# Syddansk Universitet

# Project Report

Faculty of Engineering.
BSc in Software Engineering
Supervisor: Sadok Ben Yahia
Project period: 2024.02.01 – 2024.06.03.

Group 11
Participants:

**Kacper Grzyb** (kagrz23@student.sdu.dk)
**Leonardo Gianola** (legia23@student.sdu.dk)
**Levente Sohár** (lesoh23@student.sdu.dk)
**Ignat Bozhinov** (igboz23@student.sdu.dk)
**Sebestyén Deák** (sedea23@student.sdu.dk)

# Contents

# Chapter 1

# Introduction

## Abstract

This project report aims at designing and developing a web-application with a heat production optimization system in the imaginary city of Heatington. The objective is to minimize costs and maximize profits from electricity production while ensuring consistent heat delivery all year round. The system includes multiple modules, some of the key ones being: an Asset Manager, Source Data Manager, Result Data Manager, an Optimizer, and a Data Visualization module. This project addresses the transition from manual to optimized scheduling of heat production, leveraging advanced data management and optimization techniques.

## 1a Background

Heatington's district heating network is an essential aspect of urban distribution and supplies heat to a large number of structures through a system of pipelines and production units. The Combined Heat and Power (CHP) network incorporates both conventional heat-only boilers and CHP units that produce both heat and power. Originally, the scheduling of these heat production units has been done in a manual manner, compromising efficiency and incurring high costs. New energy market conditions, with their constant calls for economic efficiency in the energy production process, demand a more complex and automated approach to the scheduling of heat production.

District heating systems centralize the production of heat and distribute it through hot water to different buildings in the city. The heat from the hot water is used for

both heating the building and preparing hot water for the occupants. After releasing its heat, the cooled water is recirculated back to the plant where it is reheated. This cycle ensures a constant provision of heat, although controlling it is always a challenge given the unpredictable nature of heat demand at different times and seasons.

# 1b Project Brief

Developing an optimization system that schedules heat production for Heatington's district heating network at the lowest possible cost while maximizing profit from electricity production is the main goal of this semester-long project. The system is designed to handle data from different production units, factoring in production costs, electricity prices, and heat demand to generate optimal schedules that ensure operational efficiency and financial profitability.

The system comprises several key modules:

- **Asset Manager (AM)**: Captures non-operational data of the heating grid and production units, including operational conditions, production rates, cost data, and CO2 emission rates. It ensures that the most up-to-date information required for optimization processes and other system elements is available.
- **Source Data Manager (SDM)**: Processes dynamic data such as heat demand and electricity prices in the form of time series, as the heating grid requirements often change over time. The SDM aids the optimization process by providing the temporal context for scheduling decisions.
- **Result Data Manager (RDM)**: Stores optimized scheduling results and performance data for each production unit. This module retains copies of all results for validation, reporting, or further analysis.
- **Optimizer (OPT)**: The core component responsible for calculating optimal heat production schedules. It considers all available production units, their costs, and electricity market dynamics to generate schedules that secure heat availability while minimizing costs. The Optimizer will be implemented in two scenarios to progressively incorporate complexity, starting with basic heat-only units and extending to include electricity-producing and consuming units.
- **Data Visualization (DV)**: Provides a graphical interface to review the heating grid format, production facilities, and optimization outcomes. This module

enhances system usability by allowing stakeholders to easily understand metrics, time series, and overall system performance through graphical representations.

This project enables Heatington's district to efficiently and automatically generate the best schedule, ensuring adequate, cost-effective, and reliable heat delivery to its residents.

# Chapter 2

# Release Planning

This chapter contains the release planning as we wrote it before the first sprint. As the project matured we made more accurate representations, which are included in the Sprint Materials chapter.

## User Stories

### Heating System Manager

As a Heating System Manager, I want to visualize real-time heat production data (produced heat, produced/consumed electricity, production costs, consumption of primary energy and produced amount of CO2) through a user-friendly dashboard that allows me to choose the period I want that helps me make informed decisions, optimize heat production and minimize costs.
HIGH Priority, 13 User Points
Connected Requirements: Result Data Manager, Optimizer

### Data Administrator

As a Data Administrator for the heat production optimization application, I need to effectively import data to ensure optimal performance and efficiency of the heat production system.
HIGH Priority, 8 User Points
Connected Requirements: Source Data Manager, Asset Manager

## Financial Analyst

As Financial Analyst I want to look at previous month's data and look for improvement. I must know when it's the best time to buy electricity, and when to sell it. I need to know which machines are the most efficient, and whether it's the best for the company to keep the already used machines, or to change some of them. For this I need to see the operation costs and a few graphs.
MEDIUM Priority, 13 User Points
Connected Requirements: Result Data Manager, Data Visualization

## Sustainability Officer

As a Sustainability Officer I want to promote sustainable energy practices. With background in environmental engineering and deep commitment to combating climate change I want an application where I can easily visualize the emission and the potential improvements I can make.
LOW Priority, 5 User Points
Connected Requirements: Result Data Manager, Data Visualization

## Definition of Done

- Result Data Manager component finished
- Asset Manager component finished
- Source Data Manager component finished
- Optimizer component finished
- Continuous Integration implemented with at least 60 % code coverage
- UI Elements are functional and responsive across all devices
- UX reviewed and tested by external users
- Reviewed by Product Owner
- Group Supervisor has reviewed and approved the features
- All known bugs and issues are resolved and documented
- Third party libraries are properly licensed
- Product increment has been built into release mode and tested by the team
- All release documentation has been written and reviewed by group supervisor
- Scenario 1 and 2 are Implemented

# Chapter 3

# Sprint Materials

In this chapter all the materials from the sprints can be found.

## 3a Sprint 1

### Sprint Review

**Project:** Semester Project Group 11
**Sprint Duration:** March 5 - March 19, 2024
**Team Members:** Kacper Grzyb, Sebestyen Deak, Ignat Bozhinov, Leonardo Gianola, Levente Sohar
**Stakeholders:** Sadok Ben Yahia

### 1. Sprint Goals and Outcomes

- **Goal 1:** Move epic and user stories into Jira
  **Status:** Completed. All the epics and user stories are in Jira now.
- **Goal 2:** Divide Roles
  **Status:** Completed. Product Owner and Scrum Master Roles have been given.
- **Goal 3:** Create .gitignore file
  **Status:** Completed. Created .gitignore file.
- **Goal 4:** Break down User Stories into requirements with MoSCoW
  **Status:** Completed. All the different User Stories have a Must Do (-M), Should Do (-S), Can Do (-C), Would Not Do (-W).

