Bring ideas to life
VIA University College

Bob the Builder Construction Company

~ Project report ~

Matej Palas, 344682; Florina Mitigus, 344692;

Michał Barczuk, 344615; Naom Octavian Giovanni, 343812;

Supervisors: Line Lindhardt Egsgaard, Steffen Vissing Andersen, Richard Brooks

Number of characters: 97.150
Number of words: 14.744

Software Technology Engineering

1st Semester

13/11/2023

Table of contents

Abstract

The construction companies need a strategic method to manage the projects and promote their business, in order to improve the organizational efficiency. This paper presents the development of a project management system and a website that satisfy these requirements, while using files as back-up procedure and to update the information displayed on the website for real-time feedback on project progress. The results of this process is a clearer overview of ongoing projects and completed projects, the user having the option to store and manage projects's details and resources, as well as access reports about the finished projects. This outcome contributes to the overall success of the project management system, aligning with the initial objectives set for enhancing operational efficiency and effectiveness in the construction company.

1. Introduction

The construction industry plays a crucial role in the progress of the society, involving planning, design and execution of various projects related to buildings, infrastructure and other structures. All these projects are carefully managed by professionals such as architects, project managers, engineers, electricians and many others (Wood, 2012).

The construction field is significantly influenced by building regulations, technological progress and economic conditions, this is the reason why all construction companies are oriented towards efficiency and effectiveness, as the management consultant Peter Drucker (1967) said "Efficiency is doing things right; effectiveness is doing the right things". Organizing the construction of a project may be difficult: supervising the budget, managing the time for meeting the deadline, checking the status of a project to determine if it is on schedule and many other tasks, so the companies must find ways to avoid eventual effectiveness issues (*The Economist*, 2017). This is the point where technological advancements are used to increase the business productivity.

Moreover, in order to attract clients, companies must find ways to advertise their services and one of the most efficient ways is online advertising. Businesses can reach a broad audience and therefore constantly increase their customer base (Oberoi, 2013).

Bob The Builder construction company's owner is currently facing operational challenges with the management of all projects. He is overloaded with work and does not possess any means to assist him in his work (see the rich picture in Appendix A). Bob is missing the clear overview of ongoing projects as well as having to initiate new projects manually entering all data.

The company also lacks real-time feedback on construction completion for their customers, along with having no archive to compile and compare statistics of completed projects. The owner must assure enough construction materials for his workers and pay them according to their working hours.

Customers are expecting exceptional services and good communication with the company, so they can be informed about the status of projects. Moreover, potential customers want more details about prices and available project types before considering initializing a new project, in order to decide if the company offers what they are looking for.

All these issues are causing Bob to lose potential profit and make an analysis for future projects impossible.

The purpose of this project is to help Bob (the owner) advertise his company and manage all his projects by supervising the budget, planning the time allocation and establishing the status related to the schedule, as well as giving the customers a way to view the progress on their projects. The project will not be used to communicate with clients, only to advertise and give them the option to see the updated stage of the projects.

In order to solve these problems, the project was divided into several phases: analysis, design, implementation and testing, each of them playing a major role in the development of this project.

2. Analysis

Since the construction industry has always faced effectiveness challenges in terms of project management (Atkinson et al., 2006), the stakeholders - including company's owner and customers - have varied requirements (Bass et al. 1998) and expectations, in order to achieve their goal, but their collective needs are related to efficient project execution and effective communication.

2.1 Requirements

The system is designed for a diverse group of users, each possessing specific objectives:

- the owner of the company - needs a clear overview of the ongoing and completed projects, to be able to manage all work and resources;
- the existing customers - want to access the real-time progress of their order;
- the potential customers - are expecting to find information about the company's services.

2.2 Functional Requirements

1) As the owner of the company, I want to keep track of the new projects (of the following types: Residential, Commercial, Industrial or Road Construction), assign them a start date and have default values for some of the new project's fields: expected duration - specific for each type and for residential projects: number of kitchens (1), bathrooms (1), other rooms with plumbing (1), if the project is a new build or a renovation (new build); for commercial projects: number of floors (1); for road construction: number of bridges (0), tunnels (0) and other environmental challenges (none), so I don't have to enter all data every time I add a new project.

2) As the owner of the company I want to modify the project's details(for residential projects: number of kitchens/bathrooms/other rooms with plumbing, size of the building; for commercial projects: the size, the use of the building and the number of floors; for industrial projects: the size and the use of the building; for road construction: the length and width of the road, number of bridges and tunnels, the environmental challenges) in case I accidentally type wrong values or the circumstances of the construction change.

3) As the owner of the company, I want to update the estimated expenses and resources in my project (expected duration, budget, expected expenses, estimated total hours), so I can get an overview of resources distribution for the whole project.

4) As the owner of the company, I want to update the materials expenses, salary expenses and man-hours used, using daily values, add notes about each value and get the system to calculate the total expenses for an ongoing project, so I don't have to calculate the expenses myself and to remember what expenses I lastly added.

5) As the owner of the company, I want to see the expected duration, expected total hours, expected expenses, man-hours used, and total expenses for an ongoing project, so I can decide if the project is behind schedule.

6) As the owner of the company, I want to mark a project as behind the schedule, so that I can see the status of the project without checking all fields.

7) As the owner of the company, I want to assign an ID to each new project, so that each project will be uniquely represented and easier to find.

8) As the owner of the company, I want to delete a project, in case I accidentally create it or if it was completed a long time ago.

9) As the owner of the company, I want to store data from a completed project, then use it to calculate the average time and expenses for similar projects, so that I can compare and better estimate the resources for similar future projects.

10) As the owner of the company, I want to mark an ongoing project as completed, so I can use it for advertising and for future estimations.

11) As the owner of the company, I want to mark a project as ongoing, if I just created the project or if I accidentally marked it as completed.

12) As the owner of the company, I want to search a project by ID and filter the projects by type, budget, start date, duration or status, to see or update project's data.

13) As the owner of the company, I want to sort the projects by ID (alphabetically), type (alphabetically), status (alphabetically), budget (numerically), duration (numerically) or start date (chronologically) , so I can easily see the projects, accordingly to the criteria I'm interested in.

14) As the owner of the company, I want to see how many ongoing or completed projects I have, so I know the total number of projects and the number of projects that match the search criteria.

15) As the owner of the company, I want a website to advertise my construction company.

16) As a customer, I want to see the company's ongoing projects to see the progress of my order or how well the company works.

17) As a customer, I want to see the company's available types of projects and some completed projects to get a better idea of what the company offers and delivers.

18) As a customer, I want to see the company's contact information, so I can contact the company if I want to hire it.

2.3 Non-Functional Requirements

19) The system must use files as a backup strategy and every update in the system includes writing to a file.

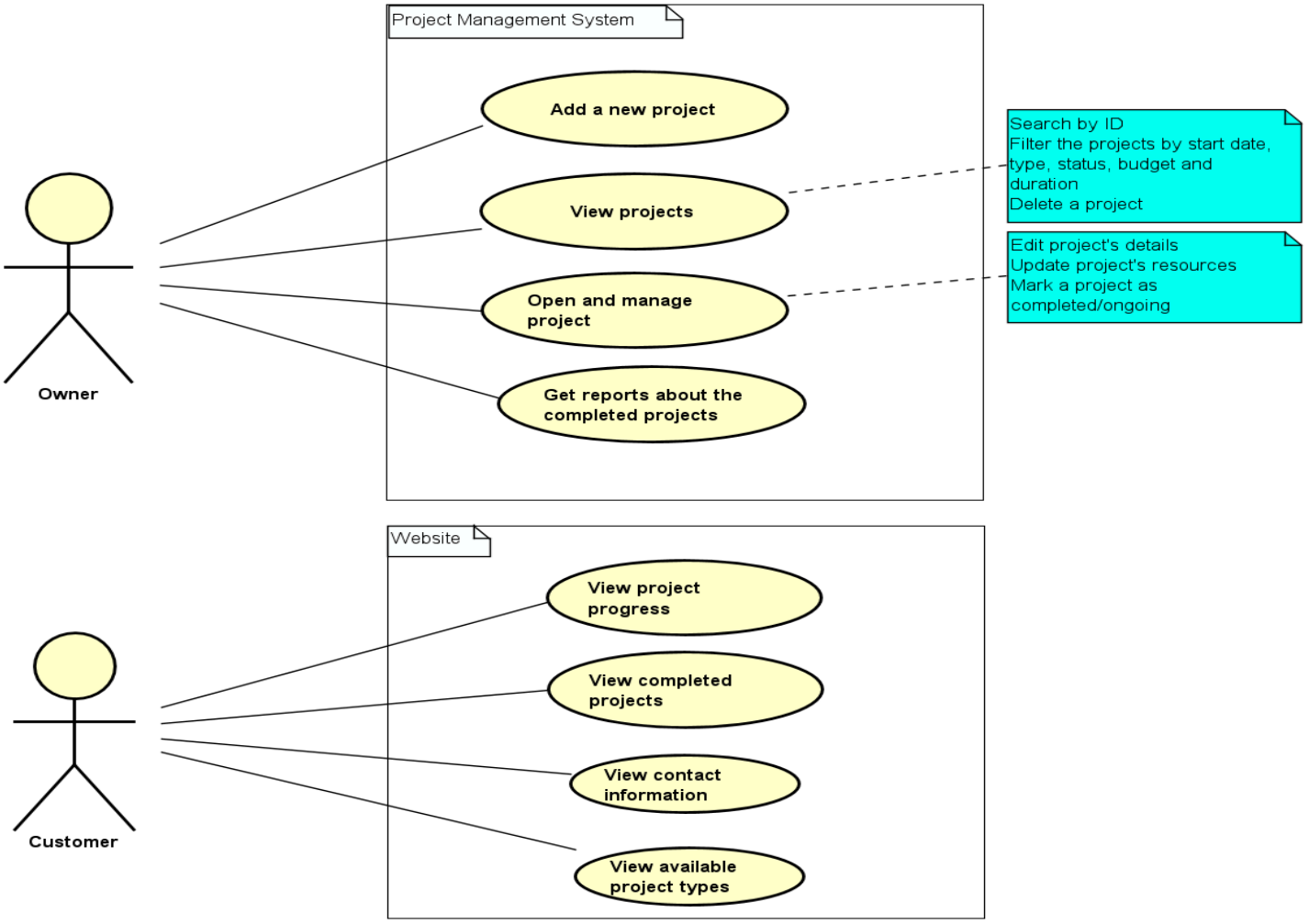


Figure 2.1 *The Use Case Diagram*

2.4 Relation between use cases and requirements

Use case	**Covered requirements**
Add a new project	1, 7, 19.
View projects	8, 12, 13, 14, 19.
Open and manage project	2, 3, 4, 5, 6, 10, 11, 19.
Get reports about completed projects.	9.
View project progress	15, 16.
View completed projects	15, 17.
View available project types	15, 17.
View contact information	15, 18.

Table 2.2 *Relation between use-cases and requirements*

The use case (Larman, 2004, chap. 6) diagram (Figure 2.1) indicates the options each user has:

- the owner can add, view and manage a project, as well as access projects' reports;
- the customers can see the company's contact information, services and portfolio, as well as the ongoing projects and their progress stage.

The table (Table 2.2) specifies the link between requirements and each use case and indicates that all requirements have been covered.

2.5 "Add a new project" Use Case Description

Use case	Add a new project
Summary	Create a new project with entered information and a set of default values (if no input is given).
Actor	Owner
Precondition	None
Postcondition	A new project has been added to the system.
Base sequence	1. System displays all ongoing projects. 2. Select project's Type {Residential, Commercial, Industrial, Road Construction}. 3. Enter data for: a) ID

	b) Budget c) Start date d) Expected duration e) Estimated total hours f) Expected expenses g1) If the project is of the Residential type then also Size, number of Kitchens, number of Bathrooms, number of other rooms with Plumbing (not kitchens and bathrooms) and if it is a new build or a renovation Default values (they can be changed): ● Expected duration is by default 9 months ● number of Kitchens is by default 1 ● number of Bathrooms is by default 1 ● number of other rooms with Plumbing is by default 1 ● the project is by default “New build” g2) If the project is of the Commercial type then also size, number of Floors and the intended Use of the building Default values (they can be changed): ● Expected duration is by default 18 months ● number of Floors is by default 1 g3) If the project is of the Industrial type then also size and intended Use of industrial facility Default values (they can be changed): ● Expected duration is by default 30 months g4) If the project is of the Road Construction type then also Length, Width, number of Bridges and Tunnels that need to be constructed and the description of environmental or geographical Challenges Default values (they can be changed): ● Expected duration is by default 18 months ● number of Bridges is by default 0 ● number of Tunnels is by default 0 ● the description of environmental or geographical Challenges is ● by default “none” 4. Approve the entered data.**[ALT1]** 5. System validates the data. **[ALT2]** 6. System adds a new project with the given data to the list of ongoing projects and to the files.
Alternate sequence	**[*ALT0]** The process can be cancelled in any step with user interaction ending the use case. The unapproved changes will not be saved. **[ALT1]** If the actor doesn’t approve the data, the project is not added. Go to Step 1. **[ALT2]** System validates all data:

	If ID is not given or it is used as ID for another project, then the System will show an error message. Go to Step 3. If the Budget is not given as a positive number, then the System will show an error message. Go to Step 3. If the Expected duration is not given as a positive number, then the System will show an error message. Go to Step 3. If Start date doesn't represent a legal date with day, month and year, then the System will show the message that the date is not legal. Go to step 3. If Estimated total hours is not given as a positive number, then the System will show an error message. Go to Step 3. If Expected expenses is not given as a positive number, then the System will show an error message. Go to Step 3. If the project is "Residential" and if the Size is not given as a positive number, then the System will show an error message. Go to Step 3. If the project is "Residential" and the number of Kitchens is not given as a non-negative number, then the System will show an error message. Go to Step 1. If the project is "Residential" and the number of Bathrooms is not given as a non-negative number, then the System will show an error message. Go to Step 3. If the project is "Residential" and the number of other rooms with Plumbing is not given as a non-negative number, then the System will show an error message. Go to Step 3. If the project is "Commercial" and if the Size is not given as a positive number, then the System will show an error message. Go to step 3. If the project is "Commercial" and if the number of Floors is not given as a positive number, then the System will show an error message. Go to Step 3. If the project is "Commercial" and the intended Use of the building is not given, then the System will show an error message. Go to Step 3. If the project is "Industrial" and the Length is not given as a positive number, then the System will show an error message. Go to Step 3. If the project is "Road Construction" and if the Width is not given as a positive number, then the System will show an error message. Go to Step 3.

	If the project is “Industrial” and if the Size is not given as a positive number, then the System will show an error message. Go to Step 3. If the project is “Industrial” and the Use of industrial facility is not given, then the System will show an error message. Go to Step 3. If the project is “Road Construction” and if the number of Bridges is not given as a non-negative number, then the System will show an error message. Go to step 3. If the project is “Road Construction” and if the number of Tunnels is not given as a non-negative number, then the System will show an error message. Go to step 3.
Notes	The Budget represents the financial baseline or the amount of money allocated for the project. The Expected expenses represent the actual amount of money that is expected to be spent for the project. This case covers requirements 1), 7), 19).

The “Add a new project” use case (Jacobsen, 1992) description explains step by step how the system is handling the data entered by the actor (Cockburn, 1997), default values and the behavior in case of invalid input.

Each of the remaining use cases is described in detail in Appendix B.

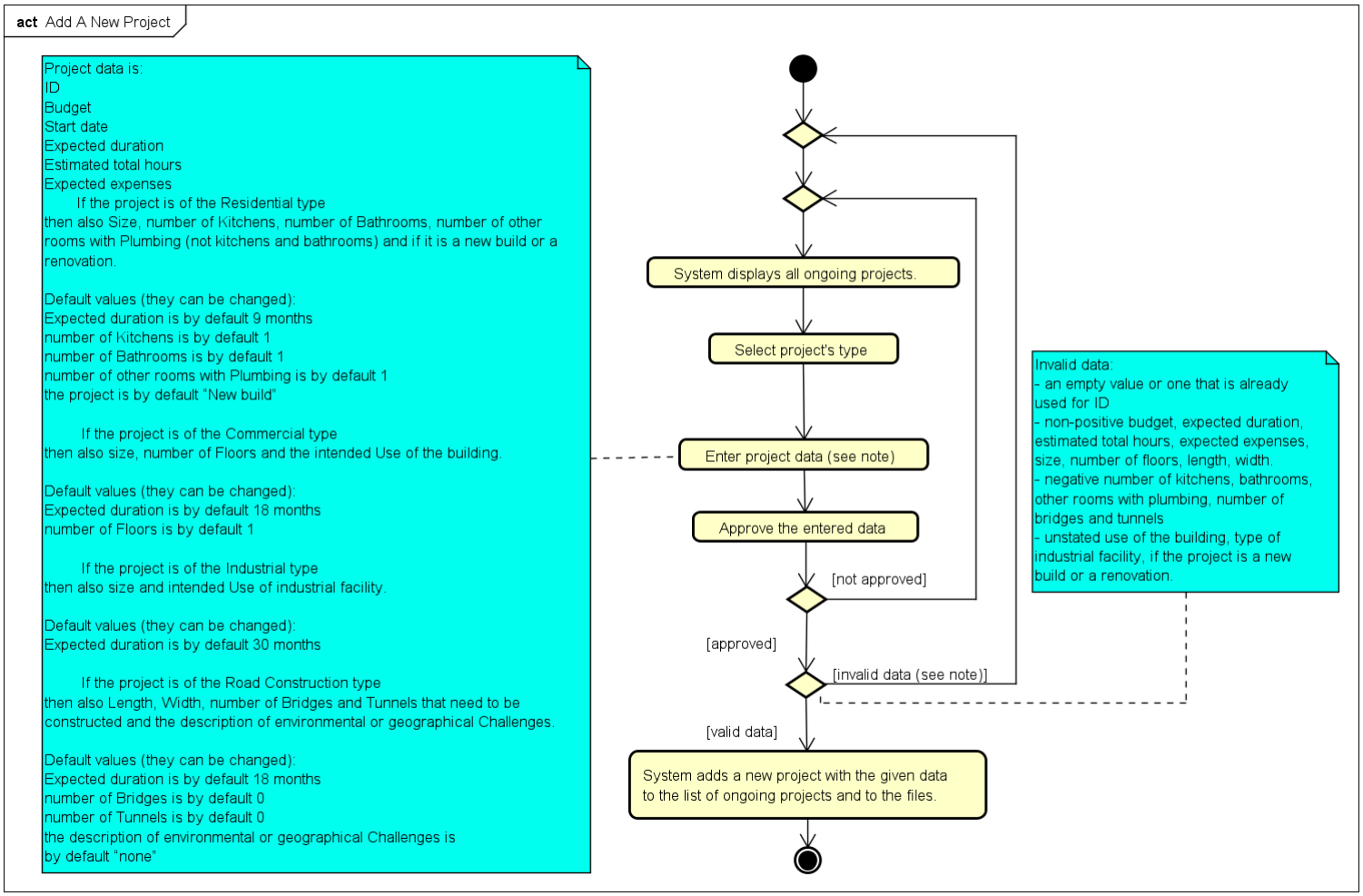


Figure 2.3 *Activity Diagram for "Add a new project" Use Case*

Figure 2.3 indicates each step of the same use case in a more dynamic manner, indicating where the type is selected, where the data must be entered and approved and when it is validated.

Each of the remaining use cases is represented through an activity diagram in Appendix C.

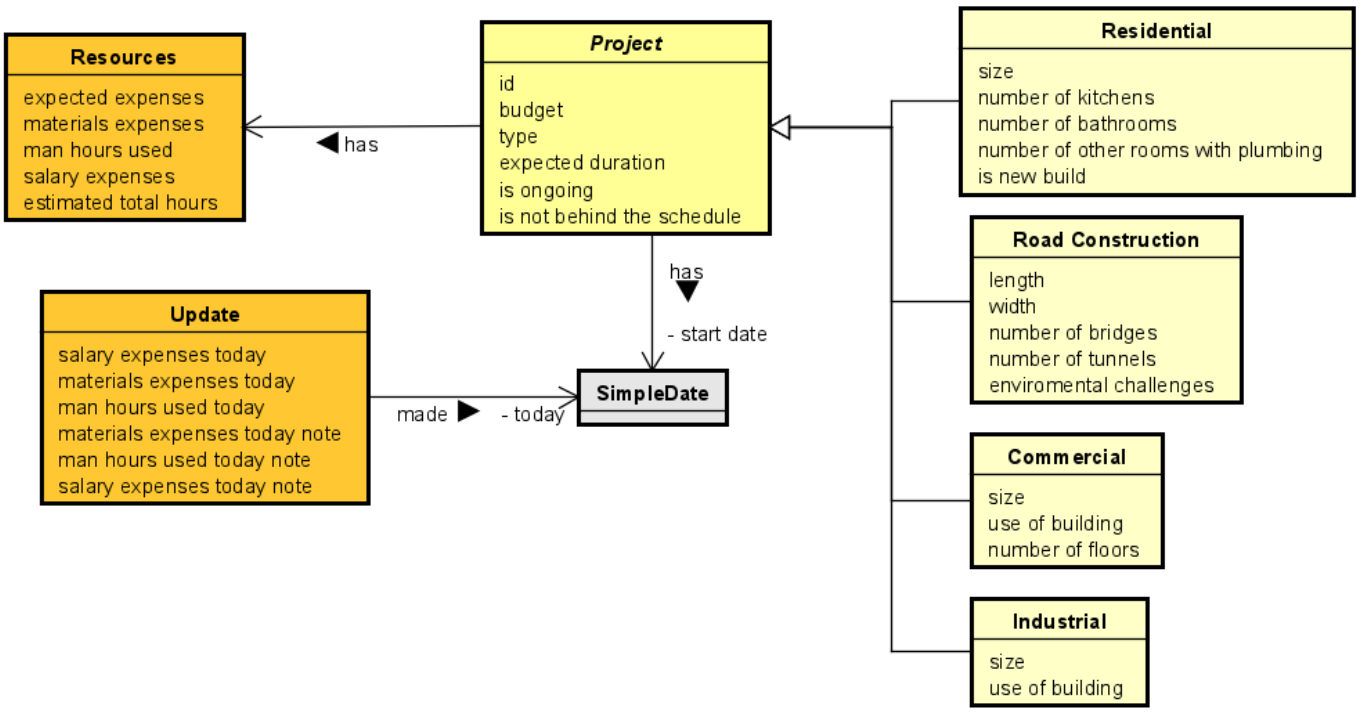


Figure 2.4 *The Domain Model*

The domain model (Martin & Odell, 1995; Fowler, 1996) - Figure 2.4, incorporates several classes: Project, which serves as a base class extended by specialized classes such as Residential, Commercial, Industrial, and Road Construction. This class hierarchy is used to effectively differentiate between diverse project categories, each having unique attributes and behaviors; Resources, which encapsulates various resources essential for project execution; SimpleDate, providing a representation for temporal aspects; and Update, responsible for tracking the progress within the projects.

In conclusion, the construction company's project management system demonstrates a well-structured domain model that aligns with industry best practices, including dedicated classes for project updates and resources that collectively contribute to an effective project management framework.

3. Design

3.1 System's design

The foundation of the project management system's design is the domain model (see Figure 2.4), which illustrates the essential structure and relationships between classes.

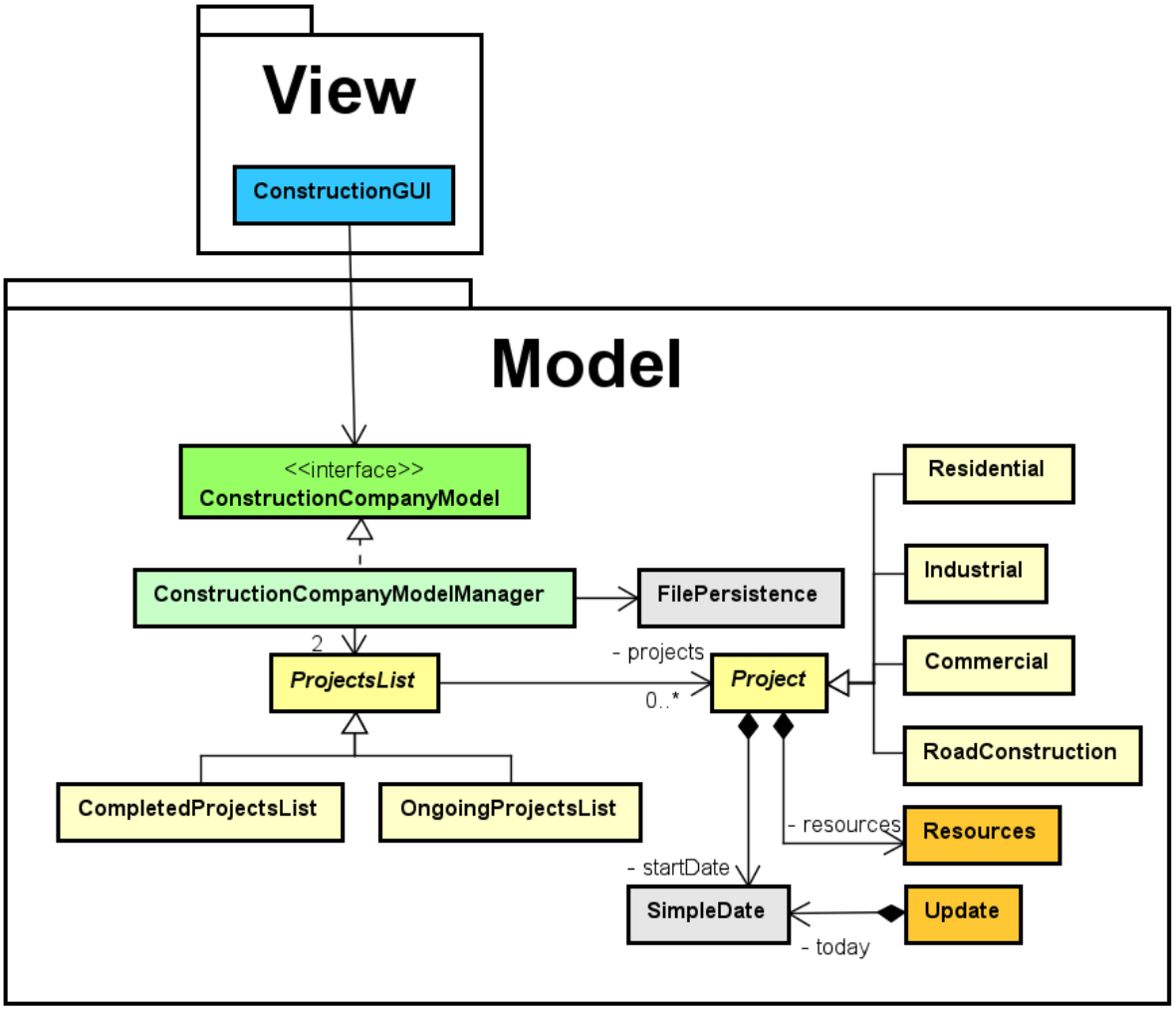


Figure 3.1 *The Class Diagram - Model*

The classes present in the domain model are also present in the class diagram (see Figure 3.1), where more other classes have been added to ensure efficient functionality and a robust structure. The projects are categorized by their progress status: ongoing and completed projects, organized in two lists that extend another general class. In the next chapter, it will be shown that the general class supports behaviors shared by both ongoing and completed projects.

The relationships (Larman, 2004, chap. 11) between classes are various: association (`ProjectsList - Project`, because the list of projects has elements of type `Project`; `ConstructionCompanyModelManager` - ProjectsLists and `ModelManager - FilePersistence`, because the model has projects that must be managed and files to store their data), composition (`Project - Resources`, `Project - SimpleDate`, since both the resources and the start date belong to the project, as well as `Update - SimpleDate`, because the current date belongs to each update), inheritance (`ProjectsList - CompletedProjectsList` and `OngoingProjectsList`, as a project is either ongoing or completed; `Project - Residential, Industrial, Commercial` and `RoadConstruction`, because a project is one of these mentioned types) and realization (where the `ModelManager` implements the functionality defined in the `ConstructionCompanyModel`/interface).

These relationships help model the structure and behavior of the system, facilitating the creation of maintainable and extensible code, as well as capturing the architecture and the functionality.

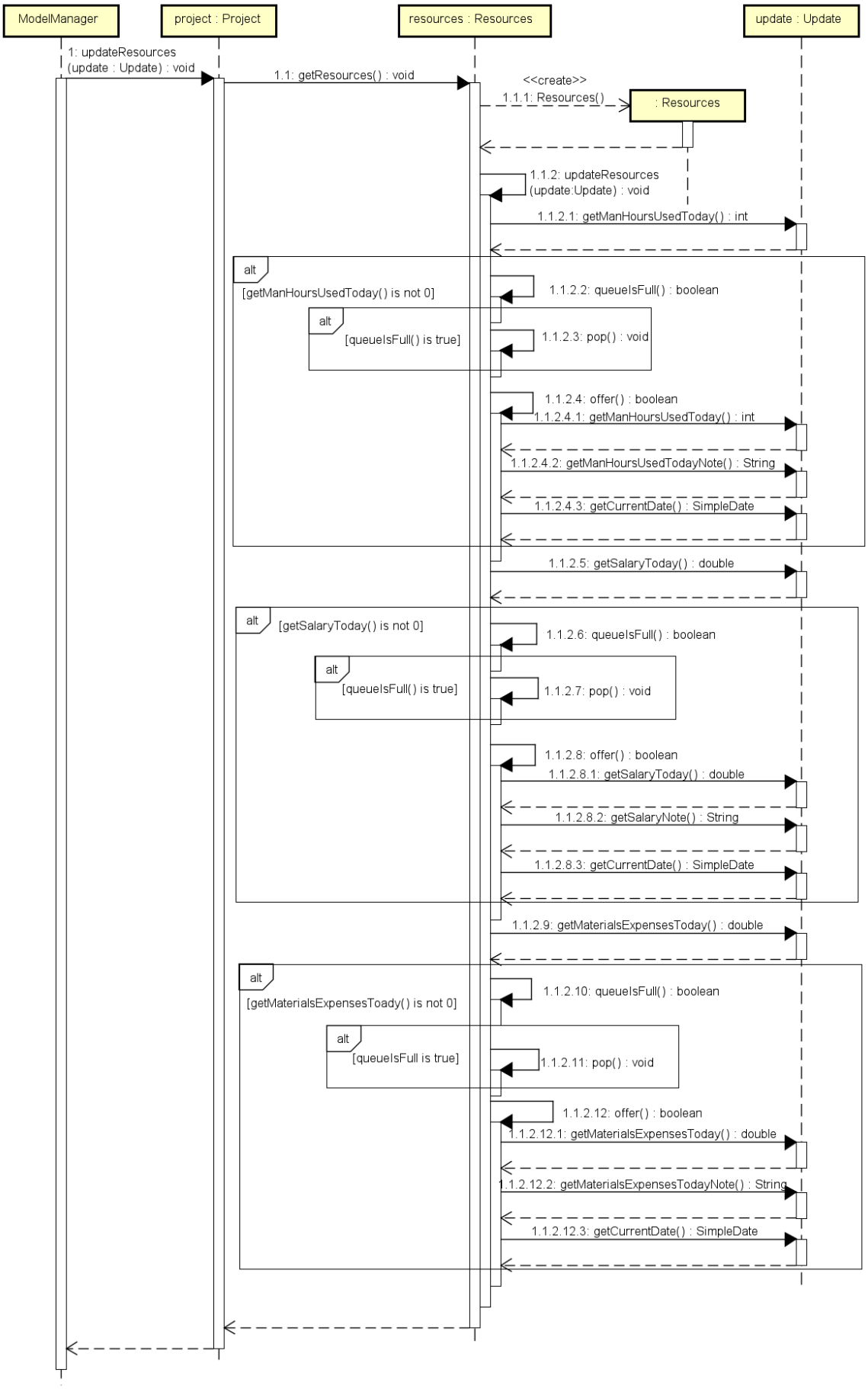


Figure 3.2 *Sequence Diagram for* `UPDATE RESOURCES` *Scenario*

The sequence diagram (Larman, 2004, chap.9) in Figure 3.2 highlights the entities that exchange messages in the diagram: the `ModelManager`, the `Project` class, its `Resources` and the `Update` class, used to modify the resources according to daily consumed values.

The `ModelManager` calls the `updateResources(Update update)` method for a project. The project is accessing its resources and updating them using the data found in the update argument: for each non-null value, the system checks if there is enough space in the updates' queue to add the new value; if the queue is full, the system removes the oldest element before adding the new data. The resources are updated and the latest values entered are displayed in the list of previous updates.

3.2 GUI's structure and functionality

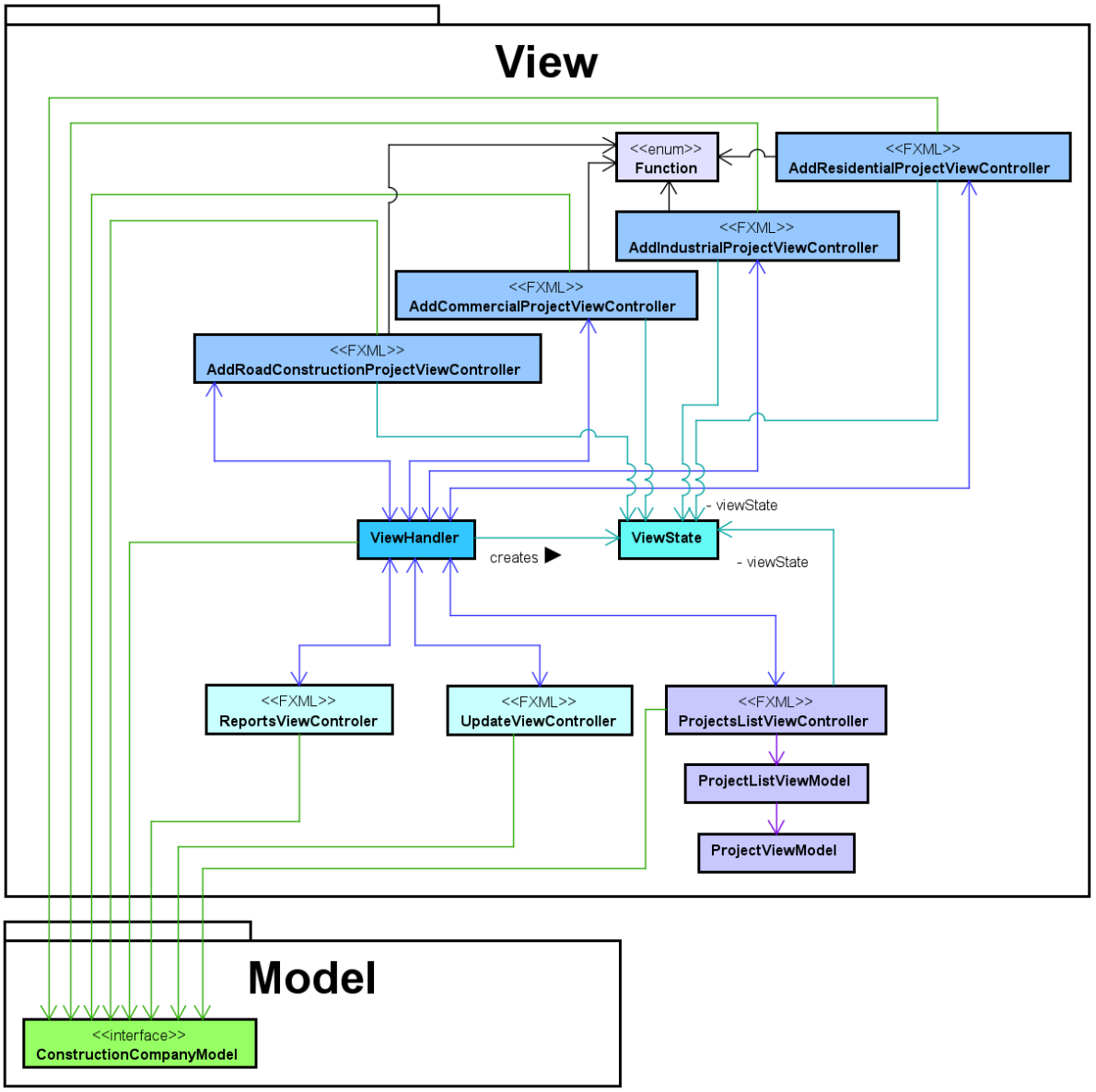


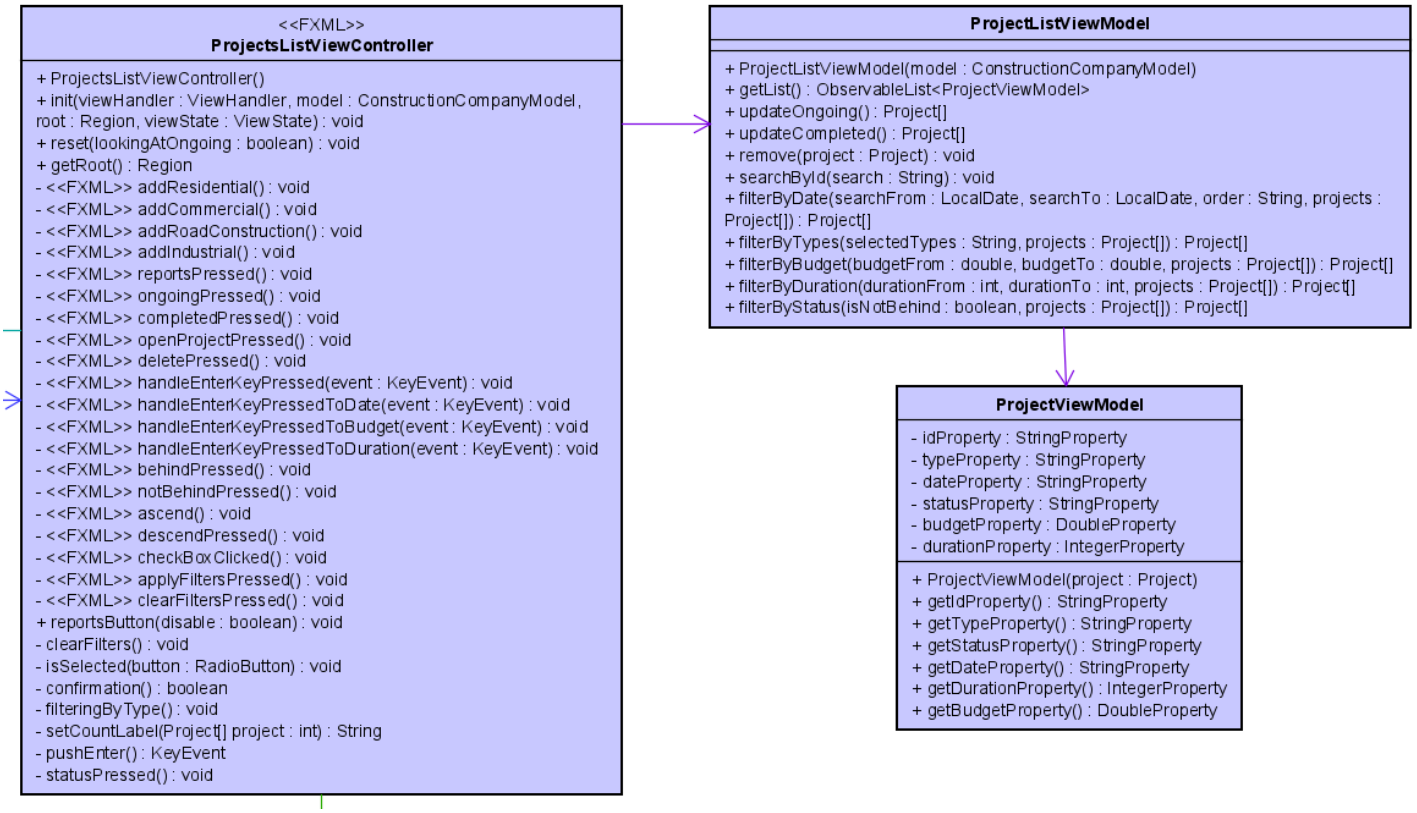
Figure 3.3 *The Class Diagram - View*

The system has seven `ViewControllers`:

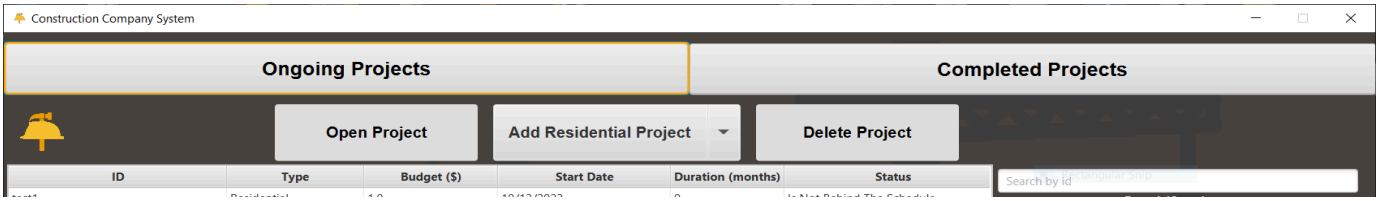
I. *`ProjectsListViewController`* - used for displaying the projects;
II. *`AddResidentialViewController`* - used for displaying, adding and editing a residential project;
III. *`AddCommercialViewController`* - used for displaying, adding and editing a commercial project;
IV. *`AddIndustrialViewController`* - used for displaying, adding and editing an industrial project;

V. `AddRoadConstructionViewController` - used for displaying, adding and editing a road construction project;
VI. `UpdateViewController` - used for updating projects' resources;
VII. `ReportsViewController` - used for displaying the reports about the completed projects.

