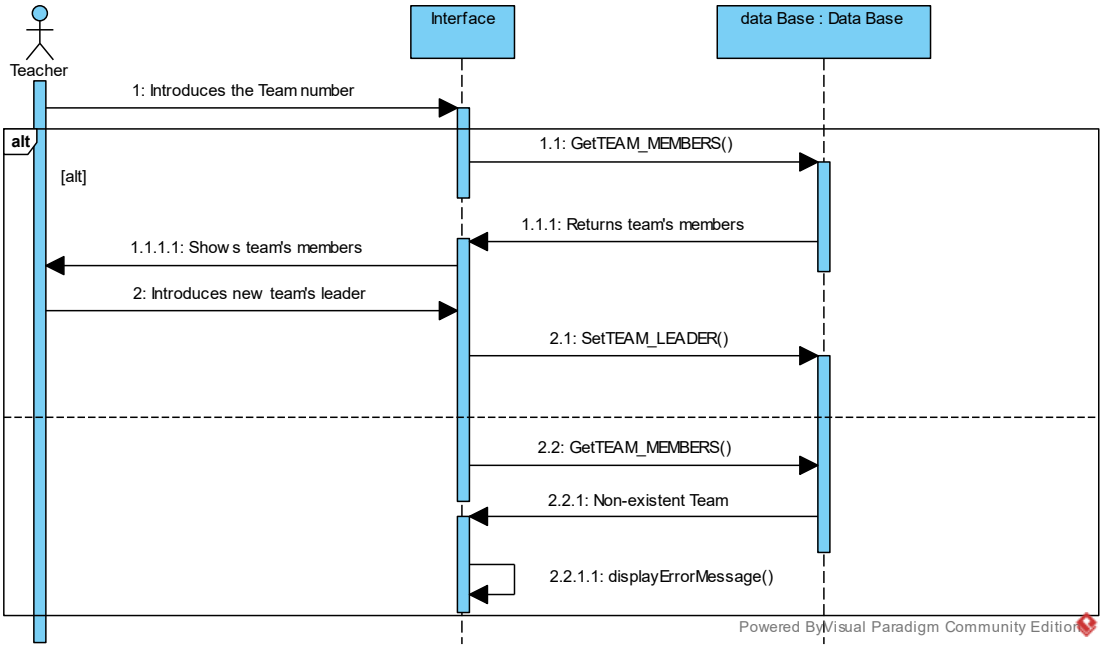
Remove all Students



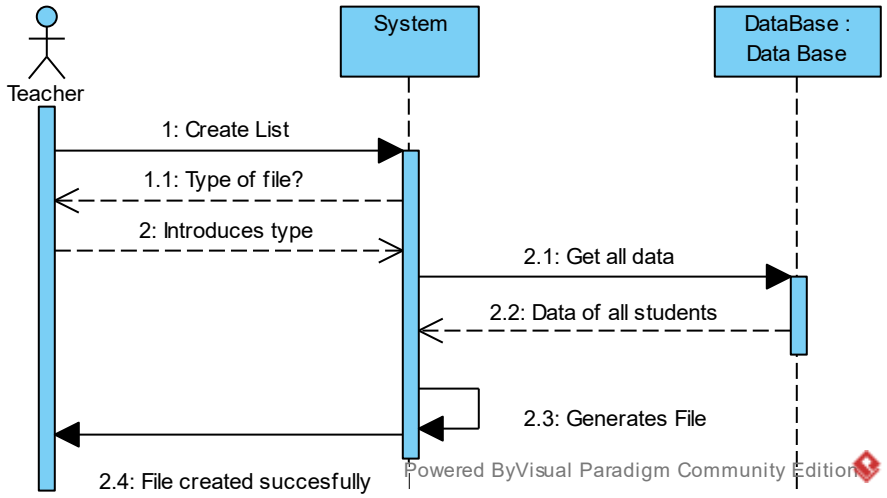
Modify a Student



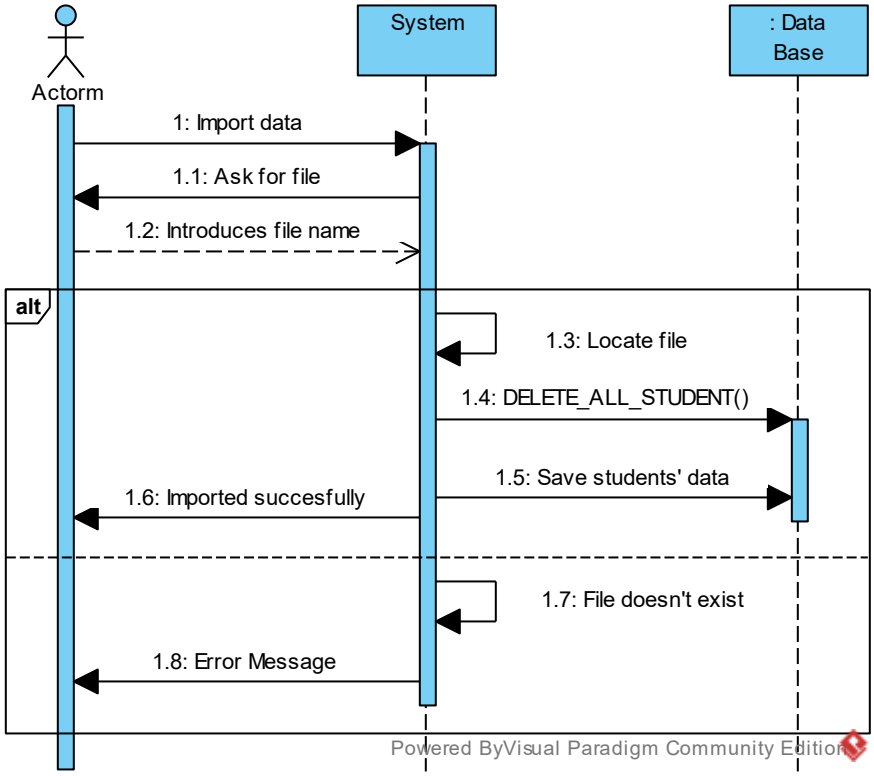
Modify the leader of a Team



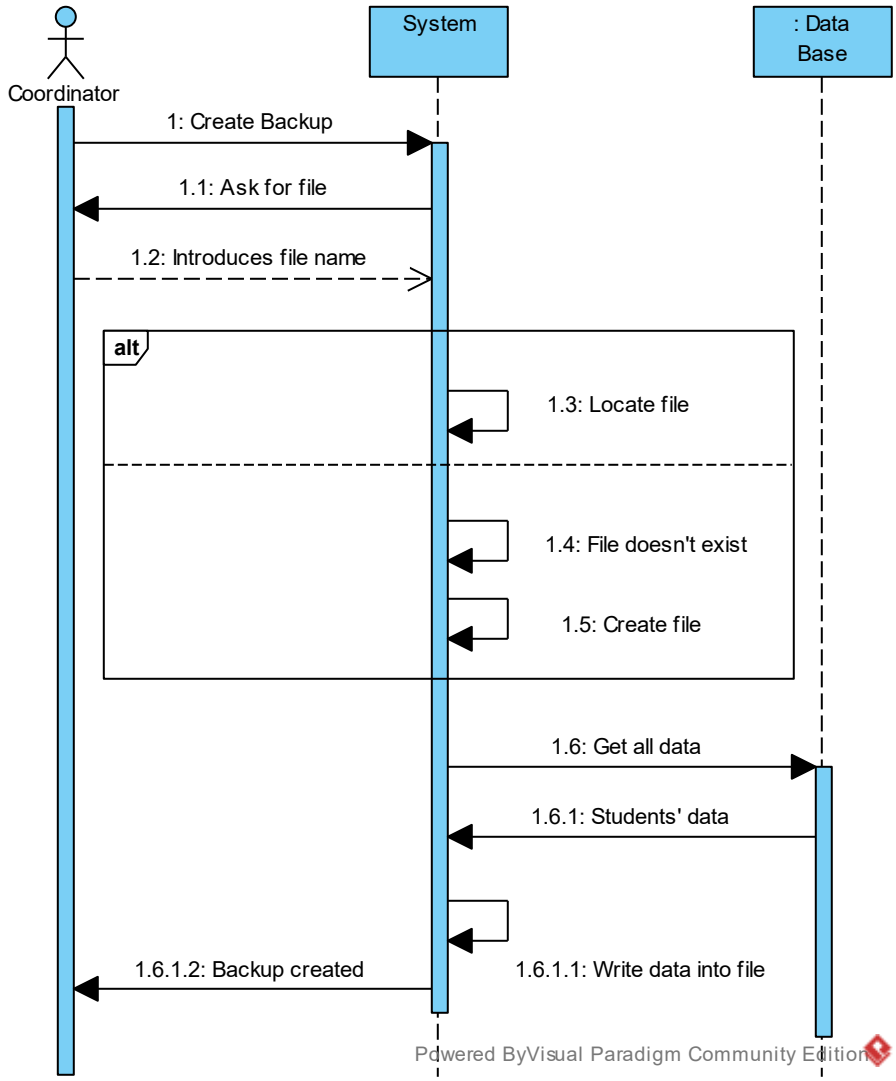
Show Student



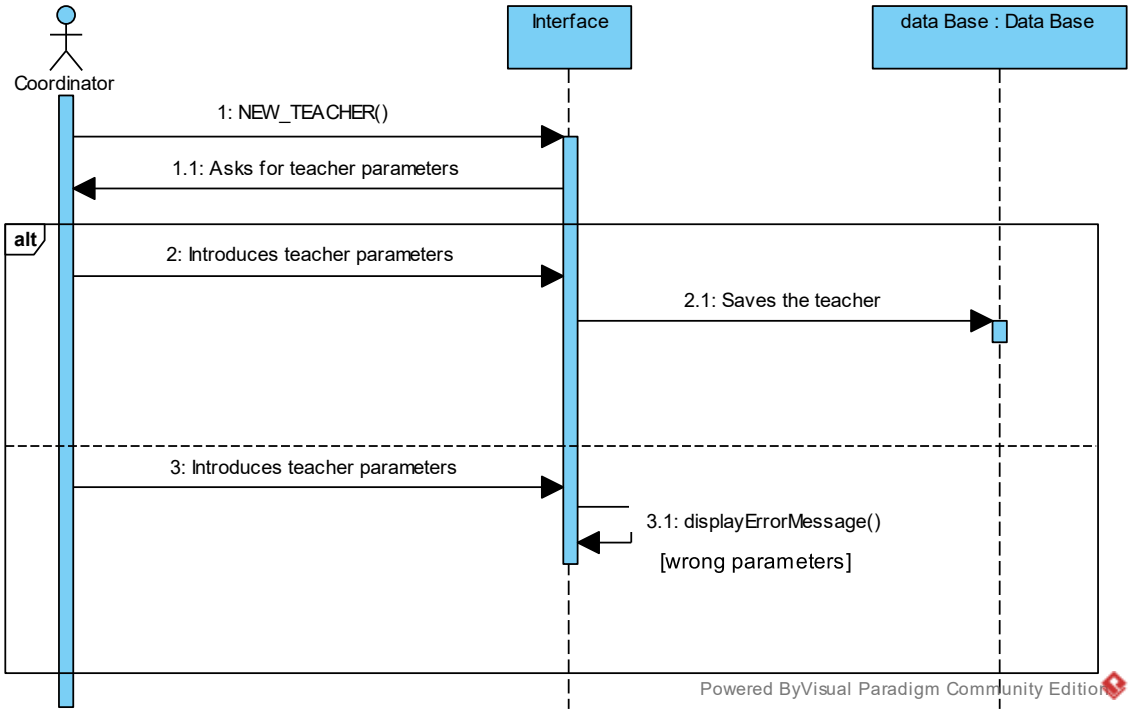
Import Data



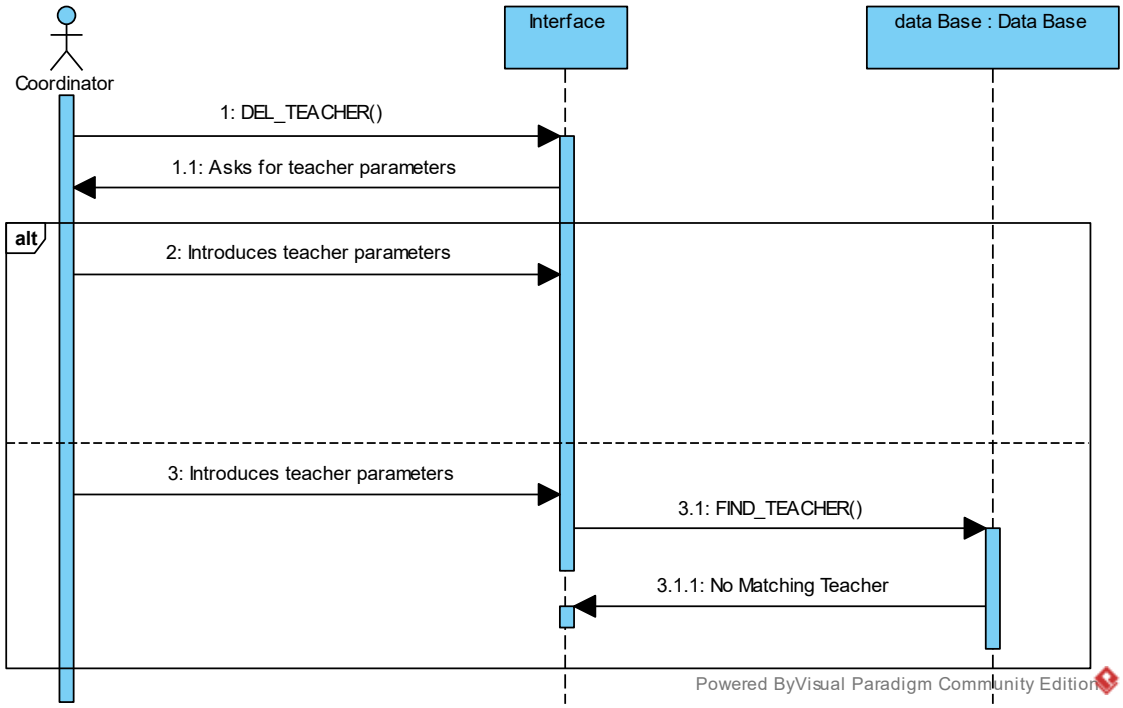
Save Data


Coordinator
System
: Data Base
1: Create Backup
1.1: Ask for file
1.2: Introduces file name
alt