- **Goal 5:** Rewrite tasks into User Stories
  **Status:** Completed.
- **Goal 6:** Add User Points to User Stories
  **Status:** Completed. Every User Story has been rated in story points.
- **Goal 7:** Gantt Chart
  **Status:** Completed. Every Task has been estimated, and a Gantt Chart has been made according to this and our timeframe.
- **Goal 8:** Create Sprint Review
  **Status:** Completed.

## 2. Completed Work

Transitioning our project management to Jira, we've streamlined our workflow and enhanced visibility into our tasks and progress. Recognizing the importance of role clarity in optimizing team performance, we successfully delineated roles and responsibilities. Implementing best practices in version control, we established a .gitignore file. Employing the MoSCoW method to prioritize requirements, we gained clarity on project scope and stakeholder expectations. Restructuring our tasks into user stories, we've shifted our focus from implementation details to user-centric outcomes, fostering a deeper understanding of user needs and motivations. Introducing user points to our user stories allowed us to quantify complexity and effort more accurately, facilitating resource allocation and sprint planning. Creating a Gantt chart provided us with a visual roadmap for project execution, enabling us to sequence tasks, allocate resources, and identify dependencies more effectively. Instituting sprint reviews has fostered transparency, accountability, and continuous improvement within our agile framework.

## 3. Unfinished Work

Everything we set out to do during this sprint we have accomplished.

## 4. Quality and Technical Issues

We haven't started coding yet, and only used already established software for our work, therefore we didn't have any technical issues.

## 5. Team Dynamics and Collaboration

Work has been mostly divided equally, with everyone doing their part. Communication was clear and to the point.

## 6. Processes and Tools

Jira helps keep track of the backlog and manage the sprint. For making the Gantt Chart, Canva was used, which helped speed up the process.

## 7. Stakeholder Feedback

When talking with our supervisor Sadok, he approved of the direction we were heading this sprint, emphasizing making Dashboards.

## 8. Obstacles and Impediments

We have been able to complete all the goals without any obstacles or impediments.

## 9. Successes and Wins

The biggest win for the team was finishing all of our goals in time.

## 10. Action Items for Improvement

Breaking the requirement into small tasks that can be worked on independently, therefore not everything has to be done in the one meeting we weekly.

16/03/2024

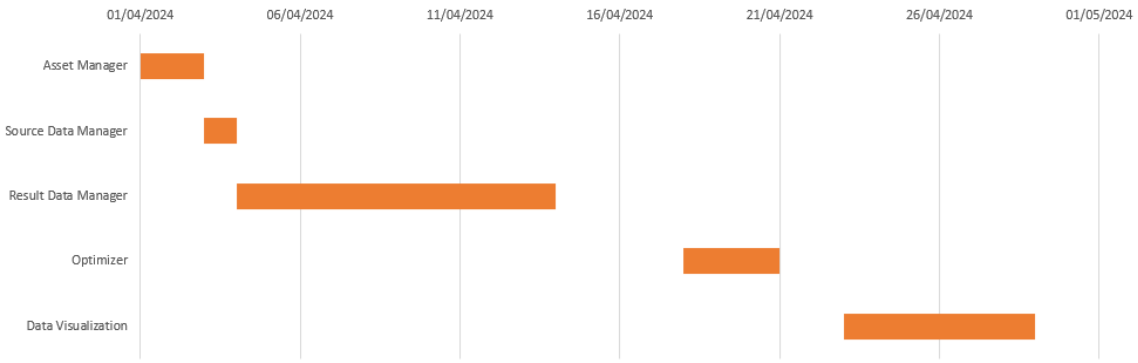



Figure 3.1: Optimal Gantt Chart


Figure 3.2: Realistic Gantt Chart

# 3b Sprint 2

## Planning

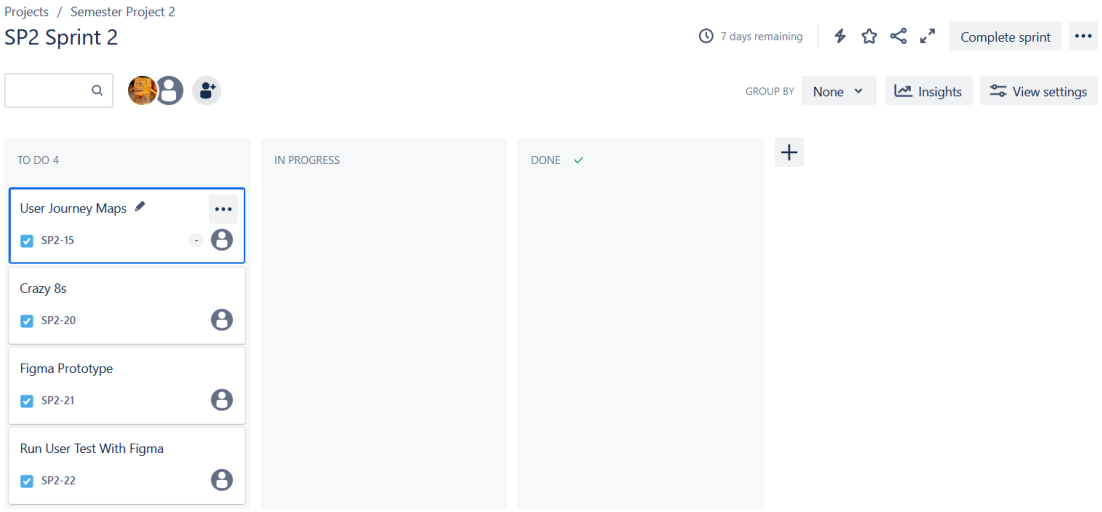



Figure 3.3: Sprint 2 Planning Package

## Daily Scrum 02/04/2024

- The team completed 2 out of 4 issues.
- 2 issues are currently in progress, one of which is very close to being finished and the other one is expected to be finished by the end of the week.