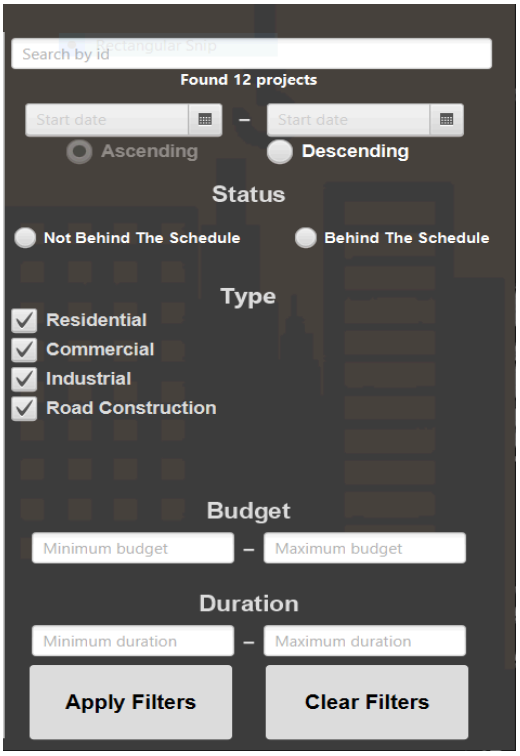
3.2.1 `ProjectsListViewController`



The system has two big sections: *Ongoing Projects* and *Completed Projects*, navigable through buttons that are using `ongoingPressed()` and `completedPressed()` fxml methods to display the projects according to their progress status. The `init(...)` method sets the non-FXML instance variables with values from its four parameters and initializes the scene with the existing projects and the method `reset(...)` manages the transition between *Ongoing Projects* and *Completed Projects,* according to the value passed in as argument.



In both sections, there are buttons such as *Open Project*, that uses the `openProjectPressed()` method to display the details and options for a selected project and *Delete Project*, that uses the `deletePressed()` method to completely remove a selected project from the system, after the user approves the deletion in a pop-up window managed by the method `confirmation()`. The drop-down menu button is used to add a project of the available types: *Residential, Commercial, Industrial or Road Construction*. Each type has its own button and method (`addResidential()`, `addCommercial()`, `addIndustrial()`, `addRoadConstruction()`) that open the specific view with specific fields.

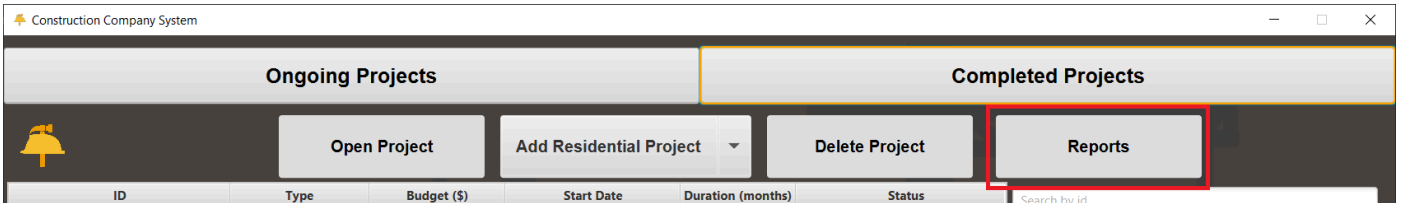


Additionally, in both sections, there is a filter area with the following functionality:

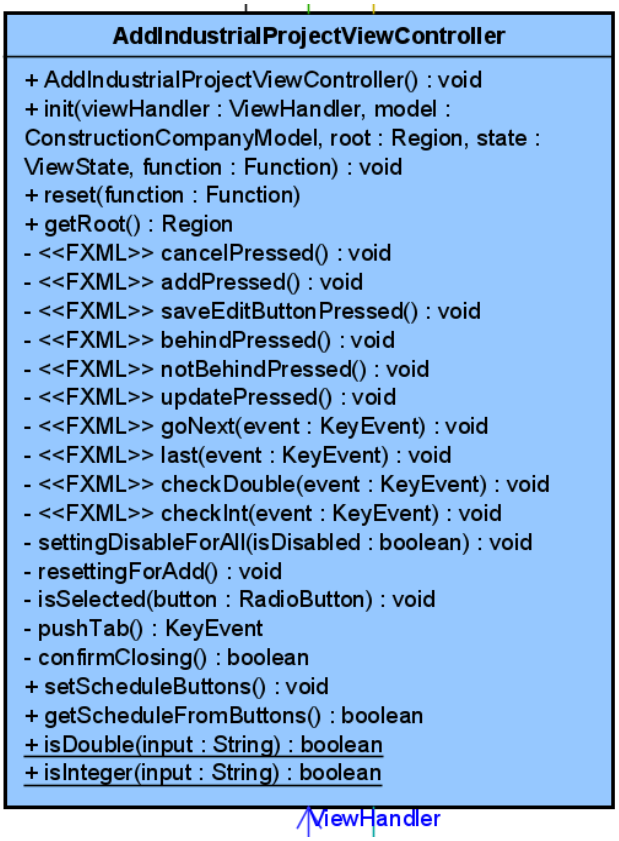
Name	Method(s) used	Functionality
Search by id	• `handleEnterKeyPressed(KeyEvent event)`	Displays the project with the entered id if the user presses enter (and if the project exists; displays a “No project with that ID exists” pop-up window otherwise).
Number of projects found	• `setCountLabel(Project[] projects)`	Counts the number of projects found before or after applying the filters.
Start date filter	• `handleEnterKeyPressedToDate(...)`	Displays the projects that have their start date in the entered interval, if the user presses enter (and if such projects exist; displays a “No projects found” pop-up window otherwise).

Ascending/Descending buttons	• `ascendPressed()` • `descendPressed()`	Display the projects shown after the start date filter was applied, in the ascending or descending order.
Status filter	• `behindPressed()` • `notBehindPressed()` • `status Pressed()`	Displays the projects with the specified status (behind or not behind) if the user presses enter.
Type filter	• `filteringByType()` • `checkBoxClicked()`	The `filteringByType()` method displays the projects that have one of the selected types. The `checkBoxClicked()` method ensures the updating of the projects displayed every time a box is selected or deselected.
Budget filter	• `handleEnterKeyPressedToBudget(...)`	Displays the projects that have their budget in the entered interval, if the user presses enter (and if such projects exist; displays a “No projects found” pop-up window otherwise).
Duration filter	• `handleEnterKeyPressedToDuration(...)`	Displays the projects that have their duration in the entered interval, if the user presses enter (and if such projects exist; displays a “No projects found” pop-up window otherwise).
Apply Filters button	• `applyFiltersPressed()`	Displays the projects that respect the entered filters if such projects exist; displays a “No projects found” pop-up window otherwise.
Clear Filters button	• `clearFiltersPressed()`	Displays the projects that respect the entered filters if such projects exist; displays a “No projects found” pop-up window otherwise.

There is one different functionality present only in the *Completed Projects* section - the *Reports* button, that uses the `reportsPressed()` method to open the view of the Reports section and the `reportsButton(...)` method, that makes the button visible only in the *Completed Projects* section, when its parameter is “true”.

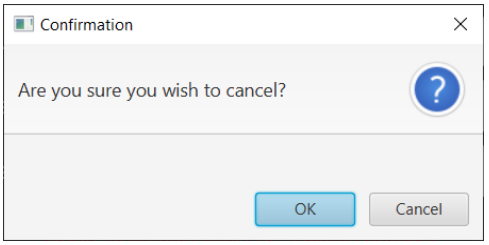


3.2.2 `AddTypeProjectViewController`



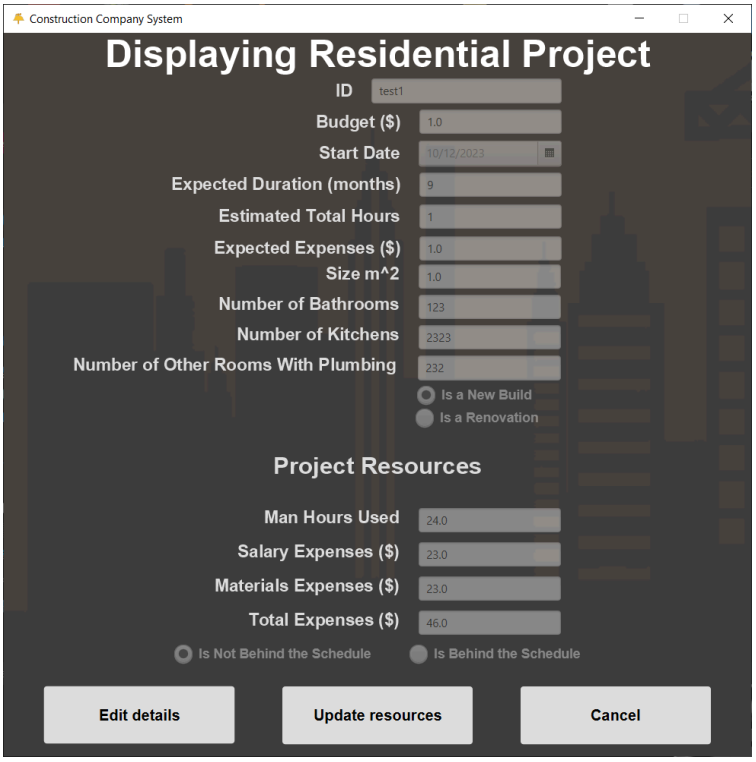
The system has four controllers for the creation of a project, one for each type: *`AddResidentialViewController`*, *`AddCommercialViewController`*, *`AddIndustrialViewController`*, *`AddRoadConstructionViewController`*. These controllers are reused for displaying and editing a project, depending on the *`Function`* enumeration's value (*`edit`*, *`display`*, *`add`*), that is passed as an argument in the *`init(...)`* and *`reset(...)`* methods, used for setting the non-FXML instance variables and initializing the scene with the needed fields and buttons.

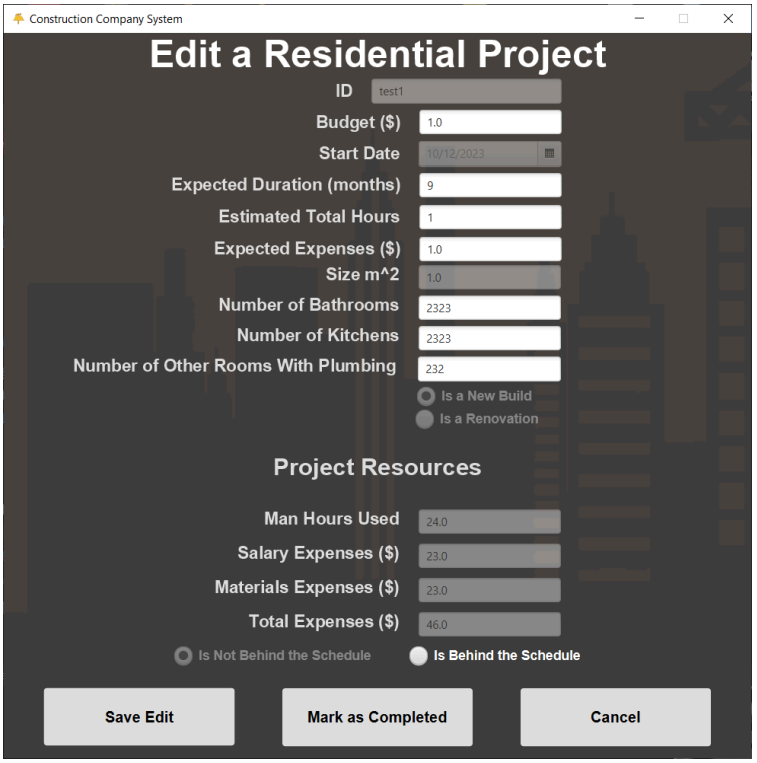
For the case *`display`* (accessed by pressing the *Open Project* button in the *ProjectsListView*), all the details and resources are shown for the selected project, but the fields are disabled by passing a "false" value as argument in the *`settingDisableForAll(...)`* method. Here, three (or two, for *Completed Projects*) buttons can be accessed:



- *Cancel*, which uses the *`cancelPressed()`* method to open the *ProjectsListView* after the user confirms in a pop-up window the intention to leave managed by the *`confirmClosing()`* method;
- *Update resources* (only visible for *Ongoing Projects*), which uses the *`updatePressed()`* method to open the *UpdateProjectView,* where the user can update the resources;
- *Edit details*, which opens the *AddTypeProjectView*, in the *`edit`* mode and makes some fields editable and offer other three buttons:
    - *Cancel*, which uses the *`cancelPressed()`* method to open the *AddTypeProjectView*, in the *`display`* mode after the user confirms in a pop-up window the intention to leave managed by the *`confirmClosing()`* method;

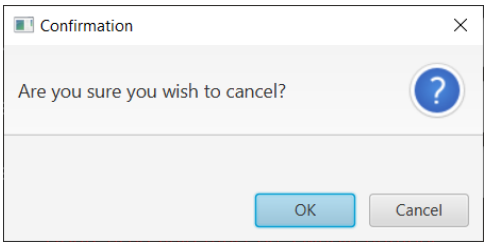
- *Save Edit*, which updates the project's details (if the entered data is valid) and open the *AddTypeProjectView*, in the `display` mode, using the `saveEditButtonPressed()` method;
- *Mark as Ongoing/Completed* - depending on project's progress (*Mark as Ongoing* for completed projects, *Mark as Completed* for ongoing project) - updating the status of the project and moving it in the new list using the `updatePressed()` method.



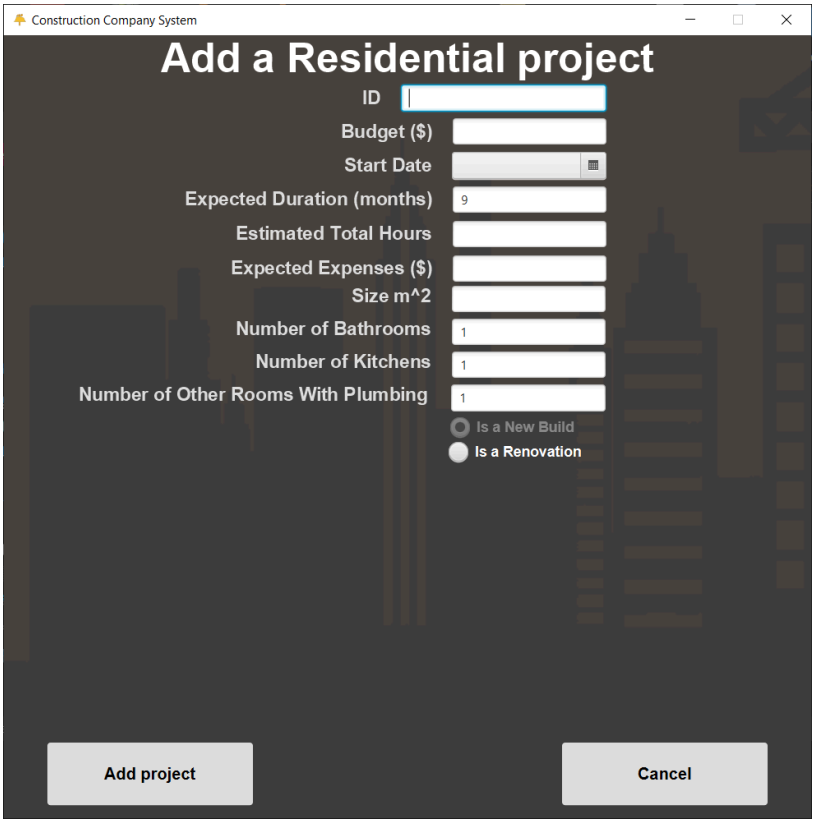


For the case `add` (accessed by pressing the drop-down menu button in the *ProjectsListView*, that uses the `addPressed()` method), all the details fields are shown - empty (using the `resettingForAdd()` *method*) and editable for the selected project type using the `settingDisableForAll(...)` method. Here, two buttons can be accessed:

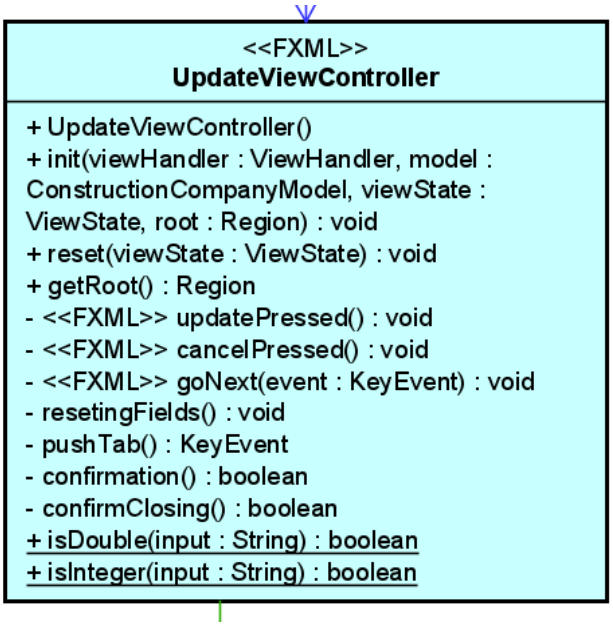
- *Cancel*, which uses the `cancelPressed()` method to open the *ProjectsListView*, after the user confirms in a pop-up window the intention to leave managed by the `confirmClosing()` method;



- *Add project*, which uses the `addPressed()` method creates and adds a new project to the list of ongoing projects, with the entered data (if the data is valid).

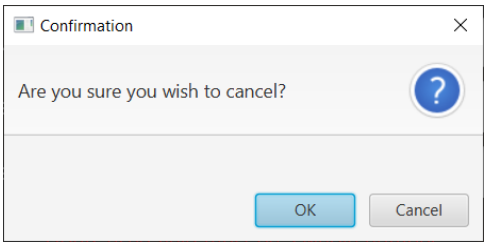


3.2.3 `UpdateViewController`

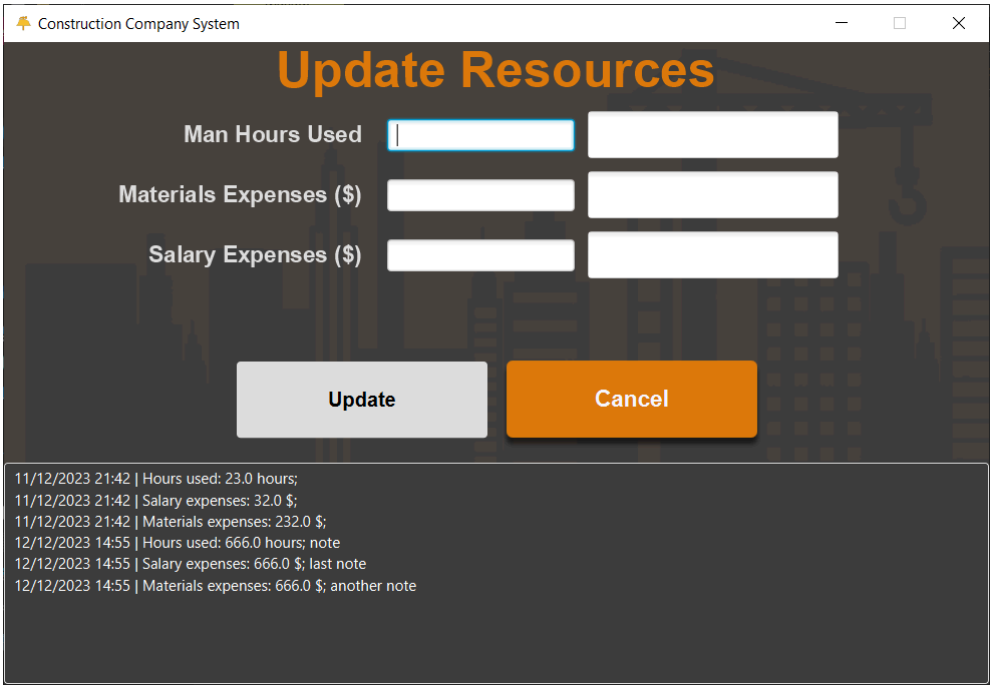


The UpdateProjectView is used to update projects' resources and it is accessed by pressing the *Update resources* button in the *AddTypeProjectView,* case `display`. In this section, some of the resources fields are shown - empty (using the `reset(...)` and `resettingFields()` *methods*) and editable.

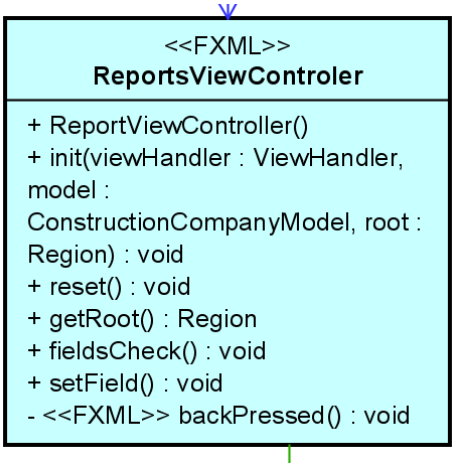
Here, two buttons can be accessed:



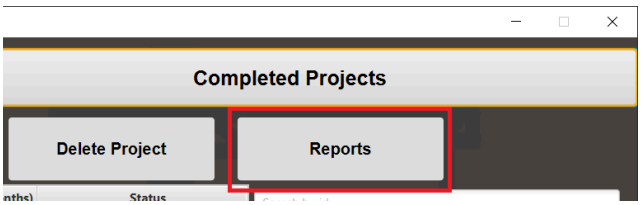
- *Cancel*, which uses the `cancelPressed()` method to open the *AddTypeProjectView*, in the *display* mode after the user confirms in a pop-up window the intention to leave managed by the `confirmClosing()` method;
- *Update*, which updates the project's resources (if the entered data is valid), saves the updates in the queue of updates and open the *AddTypeProjectView*, in the *display* mode, using the `updatePressed()` method;

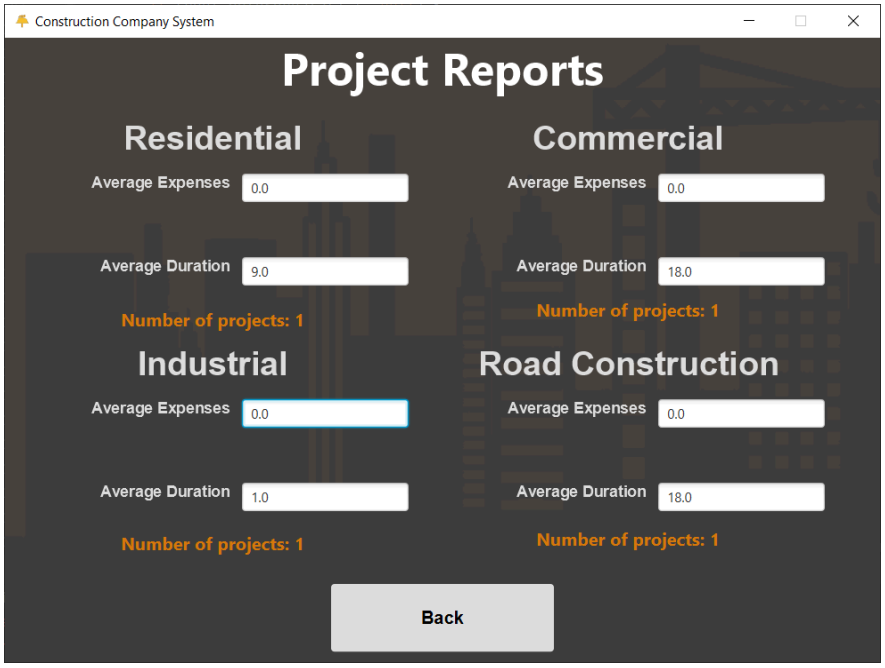


3.2.4 `ReportsViewController`



The ReportsView is used to display completed projects' reports and it is accessed by pressing the *Reports* button in the *ProjectsListView - Completed Projects* section.





Here, the system displays the *Average Duration, Average Expenses* and the *Number of projects* for each type, using the `setFields()` method. If there are no projects of any type, the system displays "No data available" using the `fieldsCheck()` method. The `backPressed()` method is used to open the *ProjectsListView - Completed Projects* section, when the user presses the *Back* button.

3.3 Website's design

The website's design is structured around the user experience (Hassenzahl, 2008):

- intuitive navigation for seamless exploration - given by the simple, precise and convenient located navigation bar (see Figure 3.4);
- useful content, so the user receives accurate and succinct details(see Figure 3.5);
- mobile responsiveness for accessibility on various devices (see Figure 3.6);
- fast loading times for a smooth experience - given by the xml files for loading small sized datasets to update the information.

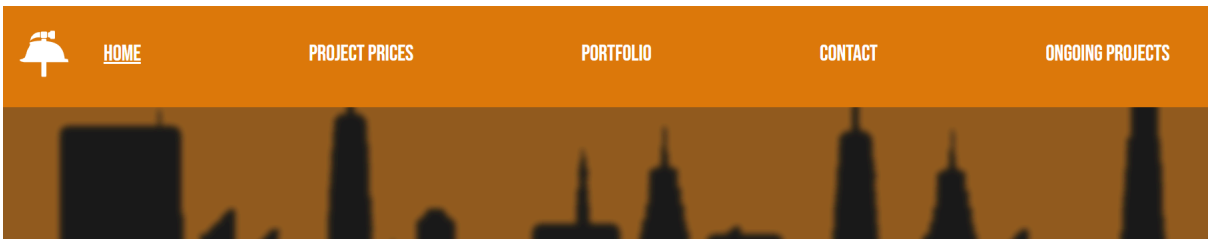


Figure 3.4 *Navigation Bar*

Figure 3.5 *Front Page's Content*

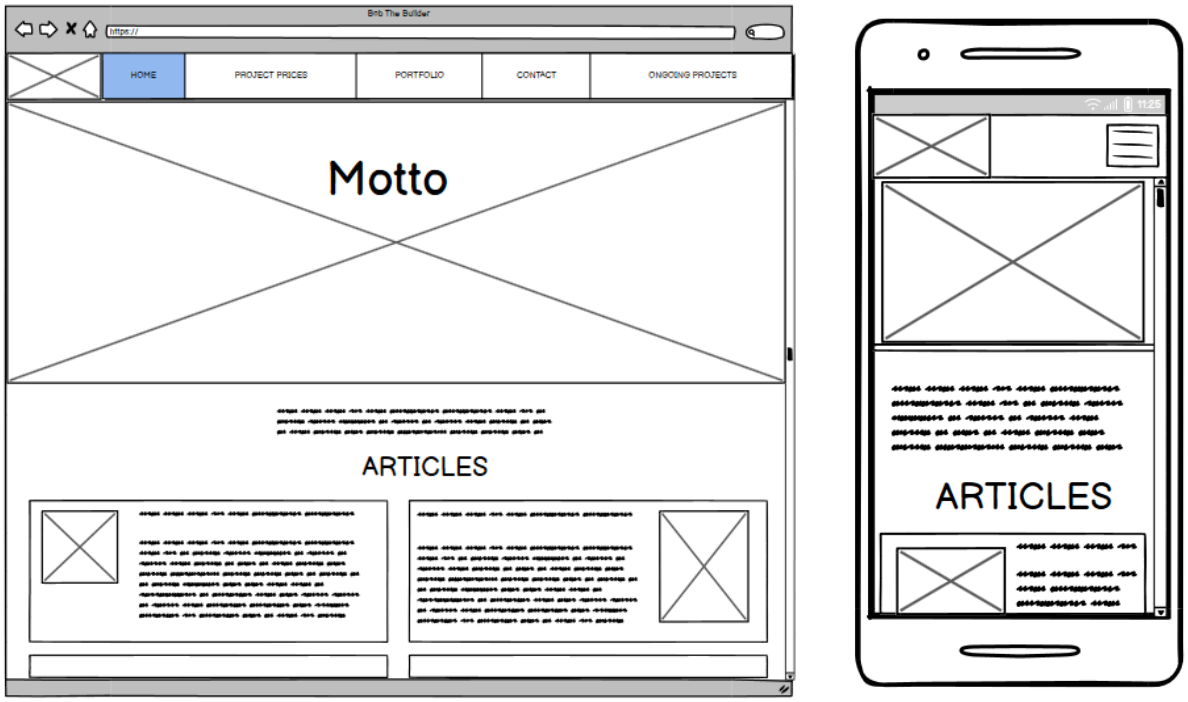


Figure 3.6 *Wireframes illustrating the layout of both desktop and mobile version of website's home page*

Website's pages have suggestive names:

- *HOME*, the front page - contains information about the company, articles and reports;
- *PROJECT PRICES* - displays the available project types the company offers;
- *PORTFOLIO* - shows the previous completed projects;
- *CONTACT* - presents the construction crew's members and displays their contact information;
- *ONGOING PROJECTS* - shows details about the ongoing projects.

As it can be seen in Figure 3.6, the website's layout adapts to different devices and screen sizes. Responsive design ensures that users can access and navigate the website seamlessly across various devices, including desktops and smartphones, therefore they receive a consistent and visually appealing experience regardless of the device they are using (see the wireframes for the remaining pages in Appendix D).

The color scheme is warm, revolving around orange and brown and aligning with the construction company's brand identity.

The website uses Bootstrap's grid system for structuring the layout. The main content is organized into sections, each occupying a specific number of grid columns to maintain consistency:

- *HOME* page - navigation bar, *ABOUT US section* - 1 column; *ARTICLES* section - 2 columns; *ANALYTIC REPORTS* section - 2 rows, with 1 and 2 columns; footer - 2 rows, with 3 and 1 columns;
- *PROJECT PRICES* page - navigation bar, header - 4 columns, *PROJECT TYPES* section - 2 rows with 2 columns each, footer - 2 rows, with 3 and 1 columns;
- *PORTFOLIO* page - navigation bar, *OUR COMPLETED PROJECTS* section - 2 rows with 2 columns each, footer - 2 rows, with 3 and 1 columns;
- *CONTACT* page - navigation bar, *MEET THE TEAM* section - 2 rows with 2 columns each, footer - 2 rows, with 3 and 1 columns;
- *ONGOING PROJECTS* page - navigation bar, footer - 2 rows, with 3 and 1 columns.

The images displayed on the website have various sources:

- *HOME* page - (HowStuffWorks, 2008)(IStock, 2021)
- *PROJECT PRICES* page - (Chloe, 2011)(Cursors4u, n.d.)(Freepik, n.d.)(Hopping, n.d.)(PxHere, n.d.)(The Opus Group, n.d.)
- *CONTACT* page - (Flickr, 2019)(Heroes Wiki, n.d.)(Bob the Builder Wiki, n.d.)(Pooh's Adventures Wiki, n.d.)

4. Implementation

The implementation is based on a clear understanding of the system requirements. This includes both functional and non-functional requirements that define what the system is expected to achieve. Moreover, the design phase serves as a blueprint for the implementation, including the data structures and algorithms that guide the development process. The alignment of the analysis and design ensures that the implementation process is well-planned, executed effectively, and ultimately delivers a successful system that meets the intended goals and requirements: adding, updating, deleting and displaying projects.

4.1 “Add a new project” use case - implementation

According to the “Add a new project” use case description (see the *Analysis* chapter for the complete version), the addition of an industrial project has 6 steps (see the code below each step):

1. System displays all ongoing projects.

```
//display the list of ongoing projects
@FXML private void ongoingPressed()
{
 //arrange the elements in the view
 projectTable.setItems(viewModel.getList());
 state.setProjects(viewModel.updateOnGoing());

 //hide the Reports button in the Ongoing Projects section
 reportsButton(false);
 lookingAtOngoing = true;

 //remove previous applied filters, to display all projects
 clearFilters();

 //display the number of projects in the list
 if(model.getAllOngoingProjects().length==1)
   countLabel.setText("Found 1 project");
 else
  countLabel.setText(
     "Found " + model.getAllOngoingProjects().length + " projects");
}
```

2. Select project’s Type (Industrial).

```
//open the view for adding an industrial project
@FXML private void addIndustrial()
{
 viewHandler.openView("addIndustrial", state);
}
```

3. Enter data

```
//clear all the fields in order to let the user enter data
private void resettingForAdd()
{
 //mark the fields that must be enabled
 List<TextField> fieldsToEnable = Arrays.asList(idField, budgetField,
     durationField, estimatedHoursField, expectedExpensesField,
sizeField,
     useOfBuildingField, manHoursField, salaryExpensesField,
     materialExpensesField, totalExpensesField);

 //clear the fields
 for (TextField field : fieldsToEnable)
 {
   field.setText("");
 }

 //name the window
 this.title.setText("Add an Industrial project");

 //name the button used for adding the project
 this.addButton.setText("Add project");

 //prepare the date picker for selecting the start date
 this.dateField.setDisable(false);
 this.dateField.setValue(null);
 this.dateField.getEditor().setDisable(true);
}
```

The `resettingForAdd()` method is used to clear the input fields before the user enters the data.

```
//disable or enable all fields, depending on the isDisabled's value
private void settingDisableForAll(boolean isDisabled)
{
 //mark the fields that must be enabled
 List<TextField> fieldsToEnable = Arrays.asList(idField, budgetField,
     durationField, estimatedHoursField, expectedExpensesField, sizeField,
     useOfBuildingField, manHoursField, salaryExpensesField,
     materialExpensesField, totalExpensesField);

 //enable the fields
 for (TextField field : fieldsToEnable)
 {
   field.setDisable(isDisabled);
 }

 //prepare the date picker for selecting the start date
```

```
 this.dateField.setDisable(isDisabled);
 this.notBehindButton.setDisable(isDisabled);
 this.behindButton.setDisable(isDisabled);
}
```

The `settingDisableForAll(...)` method is used to enable the input fields before the user enters the data.

```
//reset the view in the add, edit or display mode (function's value)
public void reset(Function function, ViewState viewState)
{
 this.errorLabel.setText(""); //set the error label
 this.function = function; //set the function {add, edit, display}
 //viewState is used to import data from a view to another (for the edit
mode)
 this.viewState=viewState;
 if (viewState != null)
   this.projectToEdit = (Industrial) viewState.getProject();

 //depending on the function's mode
 switch (function)
 {
   //the method will set the view to add a project
   case add ->
   {
     resettingForAdd(); //clear the fields
     //prepare the view's buttons and fields
     this.updateButton.setDisable(true);
     this.updateButton.setVisible(false);
     this.budgetField.setText("");
     this.durationField.setText(Industrial.defaults[0]);
     this.resourceBox.setVisible(false);
     settingDisableForAll(false); //let the user enter data
   }
   //the method will set the view to edit a project
   case edit ->
   {...}
}
```

The `reset(...)` method uses the above methods to prepare the view for entering data.