1.3: Locate file
1.4: File doesn't exist
1.5: Create file
1.6: Get all data
1.6.1: Students' data
1.6.1.1: Write data into file
1.6.1.2: Backup created
Powered ByVisual Paradigm Community Edition

Add Teacher



Delete Teacher



PRACTICE 4: IMPLEMENTATION & TESTING

In our fourth and...sadly...last practice, we're going to work together as a Scrum Team (or at least we'll try) and finish our project, working on the implementation and testing of it.

WHAT IS SCRUM?

Visit the website.

Scrum is a **framework** within which people can address complex adaptive problems, while **productively and creatively** delivering products of the highest possible value.

Scrum itself is a simple framework for effective team collaboration on complex products. Scrum co-creators *Ken Schwaber* and *Jeff Sutherland* have written **The Scrum Guide** to explain Scrum clearly and succinctly. This Guide contains the definition of Scrum. This definition consists of Scrum's roles, events, artifacts, and the rules that bind them together.

Why Scrum?

Scrum is proposed for...

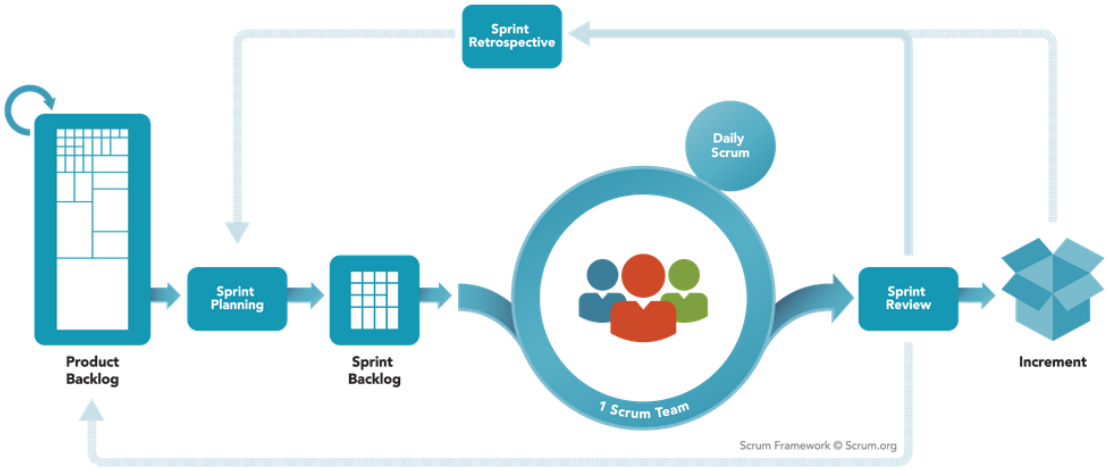
- Not wasting time.
- Starting the **quality** of the product at the start of the development.
- Creating **knowledge**.
- Take the **right decisions** at the right time.
- **Faster** delivery.
- Team **motivation**.
- Maximum **optimization** in all processes.