## Roadblocks

- The team faced a few conflicting ideas and a wrong understanding of how the Result Data Manager and Asset Manager components are supposed to look like. They were solved through an online Discord meeting.
- Some team members are still on holidays, which makes organising work a bit harder.

## Plans for the rest of the Sprint

- Polish the Figma prototype made.
- Get feedback on the Figma prototype.
- Begin discussion about starting the development phase.

## Metrics and Progress

The team has attached screenshots of the current state of the sprint backlog and the sprint status report to give information about how much work has been done and how much work still needs to be done.

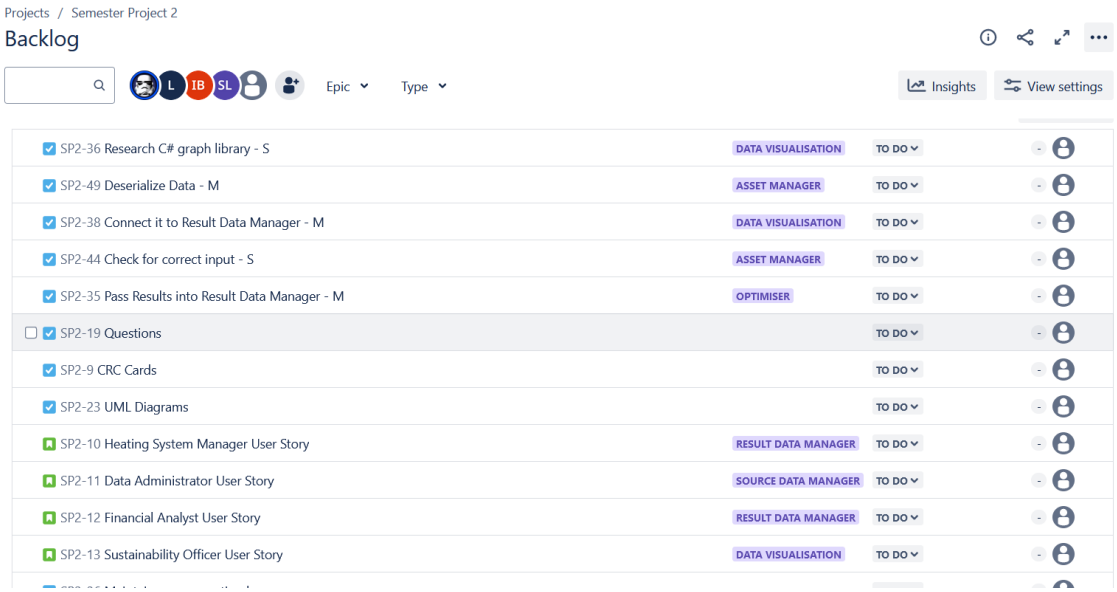



Figure 3.4: Daily Scrum Backlog 1

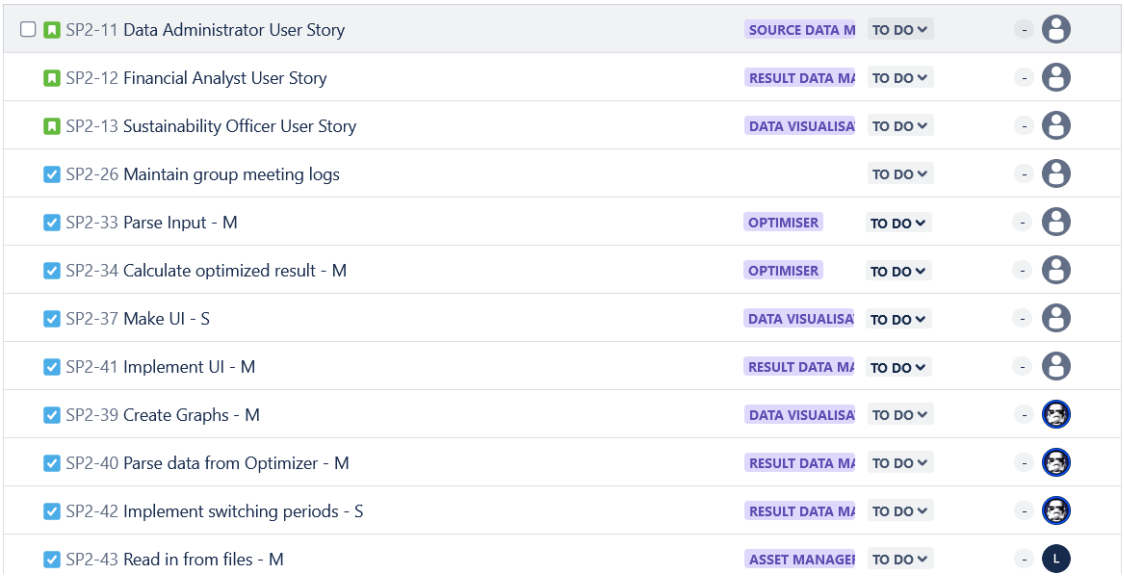



Figure 3.5: Daily Scrum Backlog 2

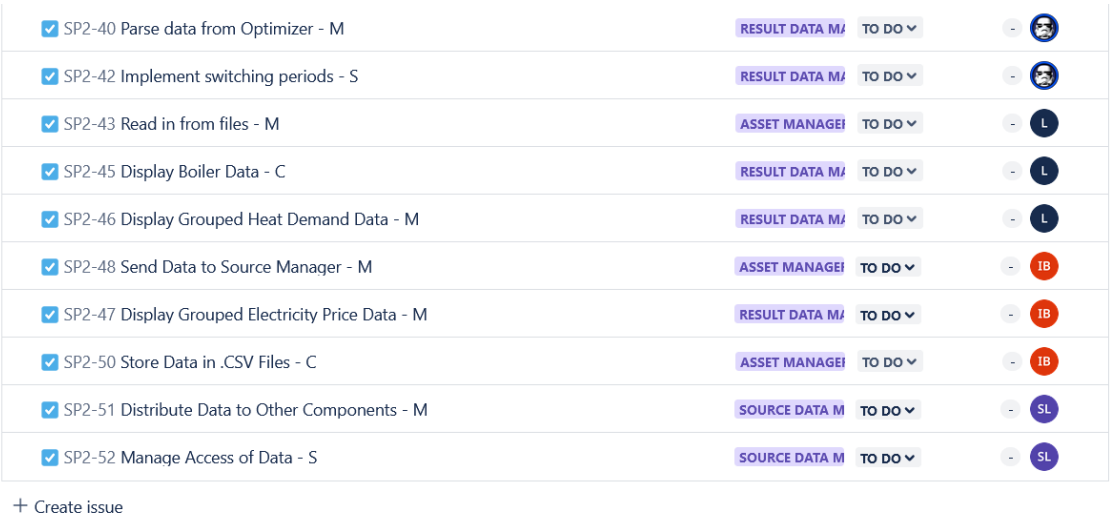



Figure 3.6: Daily Scrum Backlog 3

## Sprint Review

## What went well

- Remote meeting to re-align on the project direction
- All sprint tasks done despite remote work due to the Easter holidays
- Remote communication
- Willingness to pivot, make changes to the project