4. Approve the entered data.

The data is approved when the *Add project* button (that uses the `addPressed()` method, described at step 6) is pressed. If the *Cancel* button is pressed, the `cancelPressed()` will be called, including the `confirmClosing()` method, to confirm the intention to leave the view.

```
@FXML private void cancelPressed()
```

```
{
 //mark the fields that must have the initial appearance during leaving
confirmation
 List<TextField> fieldsToEnable = Arrays.asList(idField, budgetField,
     durationField, estimatedHoursField, expectedExpensesField, sizeField,
     useOfBuildingField, manHoursField, salaryExpensesField,
     materialExpensesField, totalExpensesField);
 //set fields' appearance
 for (TextField field : fieldsToEnable)
 {
   field.setStyle("");
 }
 //depending on the function's mode
 switch (function)
 {
   case add ->
   {
     if (confirmClosing()) //the method will ask for confirmation
     {
       viewHandler.openView("projects", null); //and open the
ProjectsListView
     }
   }
   case edit ->
   {...}
}

//display a pop-up window that asks the user to confirm the intention to
cancel
private boolean confirmClosing()
{
 Alert alert = new Alert(Alert.AlertType.CONFIRMATION);
 alert.setTitle("Confirmation");
 alert.setHeaderText("Are you sure you wish to cancel?");
 Optional<ButtonType> result = alert.showAndWait();
 return (result.isPresent()) && (result.get() == ButtonType.OK);
}
```

5. System validates the data.

```
//check if the entered data is a double
@FXML private void checkDouble(KeyEvent event)
{
 //store the text field
 TextField focusedNode = (TextField) event.getSource();
 //if data has been entered in the field, and it is not a double
 if (!isDouble(focusedNode.getText()) && !focusedNode.getText().isEmpty())
 {
   //display an error message
   errorLabel.setText("Invalid input for " +
focusedNode.getAccessibleText());
```

```
    //and set the field's color to red
    focusedNode.setStyle("-fx-border-color: red;");
  }
  else //otherwise
  {
    errorLabel.setText(""); //the label is empty
    focusedNode.setStyle(""); //and it has the initial color
  }
}
//check if the entered data is a double
@FXML private void checkInt(KeyEvent event)
{
  //store the text field
  TextField focusedNode = (TextField) event.getSource();
  //if data has been entered in the field, and it is not an integer
  if (!isInteger(focusedNode.getText()) && !focusedNode.getText().isEmpty())
  {
    //display an error message
    errorLabel.setText("Invalid input for " +
focusedNode.getAccessibleText());
    //and set the field's color to red
    focusedNode.setStyle("-fx-border-color: red;");

  }
  else //otherwise
  {
    errorLabel.setText(""); //the label is empty
    focusedNode.setStyle(""); //and it has the initial color
  }
}
```

The data is validated using the `checkDouble()` and `checkInt()` methods, that will be called each time a field gets focus.

6. System adds a new project with the given data to the list of ongoing projects and to the files.

```
//create a project with the entered data
@FXML private void addPressed()
{
  //depending on function's value {add, edit, display}
  switch (function)
  {
    //the method will create a project
    case add ->
    {
      errorLabel.setText(""); //clear the error label
      try
      {
        SimpleDate dateToAssign; //to store the start date
```

```
            if (dateField.getValue() == null) //if the start date field is empty
              dateToAssign = new SimpleDate(); //store the current date as start date
            else
              //otherwise, assign the entered start date
              dateToAssign = new SimpleDate(dateField.getValue());
            //create a Resources object using the entered values
            Resources newResources = new Resources(
                Double.parseDouble(expectedExpensesField.getText()),
                Integer.parseInt(estimatedHoursField.getText()));
            //create an industrial object using the data and resources
            Industrial newProject = new Industrial(idField.getText(),
                Double.parseDouble(budgetField.getText()), dateToAssign,
                Integer.parseInt(durationField.getText()), newResources,
                Double.parseDouble(sizeField.getText()),
                useOfBuildingField.getText());
            //add the project to the list of ongoing projects
            model.addToOngoingProjects(newProject);
            //open the ProjectsListView to show that the project has been added
            viewHandler.openView("projects", viewState);
          }
          catch (Exception e) //in case of invalid input
          {
            //display an error message
            Alert alert = new Alert(Alert.AlertType.ERROR);
            alert.setTitle("Invalid Input");
            alert.setHeaderText(e.getMessage());
            alert.showAndWait();
            errorLabel.setText(e.getMessage());
          }
        }
        case display ->
        {...}
    }
    public void addProject(Project project)
    {
     //add the project to the list of projects
     projects.add(project);
    }

    public void removeProject(Project project)
    {
     //remove the project from the list
     projects.remove(project);
    }

    @Override public void addToOngoingProjects(Project project)
    {
     //remove the project from the list of completed projects, if present
     completed.removeProject(project);
     //add the project to the list of ongoing projects
     ongoing.addProject(project);
```

```
}
```

The `addPressed()` method creates a project using the `Industrial(...)` and `Resources(...)` constructors, the `addToOngoingProjects(...)` method as well as getters defined in these classes. Additionally, the new project is saved in the files after moving to another view:

```
    /* This method opens different views according to the id and state's
values. It also ensures the windows are not resizeable as well as makes
them appear in the center of the screen, setting a title and an icon as
well.
    */
    public void openView(String id, ViewState state)
    {

       Region root = null;
       switch (id)
    {
    case "projects": {
    //open the ongoing projects section
       root = loadProjectsListView("/Scenes/ProjectsListView.fxml", true);
    //save the projects to the files
       model.saveProjects();
       break;
    }
    case "projectsComplete":
    //open the completed projects section
       root = loadProjectsListView("/Scenes/ProjectsListView.fxml", false);
    //saves the projects to the files
       model.saveProjects();
       break;
    {...}
}
    /*
    This method saves the Ongoing and Completed projects lists as well as
the list of updates of each individual project in files
    */
    @Override public void saveProjects(){
     /*  Save the ongoing projects, completed projects and the list of
updates in text files*/
     files.writeToTextFile(ongoing, "files/ongoingProjects.txt");
     files.writeToTextFile(completed, "files/completedProjects.txt");
     files.saveAllUpdateLists(ongoing,"files/listOfUpdates.txt");

     /*Save the ongoing projects in xml files, in order to be displayed on
the website*/
     try {
       files.saveXML(ongoing);
     }
     catch (ParserConfigurationException e) {
```

```
            throw new RuntimeException(e);
        } catch (TransformerException e) {
            throw new RuntimeException(e);
        }
    }

    /*
    This method write the data from list in the text file
    */
    public void writeToTextFile(ProjectsList list, String filename)
    {
        try (PrintWriter out = new PrintWriter(filename)){  //  using try
resources, declaring it in argument

            //  Writing size of the list first
        out.println(list.getSize());

        for (int i = 0; i < list.getSize(); i++) {
                //  Project
    //  Type first for reading

    out.println(list.getProject(i).getType());
    out.println(list.getProject(i).getId());
    out.println(list.getProject(i).getBudget());
    out.println(list.getProject(i).getStartDate().getDay());
    out.println(list.getProject(i).getStartDate().getMonth());
    out.println(list.getProject(i).getStartDate().getYear());
    out.println(list.getProject(i).getExpectedDurationInMonths());
                //  Status
    out.println((list.getProject(i).getIsNotBehindTheSchedule()) ? 1 : 0);

                //  Resources
out.println(list.getProject(i).getResources().getManHoursUsed());
out.println(list.getProject(i).getResources().getSalaryExpenses());
out.println(list.getProject(i).getResources().getMaterialsExpenses());
out.println(list.getProject(i).getResources().getExpectedExpenses());
out.println(list.getProject(i).getResources().getEstimatedTotalHours());
                switch (list.getProject(i).getType()) {
                    case "Residential" -> {
    out.println(((Residential) list.getProject(i)).getSize());
    out.println(((Residential) list.getProject(i)).getNumberOfKitchens());
    out.println(((Residential) list.getProject(i)).getNumberOfBathrooms());
    out.println(((Residential)
list.getProject(i)).getNumberOfOtherRoomsWithPlumbing());
    out.println((((Residential) list.getProject(i)).getIsNewBuild()) ? 1 :
0);
        }
    {...}
    }
```

The projects lists are of `ArrayList<Project>` type, so they allow calling methods such as `add(...)` and `remove(...)`.

4.2 Filters section - implementation

Some of the most complex algorithms present in the system are used in the filters section (for practical and combined utility, see the *User Manual,* chap. 2 - in Appendix E):

```
    /*This method searches for the project with the specified id and
arranges the item in the view, in order to be displayed*/
     public void searchById(String search)
     {
       list.clear(); //clear the list of projects
       /*search for the project with the specified id in the list of
ongoing projects*/
       for (Project project : model.getAllOnGoingProjects())
         if (project.getId().equals(search))
           list.add(new ProjectViewModel(project));
       /*search for the project with the specified id in the list of
completed projects*/
       for (Project project : model.getAllCompletedProjects())
         if (project.getId().equals(search))
           list.add(new ProjectViewModel(project));
       //if there is no project with the specified id, throw an exception
       if (list.isEmpty())
         throw new RuntimeException("No project with that ID exists");
     }

     /*This method filters and stores the projects with the start date in
the given interval and arranges the items in the view, in order to be
displayed*/
     public Project[] filterByDate(LocalDate searchFrom, LocalDate searchTo,
         String order, Project[] projects)
     {
       list.clear(); //clear the list of projects
       SimpleDate from = new SimpleDate(searchFrom);
       SimpleDate to = new SimpleDate(searchTo);

       //filter, store and arrange the projects
       Project[] results = model.getOngoingProjectsStartedBetween(from, to,
order, projects);
       for (Project project : results)
         list.add(new ProjectViewModel(project));
       /*if there are no projects within the specified interval, throw an
exception*/
       if (list.isEmpty())
         throw new RuntimeException("No projects found");
       return results;
     }
```

```
      /*This method filters and stores the projects with the type in the
specified string and arranges the items in the view, in order to be
displayed*/
      public void filterByTypes(String selectedTypes, Project[] projects)
      {
        list.clear(); //clear the list of projects
        //filter, store and arrange the projects
        Project[] results = model.getOngoingProjectsByType(selectedTypes,
projects);
        for (Project project : results)
          list.add(new ProjectViewModel(project));
        /*if there are no projects with the specified type, throw an
exception*/
        if (list.isEmpty())
          throw new RuntimeException("No projects found");
      }
      /*This method filters and stores the projects with budget in the
given interval and arranges the items in the view, in order to be
displayed*/
      public Project[] filterByBudget(double budgetFrom, double budgetTo,
Project[] projects)
      {
        list.clear(); //clear the list of projects
        //filter, store and arrange the projects
        Project[] results = model.getOngoingProjectsByBudget(budgetFrom,
budgetTo, projects);
        System.out.println(results.length);
        for (Project project : results)
          list.add(new ProjectViewModel(project));
        /*if there are no projects within the specified budget, throw an
exception*/
        if (list.isEmpty())
          throw new RuntimeException("No projects found");
        return results;
      }
      /*This method filters and stores the projects with duration in the
given interval and arranges the items in the view, in order to be
displayed*/
      public Project[] filterByDuration(int durationFrom, int durationTo,
Project[] projects)
      {
        list.clear(); //clear the list of projects
        //filter, store and arrange the projects
        Project[] results =
model.getCompletedProjectsByDuration(durationFrom, durationTo, projects);
        for (Project project : results)
          list.add(new ProjectViewModel(project));
        /*if there are no projects within the specified duration, throw an
exception*/
        if (list.isEmpty())
```

```
          throw new RuntimeException("No projects found");
        return results;
      }
      /*This method filters and stores the projects with the specified
status and arranges the items in the view, in order to be displayed*/
      public Project[] filterByStatus(boolean isNotBehind, Project[]
projects)
      {
        list.clear(); //clear the list of projects
        //filter, store and arrange the projects
        Project[] results = model.getProjectsByStatus(isNotBehind, projects);
        for (Project project : results)
          list.add(new ProjectViewModel(project));
        /*if there are no projects with the specified status, throw an
exception*/
        if (list.isEmpty())
          throw new RuntimeException("No projects found");
        return results;
      }
```

This implementation is made in the *View* package - *ProjectListViewModel* class and it is using methods implemented in the *Model* package - *ModelManager* by calling *ProjectsList* class' methods for each list of projects (ongoing and completed):

```
     @Override public Project getProjectById(String id)
     {
      if(ongoing.getProjectById(id)!=null)
        return ongoing.getProjectById(id);
      else if(completed.getProjectById(id)!=null)
        return completed.getProjectById(id);
      return null;
     }

     @Override public Project[] getOngoingProjectsStartedBetween(SimpleDate
startDate1, SimpleDate startDate2, String order, Project[] projects)
     {
      return ongoing.getProjectsStartedBetween(startDate1, startDate2, order,
projects);
     }
     @Override public Project[] getCompletedProjectsStartedBetween(SimpleDate
startDate1, SimpleDate startDate2, String order, Project[] projects)
     {
      return completed.getProjectsStartedBetween(startDate1, startDate2,
order, projects);
     }
     @Override public Project[] getOngoingProjectsByType(String
selectedTypes, Project[] projects)
     {
      return ongoing.getProjectsByType(selectedTypes, projects);
     }
```

```
    @Override public Project[] getCompletedProjectsByType(String
selectedTypes, Project[] projects)
    {
     return completed.getProjectsByType(selectedTypes, projects);
    }
    @Override public Project[] getOngoingProjectsByBudget(double budget1,
double budget2, Project[] projects)
    {
     return ongoing.getProjectsByBudget(budget1, budget2, projects);
    }
    @Override public Project[] getCompletedProjectsByBudget(double budget1,
double budget2, Project[] projects)
    {
     return completed.getProjectsByBudget(budget1, budget2, projects);
    }
    @Override public Project[] getOngoingProjectsByDuration(int duration1,
int duration2, Project[] projects)
    {
     return ongoing.getProjectsByDuration(duration1, duration2, projects);
    }
    @Override public Project[] getCompletedProjectsByDuration(int duration1,
int duration2, Project[] projects)
    {
     return completed.getProjectsByDuration(duration1, duration2, projects);
    }
    @Override public Project[] getProjectsByStatus(boolean
isNotBehindTheSchedule, Project[] projects)
    {
     return ongoing.getProjectsByStatus(isNotBehindTheSchedule, projects);
    }
```

The initial methods in the ProjectsList class:

```
    /*
    This method takes as parameters an array of projects, two dates and a
string that
    specify an order {Ascending, Descending} and returns all projects that
have their
    start date in the interval [startDate1; startDate2], in the specified
order, in a new array
    */
    public Project[] getProjectsStartedBetween(SimpleDate startDate1,
SimpleDate startDate2, String order, Project[] projects)
    {
     /*We create a new array that contains all the projects that respect
the date range condition*/
     Project[] filteredProjects = Arrays.stream(projects).filter(project ->
startDate1.isBefore(project.getStartDate()) &&
startDate2.isAfter(project.getStartDate())).toArray(Project[]::new);

     //We sort the projects according to the specified order
     if(order.equals("Ascending"))
```

```
       Arrays.sort(filteredProjects,
Comparator.comparing(Project::getStartDate));
      else
        Arrays.sort(filteredProjects,
Comparator.comparing(Project::getStartDate).reversed());

      //And return the new array
      return filteredProjects;
     }

     public Project[] getProjectsByType(String selectedTypes, Project[]
projects)
     {
      return Arrays.stream(projects).filter(project ->
selectedTypes.contains(project.getType())).toArray(Project[]::new);
     }
     public Project[] getProjectsByBudget(double budget1, double budget2,
Project[] projects)
     {
      return Arrays.stream(projects).filter(project ->
project.getBudget()>=budget1 &&
project.getBudget()<=budget2).toArray(Project[]::new);
     }
     public Project[] getProjectsByDuration(double duration1, double
duration2, Project[] projects)
     {
      return Arrays.stream(projects).filter(project ->
project.getExpectedDurationInMonths()>=duration1 &&
project.getExpectedDurationInMonths()<=duration2).toArray(Project[]::new);
     }
```

4.3 `getAverageExpenses(...)` method - complexity analysis

```
     // This method calculates and returns the average expenses value for
the type given as argument
     public double getAverageExpenses(String type)
     {
      //here we store the total expenses
      double expenses=0; // initialization - 1 step

      //here we store the number of projects with the specified type
      int count=0;// initialization - 1 step

      //we check all completed projects
      for(int i=0; i<getSize(); i++) //n iterations
        //if the type corresponds
        /*
        getProject() - 1 step (Oracle, n.d)
```

```
        getType() - 1 step
        equals() - m steps, where m is the number of characters (here, at
most 17)
        =19 steps
         */
        if(super.getProject(i).getType().equals(type))
        {
          //we update the expenses and increment the counter
          /*
          getProject() - 1 step;
          getResources() - 1 step;
          getTotalExpenses() - 2 steps;
          =4 steps
           */

expenses=expenses+super.getProject(i).getResources().getTotalExpenses();
          count++; //1 assignment and 1 addition = 2
        }
      //we return the result
      return expenses/count; //1 return, 1 division = 2
    }
    /*
    T(n)=1 + 1 + n * (19+4+2) + 2 = 25n + 4.
    Since we can ignore the constants and coefficients => O(n),
    where n is the number of completed projects.
    We chose this method because we had to iterate through the whole list
of
    completed projects.
    */
```

4.4 `getProjectsStartedBetween(...)` method - complexity analysis

The most complex method in the system has a n*log n complexity and it is listed below, along with comments describing the logic and the complexity behind the algorithm.

```
    /*
    This method takes as parameters an array of projects, two dates and a
string that specify an order {Ascending, Descending} and returns all
projects that have their start date in the interval [startDate1;
startDate2], in the specified order, in a new array
    */
    public Project[] getProjectsStartedBetween(SimpleDate startDate1,
SimpleDate startDate2, String order, Project[] projects)
    {
     //We create a new array that contains all the projects that respect
the date range condition
      /*
```

```
        Initialization - 1 step
        The stream().filter() methods involve iterating through the array of
projects once and applying the date range condition => n iterations, where
n is the number of projects (Oracle, n. d.)
        */
        Project[] filteredProjects = Arrays.stream(projects).filter(project ->
            // isBefore() and isAfter() methods have a constant time
complexity, both taking up to 4 steps (see SimpleDate class)
            //getStartDate() has a constant time complexity (3 steps)
            /*
            toArray() method creates the new array and copies the elements
from the stream into that array => n steps (Oracle, n. d.), where n is the
number of projects that respect the date range condition
            */
        startDate1.isBefore(project.getStartDate()) &&
startDate2.isAfter(project.getStartDate())).toArray(Project[]::new);

        //We sort the projects according to the specified order
        if(order.equals("Ascending"))//1 comparison
         Arrays.sort(filteredProjects,
Comparator.comparing(Project::getStartDate));
        else
          //sort() method takes n*log n iterations (Oracle, n. d.), where n
is the number of projects that respect the date range condition
          //reversed() method has a constant time complexity , because it
simply changes the sorting order (Oracle, n. d.)
          Arrays.sort(filteredProjects,
Comparator.comparing(Project::getStartDate).reversed());

        //And return the new array
        return filteredProjects;//1 return statement
        /*
        T(n)= 1 + n + 4 + 4 + 3 + n + 1 + (n*log n) + 1 + 1 = 15 + 2n +
(n*log n)
        Ignoring the constants and coefficients, we get O(n)=n*log n,
        where n is the number of projects that have to be sorted.
         */
        /*
        We chose this method since it is efficient, especially for large
projects arrays, where the sorting section dominates the complexity
         */
    }
```

4.5 `updateResources(...)` method - complexity analysis

This method uses a queue to store the previous updates, with a 12 elements limit. This method was chosen for data structure diversity.

```
    /*
    This method updates the worked hours, salary expenses and materials
expenses using the
    values found in the update and display all updates as strings in the
queue, as long as the
    queue is not full
    */
    public void updateResources(Update update)
    {
     //If this field is not 0
     if(update.getManHoursUsedToday()!=0)//1 variable accessed and 1
comparison
     {
       //We prepare to add it in the queue, by removing the older updates
       if(queueIsFull())//constant time - at most 2 steps
       //independent of the queue size - constant time (Oracle, n. d.)
         updates.pop();
       //We update the value
       //1 assignment, 1 addition, 1 variable accessed = 3
       manHoursUsed = manHoursUsed + update.getManHoursUsedToday();

       //And add the update's information to the queue
       //offer() - independent of the queue size => constant time (Oracle, n. d.)
       //3 variable accessed
       updates.offer(update.getCurrentDate() + " | Hours used: " +
update.getManHoursUsedToday() + " hours; " +
update.getManHoursUsedTodayNote());
     }
     //If this field is not 0
     if(update.getSalaryToday()!=0)//1 variable accessed and 1 comparison
     {
       //We prepare to add it in the queue, by removing the older updates
       if(queueIsFull())//constant time - at most 2 steps
         updates.pop();//independent of the queue size - constant time

       //We update the value
       //1 assignment, 1 addition, 1 variable accessed = 3
       salaryExpenses=salaryExpenses+update.getSalaryToday();

       //And add the update's information to the queue
       //offer() - independent of the queue size => constant time
       //3 variable accessed
       updates.offer(update.getCurrentDate() + " | Salary expenses: " +
update.getSalaryToday() + " $; " + update.getSalaryNote());
     }
     //If this field is not 0
     if(update.getMaterialsExpensesToday()!=0)//1 variable accessed and 1
comparison
     {
       //We prepare to add it in the queue, by removing the older updates
       if(queueIsFull())//constant time - at most 2 steps
```

```
            updates.pop();//independent of the queue size - constant time

          //We update the value
          //1 assignment, 1 addition, 1 variable accessed = 3
          materialsExpenses = materialsExpenses +
update.getMaterialsExpensesToday();

          //And add the update's information to the queue
          //offer() - independent of the queue size => constant time
          //3 variable accessed
          updates.offer(update.getCurrentDate() + " | Materials expenses: " +
update.getMaterialsExpensesToday() + " $; " +
update.getMaterialsExpensesTodayNote() );
        }
      }

      /*
      T(n)=(1+2+1+3+1+3)*3=33 steps => O(1)
      We chose this method because the queue is a very efficient data
structure to store information,
      allowing us to easily remove the first element
      */
```

4.6 Website's portfolio page - carousels implementation

When introducing new customers to the company's already completed projects, the layout and some design parts are handled using the Bootstrap framework, due to its reputation of facilitating efficient front-end development. Using Bootstrap's carousels allows switching sections on button click. Additionally, Bootstraps Cards are used to design the carousels' structure, making use of pre-built classes.

```
<div class="carousel-inner">
      <div class="carousel-item active">
            <div class="card">
                  <img src="#" alt="buildingImage1"
class="card-img-top">
                  <div class="card-body text-center" id="cardBody">
                        /*Information about the first building*/
                  </div>
            </div>
      </div>
      <div class="carousel-item">
            <div class="card">
                  <img src="#" alt="buildingImage1"
class="card-img-top">
                  <div class="card-body text-center" id="cardBody">
                        /*Information about the second building*/
                  </div>
            </div>
```

```
        </div>
    </div>
```

To make the carousel functional, so the cards can be switched around (elements with `carousel-item` class) on button click, specific classes for two buttons functioning as "*previous*" and "*next*" are needed. Additionally, an ID referencing the carousel must be assigned.

```
<button class="carousel-control-prev"
      type="button"
      data-bs-target="#myCarousel"
      data-bs-slide="prev"  >
            /* according image */
</button>
<button class="carousel-control-next"
      type="button"
      data-bs-target="#myCarousel"
      data-bs-slide="next"  >
            /* according image */
</button>
```

4.7 Website's contact page - member's pictures

The Contact page uses the simple design of splitting the window in four equal parts, that shows information about all four team members, displaying their name, their function and email, to easily contact them. Additionally, images from *Bob the Builder* cartoon, produced by HiT Entertainment.

Bootstrap's grid system was the most suitable option to use. In this case, a small break-point with a six column width was assigned to all member sections, first two and then the last two being wrapped in a container with class `row`, resulting in positioning the sections next to each other across full page width on a large screen.

The sections move when the user places the mouse over them, this being implemented by adjusting the CSS style-sheet using pseudo-class `:hover`, which allows the selection of an element that is being hovered over. In this case, this visual feedback is handled by slightly moving positions of section layout, which consists of member pictures alongside a background shape.

To make the whole interaction better, the `transition` CSS property is added, which smoothly animates changes over a specific duration. When combined with the `:hover` pseudo-class, it creates a smooth transition effect, when the user interacts with the element

```
.portrait .shape .self{
      position: absolute;
      max-width: 400px;
      top: 30px;
      transition: 1s ease;
}
```

```
.portrait .shape .self{
        top: 50px;
}
```

5. Test

The testing phase is integrated as a parallel and complementary process during the implementation phase. Unit (Elims, Bridges & Ince, 2004) and integration (Reis, Metzger & Ince, 2007) tests were conducted to validate individual components of code, as well as verify that different modules work together seamlessly. Unit testing covers the importance of catching and fixing issues at the unit level before integration testing, which ensures that the combined use cases function as intended.

Use Case	Test results	Comments
Add a new project	Works.	A new project can be added to the system, with all the entered information.
View projects	Works.	System displays all the ongoing/completed projects with ID, type, budget, start date, expected duration and status. It is possible to search for a project by ID, filter the projects by type, budget, start date, expected duration or status, delete or see all information about a project.
Open and manage project	Works.	It is possible to edit a project's details, update the resources or mark a project as ongoing or completed.
Get reports about completed projects	Works.	System generates and displays reports about the average expenses and the average duration for each project type.
View project progress	Works.	Website displays and updates the ongoing projects.
View completed projects	Works.	Website displays the ongoing projects.
View available project types	Works.	Website displays the available project types.
View contact information	Works.	Website displays the company’s contact information.

6. Results and Discussions

The implementation and testing phases resulted in the development of a project management system that satisfies all requirements:

Project and resources management

1) default values when the user add a project
2) the option to edit project's details
3) the option to edit project's resources
4) the option to update project's resources
5) display project's resources to let the user decide if the project is behind the schedule
6) the option to mark a project as behind the schedule
7) unique representation of a project in the system through unique ids
8) the option to delete a project
9) reports about the average expenses and duration for each project type
10) the option to mark a project as ongoing
11) the option to mark a project as completed
12) the option to search for a project by ID and filter the projects by type, budget, start date, duration or status
13) the option to sort the projects
14) display the number of projects

Advertising

15) the development of a website
16) display the progress of projects on the website
17) display the available project types of projects and some completed projects
18) display the company's contact information

Data storage

19) file persistence

The system uses efficient algorithms to implement all the functionalities and promotes task efficiency, reducing the number of steps required to accomplish common user goals. The design is intuitive, allowing the users to easily navigate and interact with the system, to find what they need without unnecessary difficulty.

The layout is clear and consistent throughout the interface, improving predictability and ease of use. The user interface provides clear and concise error messages, helping the user to understand and recover from any issues encountered. Colors and contrast are carefully chosen, to assure clarity, readability and create a visually pleasing environment.

7. Conclusions

The construction sector embraces technological innovations in order to reduce costs, improve the productivity in the construction site and the communication with the clients (Loosemore, 2004). Construction companies' general and particular problems were carefully

analyzed in order to clearly establish the requirements that must be covered when designing this project. The structure is complex, represented by multiple interconnected classes, each serving for a well-defined purpose, in order to implement the project management system and convert the design specifications into a functional software that satisfies user's needs. After many tests conducted to verify the functionality, the final version meets the user's requirements.

This project represents a management tool that helps the owner of the company to organize and administer the resources needed for the completion of a project, track its progress and access reports about the company's performances.

8. Sources of information

Atkinson, R., Crawford, L., & Ward, S. (2006). *Fundamental uncertainties in projects and the scope of project management. International Journal of Project Management*, 24(8), 687-698,

Bass, L., Clements, P., & Kazman, R. (1998). *Software Architecture in Practice*. Reading, MA.: Addison-Wesley.

Bob the Builder Wiki. (n.d.). *Scrambler/Gallery*. Retrieved from https://btb.fandom.com/wiki/Scrambler/Gallery

Chloe. (2021). *Everything You Need To Know About Commercial Architecture* -. Pro Arkitects. https://www.proarkitects.co.uk/everything-you-need-to-know-about-commercial-architecture/

Cockburn, A. (1997). *Structuring Use Cases with Goals. Journal of Object-Oriented Programming*, Sep-Oct, and Nov-Dec. SIGS Publications.

Cursors4u. *Hammer Cursor*. (n.d.). Www.cursors-4u.com. Retrieved from https://www.cursors-4u.com/cursor/2009/09/02/hammer-6.html#google_vignette

Drucker P.F. (1967). *The effective executive*. HarperCollins Publishers, New York. (p. 2).

Ellims, M., Bridges, J. & Ince, D.C. (2004). *Unit testing in practice*. Proceedings - International Symposium on Software Reliability Engineering, ISSRE. 3- 13. DOI:10.1109/ISSRE.2004.44

Flickr. (2019). *Bob el constructor*. Animados 2, D. https://www.flickr.com/photos/156142221@N02/46986149844

Fowler, M. (1996). *Analysis Patterns: Reusable Object Models*. Reading, MA.: Addison-Wesley.

Freepik. (n.d.). *Premium Photo | The local structure of urban modern architecture and the background of science and technology*. Retrieved from https://www.freepik.com/premium-photo/local-structure-urban-modern-architecture-background-science-technology_1346589.htm#from_view=detail_alsolike

Hassenzahl, M. (2008). *User experience (UX): Towards an experiential perspective on product quality*. New York: Association for Computing Machinery. https://doi.org/10.1145/1512714.1512717 [GS Search]

Heroes Wiki. (n.d.). *Muck*. https://hero.fandom.com/wiki/Muck

Hopping, L. (n.d.). *Modern Homes in Colorado (33 Photos)*. Dwell. Retrieved from https://www.dwell.com/collection/modern-homes-in-colorado-8a8bdb99

HowStuffWorks. (2008).*How Urban Landscape Design Works*. https://home.howstuffworks.com/lawn-garden/professional-landscaping/urban-landscape-design.htm

IStock, (2021). *Effective collaboration makes for quick turnaround time*. https://www.istockphoto.com/photo/shot-of-a-group-of-builders-having-a-meeting-at-a-construction-site-gm1327011873-411501878

Jacobson, I. (1992). *Object-Oriented Software Engineering: A Use Case Driven Approach.* Reading, MA.: Addison-Wesley.

Larman, C. (2004). *Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development*,

Loosemore, M. (2014), *Improving construction productivity: a subcontractor's perspective*, Engineering, Construction and Architectural Management, Vol. 21 No. 3, pp. 245-260.

Martin, J., and Odell, J. (1995). *Object-Oriented Methods: A Foundation*. Englewood (Miffs, NJ.: Prentice-Hall.

Oberoi, A. (2013). *The history of online advertising*. https://www.adpushup.com/blog/the-history-of-online-advertising/#Social_Media_Advertising

Oracle, n. d., *ArrayDeque* (Java Platform SE 7). Retrieved from https://docs.oracle.com/javase/7/docs/api/java/util/ArrayDeque.html

Oracle, n. d., *Comparator* (Java Platform SE 7). Retrieved from https://docs.oracle.com/javase/7/docs/api/java/util/Comparator.html

Oracle, n. d., *Arrays* (Java Platform SE 7). Retrieved from https://docs.oracle.com/javase/7/docs/api/java/util/Arrays.html#sort(java.lang.Object[])

Oracle, n. d. , *ArrayList* (Java Platform SE 8). Retrieved from https://docs.oracle.com/javase/8/docs/api/java/util/ArrayList.html

Pooh's Adventures Wiki. (n.d.). *Skip (Bob the Builder)*. Retrieved from https://poohadventures.fandom.com/wiki/Skip_(Bob_the_Builder)

PxHere. (n.d.). *Free Images : sea, coast, horizon, structure, sunrise, sunset, road, bridge, night, sunlight, morning, dawn, pier, walkway, cityscape, dusk, transportation, transport, evening, construction, modern, engineering, roadway, infrastructure, maryland, architectural, architecture design 4236x2814 - - 1108088 - Free stock photos -* Pxhere.com. Retrieved from https://pxhere.com/en/photo/1108088

Reis, S., Metzger, A. & Pohl, K. (2007). *Integration Testing in Software Product Line Engineering: A Model-Based Technique*. 321-335. DOI:10.1007/978-3-540-71289-3_25

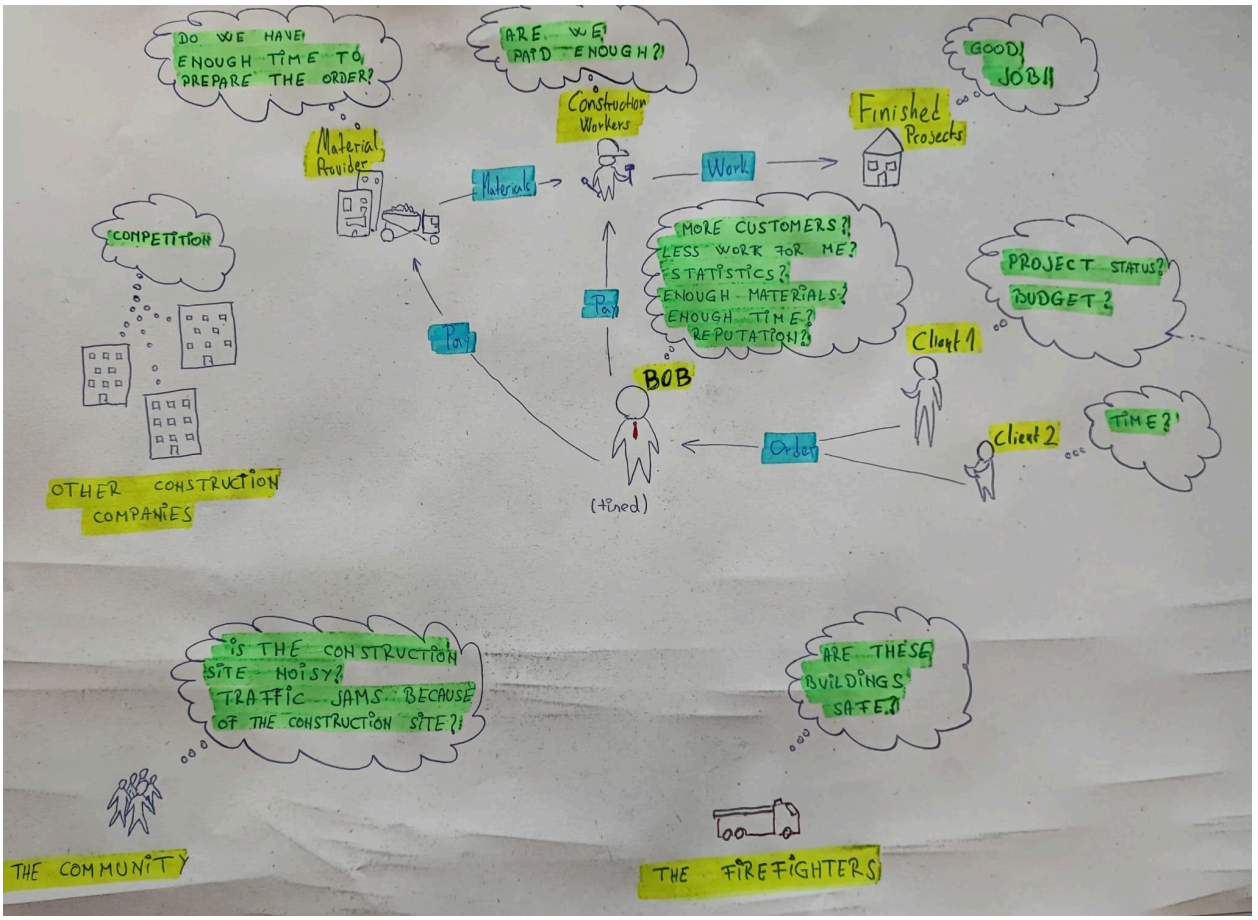
The Economist. (2017). *The construction industry's productivity problem.* https://www.economist.com/leaders/2017/08/17/the-construction-industrys-productivity-problem

The Opus Group. (n.d.). *Construction Begins on Industrial Build-to-Suit in Ankeny, Iowa*. Retrieved from https://www.opus-group.com/News/Opus-Begins-Construction-on-Industrial-Build-to-Suit-in-Ankeny-Iowa

Wood H. (2012). *UK construction careers, certifications/degrees and occupations*.https://web.archive.org/web/20120304075330/http://blog.thservices.co.uk/uk-construction-careers-certificationsdegrees-and-occupations/

9. Appendices

Appendix A - The Rich Picture



Appendix B - Use Case Descriptions

Name	View projects
Summary	System displays all the ongoing/completed projects with ID, type, budget, start date, expected duration and status, allows the actor to search for a project by ID, filter the projects by type, budget, start date, expected duration or status, delete or see all information about a project.
Actor	Owner
Precondition	The project has already been added and stored in the system.
Postcondition	A project has been opened, deleted, searched by ID, or the list of projects was filtered by type, budget, start date, duration and status.
Base Sequence	1. System shows the list of ongoing/completed projects. For each project, the ID, Type, Budget, Start date, Expected duration and Status are shown. 2. System calculates and displays the number of projects in the list. 3. System allows the actor to sort the projects by ID (alphabetically), type (alphabetically), status (alphabetically), budget (numerically), duration (numerically) or start date (chronologically). 4. If SEARCH BY ID then go to Scenario A SEARCH BY ID (step A1.) If FILTER BY START DATE, go to Scenario B FILTER BY START DATE (step B1.) If FILTER BY STATUS, go to Scenario C FILTER BY STATUS (step C1.) If FILTER BY TYPE, go to Scenario D FILTER BY TYPE (step D1.) If FILTER BY BUDGET, go to Scenario E FILTER BY BUDGET (step E1.) If FILTER BY DURATION, go to Scenario F FILTER BY DURATION (step F1.) If CLEAR FILTERS, the applied filters are deselected. Go to Step 1. 5. Select a project from the list. **[ALT7]** 6. If OPEN PROJECT, the System displays all the information about the project: ID, Start date, Budget, Expected duration, Expected expenses, Estimated total hours, Status - if the project is behind the schedule, Materials expenses, Salary expenses, Man-hours used, Total expenses and the specific information: for residential projects: number of Kitchens/Bathrooms/other rooms with Plumbing, Size of the building; for commercial projects: the Size, the Use of the building and the number of Floors; for industrial projects: the Size and the Use of the building; for road construction: the Length and Width of the road, number of Bridges and Tunnels, the environmental Challenges. If DELETE PROJECT, the System asks for confirmation and deletes the project from the System and files.**[ALT8].**

	Scenario A SEARCH BY ID — A1. Enter ID. A2. The system displays the project with the entered ID.**[ALT1]** A3. Go to Step 2.
	Scenario B FILTER BY START DATE — B1. Enter the Start date interval. B2. Select the order {Ascending, Descending}. B3.The system displays all the projects with their start date in the entered interval, in the specified order.**[ALT2]** B4. Go to Step 2.
	Scenario C FILTER BY STATUS — C1. Select Status {Behind the schedule, Not behind the schedule} C2. The system displays all projects with the same status.**[ALT3]** C3. Go to Step 2.
	Scenario D FILTER BY TYPE — D1. Select the Type from the list {Residential, Commercial, Industrial, Road Construction}. D2. The system displays all the projects with the selected types.**[ALT4]** D3. Go to Step 2.
	Scenario E FILTER BY BUDGET — E1. Enter the Budget interval. E2. The system displays all the projects with their Budget in the entered interval.**[ALT5]** E3. Go to Step 2.
	Scenario F FILTER BY DURATION — F1. Enter the Expected duration interval. F2. The system displays all the projects with their Expected Duration in the entered interval.**[ALT6]** F3. Go to Step 2.
	Use case ends.
Alternate Sequence	**[*ALT0]** The process can be canceled in any step with user interaction, after confirming the intention to cancel. **[ALT1]** If the entered ID doesn't match any project's ID, then the System shows an error message and displays an empty list. Go to Step 2. **[ALT2]** If there are no projects with their Start date in the specified interval, then the System shows an error message and displays an empty list. Go to Step 2. **[ALT3]** If there are no projects with the specified Status, then the System shows an empty list. Go to Step 2. **[ALT4]** If there are no projects of the same Type as those selected, then the System shows an empty list. Go to Step 2.