Scrum roles

- **Product Owner**: Sets the project action guidelines.
- **Scrum Master**: Guides the reunions and coordinates the team.
- **Team**: Implement the functionalities.
- **Users**: Final beneficers.

Scrum tools:

- **Product Backlog**: Contains all the functionalities ordered by priority. It's the group of the product user stories.
- **Sprint Backlog**: Functionality to develop in a specific sprint.
- **Burndown Chart**: Graphical representation of work left to do versus time.



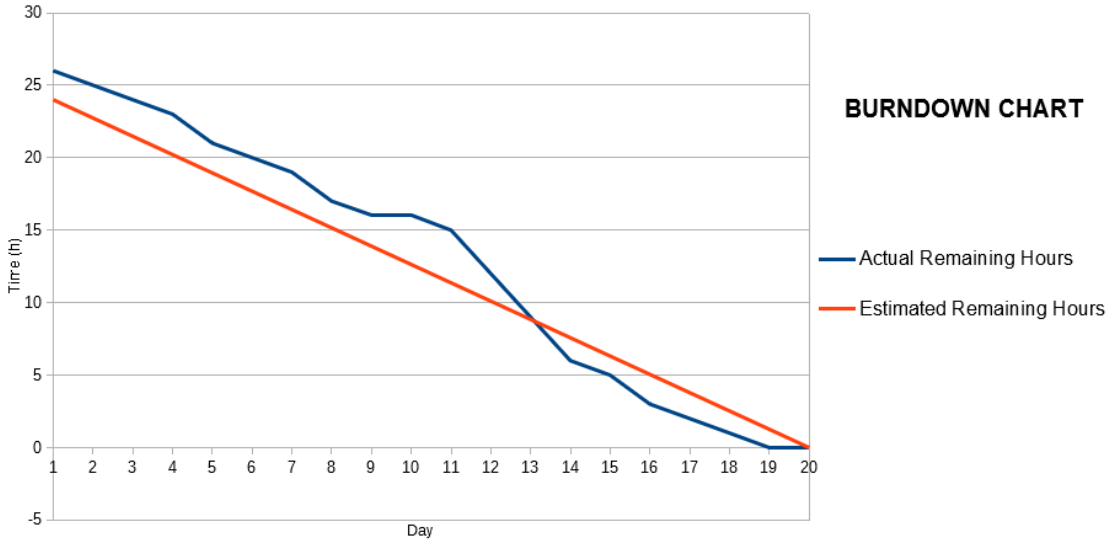
PRODUCT BACKLOG

ID	NAME	ESTIMATION	PRIORITY
	System Foundations	5	0
010	Login	2	0
001	Student Search	2	1
000	Insert Student	2	1
004	Modify Leader	2	2
002a	Remove Student	2	2
002b	Remove all students	1	2
003	Modify Student	3	2
005	Show Students	1	3
008	Add Teacher	1	3
009	Delete Teacher	1	3
006	Import Data	2	4
007	Save Data	2	4

SPRINT BACKLOG

SPRINT	ID	NAME	ASSIGNEES
1		System Foundations	@AdoenLunnae @RexusWolf @ErTonix12
1	010	Login	@AdoenLunnae
1	001	Student Search	@ErTonix12
1	000	Insert Student	@RexusWolf
2	004	Modify Leader	@ErTonix12
2	002a	Remove Student	@RexusWolf
2	002b	Remove all students	@RexusWolf
2	003	Modify Student	@RexusWolf
3	005	Show Students	@RexusWolf @AdoenLunnae
3	008	Add Teacher	@AdoenLunnae
3	009	Delete Teacher	@AdoenLunnae
4	006	Import Data	@RexusWolf @AdoenLunnae
4	007	Save Data	@AdoenLunnae

BURNDOWN CHART



VALIDATION MATRICES

Uses Cases/ Requirement	ID: 000	ID:001	ID:002.a	ID: 002.b	ID: 003	ID: 004	ID: 005	ID: 006	ID: 007	ID: 008	ID: 009	ID: 010
0	✓											
1		✓										
2.a			✓									
2.b				✓								
3					✓							
4						✓						
5							✓					
6								✓				
7									✓			

Requirements:

0. Insert a Student
1. Search a Student
2. A. Delete a Student
2. B. Delete all the Student
3. Edit Student's data
4. Edit group leader
5. Show all Student
6. Import data from backup
7. Create a backup