## What to improve

- Spend more time on understanding the project requirements – the team had a wrong idea of what the Result Data Manager, Asset Manager and Source Data Manager should consist of which created a setback and meant some of the plans for the project need to be remade, such as the tasks on Jira
- Pay attention to time zones when doing remote work – the time zone difference created a minor issue during one of the team's remote meetings
- Plan out and divide work more carefully to avoid misunderstandings and vagueness

## What the team aims to improve in the next Sprint

- Align the project with the requirements
- Remove vagueness from the project direction
- Remove vagueness from tasks for each team member

# 3c Sprint 3

## Planning

The team's goal for this sprint is to be ready for the Midterm Evaluation that is to happen on Wednesday the 17th of March. The team has planned out to work on the 3 required components: Result Data Manager, Asset Manager and Source Data Manager. The components will be created in Asp.Net using Razor Pages and only the boiler configuration for the first iteration will be supported by the app created. The components will be developed throughout the week depending on the time of each individual developer . Only one team member was missing during the Sprint 3 Planning Meeting, and he will be informed of everything discussed through online matters. The only issue that made the planning process harder was the Jira backlog being outdated due to the product vision changes. This will be fixed in a future sprint.

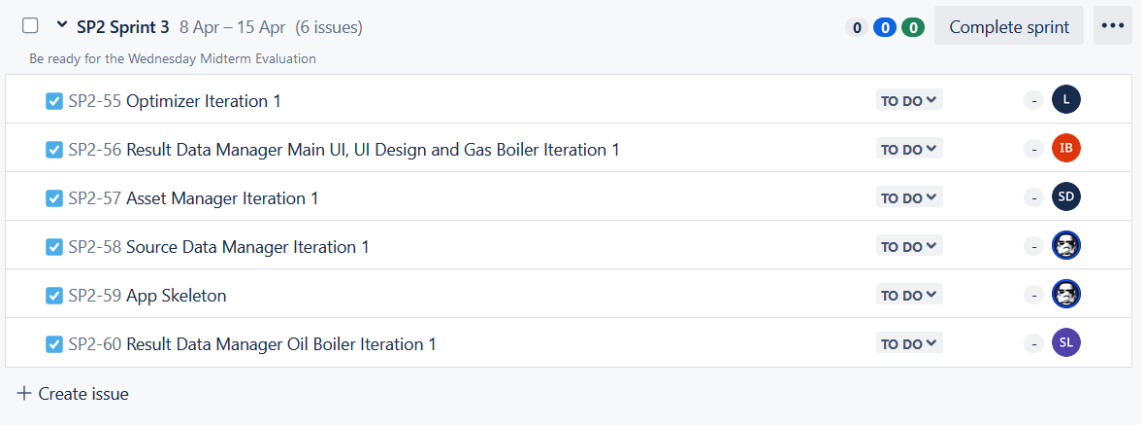



Figure 3.7: Sprint 3 Planning

## Daily Scrum 16/04/2024

### Team Update

- The team completed 6 out of 6 issues.
- The goal for the sprint of being ready for the first iteration presentation has been met.

### Roadblocks

- Most of the development did not see any roadblocks due to deliberate planning done beforehand.
- Team members helped each other to make sure no one is stuck, and the tasks are finished on time.

### Plans for the Next Sprint

- Fix bugs.
- Continue development.

### Metrics and Progress

The team has attached screenshots of the current state of the sprint backlog and the sprint status report to give information about how much work has been done and how much work still needs to be done.