	[ALT5] If there are no projects with their Budget in the specified interval, then the System shows an error message and displays an empty list. Go to Step 2. **[ALT6]** If there are no projects with their Expected duration in the specified interval, then the System shows an error message and displays an empty list. Go to Step 2. **[ALT7]** If the list is empty, no project can be selected. If no project has been selected, the System will display an error message. Go to Step 2. **[ALT8]** If the actor doesn't confirm the deletion, the project is not deleted. Go to Step 6.
Notes	This Use Case covers requirements 8), 12), 13), 14), 19).

Name	Open and manage project
Summary	The actor is allowed to edit project's details (budget, expected duration, expected expenses, estimated total hours, status - if the project is behind the schedule and the specific information: for residential projects: number of kitchens/bathrooms/other rooms with plumbing, size of the building; for commercial projects: the size and the purpose of the building and the number of floors; for industrial projects: the size and the use of the building; for road construction: the length and width of the road, number of bridges and tunnels, the environmental challenges), update the resources (materials expenses, salary expenses, man-hours used, total expenses) and mark a project as ongoing or completed.
Actor	Owner
Precondition	The project has already been added and stored in the system.
Postcondition	The project has been updated in terms of expenses, time, progress and details, marked as completed or ongoing.
Base Sequence	1. System displays all the information about the project: ID, Start date, Budget, Expected duration, Expected expenses, Estimated total hours, Status - if the project is behind the schedule, Materials expenses, Salary expenses, Man-hours used, Total expenses and the specific information: for residential projects: number of Kitchens/Bathrooms/other rooms with Plumbing, Size of the building; for commercial projects: the Size, the Use of the building and the number of Floors; for industrial projects: the Size and the Use of the building; for road construction: the Length and Width of the road, number of Bridges and Tunnels, the environmental Challenges. 2. If EDIT DETAILS then go to Scenario A. EDIT DETAILS (step A1.) If the project is ongoing If MARK AS COMPLETED, then go to Scenario B. MARK AS COMPLETED(step B1.) If the project is completed If MARK AS ONGOING, then go to Scenario C. MARK AS ONGOING(step C1.) 3. If the project is ongoing If UPDATE RESOURCES then go to Scenario D. UPDATE RESOURCES (step D1.)

	Scenario A EDIT DETAILS	A1. System allows the actor to edit information for Budget, Expected duration, Expected expenses, Estimated total hours and the specific information: for residential projects: number of Kitchens/Bathrooms/other rooms with Plumbing, Size of the building; for commercial projects: the Size and the Use of the building and the number of floors; for industrial projects: the Size and the Use of the building; for road construction: the Length and Width of the road, number of Bridges and Tunnels, the environmental Challenges. A2. Approve the changes.**[ALT1]** A3. System validates the data. **[ALT2]** A3. System updates the project in the System and files.
	Scenario B MARK AS COMPLETED	B1. System adds the project to the list of completed projects in the system and files. B2. System removes the project from the list of ongoing projects in the system and files. Use case ends.
	Scenario C MARK AS ONGOING	C1. System adds the project to the list of ongoing projects in the system and files. C2. System removes the project from the list of completed projects in the system and files. Use case ends.
	Scenario D UPDATE RESOURCES	D1. System allows the actor to enter values for Man hours used, Salary expenses and Materials expenses along with a short note for each value. D2. Approve the entered data.**[ALT3]** D3. System validates the data.**[ALT4]** D4. If the entered values are negative, the System will ask for confirmation.**[ALT5]** D5. System updates the values of Man hours used, Salary expenses, Materials expenses and Total expenses in the System and files. D6. System adds the update to the list of previous updates and displays the list. D7. Go to Step 1.
	Use case ends.
Alternate Sequence	**[*ALT0]** The process can be cancelled in any step with user interaction, after confirming the intention to cancel. The unapproved changes will not be saved.

	[ALT1] If the entered data is not approved by the actor, the changes are not saved. Go to Step 1. **[ALT2]** If the Budget is not given as a positive number, then the System will show an error message. Go to Step A1. If the Expected duration is not given as a positive number, then the System will show an error message. Go to Step A1. If Estimated total hours is not given as a positive number, then the System will show an error message. Go to Step A1. If Expected expenses is not given as a positive number, then the System will show an error message. Go to Step A1. If the project is “Residential” and if the Size is not given as a positive number, then the System will show an error message. Go to Step A1. If the project is “Residential” and the number of Kitchens is not given as a non-negative number, then the System will show an error message. Go to Step A1. If the project is “Residential” and the number of Bathrooms is not given as a non-negative number, then the System will show an error message. Go to Step A1. If the project is “Residential” and the number of other rooms with Plumbing is not given as a non-negative number, then the System will show an error message. Go to Step A1. If the project is “Commercial” and if the Size is not given as a positive number, then the System will show an error message. Go to step A1. If the project is “Commercial” and if the number of Floors is not given as a positive number, then the System will show an error message. Go to Step A1. If the project is “Commercial” and the intended Use of the building is not given, then the System will show an error message. Go to Step A1. If the project is “Industrial” and if the Length is not given as a positive number, then the System will show an error message. Go to Step A1. If the project is “Industrial” and if the Size is not given as a positive number, then the System will show an error message. Go to Step A1. If the project is “Industrial” and the Use of industrial facility is not given, then the System will show an error message. Go to Step A1.

	If the project is “Road Construction” and if the Width is not given as a positive number, then the System will show an error message. Go to Step A1. If the project is “Road Construction” and if the number of Bridges is not given as a non-negative number, then the System will show an error message. Go to step A1. If the project is “Road Construction” and if the number of Tunnels is not given as a non-negative number, then the System will show an error message. Go to step A1. **[ALT3]** If the entered data is not approved by the actor, the values of Man hours used, Salary expenses, Materials expenses and Total expenses are not updated. Go to Step 1. **[ALT4]** If the entered values for Man hours used, Salary expenses and Materials expenses are not given as numbers, then the System will show an error message. Go to Step D1. **[ALT5]** If the entered negative values are not approved by the actor, the values of Man hours used, Salary expenses, Materials expenses and Total expenses are not updated. Go to Step 1.
Notes	This Use Case covers requirements 2), 3), 4), 5), 6), 10), 11), 19).

Name	Get reports about the completed projects
Summary	Generate and display reports about the average expenses and the average duration for each project type.
Actor	Owner
Precondition	None
Postcondition	The average expenses and The average time spent are calculated using all the similar completed projects.
Base sequence	1. System calculates Average expenses for each project type by dividing the sum of all completed projects’ Total Expenses by the number of completed projects with the same type. 2. System displays Average expenses for each project type.**[ALT1]** 3. System calculates Average duration for each project type by dividing the sum of all completed projects’ Expected duration by the number of completed projects with the same type. 4. System displays Average duration for each project type.**[ALT2]** 5. System displays the number of projects of each type. Use case ends.

Alternate sequence	**[*ALT0]** The process can be cancelled in any step with user interaction ending the use case. **[ALT1]** If there are no projects of a specific Type to calculate Average expenses, the System will display a message that no data is available. **[ALT2]** If there are no projects of a specific Type to calculate Average duration, the System will display a message that no data is available.
Notes	This Use Case covers requirement 9).

Name	View project progress
Summary	Allows the actor to view their project's progress on the website.
Actor	Customer
Precondition	There must be ongoing projects
Postcondition	The actor is able to see all the available information about the project (ID, type, expected expenses, man-hours used, estimated total hours and status).
Base sequence	1. System shows all ongoing projects and their details: ID, Type, Expected expenses, Man-hours used, Estimated total hours and Status. Use case ends.
Alternate sequence	**[*ALT0]** The process can be cancelled in all steps with user interaction. Use case ends.
Notes	Viewing ongoing projects is available for everyone to see. For a project to show up in this section of the website it has to be added as a new project in the system and not be marked as completed. This Use case covers requirements 15), 16).

Name	View completed projects
Summary	Allows the actor to see basic information (project type, total expenses and duration) about all the projects that the company has completed.
Actor	Customer
Precondition	There must be completed projects
Postcondition	The customer is able to see the information about completed projects (ID, project type, total expenses and expected duration).

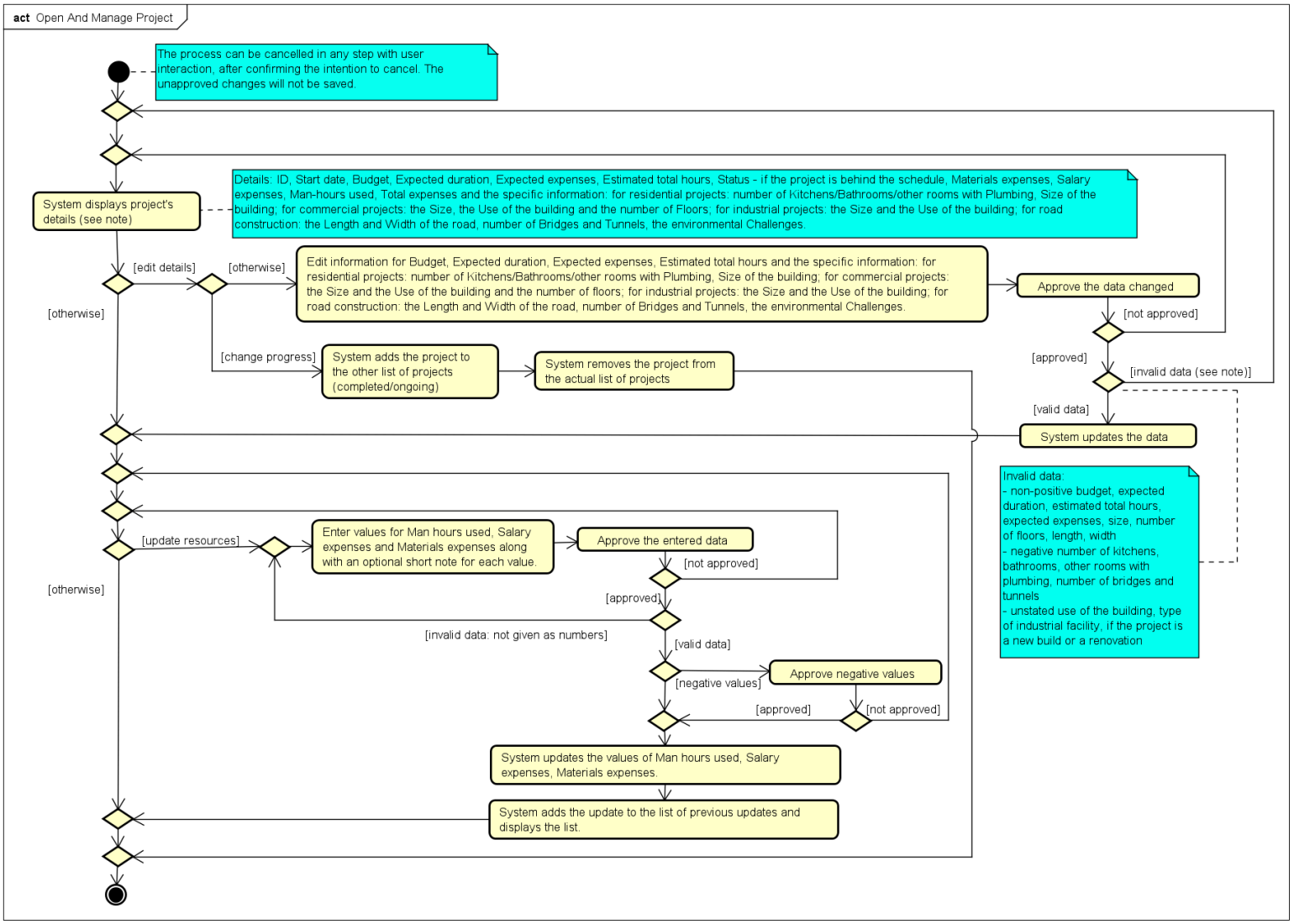
Base sequence	1. System shows the list of completed projects sorted by Type. For each project, the Total expenses and Expected duration are shown. Use case ends.
Alternate sequence	**[*ALT0]** The process can be cancelled in all steps with user interaction. Use case ends.
Notes	Viewing completed projects is available for everyone to see. For a project to show up in this section it has to be marked as completed in the system by the owner. This Use case covers requirements 15), 17).

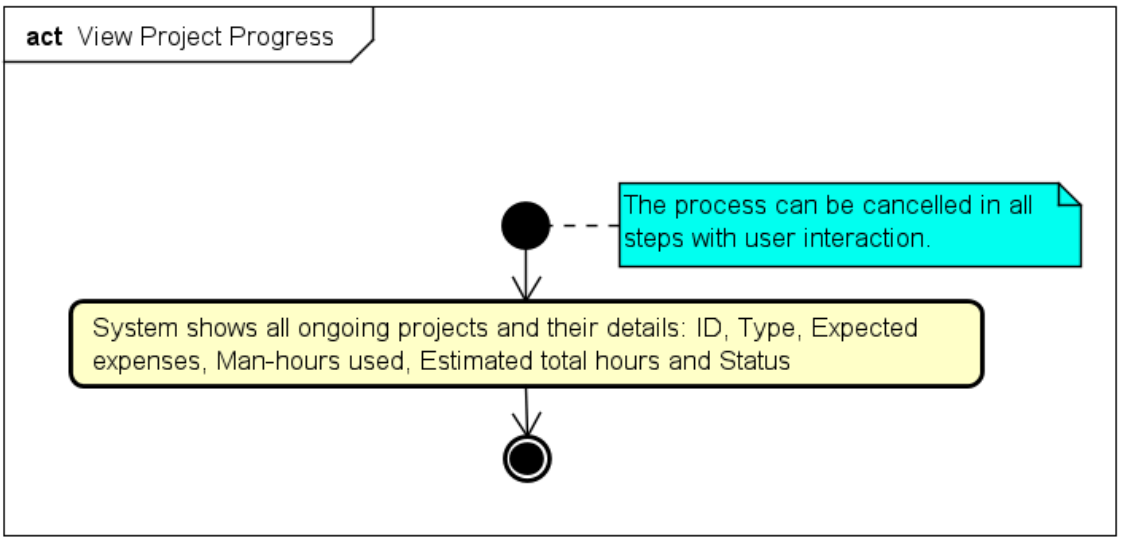
Name	View contact information
Summary	Allows customers to see company's contact information
Actor	Customer
Precondition	None
Postcondition	Customer can see the contact information displayed.
Base sequence	1. System displays the company's contact information(name of the employee, their function and their mail address). Use case ends.
Alternate sequence	**[*ALT0]** The process can be cancelled in all steps with user interaction. Use case ends.
Notes	The contact information is available for everyone to see. This use case covers requirements 15) and 18).

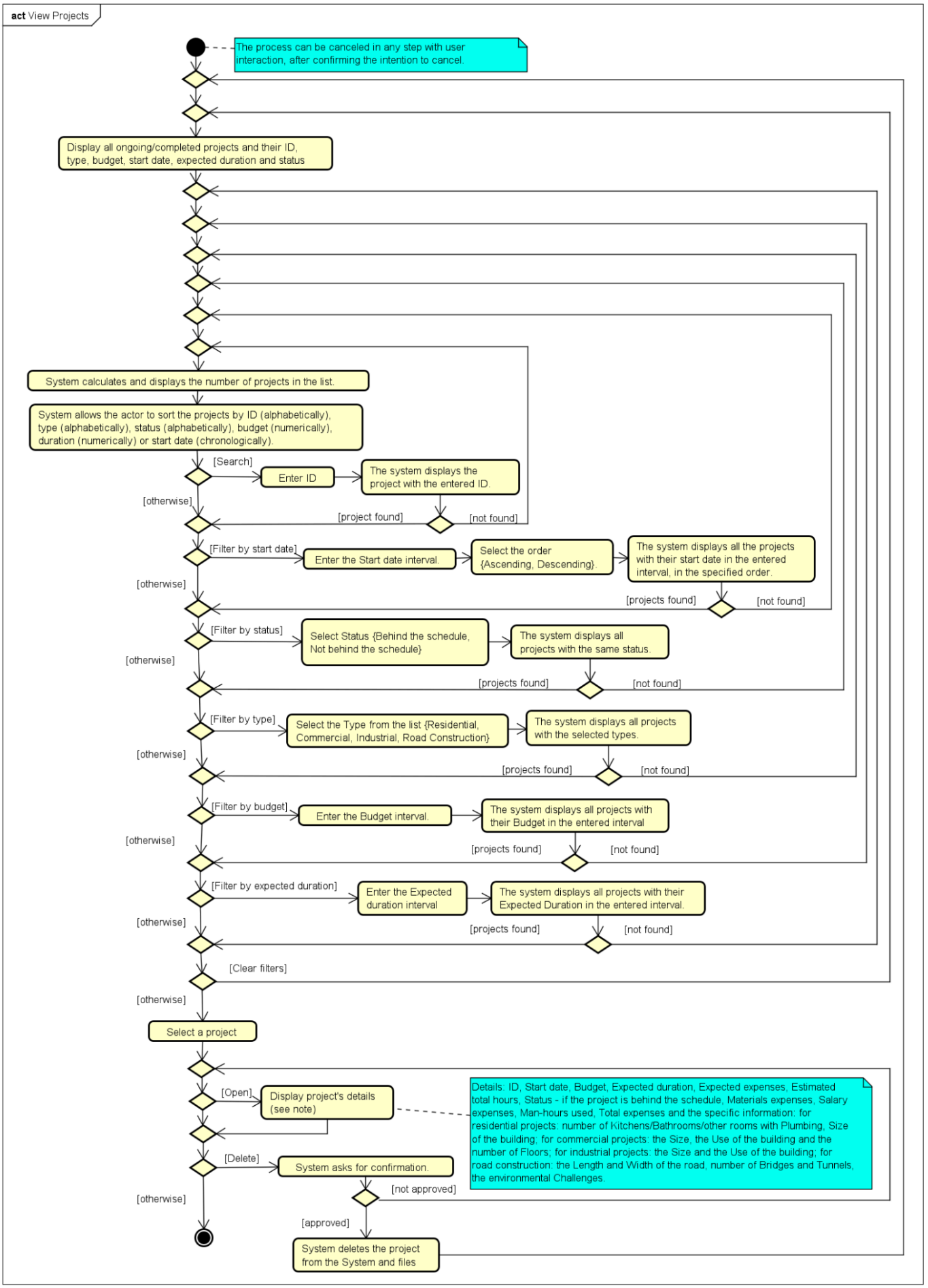
Name	View available project types
Summary	Allows the customers to see the company's available project types and an approximate price range for each type.
Actor	Customer
Precondition	None.
Postcondition	Customer can see the company's available project types and an approximate price range

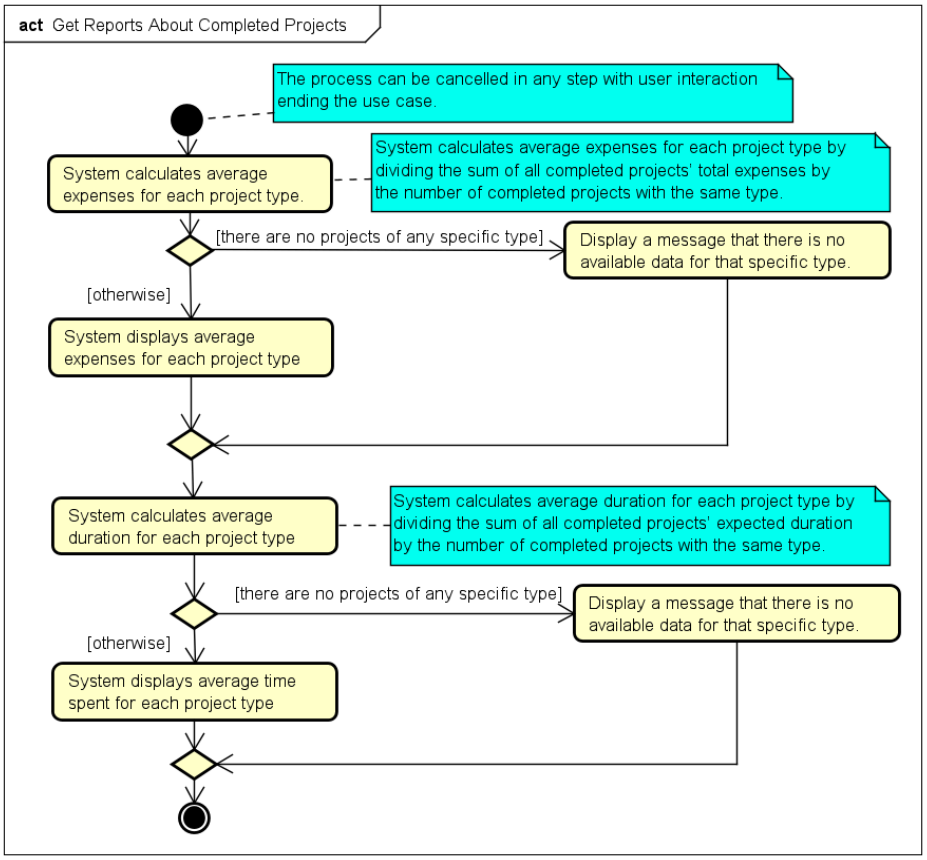
	for each type.
Base sequence	1. System displays the company's available project types (Residential, Commercial, Industrial and Road Construction) and an approximate price range for each project type. Use case ends.
Alternate sequence	**[*ALT0]** The process can be cancelled in all steps with user interaction. Use case ends.
Notes	The available project types are available for everyone to see. This use case covers requirements 15) and 17).

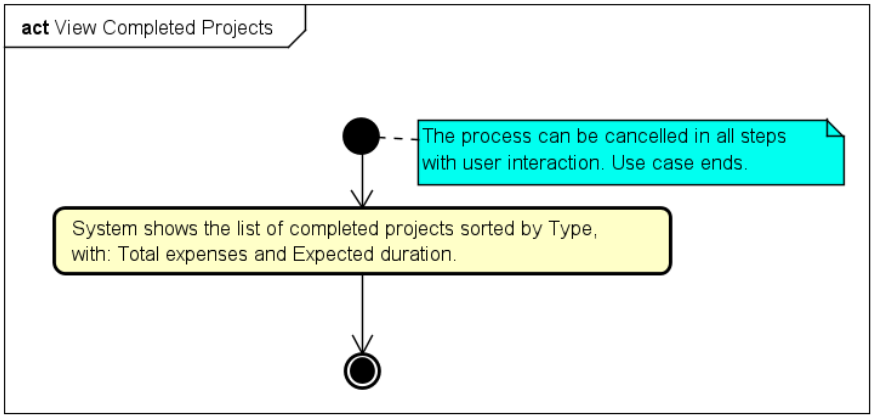
Appendix C - Activity Diagrams

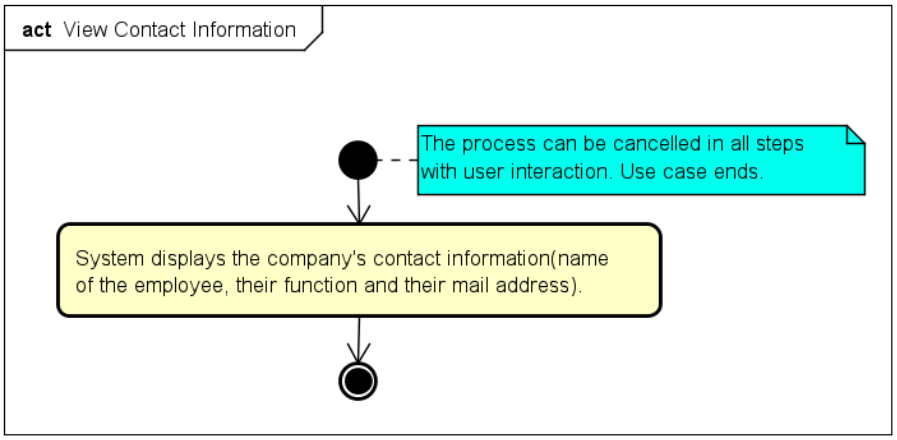


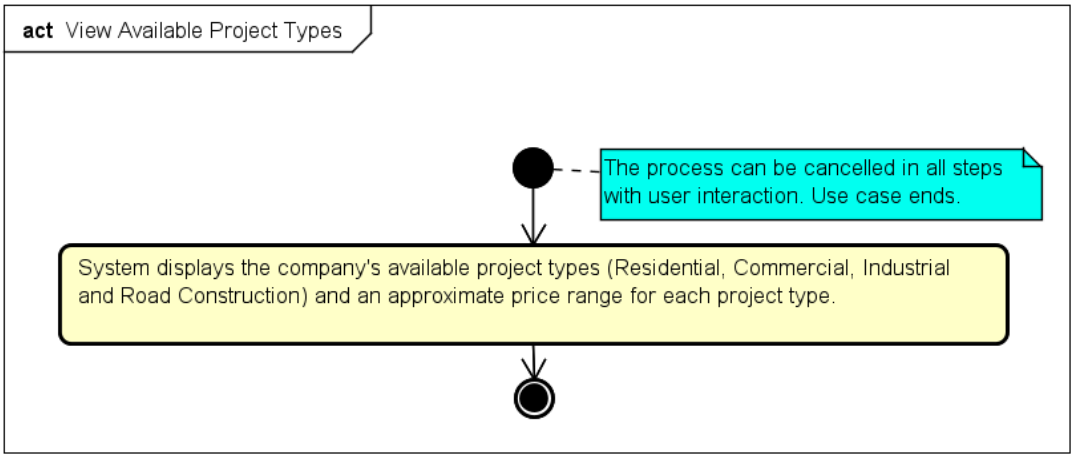



act View Projects
The process can be canceled in any step with user interaction, after confirming the intention to cancel.
Display all ongoing/completed projects and their ID, type, budget, start date, expected duration and status
System calculates and displays the number of projects in the list.
System allows the actor to sort the projects by ID (alphabetically), type (alphabetically), status (alphabetically), budget (numerically), duration (numerically) or start date (chronologically).
[Search]
Enter ID
The system displays the project with the entered ID.
[otherwise]
[project found]
[not found]
[Filter by start date]
Enter the Start date interval.
Select the order {Ascending, Descending}.
The system displays all the projects with their start date in the entered interval, in the specified order.
[otherwise]
[projects found]
[not found]
[Filter by status]
Select Status {Behind the schedule, Not behind the schedule}
The system displays all projects with the same status.
[otherwise]
[projects found]
[not found]
[Filter by type]
Select the Type from the list {Residential, Commercial, Industrial, Road Construction}
The system displays all projects with the selected types.
[otherwise]
[projects found]
[not found]
[Filter by budget]
Enter the Budget interval.
The system displays all projects with their Budget in the entered interval
[otherwise]
[projects found]
[not found]
[Filter by expected duration]
Enter the Expected duration interval
The system displays all projects with their Expected Duration in the entered interval.
[otherwise]
[projects found]
[not found]
[Clear filters]
[otherwise]
Select a project
[Open]
Display project's details (see note)
Details: ID, Start date, Budget, Expected duration, Expected expenses, Estimated total hours, Status - if the project is behind the schedule, Materials expenses, Salary expenses, Man-hours used, Total expenses and the specific information: for residential projects: number of Kitchens/Bathrooms/other rooms with Plumbing, Size of the building; for commercial projects: the Size, the Use of the building and the number of Floors; for industrial projects: the Size and the Use of the building; for road construction: the Length and Width of the road, number of Bridges and Tunnels, the environmental Challenges.
[Delete]
System asks for confirmation.
[not approved]
[otherwise]
[approved]
System deletes the project from the System and files

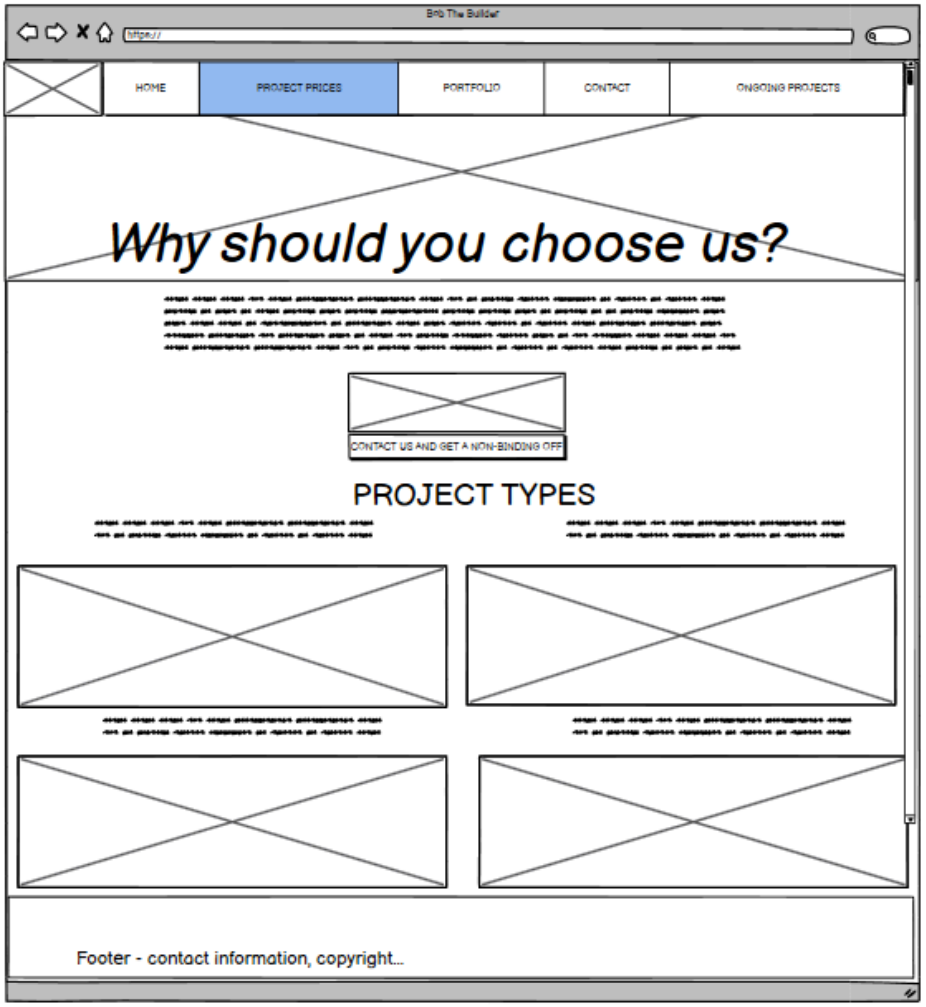

act Get Reports About Completed Projects
The process can be cancelled in any step with user interaction ending the use case.
System calculates average expenses for each project type.
System calculates average expenses for each project type by dividing the sum of all completed projects' total expenses by the number of completed projects with the same type.
[there are no projects of any specific type]
Display a message that there is no available data for that specific type.
[otherwise]
System displays average expenses for each project type
System calculates average duration for each project type
System calculates average duration for each project type by dividing the sum of all completed projects' expected duration by the number of completed projects with the same type.
[there are no projects of any specific type]
Display a message that there is no available data for that specific type.
[otherwise]
System displays average time spent for each project type

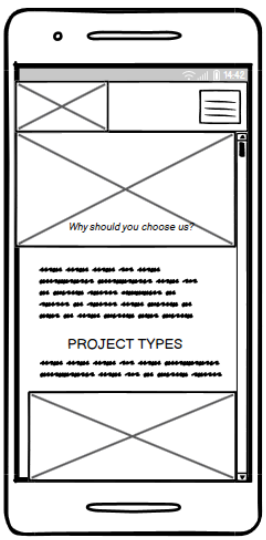

act View Completed Projects
The process can be cancelled in all steps with user interaction. Use case ends.
System shows the list of completed projects sorted by Type, with: Total expenses and Expected duration.

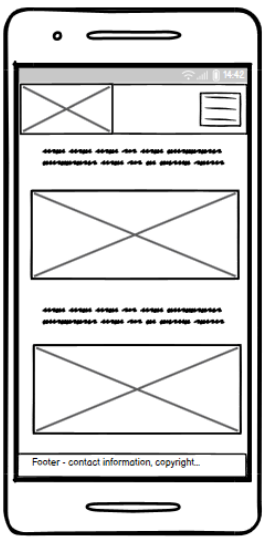

act View Contact Information
The process can be cancelled in all steps with user interaction. Use case ends.
System displays the company's contact information(name of the employee, their function and their mail address).