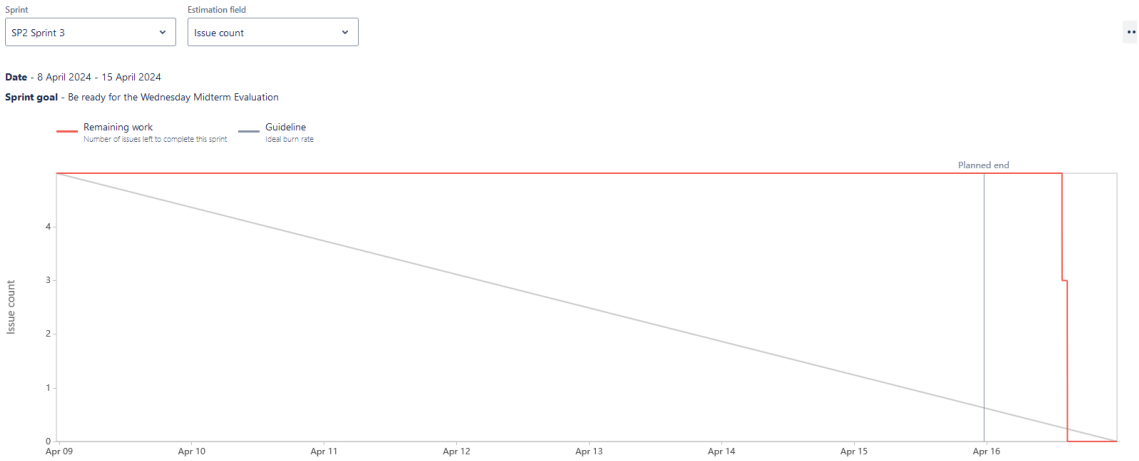



Figure 3.8: Daily Scrum Burndown Chart

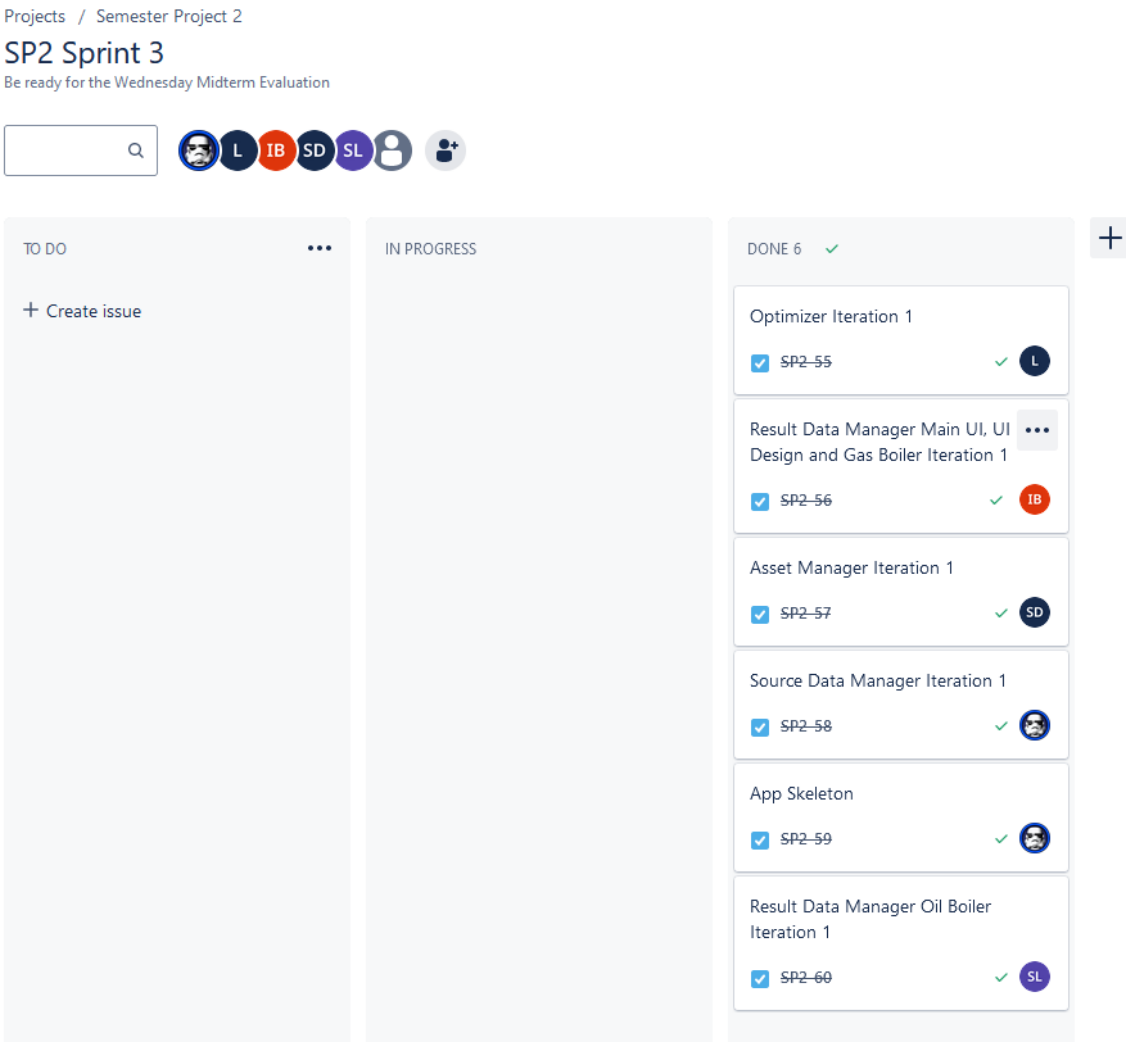



Figure 3.9: Daily Scrum Backlog

## Sprint Review

**Project:** Semester Project Group 11
**Sprint Duration:** April 9 - April 23, 2024
**Team Members:** Kacper Grzyb, Sebestyen Deak, Ignat Bozhinov, Leonardo Gianola, Levente Sohar
**Stakeholders:** Sadok Ben Yahia

## 1. Sprint Goals and Outcomes

During this sprint, we aimed to iterate our plans for the optimizer program and make a working prototype for the presentation.

- **Goal 1:** Optimizer iteration 1
  **Status:** Completed. The optimizer (for now) looks at the heat demands and if it's below the gas boiler capacity, it only uses that. If exceeded, the other boiler turns on.

- **Goal 2:** UI, UI Design and Gas Boiler Iteration 1
  **Status:** Completed. The app skeleton has been created using Bootstrap for better UX.

- **Goal 3:** Asset Manager Iteration 1
  **Status:** Completed. Created classes for all the boilers, both for iteration 1 and 2.

- **Goal 4:** Source Data Manager Iteration 1
  **Status:** Completed - with minor issue. The Source Data Manager reads in the data from CSV files and creates objects from it. The only issue we have with it is that since Apple's MacOS uses a different DateTime format than Windows, it throws an exception for some of the dates.

- **Goal 5:** Result Data Manager Oil Boiler Iteration 1
  **Status:** Completed.

## 2. Completed Work

We had the midterm presentation during this sprint, so our main focus was on getting the program in a state that can be presented and making the presentation. We focused on not just making the program work, but also making it easily expandable,

therefore we have less work to do in the second iteration. For the visual UI, we used Razor pages, and in that Bootstrap. We made all the components work almost flawlessly, and the end result visually remained close to our Figma prototype.

## 3. Unfinished Work

Everything we set out to do during this sprint we have accomplished.

## 4. Quality and Technical Issues

There remained to be a bug, where Mac devices aren't able to read in all the data from the CSV file, since the OS expects the months to be where the days are in the source data. So after the day exceeds the 13th day, it throws an exception.

## 5. Team Dynamics and Collaboration

Work has been mostly divided equally, with everyone doing their part. Communication was clear and to the point. We had weekly meetings for scrum.

## 6. Processes and Tools

Jira helps keep track of the backlog and manage the sprint. Razor pages and Bootstrap have been used for UI. We sometimes looked back at our Figma prototype for reference.

## 7. Stakeholder Feedback

After our midterm presentation, we got feedback from 2 supervisors, both were supportive of our development methods and the state of the program. The only criticism we got was regarding our presentation style, and we will try to keep that in mind for the next time.