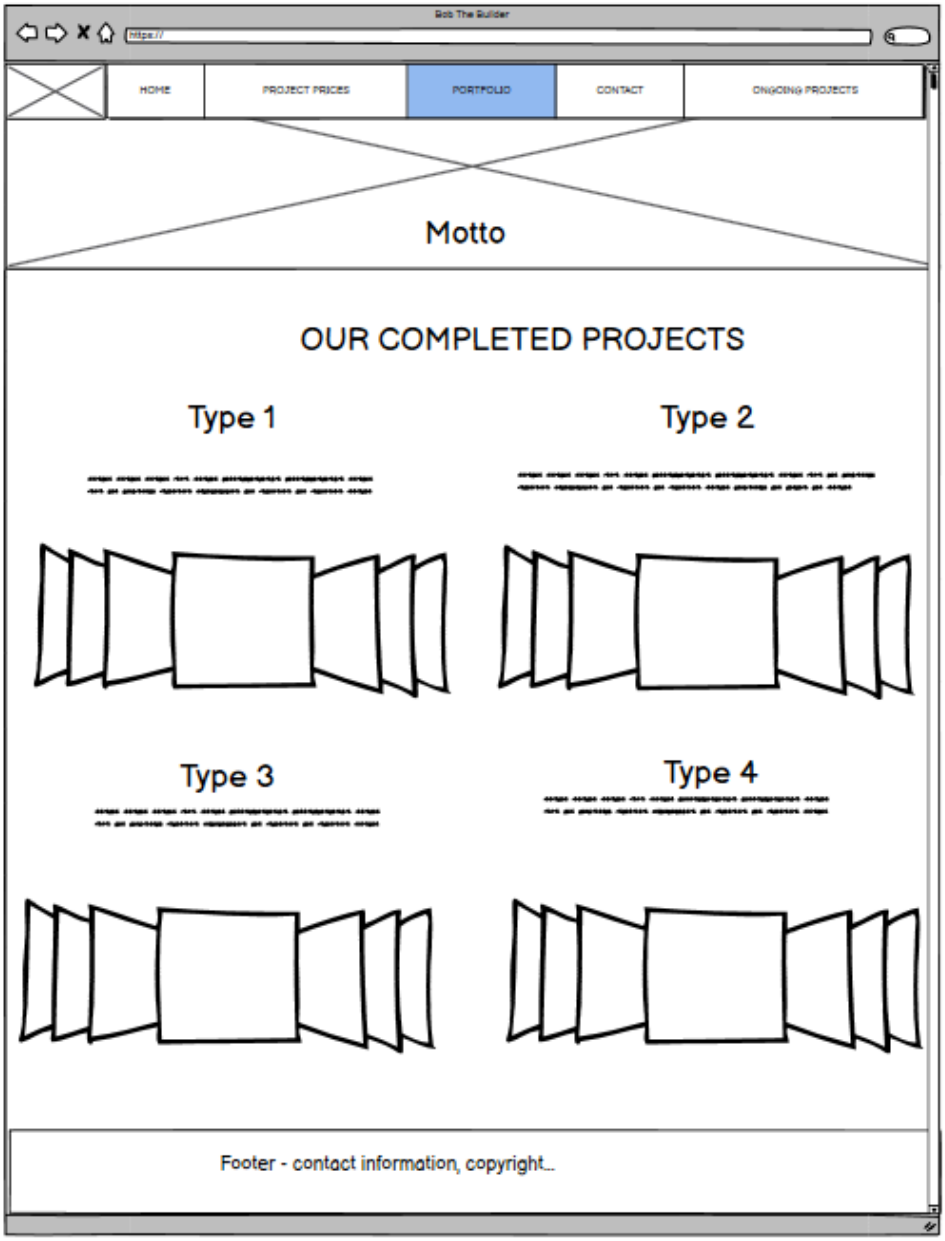

act View Available Project Types
The process can be cancelled in all steps with user interaction. Use case ends.
System displays the company's available project types (Residential, Commercial, Industrial and Road Construction) and an approximate price range for each project type.

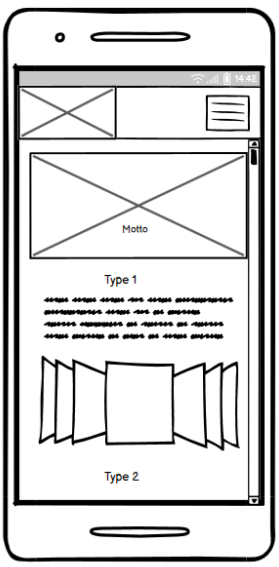
Appendix D - Website's Wireframes

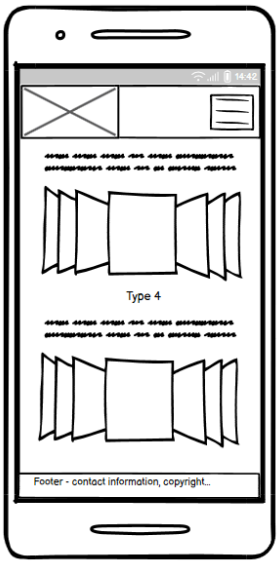


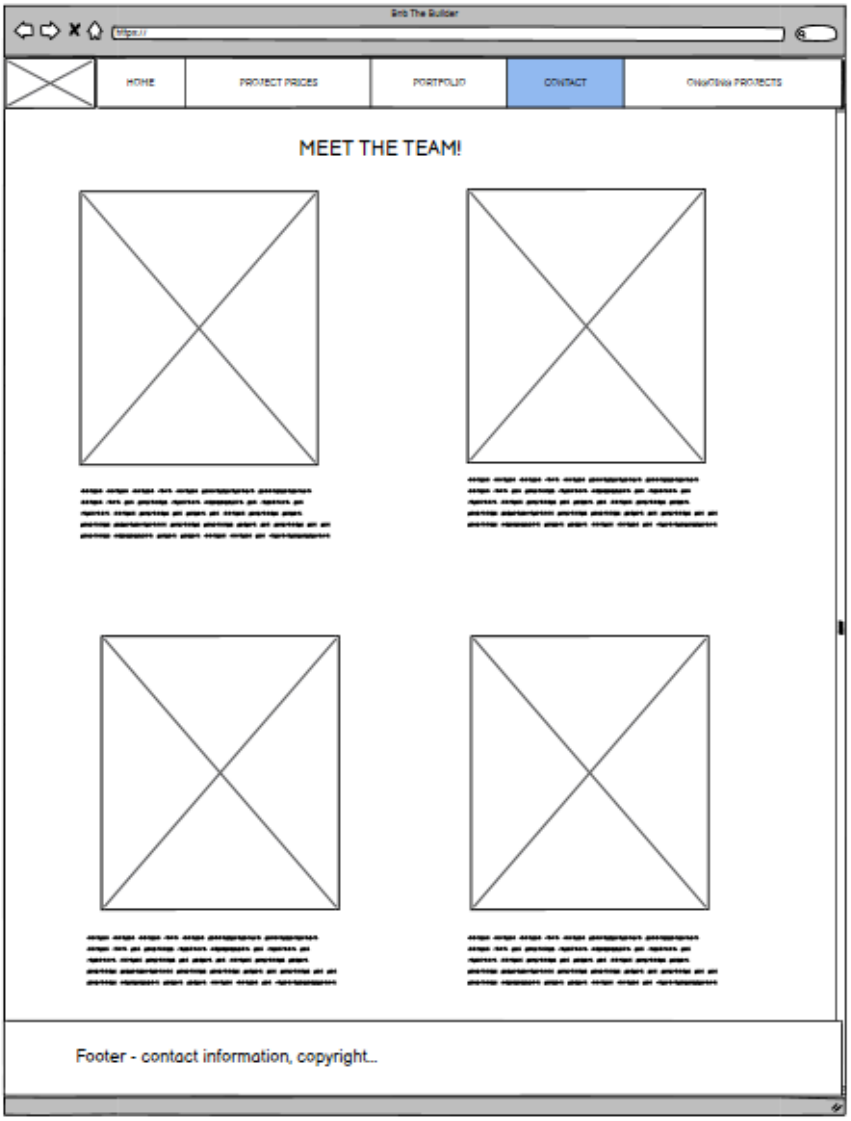


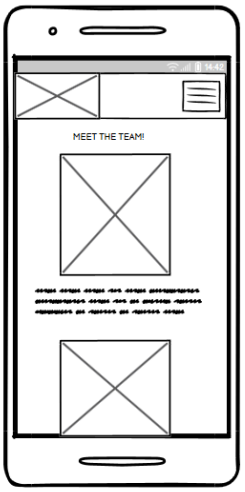


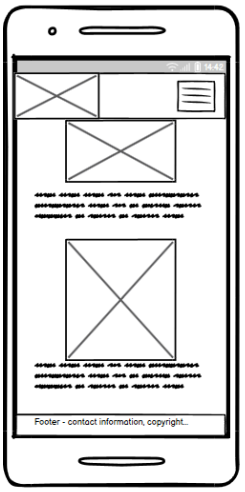

Bob The Builder
https://
HOME
PROJECT PRICES
PORTFOLIO
CONTACT
ONGOING PROJECTS
Motto
OUR COMPLETED PROJECTS
Type 1
Type 2
Type 3
Type 4
Footer - contact information, copyright...

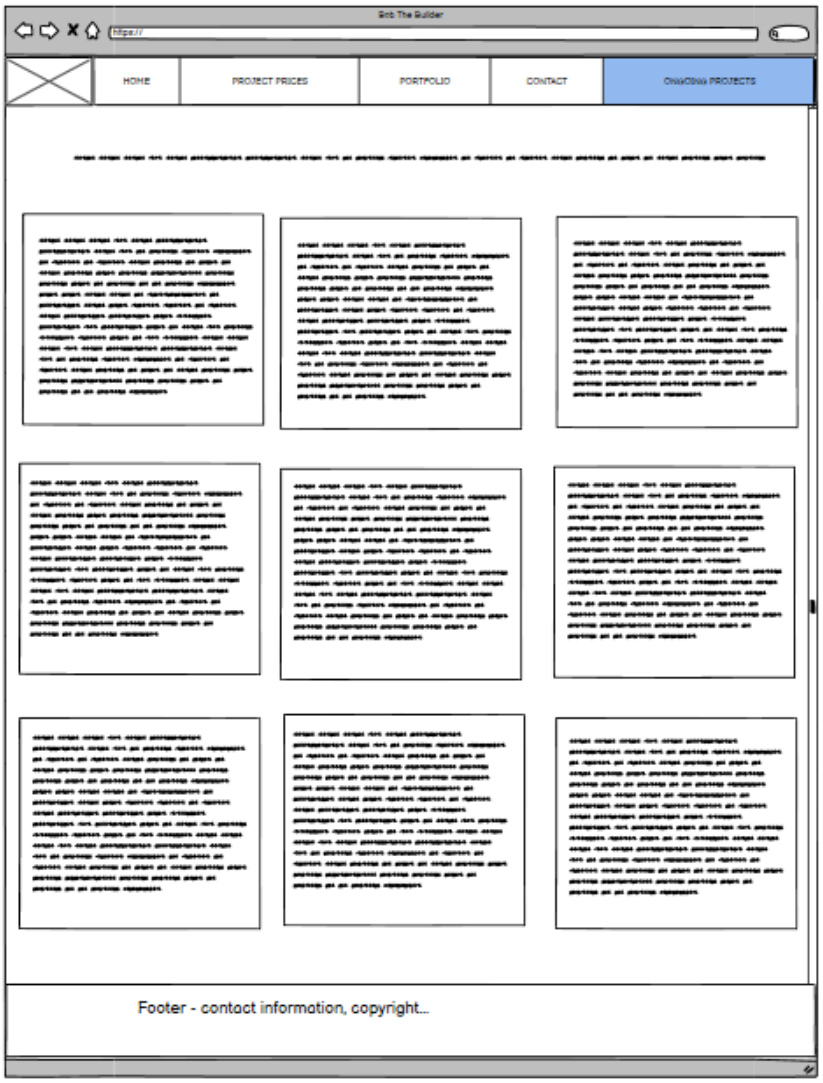

Motto
Type 1
Type 2

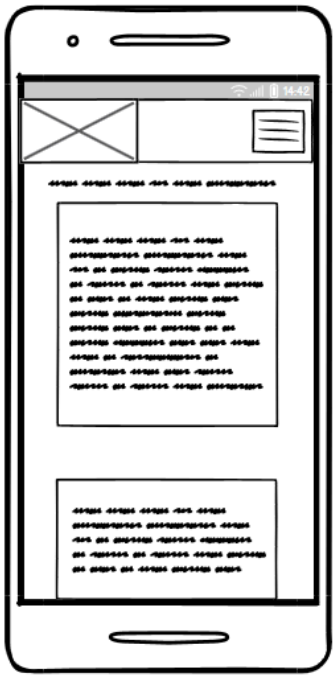

Type 4
Footer - contact information, copyright...

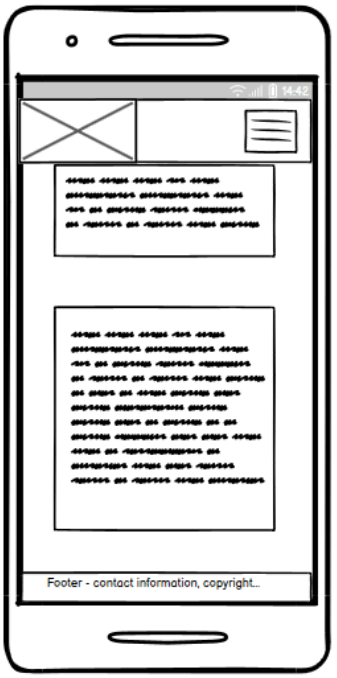

Bob The Builder
HOME
PROJECT PRICES
PORTFOLIO
CONTACT
MEET THE TEAM!
Footer - contact information, copyright...


MEET THE TEAM!


Footer - contact information, copyright...


Bob The Builder
HOME
PROJECT PRICES
PORTFOLIO
CONTACT
ONGOING PROJECTS
Footer - contact information, copyright...




Footer - contact information, copyright...

Appendix E - The User Manual

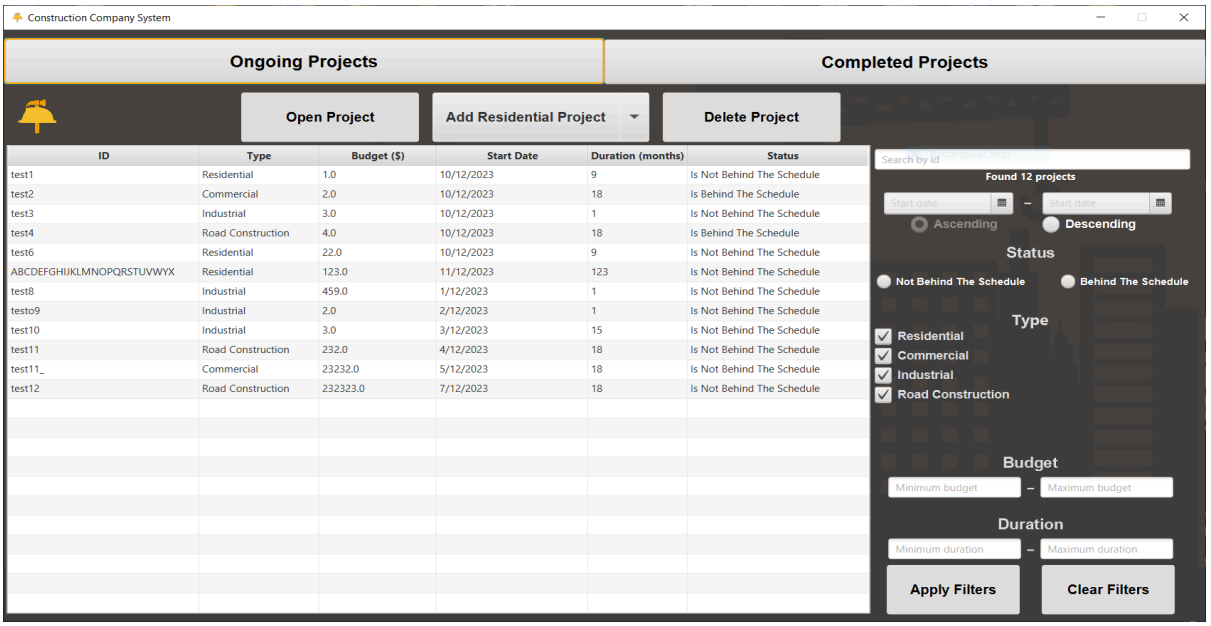
Bob the Builder Construction Company

~ User manual ~

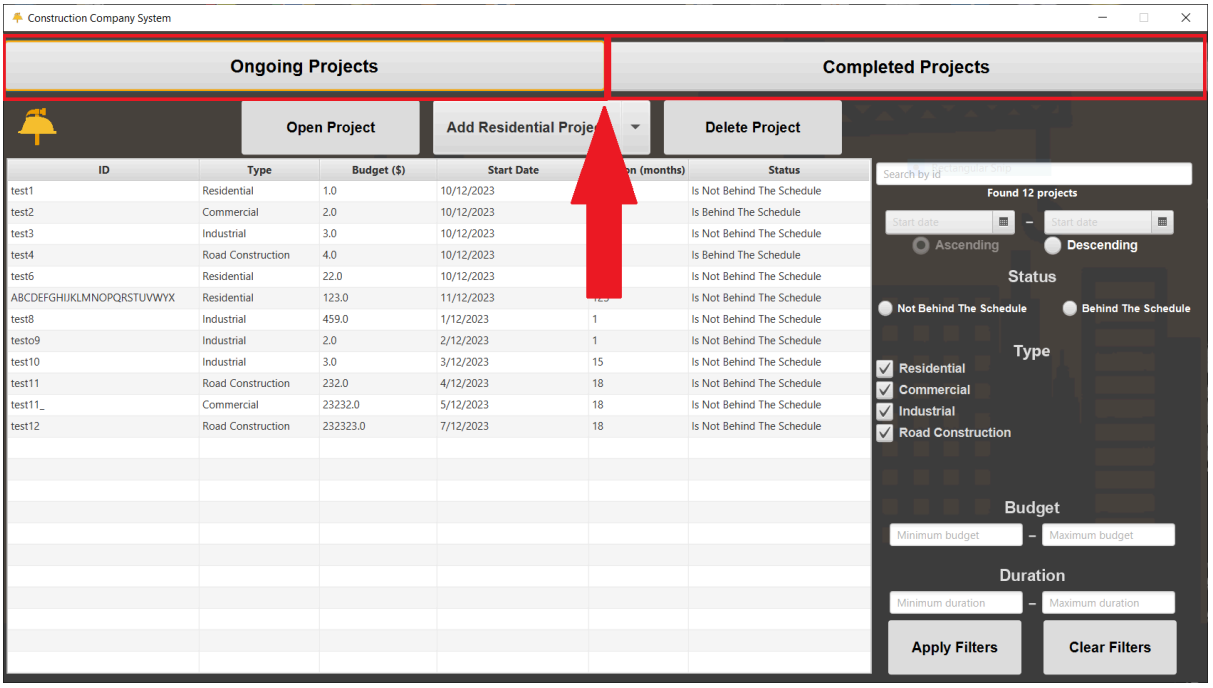
Table of contents

1. Display the projects

Once the program loads, you will be greeted by the view of the ongoing projects list. This is the default view that will load every time you reopen the application. From here, you can select a few options. You can add a new project, open an existing one to get more details on it, update or edit it or you can simply delete it from the system.



For now let's focus on displaying, the rest will be covered further in the manual. The two buttons at the top labeled *Ongoing Projects* and *Completed Projects* are your main navigation tools for displaying projects that are either ongoing or have been marked as completed. Every time you press one of them, the list of projects will adjust accordingly and the filters you have applied will also be cleared.



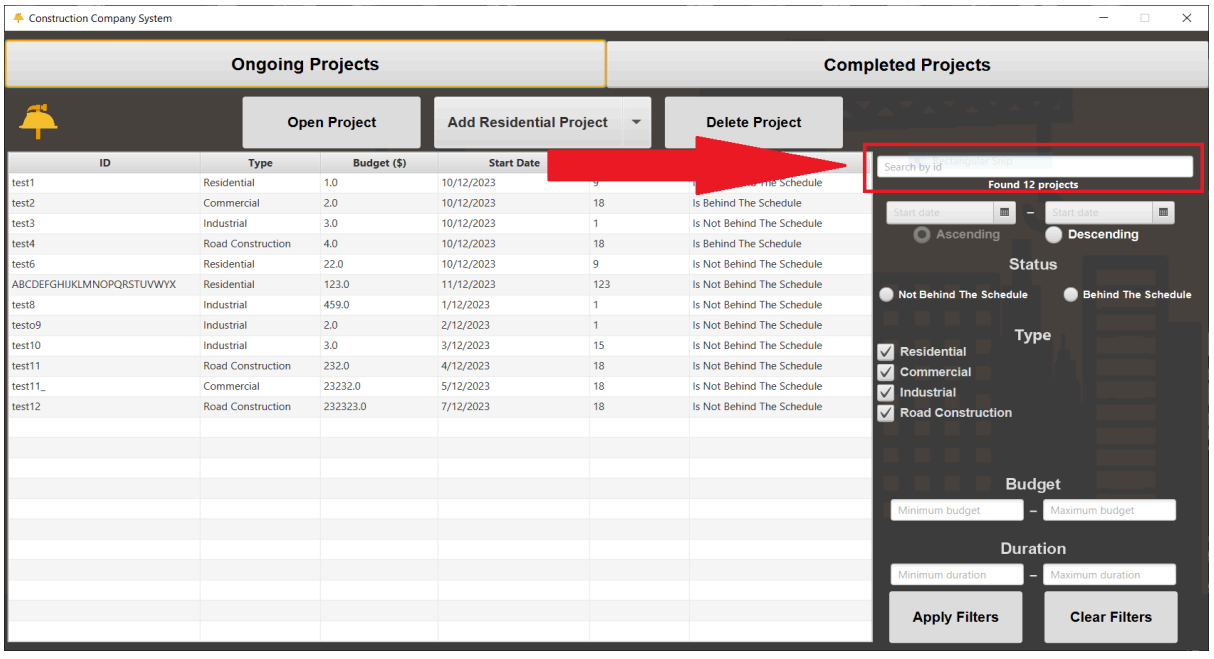
2. Filter the projects

Once you have selected either the ongoing or completed projects list you can then apply filters using the panel on the right, to search for more specific projects.

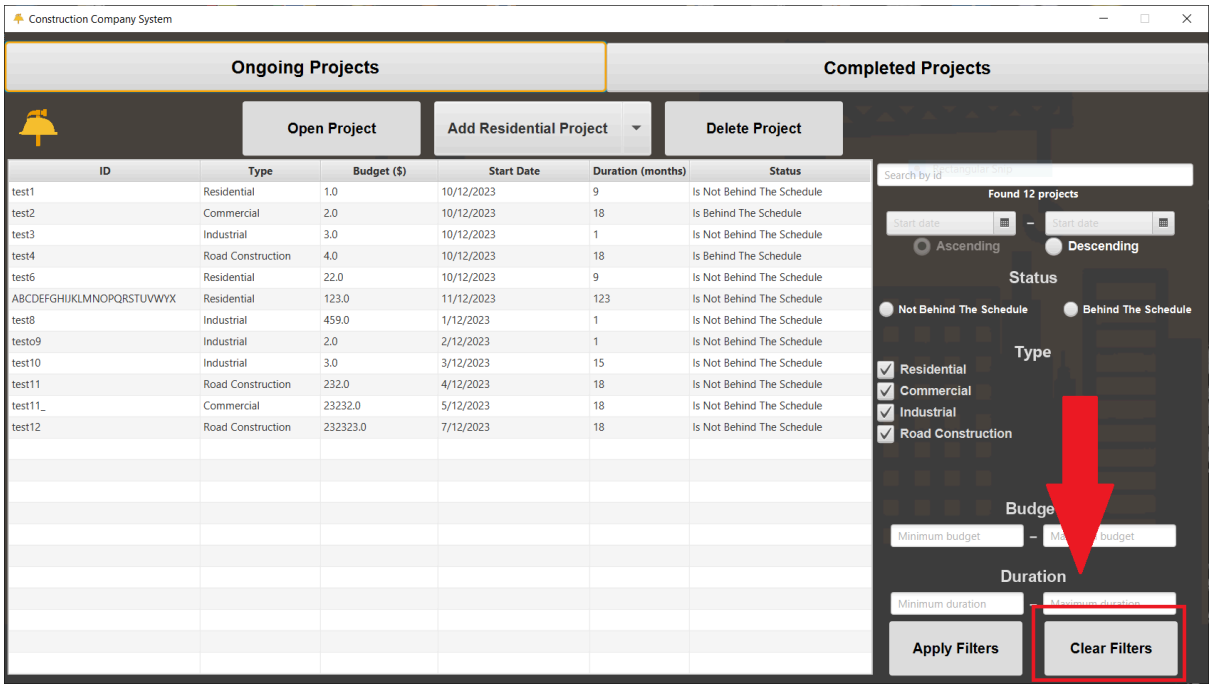
A. Search by ID

Step 1. Enter the ID in the field at the top panel.

Step 2. Press Enter.



If there are no projects with matching ID an error will be shown. To reload projects, press Enter once the ID field is empty, press *Clear Filters* button or either *Ongoing Projects* or *Completed Projects* button.



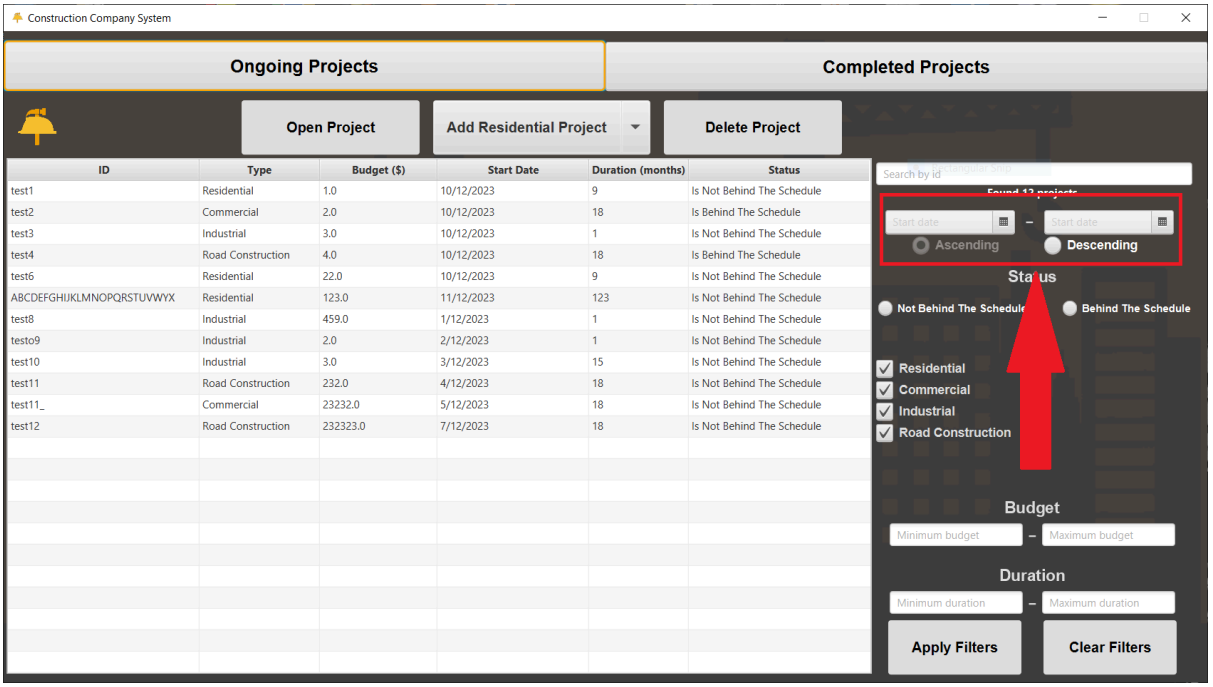
B. Filter by start date

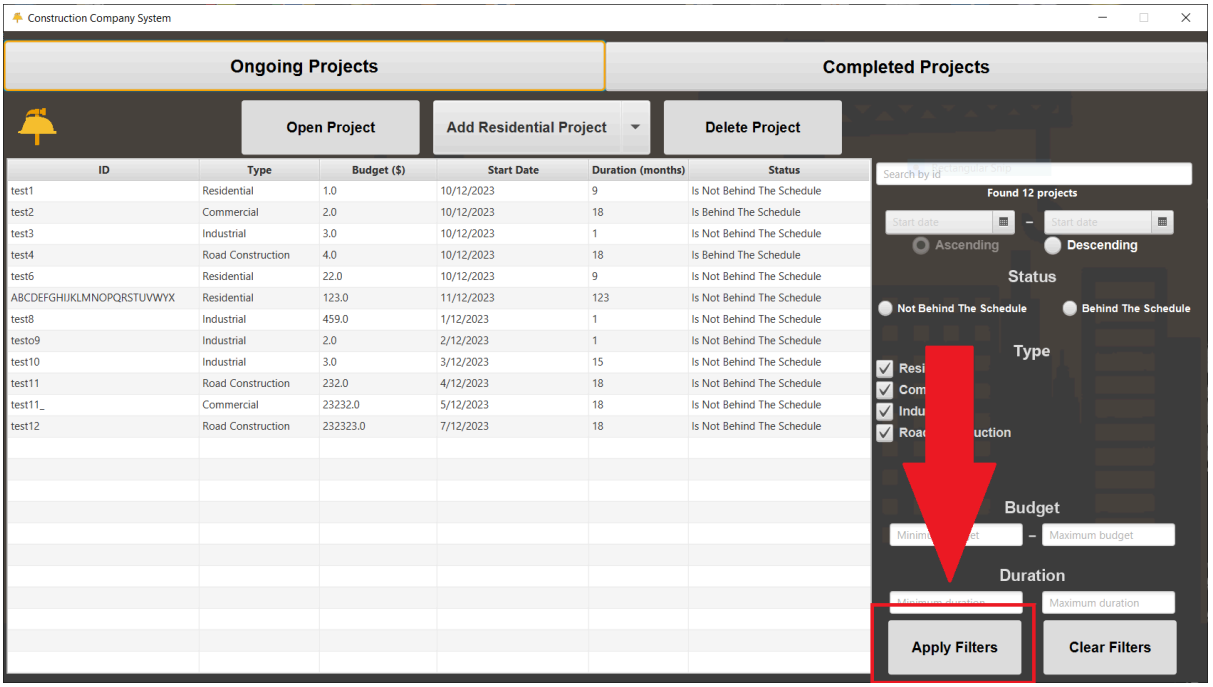
Step 1. Select the earliest start date using the first date picker on the right panel right under the *Search by ID* field.

Step 2. Select the latest start date using the second date picker.

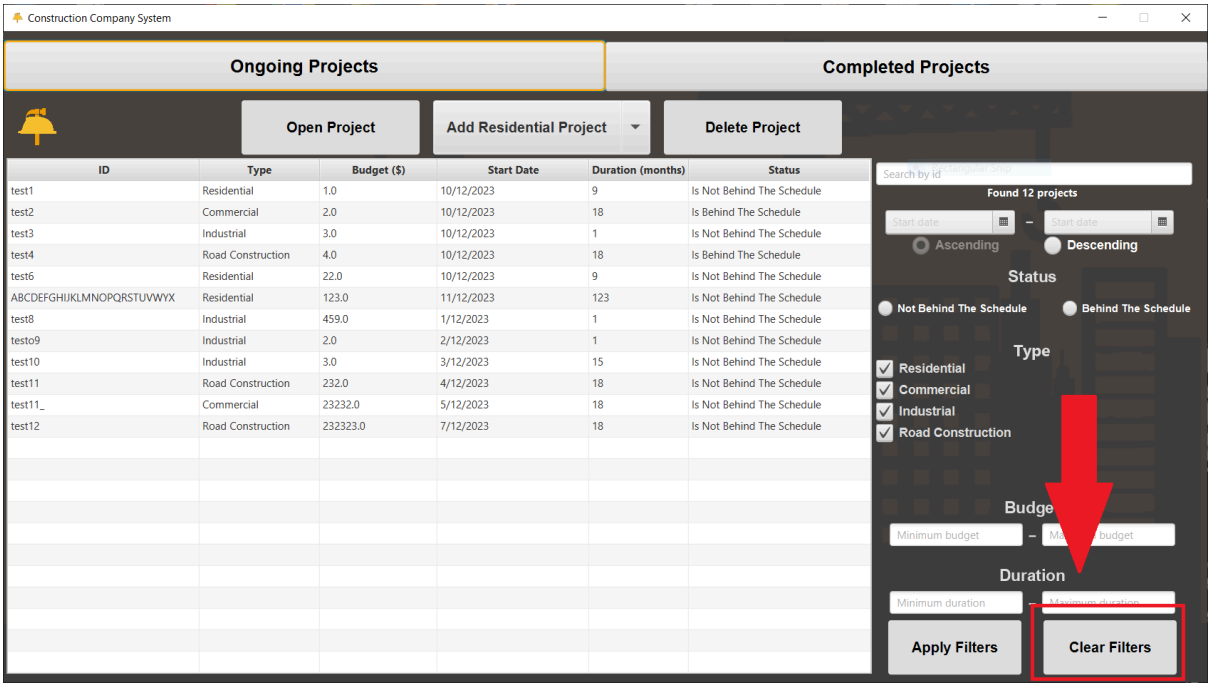
Step 3. Press Enter on any of them or press the *Apply Filters* button at the bottom of the panel.

You can also choose if the results should be ordered in descending or ascending order for the date filter by selecting one of the two options right under the date pickers.



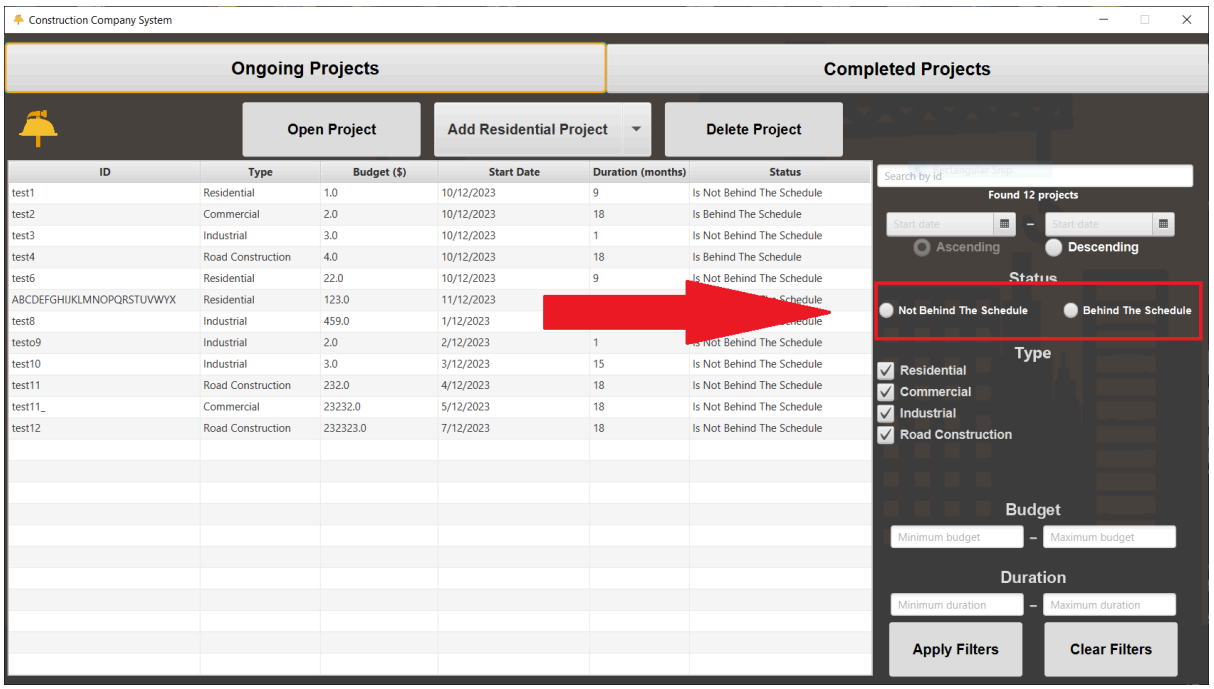


You do not have to choose the bottom or the maximal date, both fields can be left blank if you wish to not have an upper or lower bound. If you wish to further shorten the current range, enter the new dates and filter again. Once you wish to change the range of the search, remember to clear the filters first by pressing the *Clear Filters* button or either *Ongoing Projects* or *Completed Projects* button.

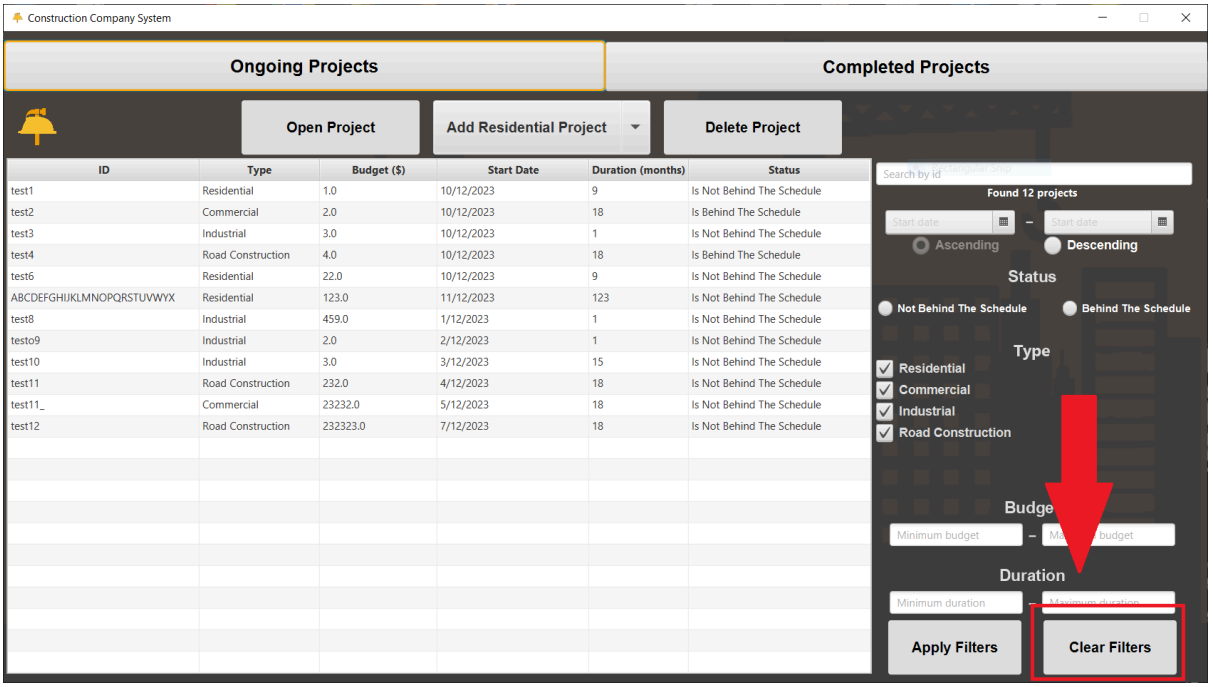


C. Filter by status

To filter by status, simply select one of the two buttons to search for projects behind or not behind the schedule. The buttons are located under the date search fields. Note that you can only select one button at a time.

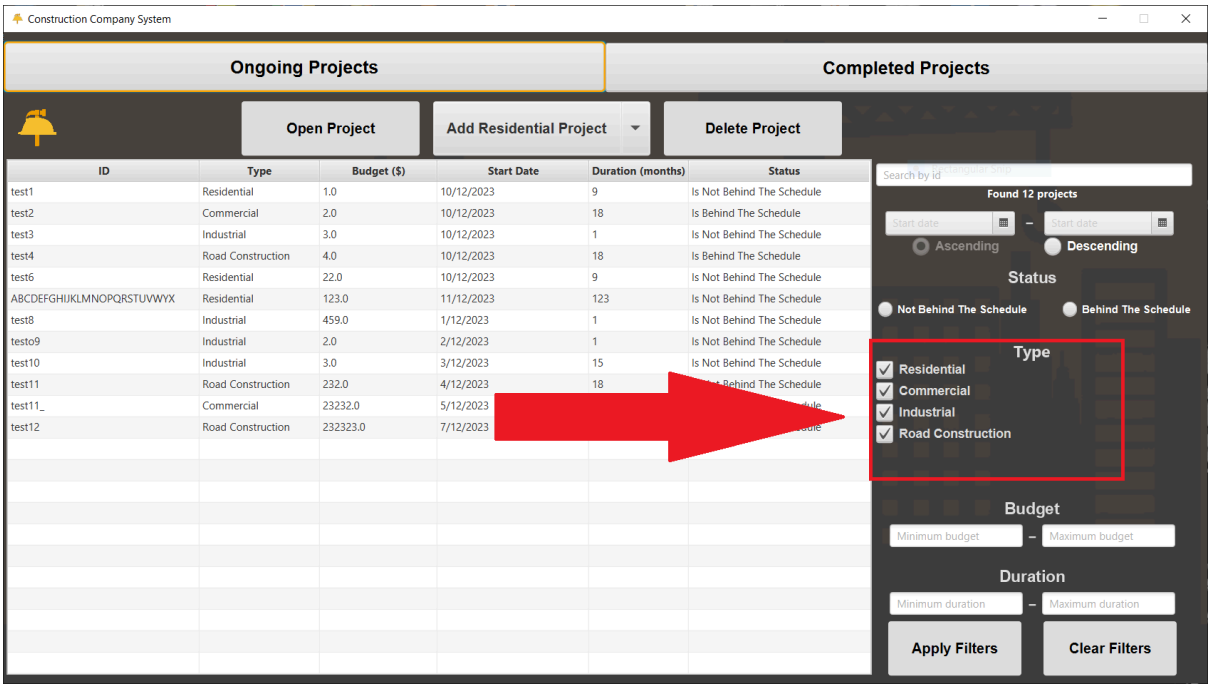


To reset it. simply clear the filters first by pressing the *Clear Filters* button or either *Ongoing Projects* or *Completed Projects* button.

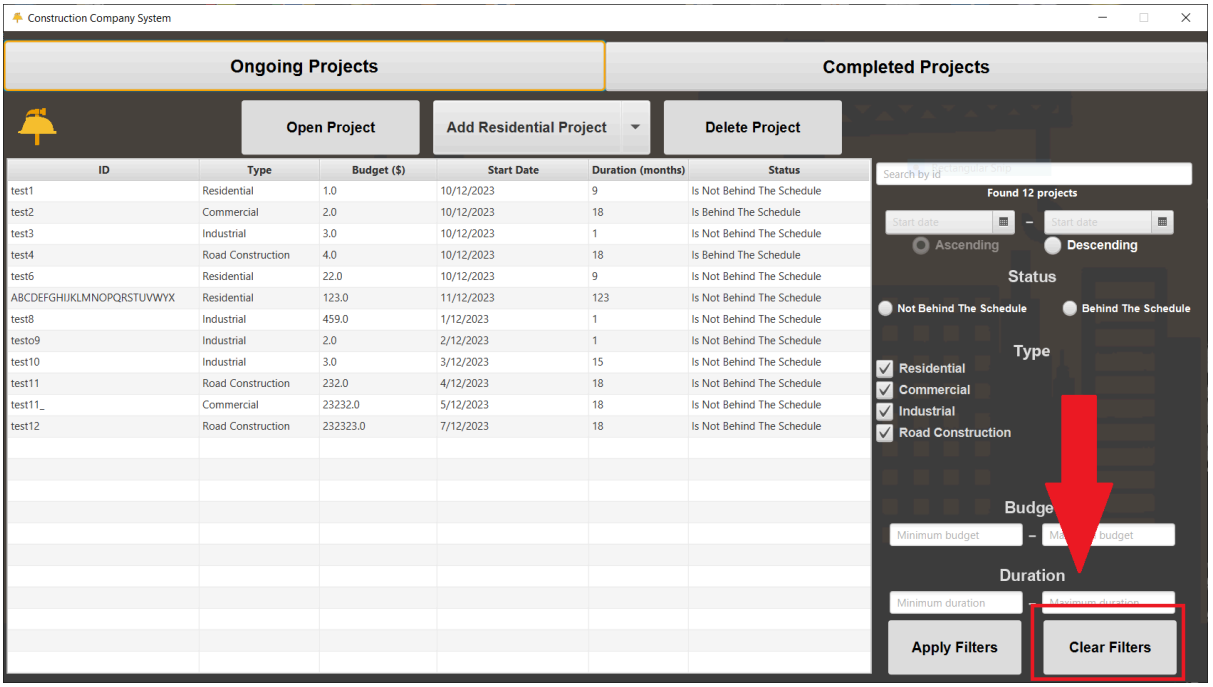


D. Filter by type

To filter by type, use the check boxes on the right panel that are located under the status buttons. To exclude the type from the displayed results, simply uncheck the box. To again include it check the box again.



If you wish to check all boxes on again click the *Clear Filters* button or either *Ongoing Projects* or *Completed Projects* button.



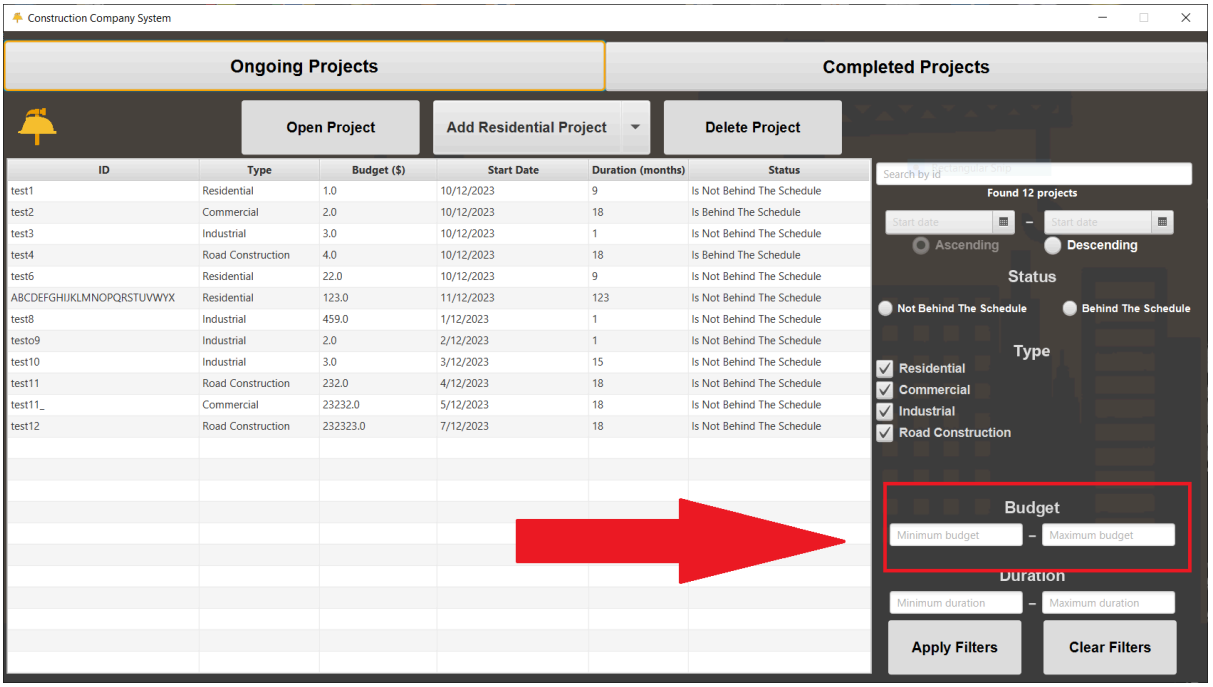
E. Filter by budget

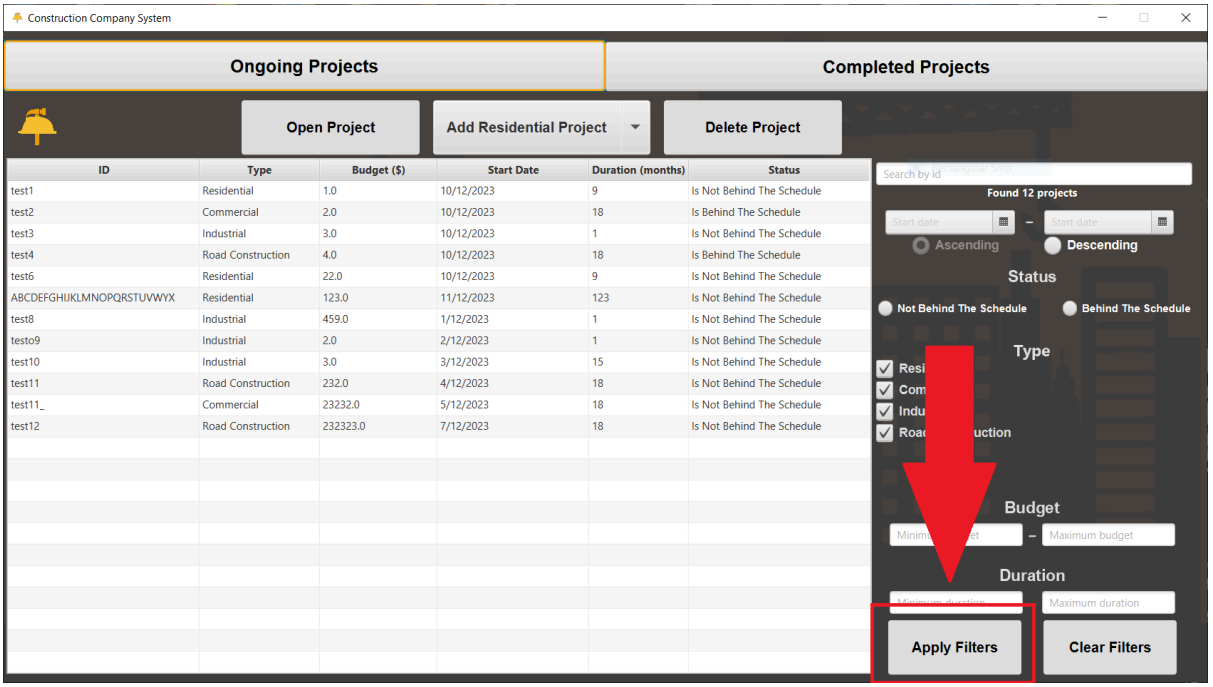
Step 1. Enter the lowest budget value in the first field, located on the right panel under the type check boxes.

Step 2. Enter the upper bound of the range in the second field.

Step 3. Press Enter on any of them or press the *Apply Filters* button at the bottom of the panel.

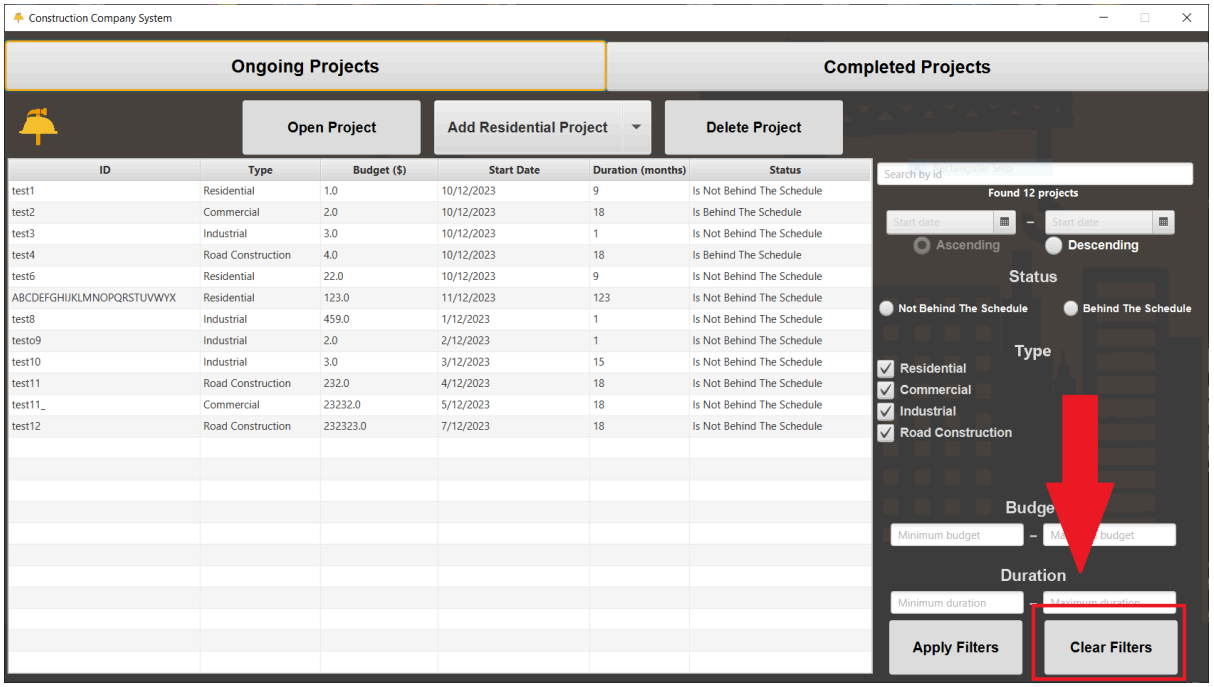
The same as with the date, you don't have to fill both fields if you don't wish to have the bottom or the upper bound in your search. If you wish to further shorten the current range, enter the new budget values and filter again.





Once you wish to change the range of the search, remember to clear the filters first by pressing the *Clear Filters* button or either *Ongoing Projects* or *Completed Projects* button.

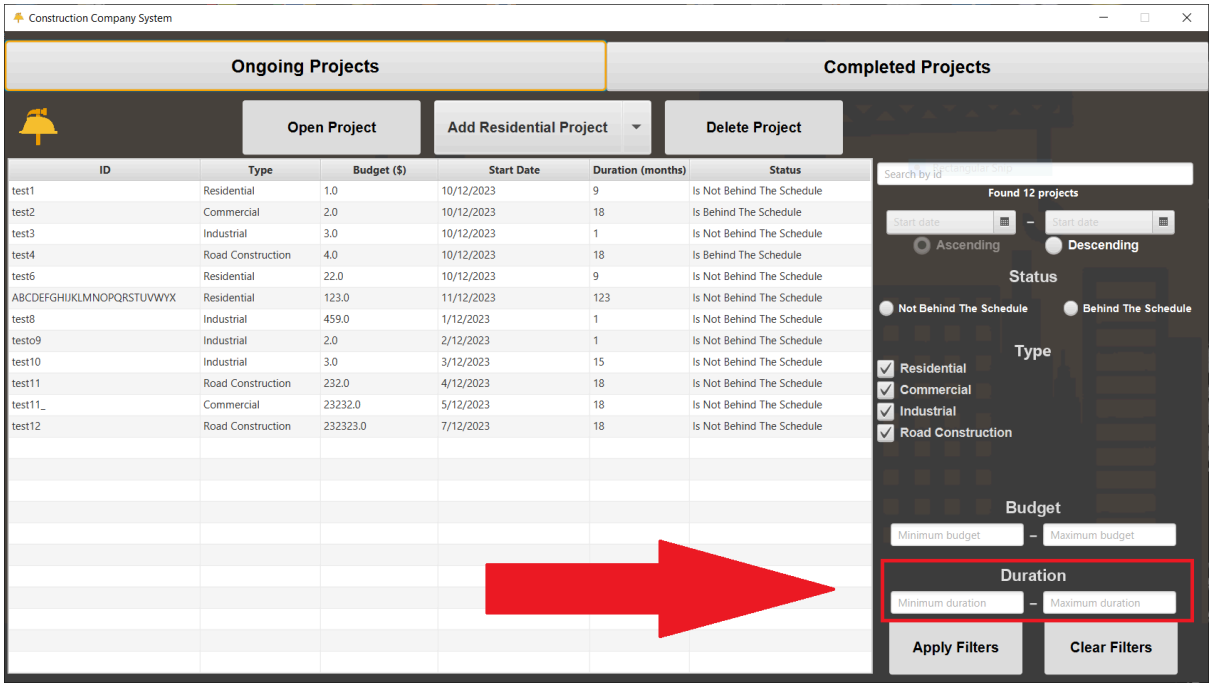
F. Filter by duration



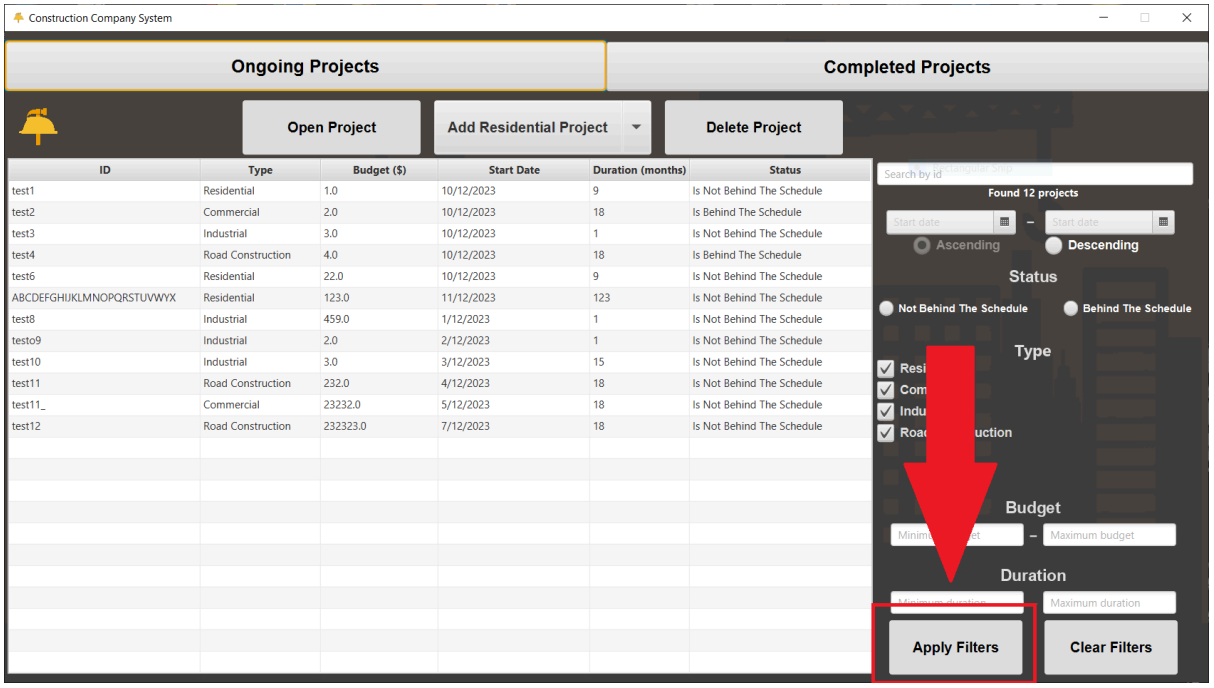
Step 1. Enter the lowest duration value in the first field, located at the bottom of the panel.

Step 2. Enter the upper bound of the range in the second field.

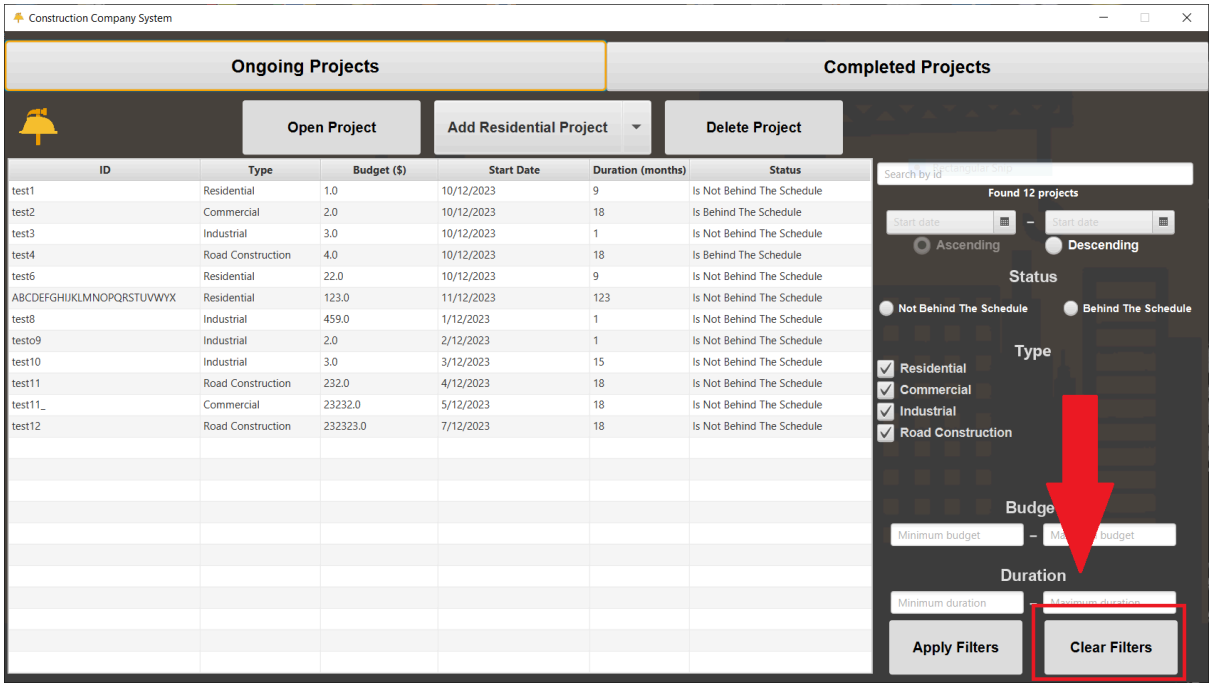
Step 3. Press Enter on any of them or press the *Apply Filters* button at the bottom of the panel.



The same as with the date and budget, you don't have to fill both fields if you don't wish to have the bottom or the upper bound in your search. If you wish to further shorten the current range, enter the new duration values and filter again.



Once you wish to change the range of the search, remember to clear the filters first by either pressing the *Clear Filters* button or either *Ongoing Projects* or *Completed Projects* button.

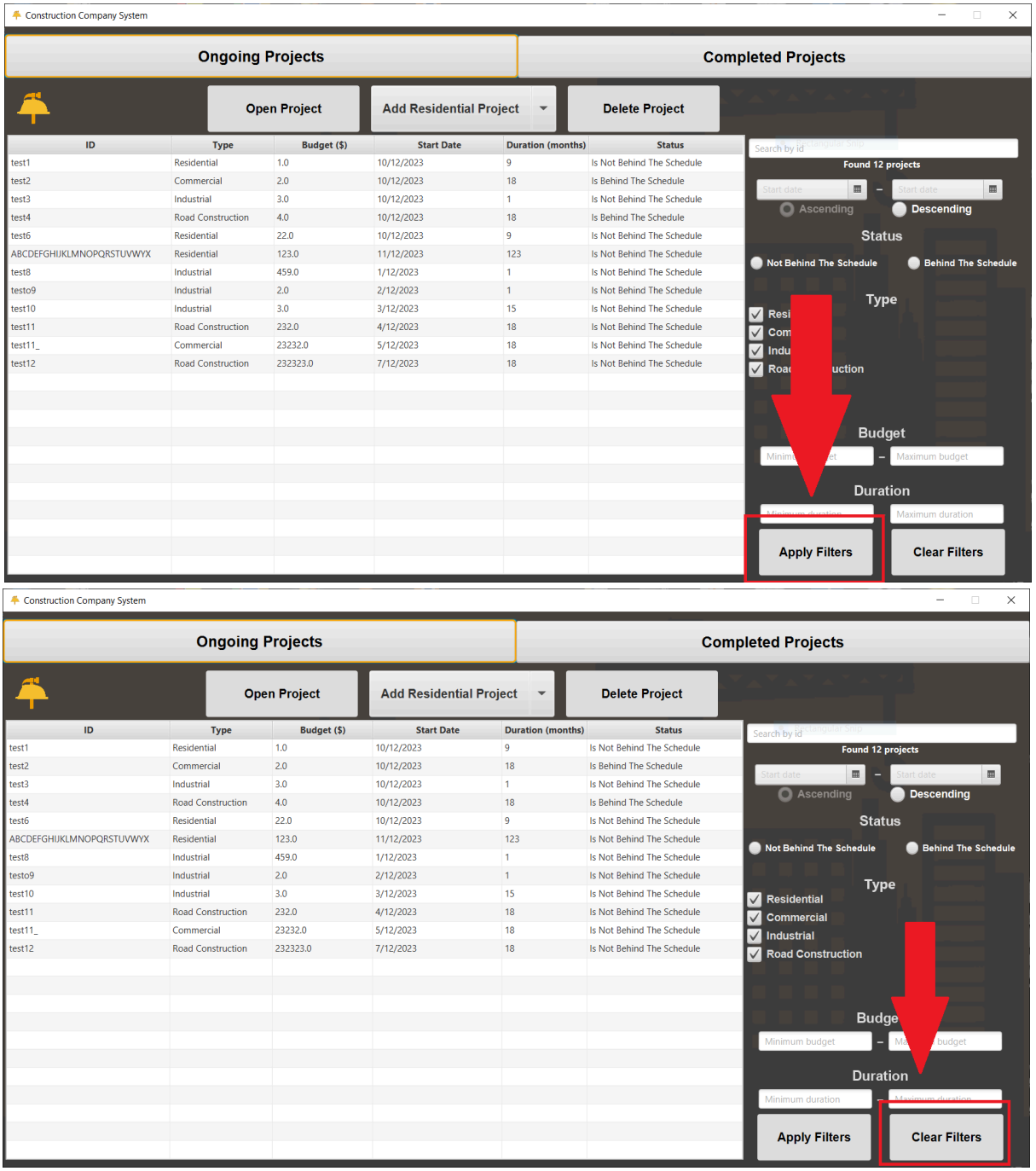


G. Apply multiple filters at the same time

Step 1. Fill out the fields with filters you wish to apply.

Step 2. Press the *Apply Filters* button at the bottom of the panel.

Alternatively, you can do it manually by hitting Enter in each field Once you wish to change the range of the search, remember to clear the filters first by pressing the *Clear Filters* button or either *Ongoing Projects* or *Completed Projects* button.



3. Add a new Project

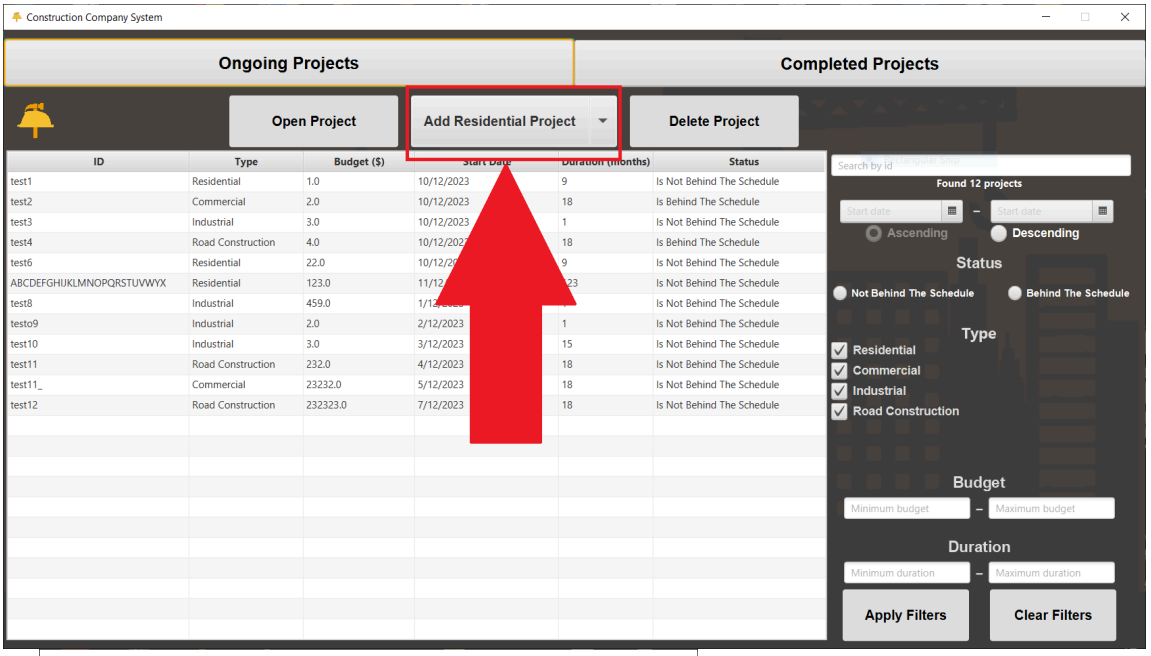
When you want to create a new project, use the middle button located under the *Ongoing Projects* and *Completed Projects* buttons. Press it to add a project of type Residential or use the small arrow next to it for other options.

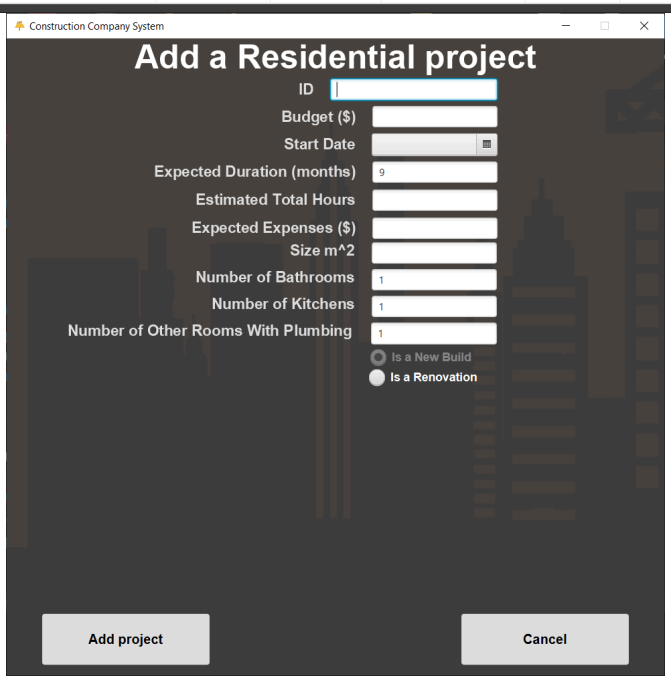
A. Add a Residential Project

Step 1. Press the *Add Residential Project* button.

Step 2. Enter the values.

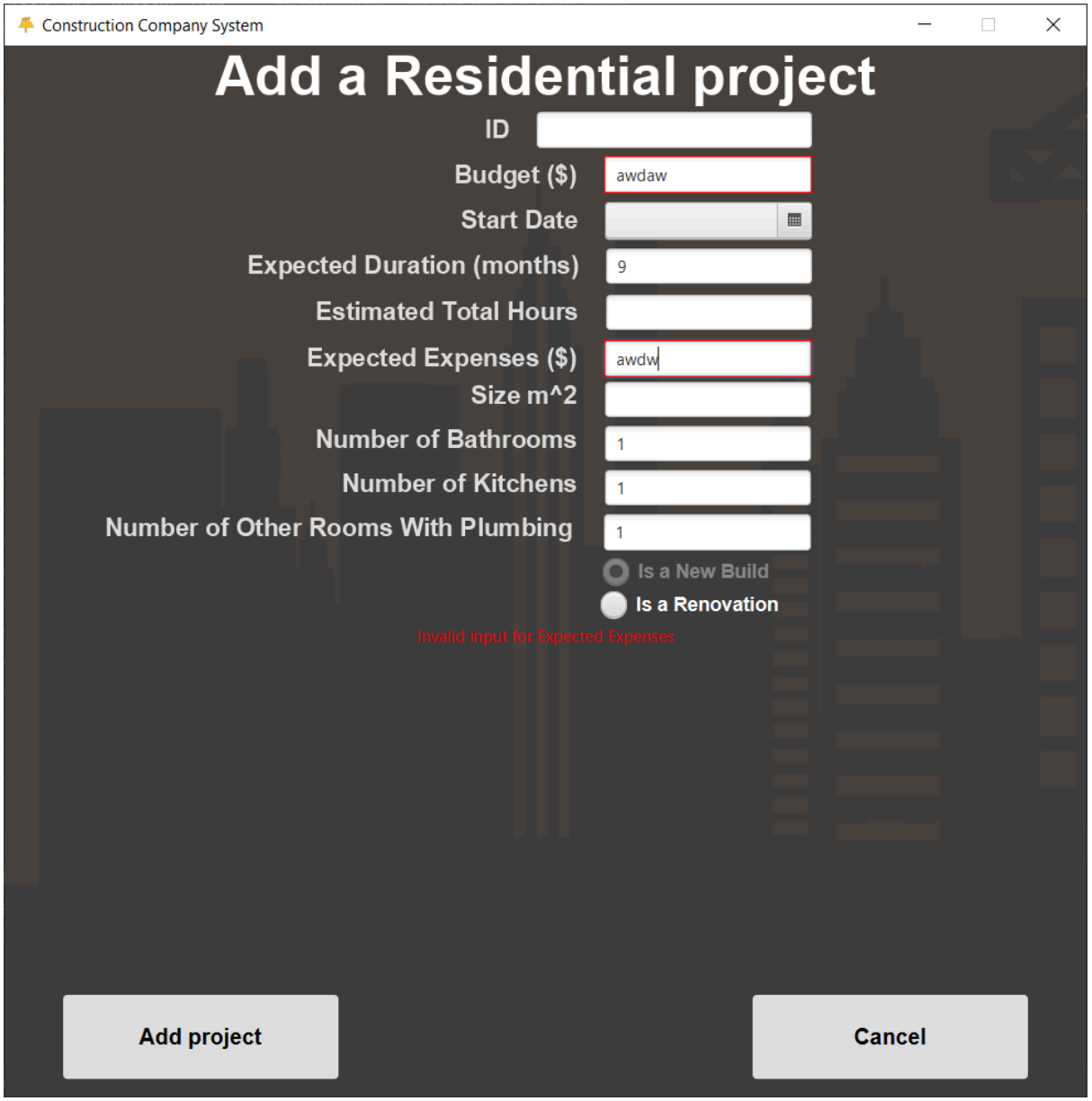
Step 3. Press the *Add Project* button.





Note that some fields will have default values already entered, but they can be changed. This is the case with the start date as well: if it remains empty, the start date will be set to the current date. You can jump to the next field by pressing Enter; pressing Enter on the last field will automatically hit the *Add project* button for you.

If you enter an invalid data (e. g. letters, where the system requires numbers), the fields will be highlighted until you fix the issue. When entering illegal values, the system will display an error and allow you to fix the values in question.