## 8. Obstacles and Impediments

We have been able to complete all the goals without any obstacles or impediments.

## 9. Successes and Wins

The biggest win for the team was the feedback we got after the presentation both from the supervisors and the other students.

## 10. Action Items for Improvement

Setting a hierarchy amongst tasks so no one has to wait for the other to finish.

24/04/2024

# 3d Sprint 4

## Planning

The team's aim for this sprint is to try and make the final product, since we only have about 4-5 weeks before having to present it in front of the other students and teachers. This means updating the optimizer, and the UI. We also plan on adding graphs which show the given data, in various configurations. We also plan to fix the bug with the Source Data Manager. Two team members were missing during the Sprint 4 Planning Meeting, but we talked about our goals previously and they will be informed of everything discussed through online matters.

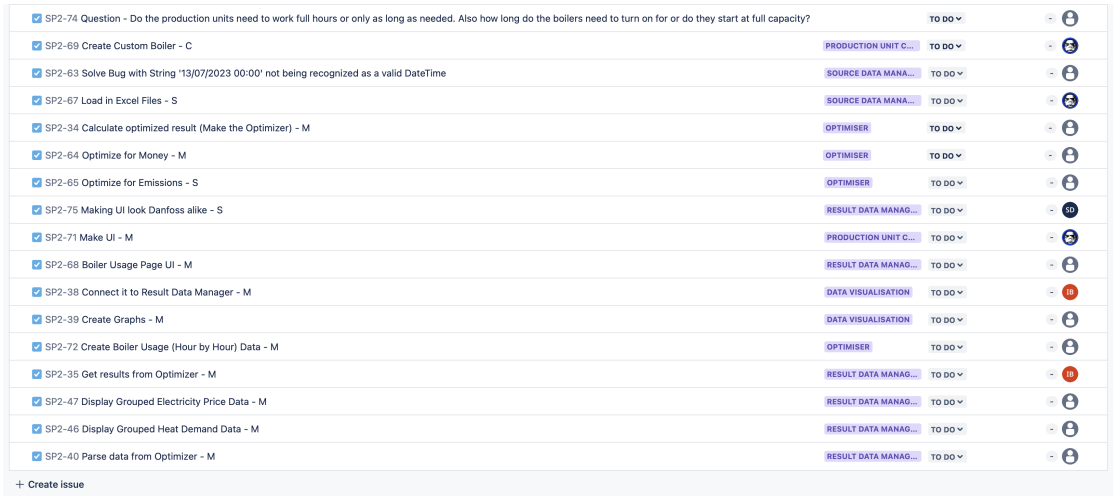



Figure 3.10: Planning Backlog

## Daily Scrum 29/04/2024

## Team Update

- The team finished 7 issues and made major progress towards the biggest issue out of 24 issues in the current sprint.
- Kacper and Sebestyén will finish work on the optimizer and work on other tasks while Leonardo will continue to work on his neural network optimizer until the end of the sprint. Ignat and Levente are moving on to other tasks.
- The goal for the sprint is to complete as many issues as possible.

## Roadblocks

- The team needed to realign on the approach for the implementation of the optimizer, change the implementation of some of the data structures and get everyone on the same page in the code structure.
- Every roadblock was talked about and resolved on this Monday's meeting.

## Plans for the Next Sprint

- Continue completing issues from the backlog, while focusing on the must-have features.
- Come up with a solution for having multiple optimizers and custom production units.
- Make sure all requested features are accounted for in the sprint backlog.

## Metrics and Progress

The team has attached screenshots of Jira for progress metrics.

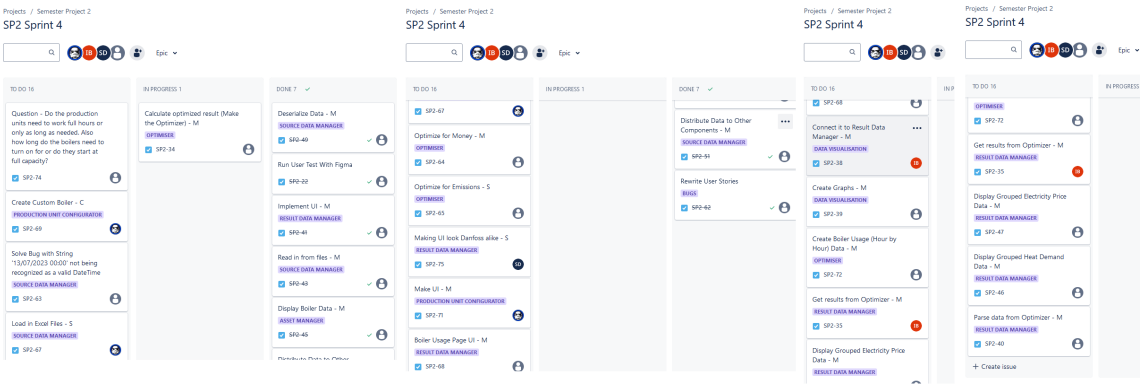



Figure 3.11: Daily Scrum Backlog

## Sprint Review

**Project:** Semester Project Group 11
**Sprint Duration:** April 23 - May 7, 2024
**Team Members:** Kacper Grzyb, Sebestyen Deak, Ignat Bozhinov, Leonardo Gianola, Levente Sohar
**Stakeholders:** Sadok Ben Yahia

## 1. Sprint Goals and Outcomes

The goal for this sprint was to start fully developing the program. For this, we added every issue that's a must (according to the MoSCoW breakdown we made) for the minimal viable product. We overshot our capabilities on purpose, so we see what to do, and we will continue working on this in the next sprint as well.

- **Goal 1:** Create Comparable Data
  **Status:** Completed. Created two additional classes based on the Optimizer class, that create scenarios which are not the optimal case, to have something to compare our solution to.

- **Goal 2:** Boiler Usage Data
  **Status:** In Progress. The data of which boiler is running when is created, it needs to be grouped and displayed to the user.

- **Goal 3:** Neural Network Optimizer
  **Status:** In Progress. The program is written for a neural engine to find the optimal solution, it just needs to be trained, and then introduced to the project environment.

- **Goal 4:** Create Graphs
  **Status:** To Do. We plan on displaying the different scenarios for the user next to each other in bar graphs.