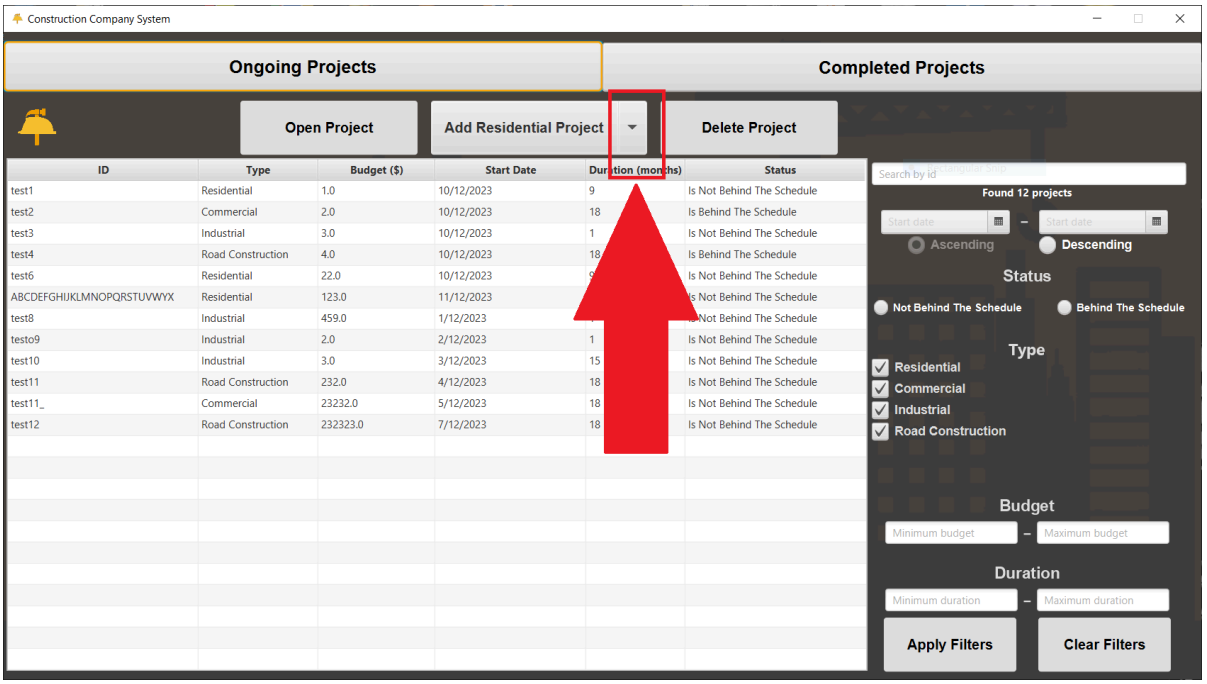
B. Add an Industrial, Commercial or Road Construction Project

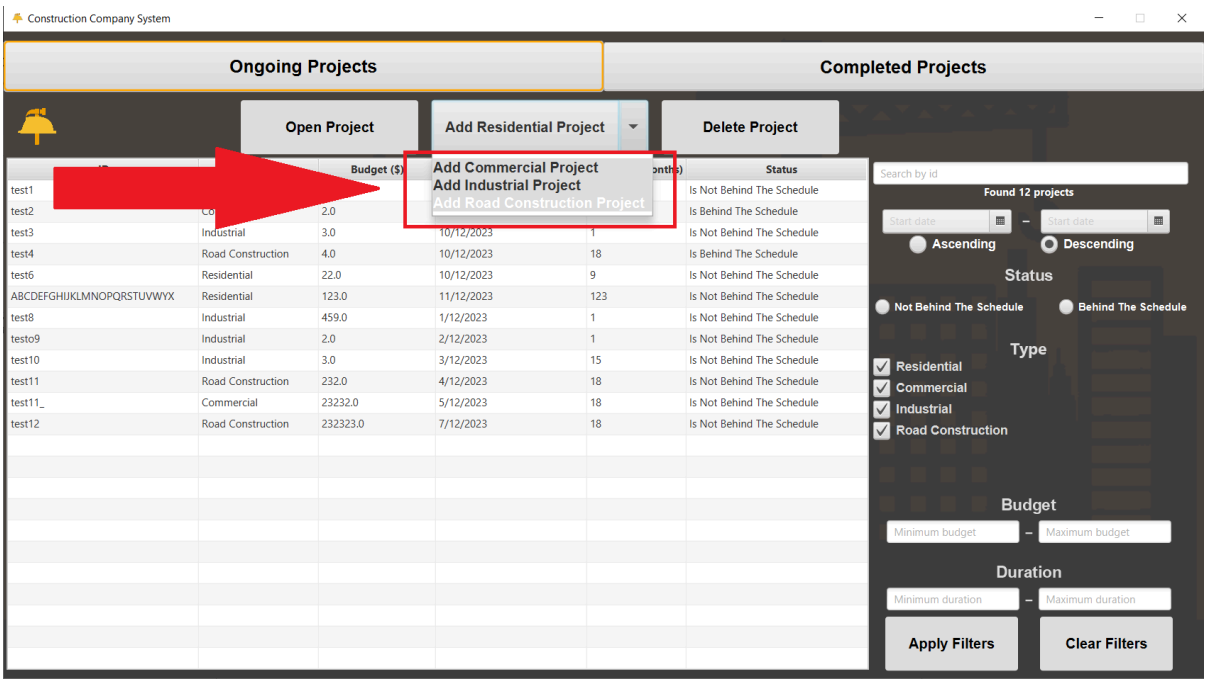
Step 1. Press the small arrow that is part of the *Add Residential Project* button.

Step 2. Select one of the options depending on what type you want to add.

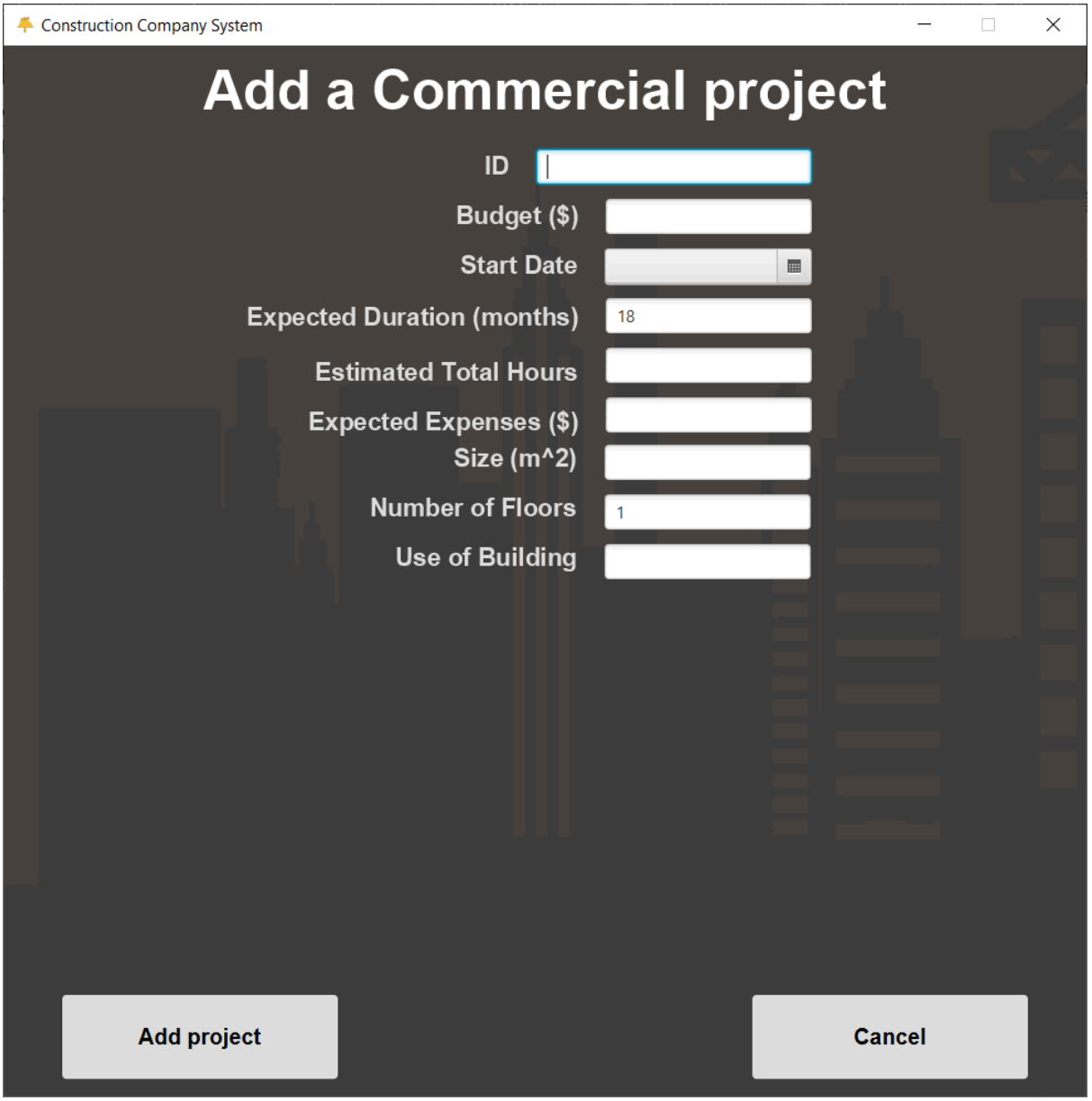
Step 3. Enter the values.

Step 4. Press the *Add Project* button.

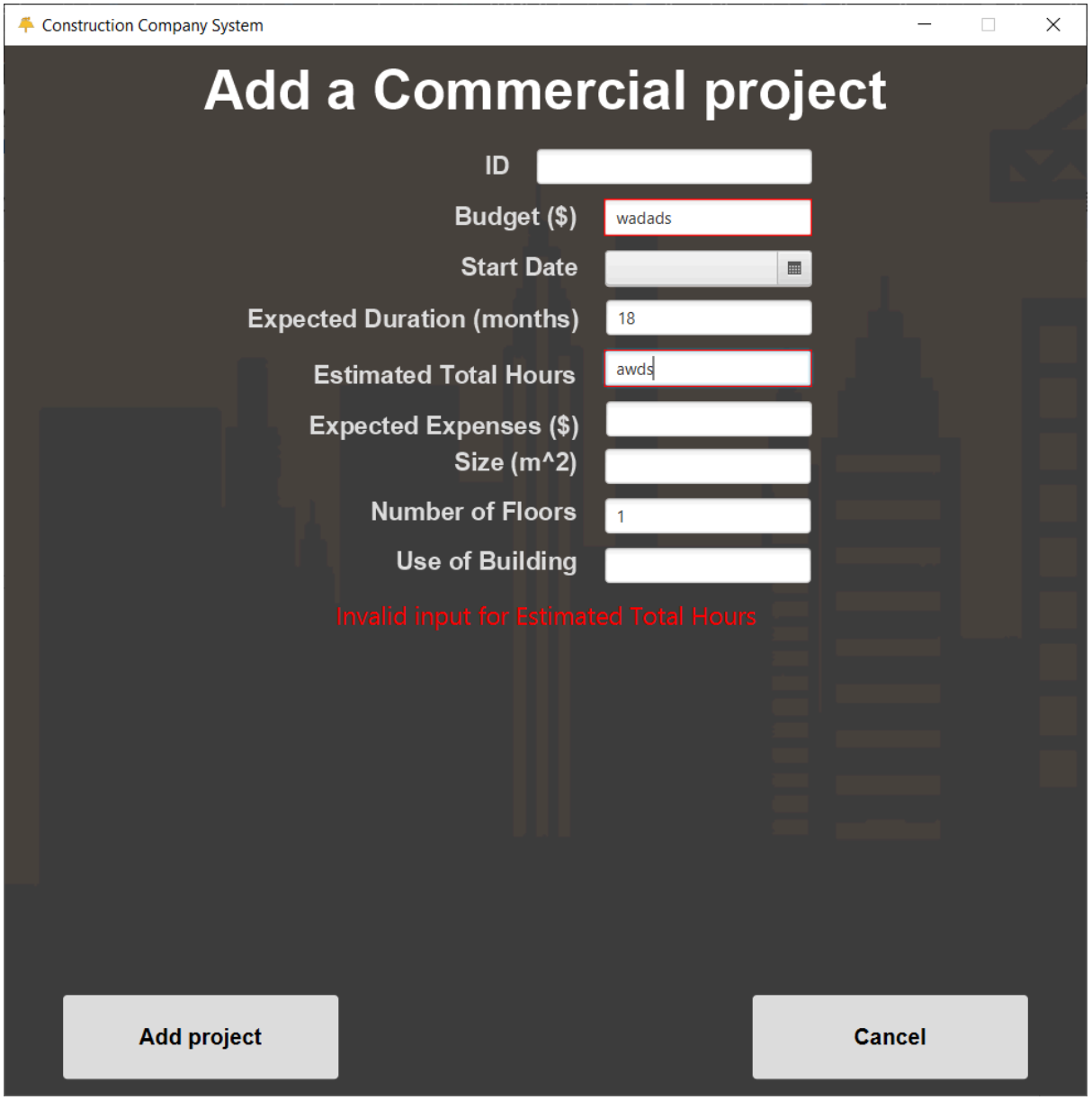




Note that some fields will have default values already entered, but they can be changed. This is the case with the start date as well: if it remains empty, the start date will be set to the current date. You can jump to the next field by pressing Enter; pressing Enter on the last field will automatically hit the *Add project* button for you.



If you enter an invalid data (e. g. letters, where the system requires numbers), the fields will be highlighted until you fix the issue. When entering illegal values, the system will display an error and allow you to fix the values in question.

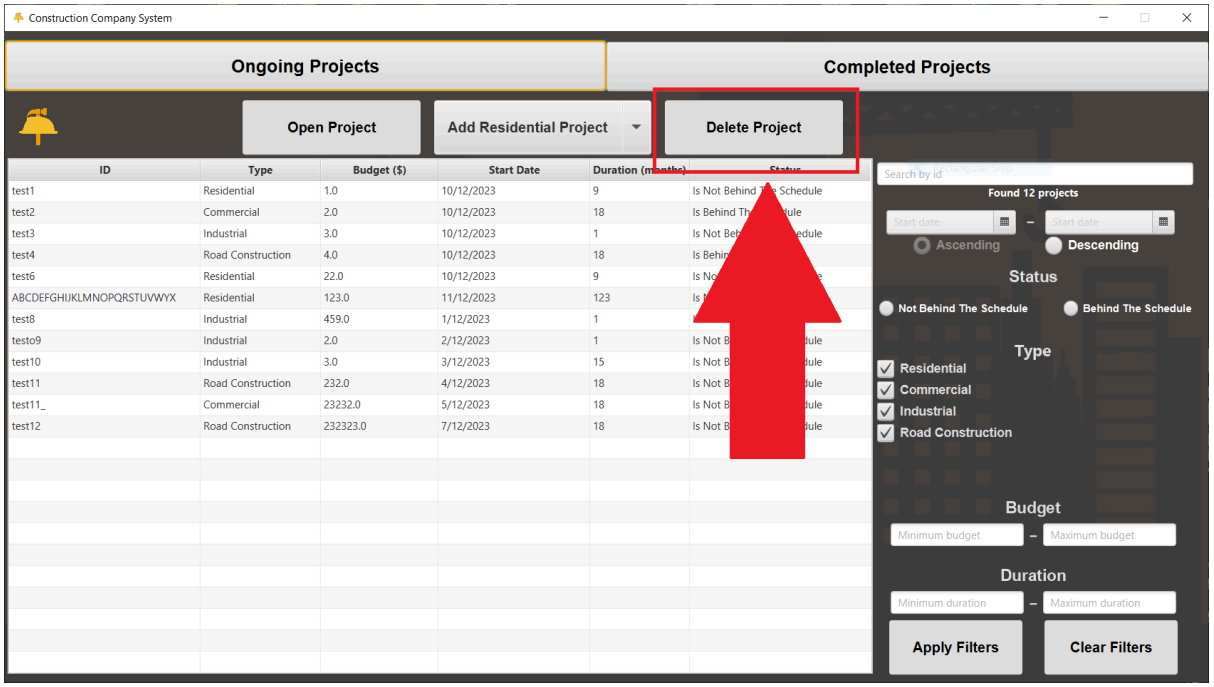


4. Delete a project

Step 1. Select (click) the project from the list.

Step 2. Press the *Delete Project* button, located under the *Ongoing Projects* and *Completed Projects* buttons.

Step 3. Press *OK* in the new pop-up window.

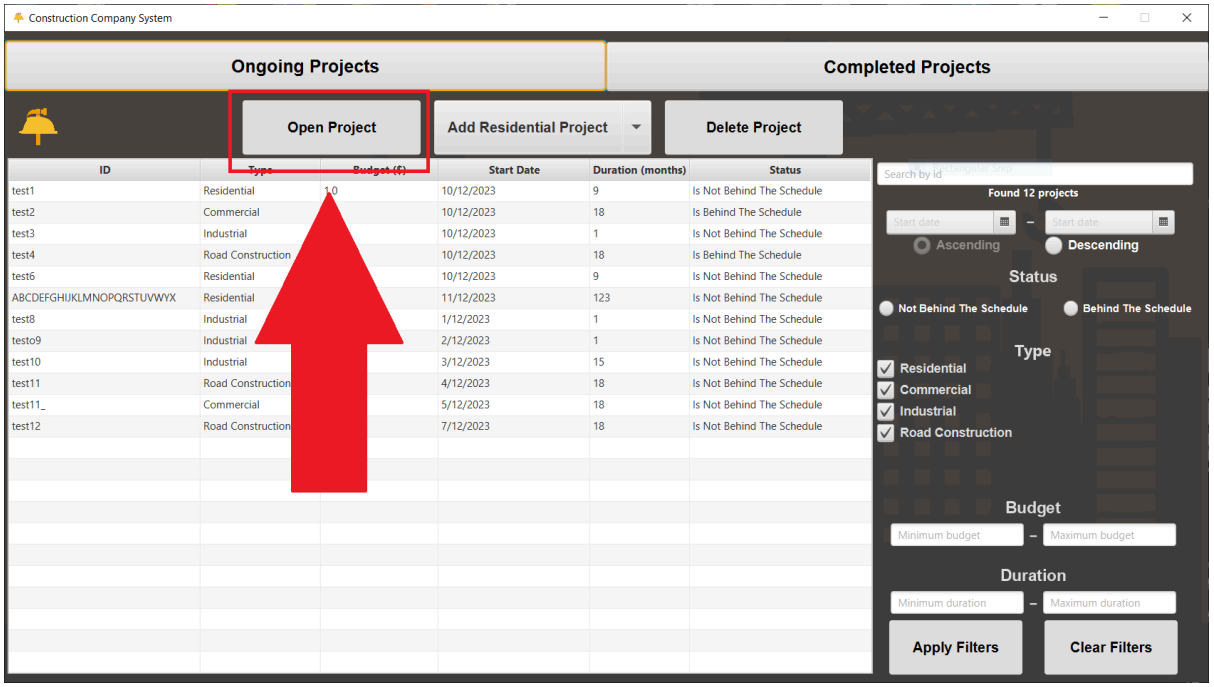


Note that deleting a project is **permanent** and **cannot be undone**. Once the project is deleted, it is removed from the system completely, which frees up the ID it had been assigned.

5. Open a project

Step 1. Select (click) the project from the list.

Step 2. Press the *Open Project* button, located under the *Ongoing Projects* and *Completed Projects* buttons.



From here, you will have a few options:

- cancel and go back to the list you were looking at;
- update project's resources;
- edit project's details.

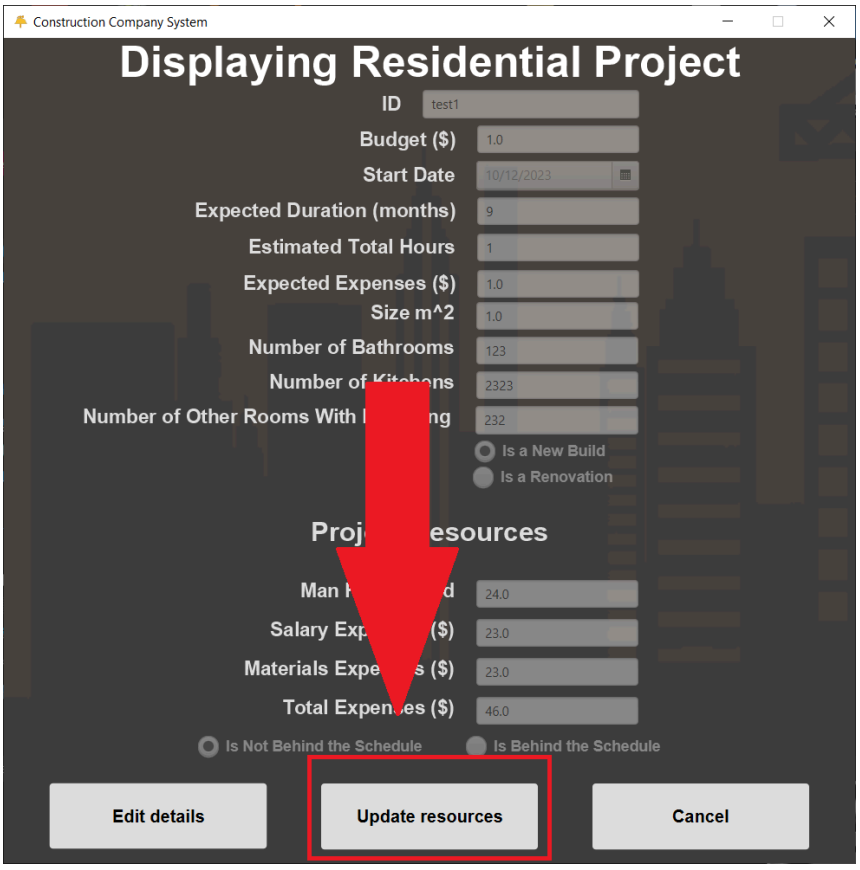
6. Update the resources

Step 1. Open the project (see chapter 5).

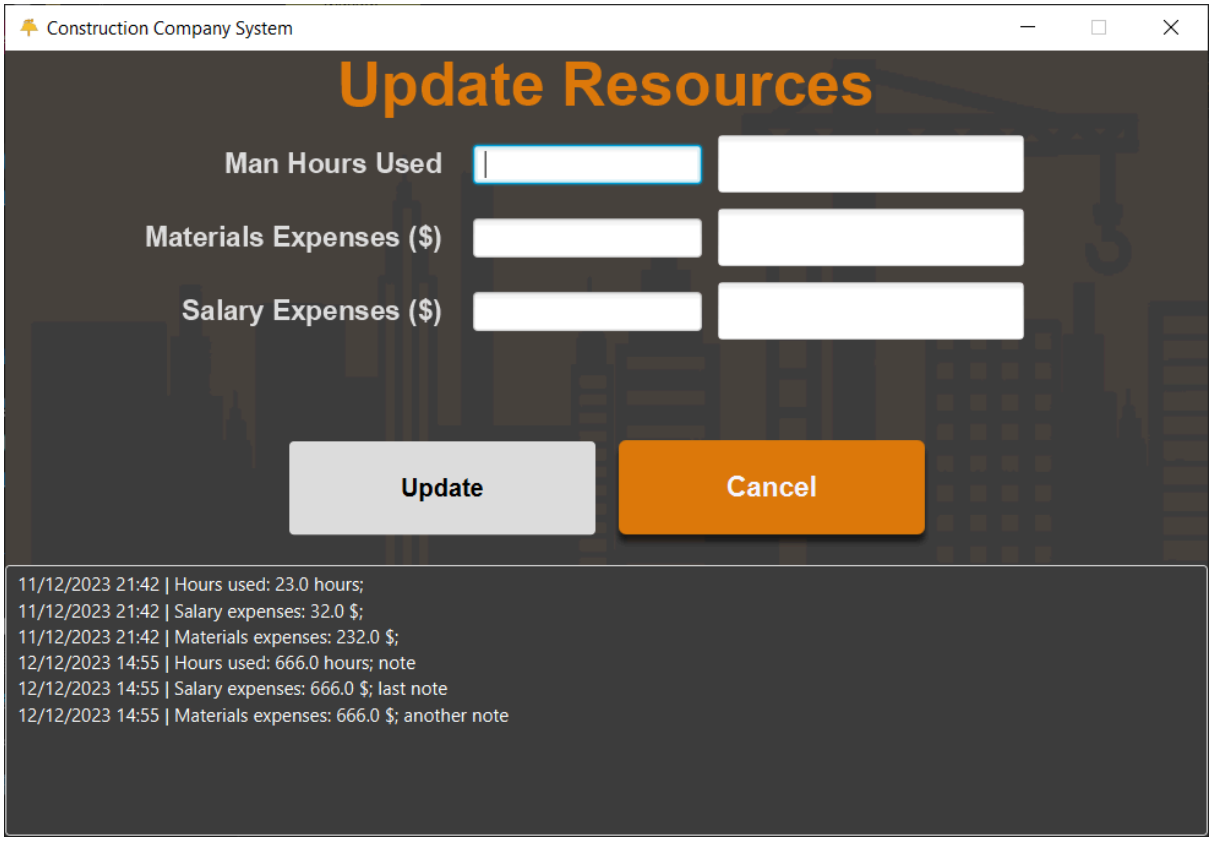
Step 2. Press the middle button labeled as *Update Resources*.

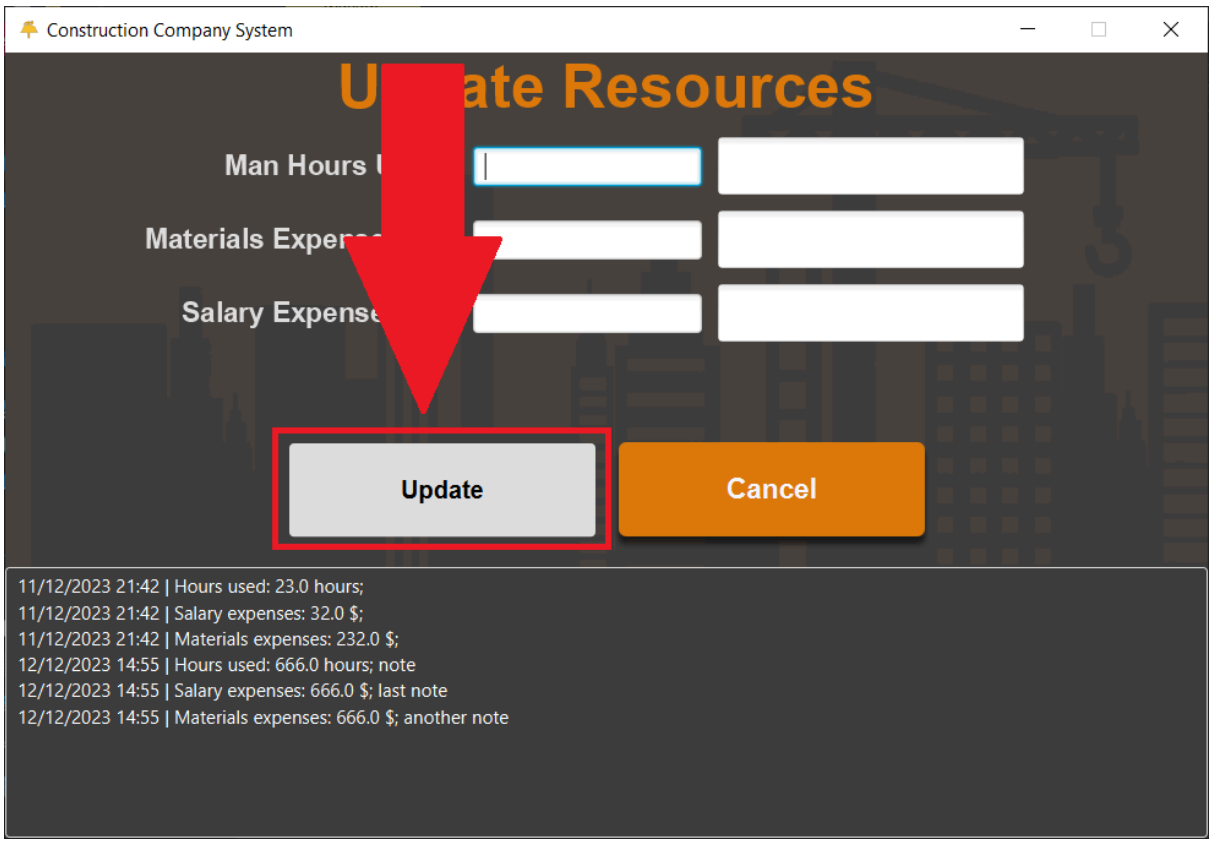
Step 3. Enter the values and optionally, add notes.

Step 4. Press the Update button.

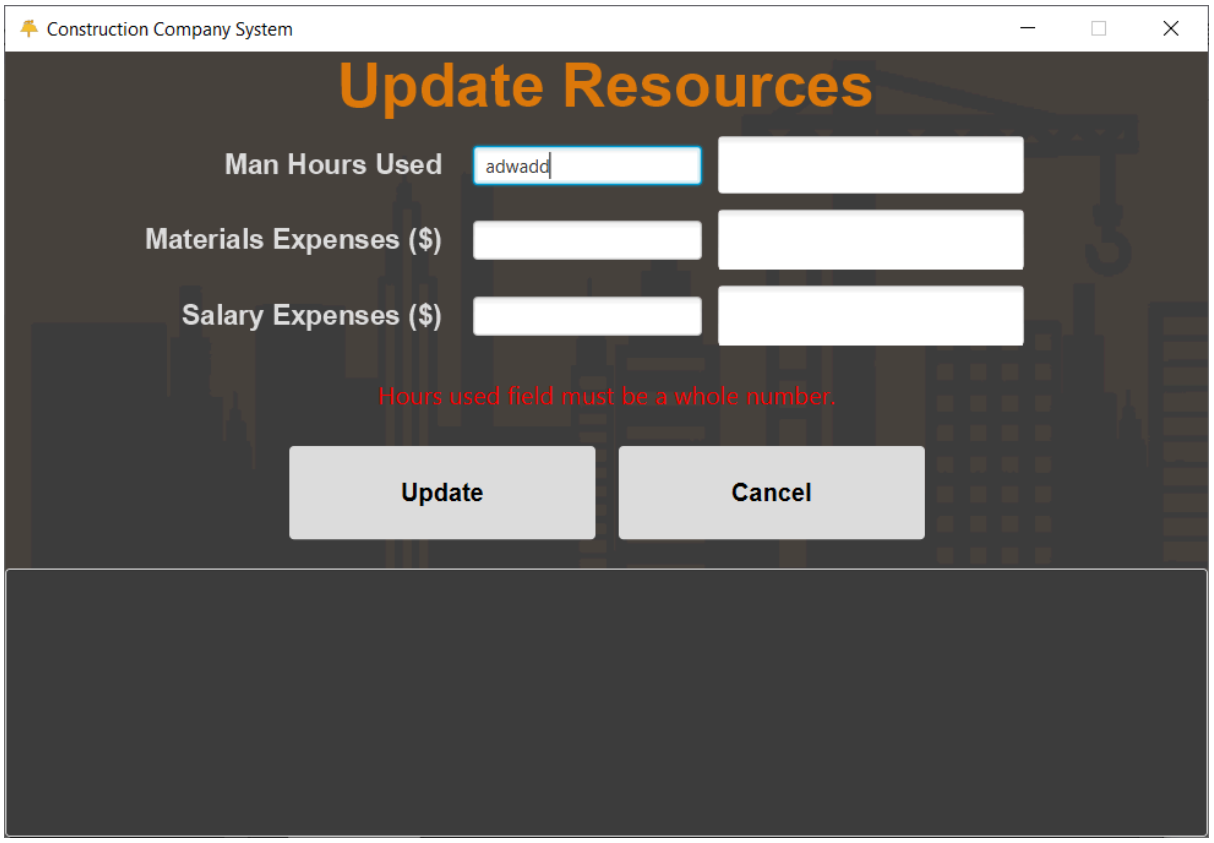


You can jump to the next field by pressing Enter; pressing Enter on the last field will automatically hit the *Update* button for you.

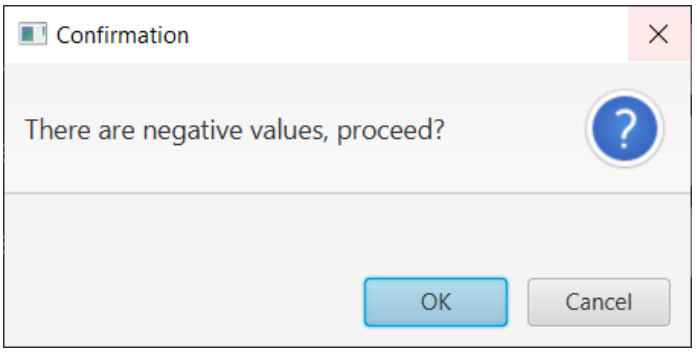

Construction Company System
Update Resources
Man Hours Used
Materials Expenses ($)
Salary Expenses ($)
Update
Cancel
11/12/2023 21:42 | Hours used: 23.0 hours;
11/12/2023 21:42 | Salary expenses: 32.0 $;
11/12/2023 21:42 | Materials expenses: 232.0 $;
12/12/2023 14:55 | Hours used: 666.0 hours; note
12/12/2023 14:55 | Salary expenses: 666.0 $; last note
12/12/2023 14:55 | Materials expenses: 666.0 $; another note


Construction Company System
Update
Cancel
11/12/2023 21:42 | Hours used: 23.0 hours;
11/12/2023 21:42 | Salary expenses: 32.0 $;
11/12/2023 21:42 | Materials expenses: 232.0 $;
12/12/2023 14:55 | Hours used: 666.0 hours; note
12/12/2023 14:55 | Salary expenses: 666.0 $; last note
12/12/2023 14:55 | Materials expenses: 666.0 $; another note

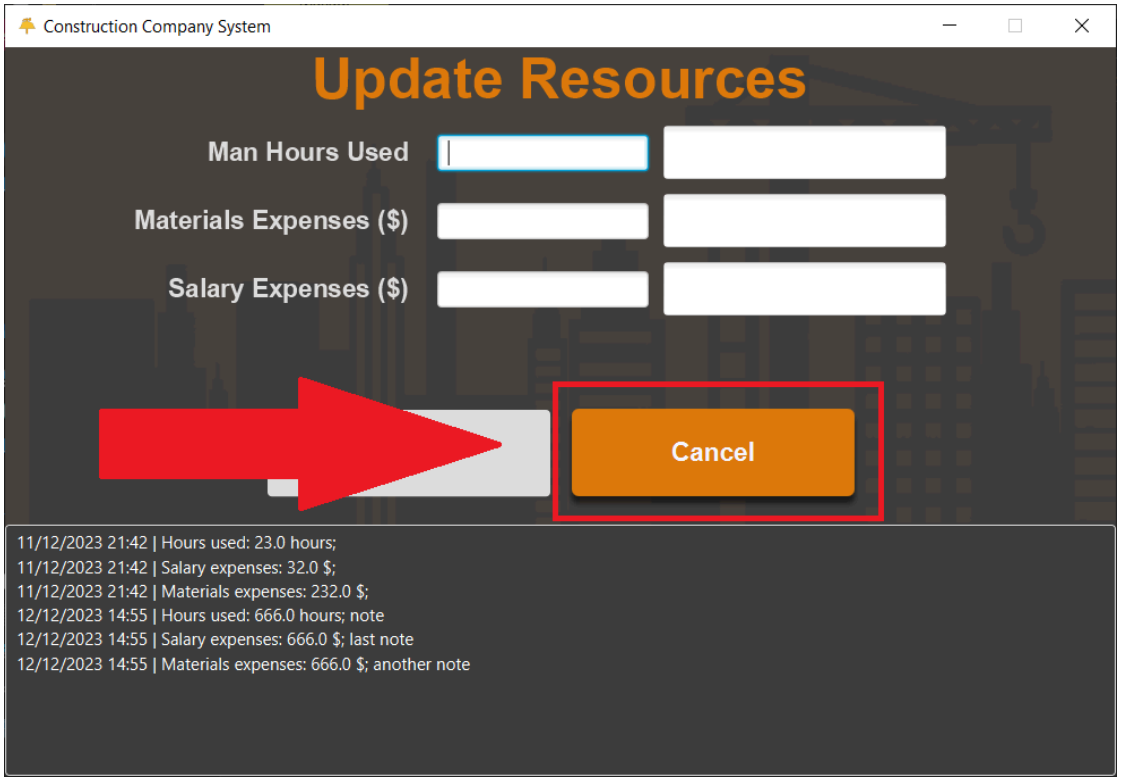
If you enter an invalid data (e. g. letters, where the system requires numbers), the fields will be highlighted until you fix the issue. When entering illegal values, the system will display an error and allow you to fix the values in question.



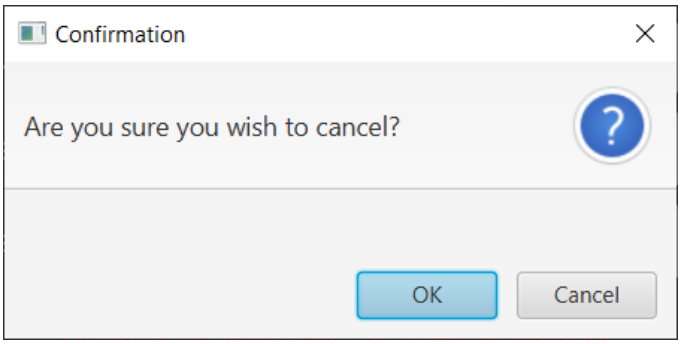
When entering a negative value for the update, the system will ask for confirmation.



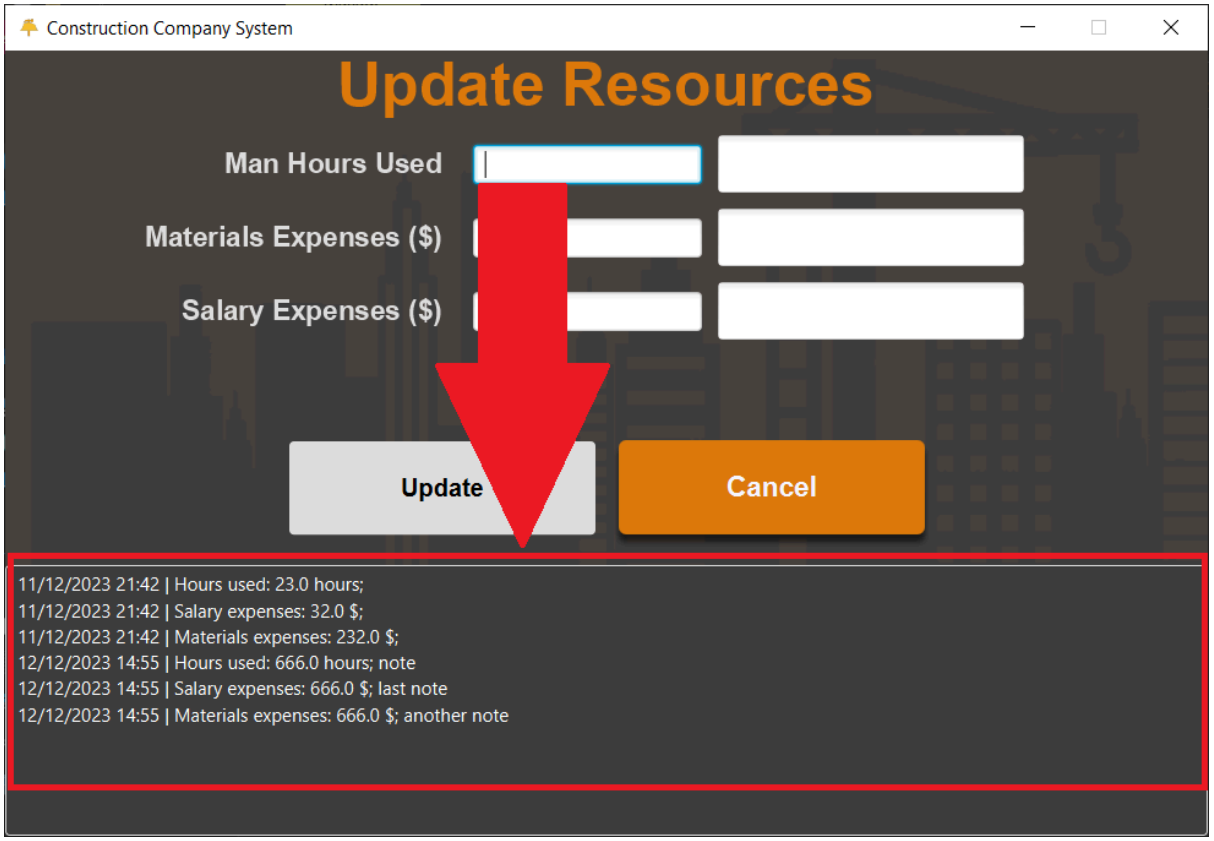
Note that if you don't want to add anything to a specific attribute you can leave it blank.



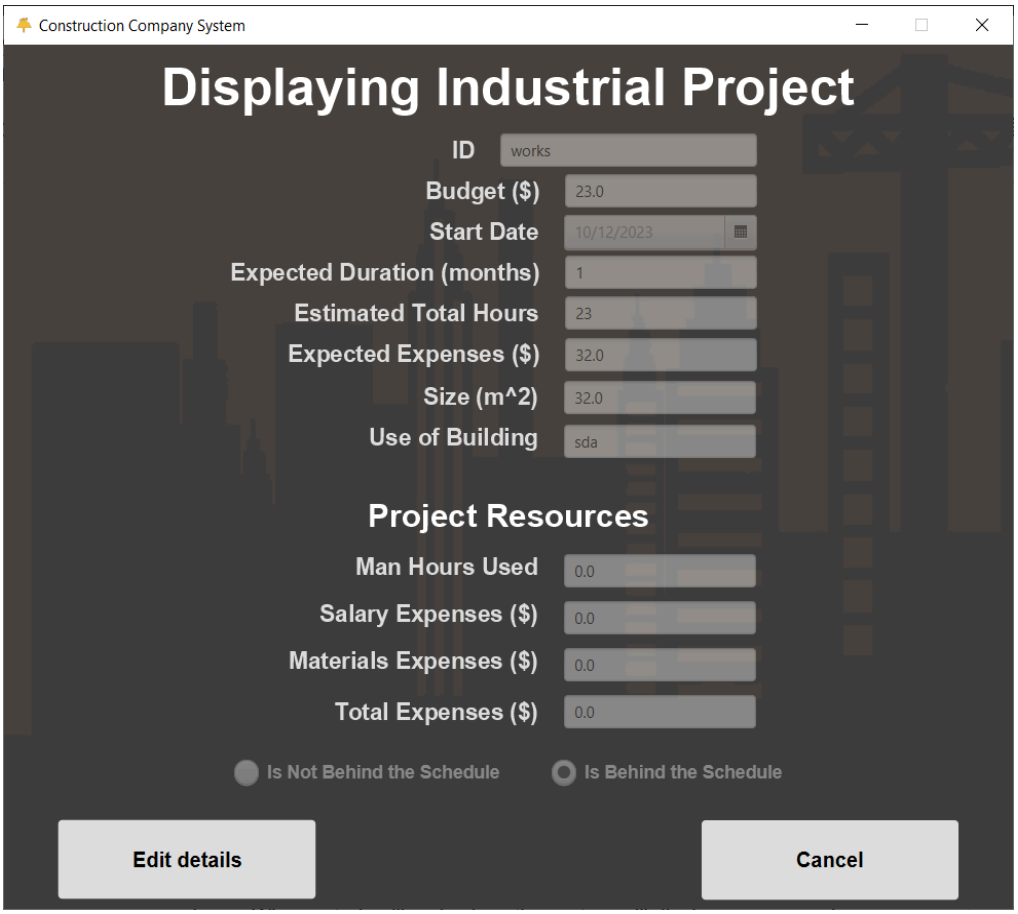
If you wish to cancel the update, simply press the *Cancel* button and confirm canceling.



When you either complete or cancel the update, you will be brought back to the project's details. Under the two buttons, there is a text area that will display the last updates you made to the project along with the notes, date and time.



Note that you do not have the option to update a project that has been marked as completed.



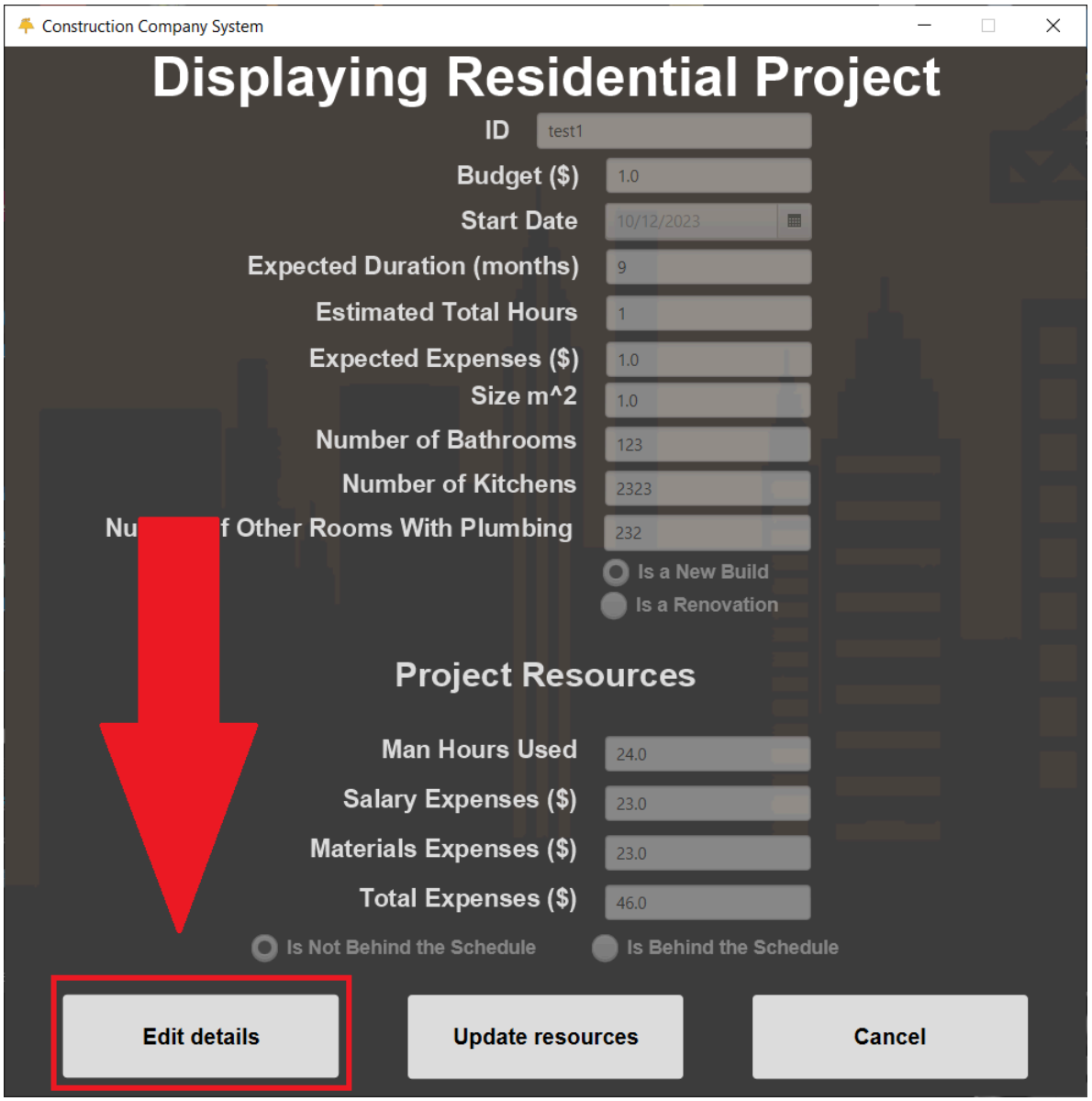
7. Edit the details

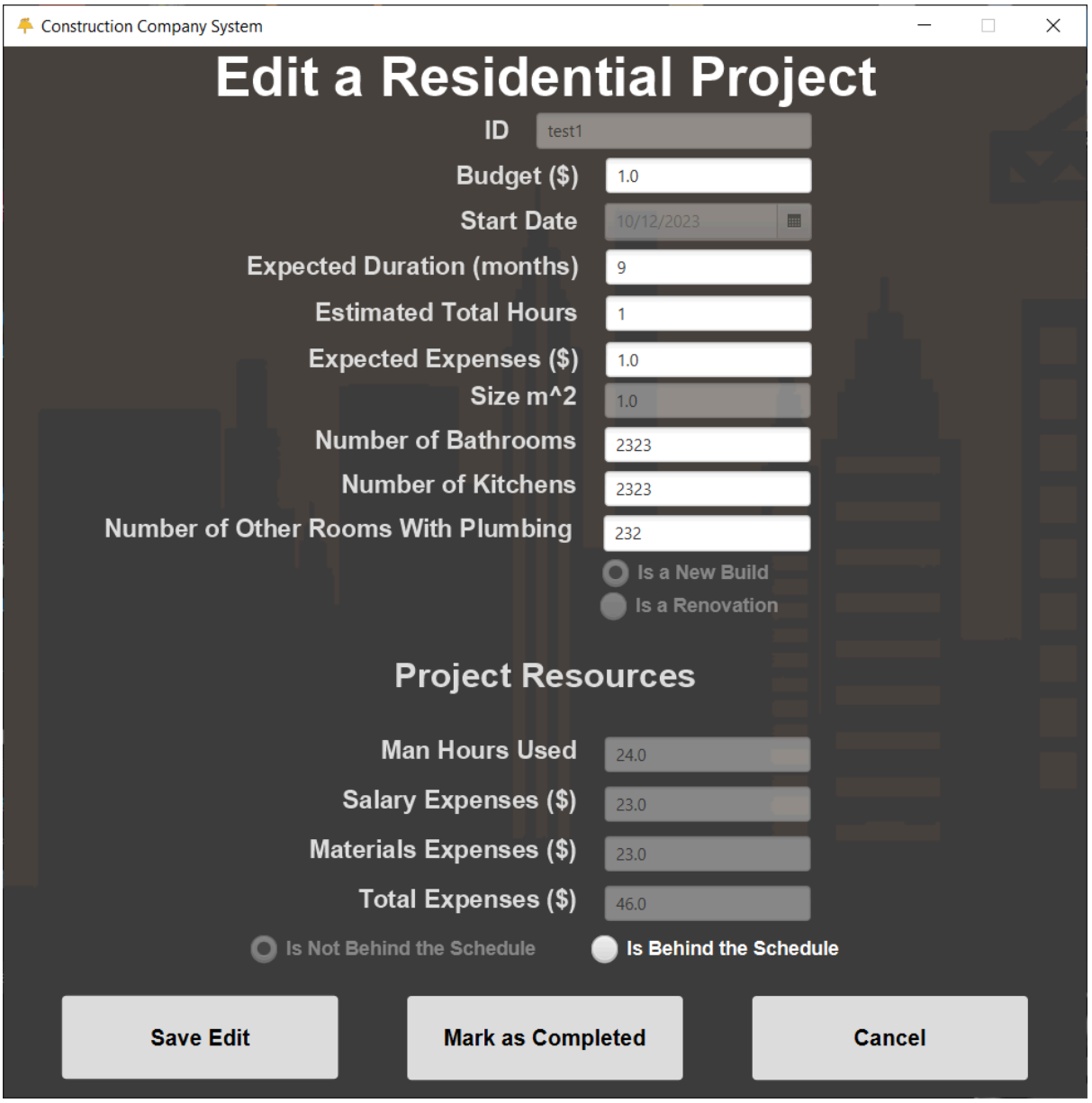
Step 1. Open the project (see chapter 5).

Step 2. Press the left button labeled as *Edit Details*.

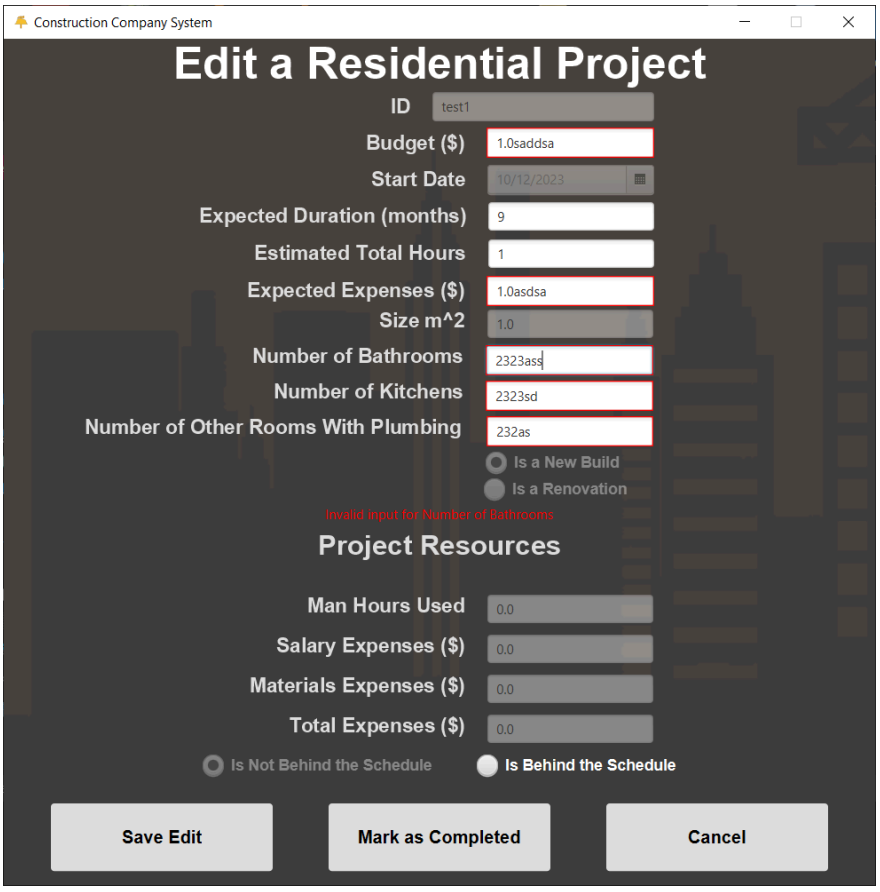
Step 3. Change the values or/and mark the project as ongoing or completed (depending on current status), by pressing the middle button labeled accordingly: *Mark as Completed* or *Mark as Ongoing*.

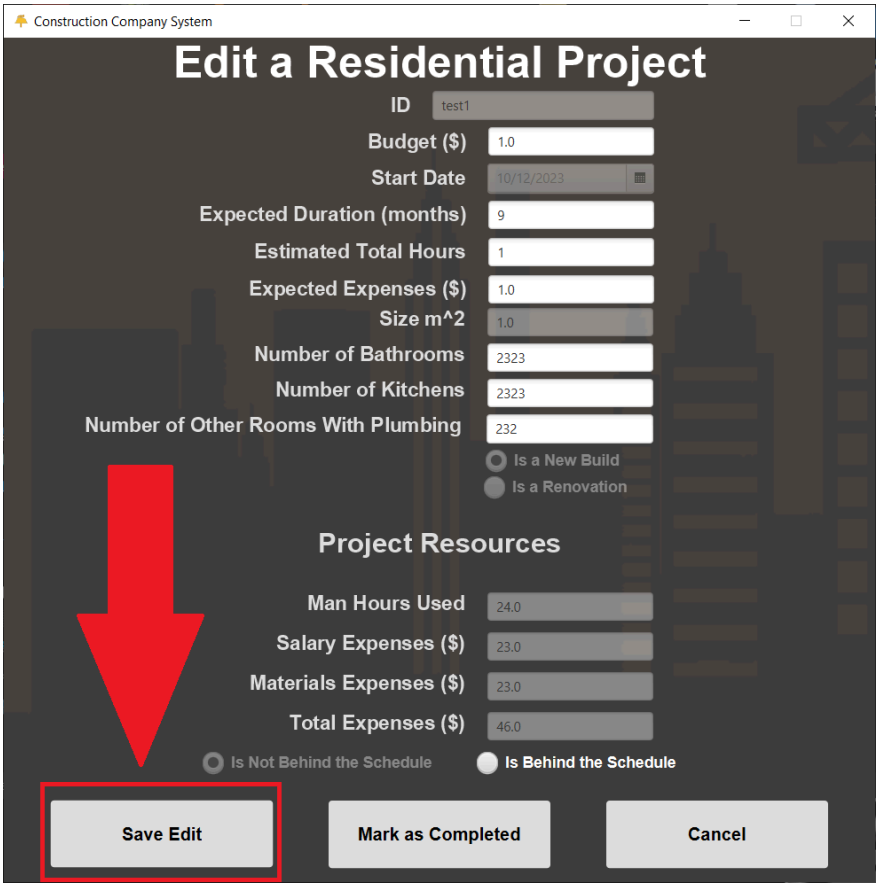
Step 4. Press the *Save Edit* button.

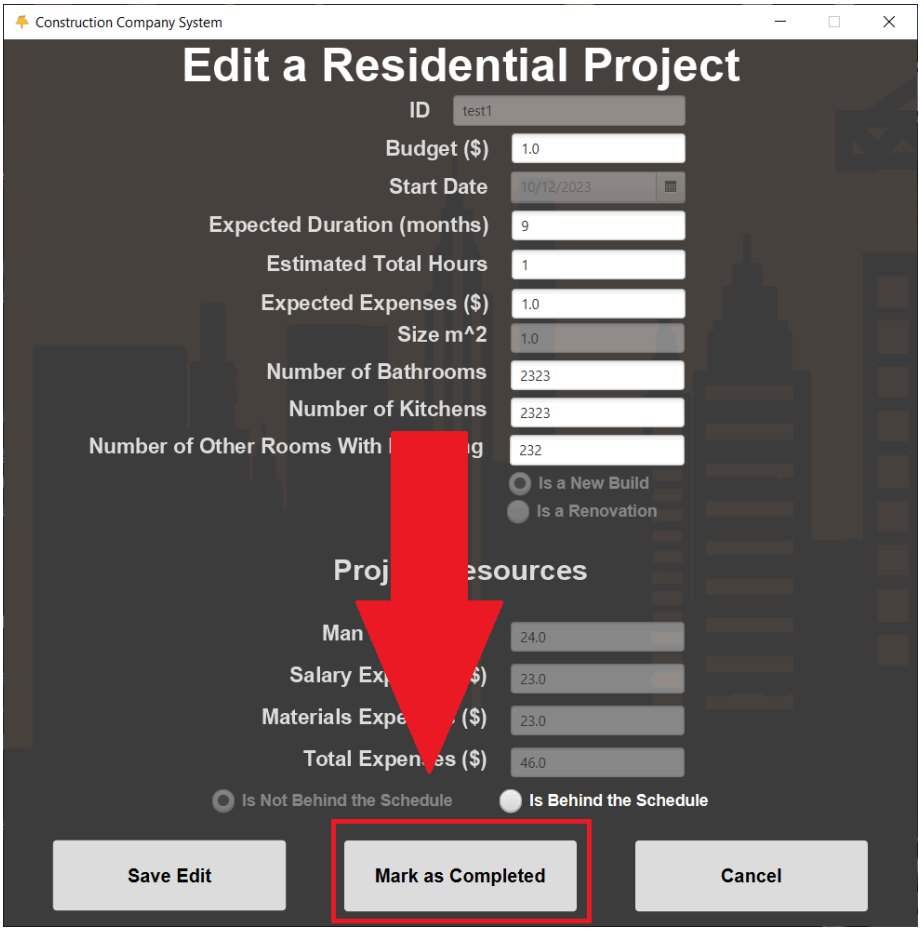


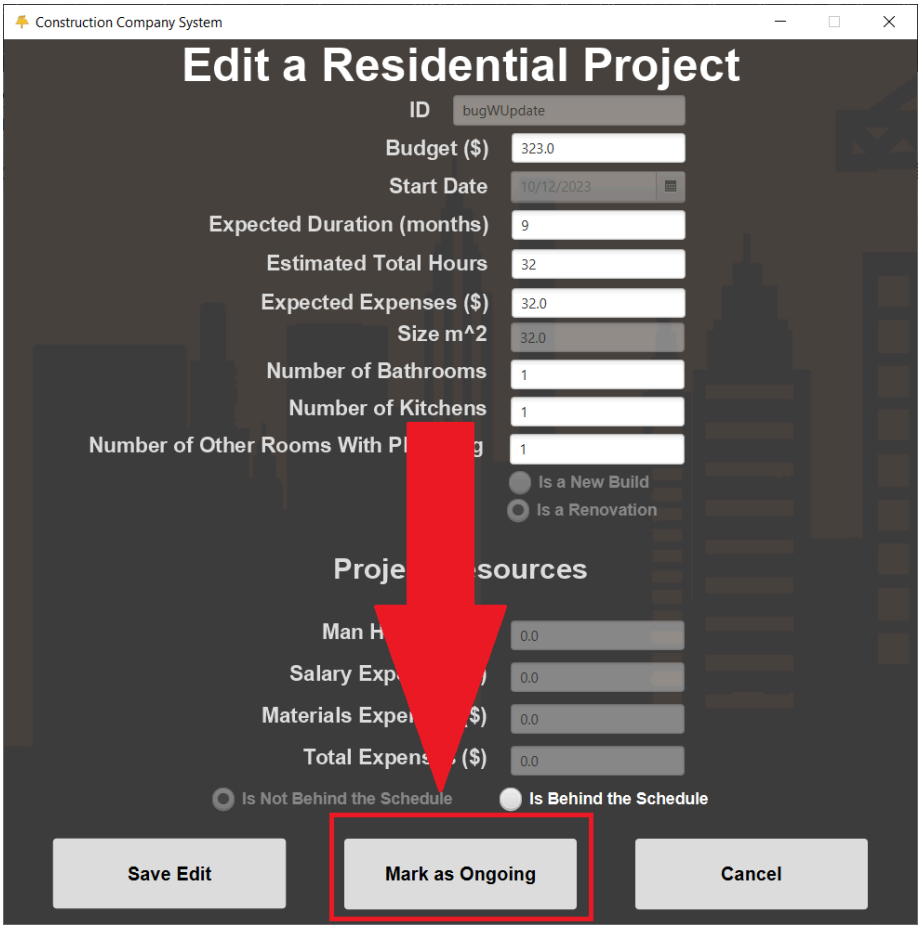


If you enter invalid data (e.g. letters, where the system requires numbers) the fields will be highlighted until you fix the issue. When entering illegal values the system will display an error and allow you to fix the values in question.


Construction Company System
Edit a Residential Project
ID
test1
Budget ($)
1.0saddsa
Start Date
10/12/2023
Expected Duration (months)
9
Estimated Total Hours
1
Expected Expenses ($)
1.0asdsa
Size m^2
1.0
Number of Bathrooms
2323ass
Number of Kitchens
2323sd
Number of Other Rooms With Plumbing
232as
Is a New Build
Is a Renovation
Invalid input for Number of Bathrooms
Project Resources
Man Hours Used
0.0
Salary Expenses ($)
0.0
Materials Expenses ($)
0.0
Total Expenses ($)
0.0
Is Not Behind the Schedule
Is Behind the Schedule
Save Edit
Mark as Completed
Cancel

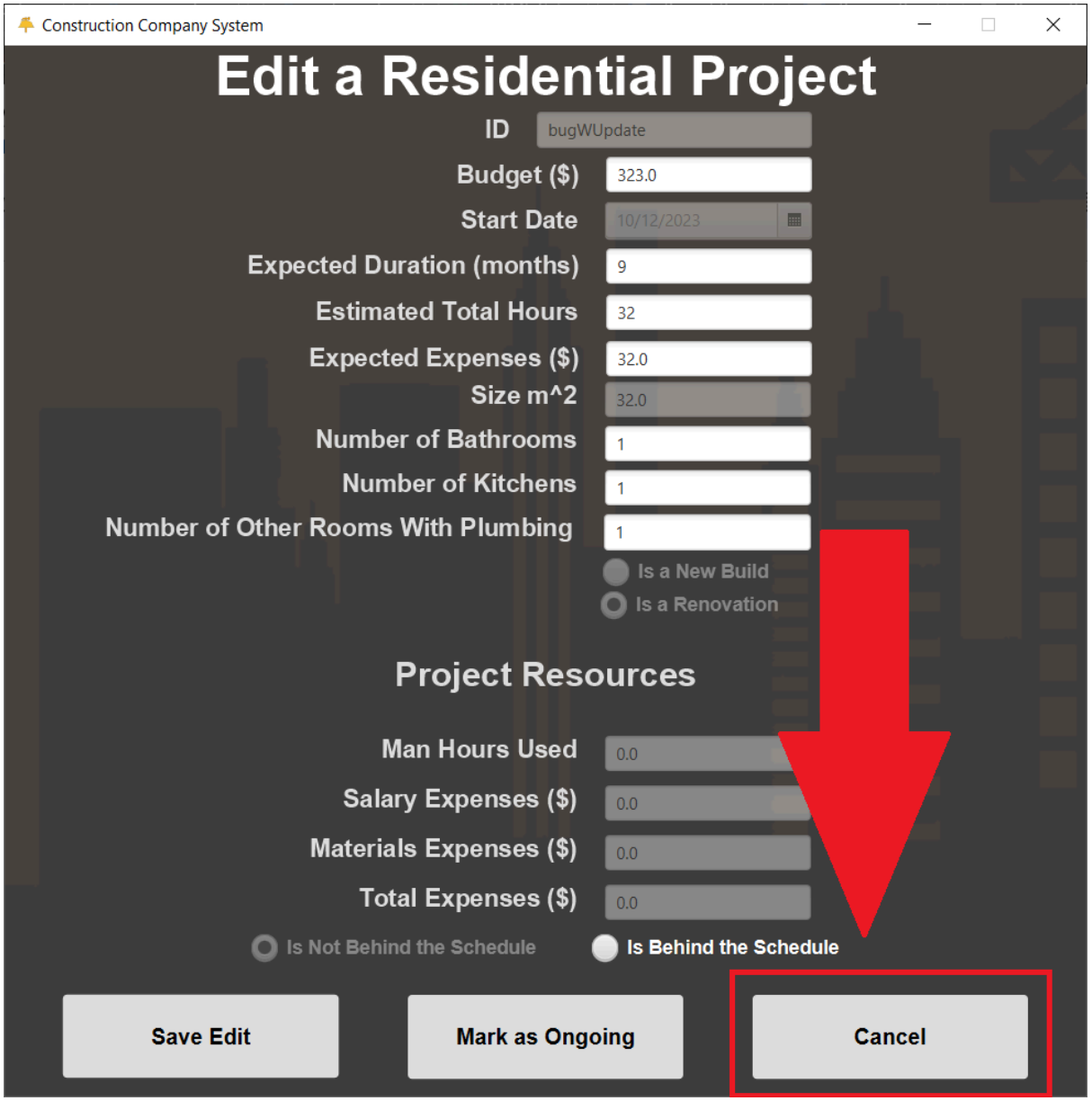

Construction Company System
Edit a Residential Project
ID
test1
Budget ($)
1.0
Start Date
10/12/2023
Expected Duration (months)
9
Estimated Total Hours
1
Expected Expenses ($)
1.0
Size m^2
1.0
Number of Bathrooms
2323
Number of Kitchens
2323
Number of Other Rooms With Plumbing
232
Is a New Build
Is a Renovation
Project Resources
Man Hours Used
24.0
Salary Expenses ($)
23.0
Materials Expenses ($)
23.0
Total Expenses ($)
46.0
Is Not Behind the Schedule
Is Behind the Schedule
Save Edit
Mark as Completed
Cancel


Construction Company System
Edit a Residential Project
ID
test1
Budget ($)
1.0
Start Date
10/12/2023
Expected Duration (months)
9
Estimated Total Hours
1
Expected Expenses ($)
1.0
Size m^2
1.0
Number of Bathrooms
2323
Number of Kitchens
2323
Number of Other Rooms With
232
Is a New Build
Is a Renovation
Man
24.0
23.0
23.0
46.0
Is Not Behind the Schedule
Is Behind the Schedule
Save Edit
Mark as Completed
Cancel


Construction Company System
Edit a Residential Project
ID
bugWUpdate
Budget ($)
323.0
Start Date
10/12/2023
Expected Duration (months)
9
Estimated Total Hours
32
Expected Expenses ($)
32.0
Size m^2
32.0
Number of Bathrooms
1
Number of Kitchens
1
Number of Other Rooms With
1
Is a New Build
Is a Renovation
Man
0.0
0.0
0.0
0.0
Is Not Behind the Schedule
Is Behind the Schedule
Save Edit
Mark as Ongoing
Cancel

Upon doing so, the project will get moved to either the completed projects list or the ongoing list and you will be sent back to the list of projects.

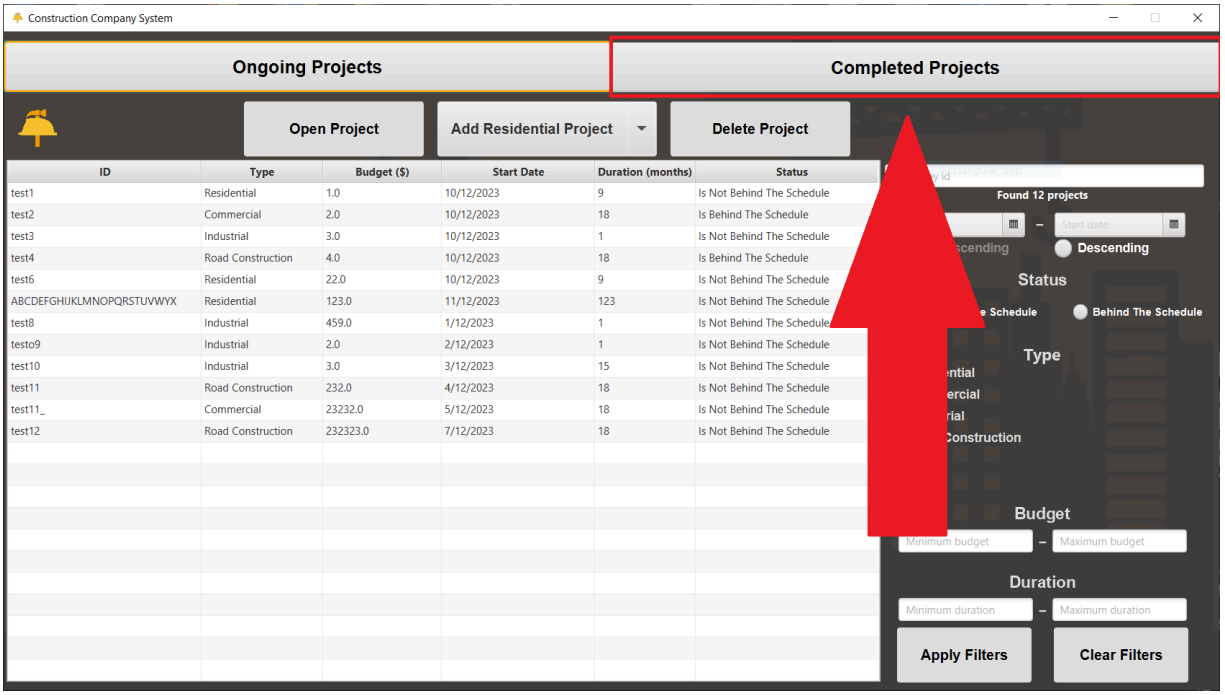
To cancel the editing without saving the values, simply press the *Cancel* button and confirm the intention to leave. This will bring you back to the project's display window.

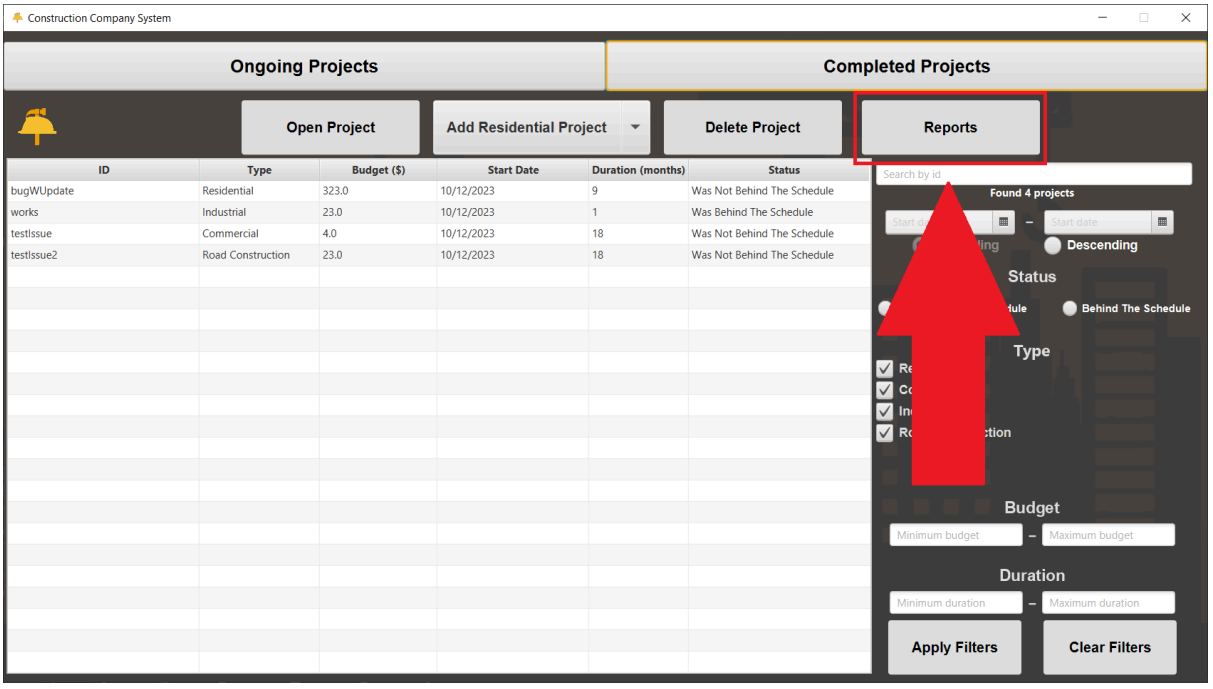


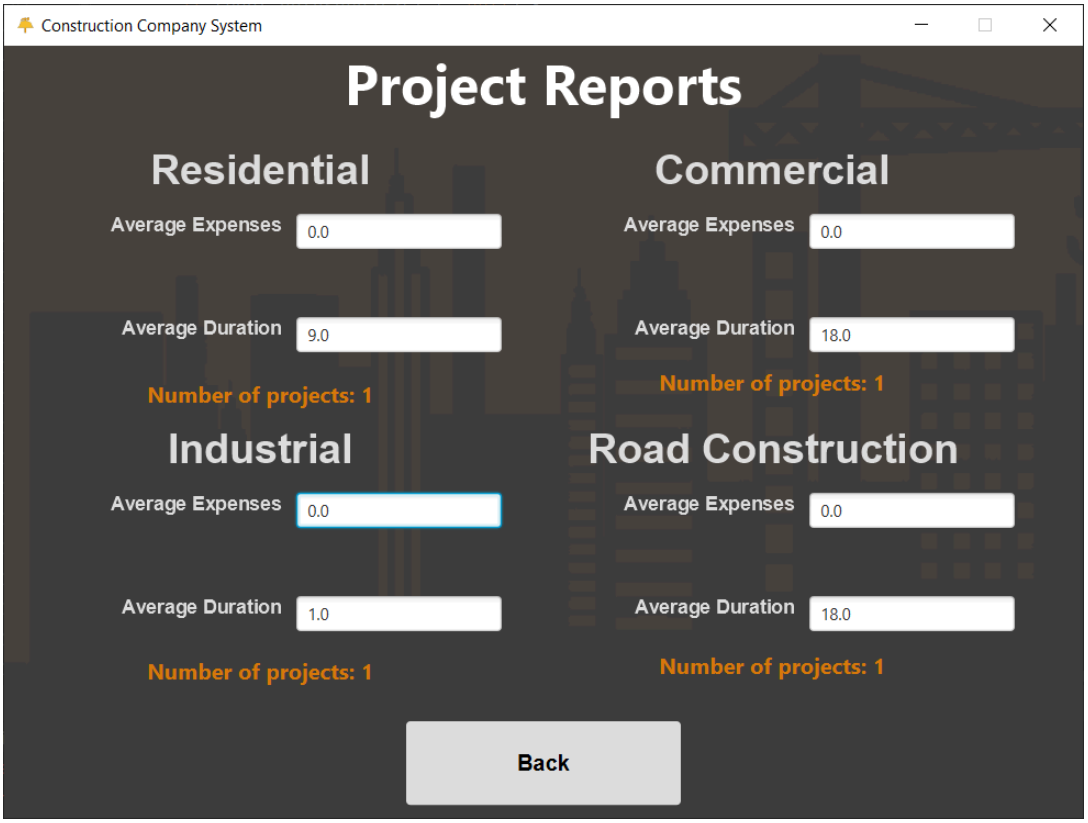
8. Access reports

Step 1. Press the Completed Projects button.

Step 2. Press the right button labeled as Reports. The system will display the average duration and expenses for each type of project, along with labels displaying how many projects of that specific type were completed.







If you wish to close the window press the *Back* button that will bring you back to the completed projects display.