- **Goal 5:** Choosing Boilers for the Optimization
  **Status:** To Do. We want the user to be able to choose which boilers they want to use to get the optimized results.

- **Goal 6:** Save to CSV files
  **Status:** To Do.

## 2. Completed Work

The members of the group are focusing on the upcoming Mathematics Exam, not on the project. The next sprint is planned to be more productive. Still, everyone is moving slowly forward with their to-dos. The only task that has been fully accomplished was requested by our supervisor.

## 3. Unfinished Work

Many things, including the Data Visualization, Creating and Choosing boilers.

## 4. Quality and Technical Issues

All the bugs from the last sprint have been fixed. There are no known issues at the moment.

## 5. Team Dynamics and Collaboration

Work has been mostly divided equally, with everyone doing their part. Communication was clear and to the point. We had weekly meetings for scrum.

## 6. Processes and Tools

Jira helps keep track of the backlog and manage the sprint. Razor pages and Bootstrap have been used for UI. We sometimes looked back at our Figma prototype for reference.

## 7. Stakeholder Feedback

The feedback of our supervisor has been to provide some reference point for the data that our optimizer gives as the end result. This has been mostly accomplished in this sprint.

## 8. Obstacles and Impediments

The pressure of the upcoming math test reflected on the amount of work done.

## 9. Successes and Wins

There has not been any outstanding win or success during this sprint.

## 10. Action Items for Improvement

Pass the exam with good grades, so all energy can be focused on the project. 07/05/2024


Figure 3.12: Sprint 4 Burnup Report

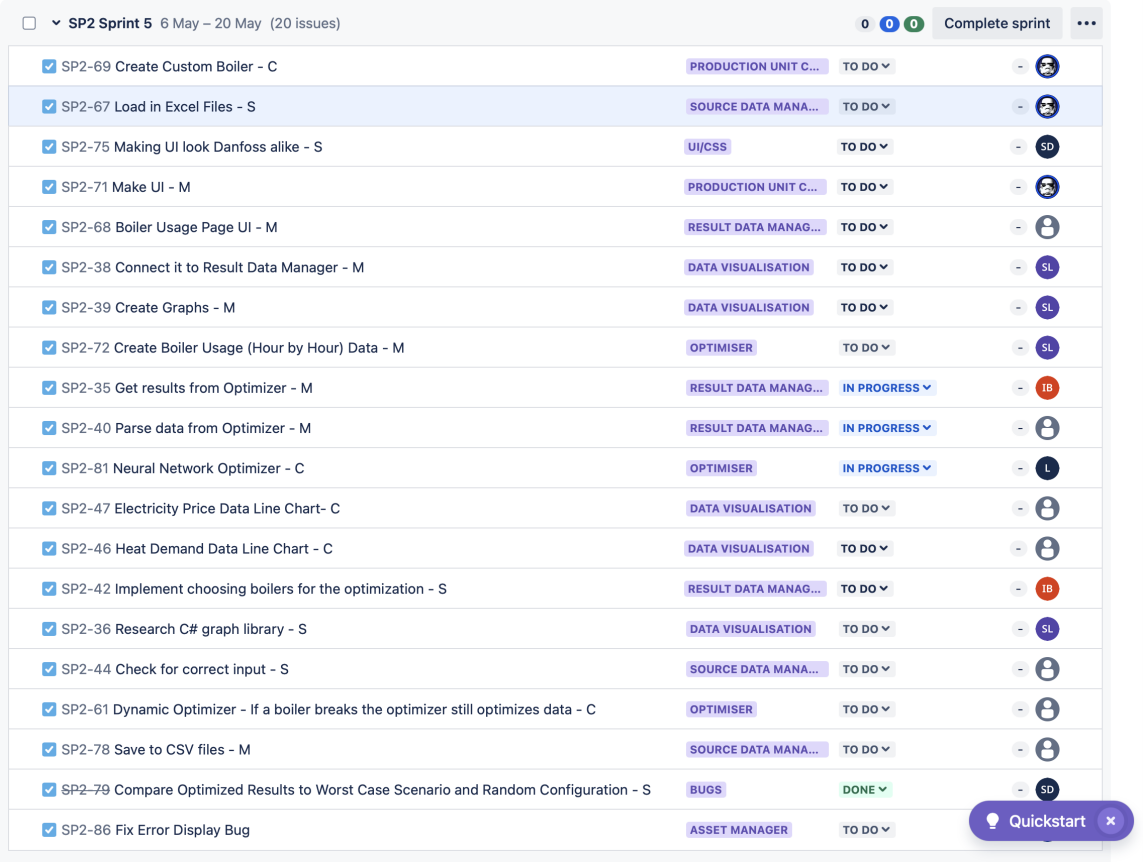



Figure 3.13: Sprint 4 Issues

# 3e Sprint 5

## Planning

We aim on delivering an almost final version of our product. The Danfoss Demo is going to take place at the end of this Sprint, and we want more than the MVP to be ready by then. We plan on making the Data Visualization with showing the end result and maybe the initial data given in, boiler customization and choosing which one to run for the optimization and a few more smaller things. All members were present on the meeting, and issues have been distributed among us.

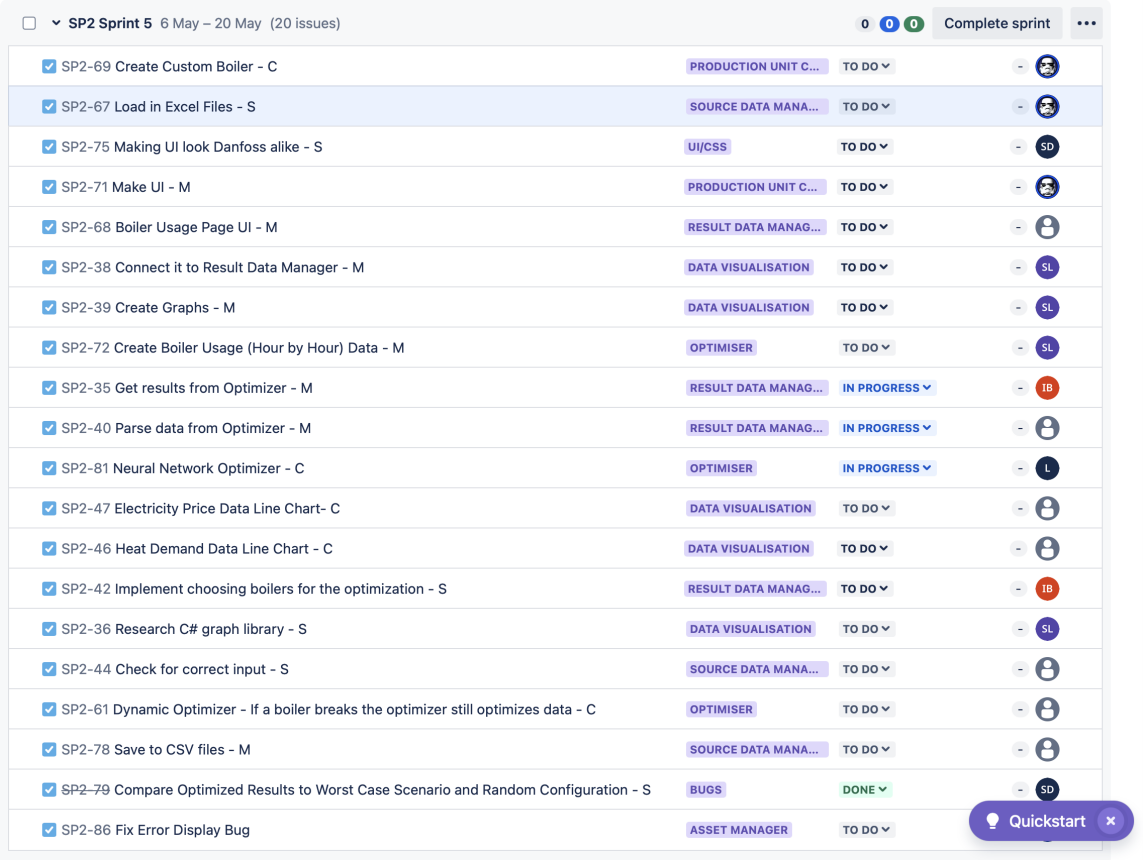



Figure 3.14: Planning Backlog

## Daily Scrum 19/05/2024

## Team Update

- The team finished 7 issues and made major progress towards the biggest issue out of the 24 issues in the current sprint.
- Leonardo is continuing to work on the Neural Network solution for the optimizer. Levente is making the graphs to show and compare the data from the program. Ignat is finalizing the looks of the pages. Kacper and Sebi are working on being able to read in and to save to different files.
- The goal for the sprint is to get as close to the final product as possible for the presentation at the end of this sprint.

## Roadblocks

- The team was slowed down by the math exam last week.
- Every roadblock was talked about and resolved in this meeting.

## Plans for the Next Sprint

- Because we plan on finishing all the must features next sprint, we plan on polishing any mistakes, and maybe adding a few of the Could features.
- Build on the feedback given at the presentation.
- Make sure all requested features are accounted for in the sprint backlog.

## Metrics and Progress

The team has attached screenshots of Jira for progress metrics.

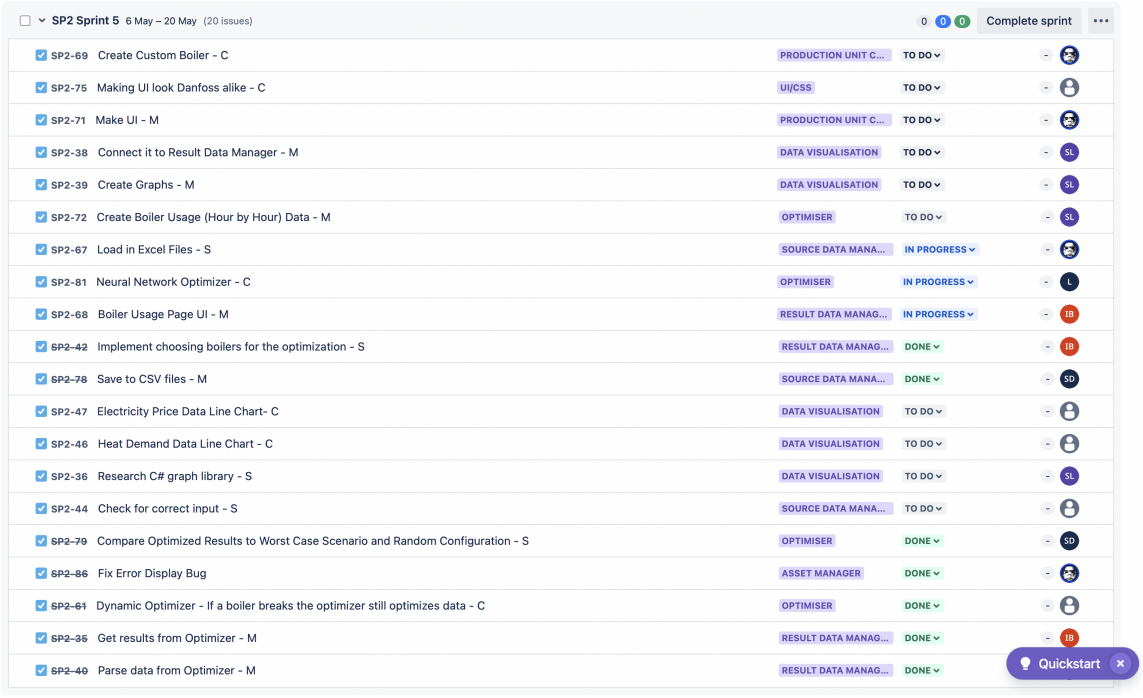



Figure 3.15: Daily Scrum Backlog

## Sprint Review

**Project:** Semester Project Group 11
**Sprint Duration:** May 7 - May 21, 2024
**Team Members:** Kacper Grzyb, Sebestyen Deak, Ignat Bozhinov, Leonardo Gianola, Levente Sohar
**Stakeholders:** Sadok Ben Yahia

## 1. Sprint Goals and Outcomes

The goal for this sprint was to finish developing the program's must-have features and polish them to be acceptable for the presentation. We carried on with the remaining tasks from Sprint 4 and continued the development of the full program.

- **Goal 1:** Create Boiler Usage
  **Status:** Completed. Created an additional page inside the program, where the distribution amongst boilers can be viewed, in addition to the saving options.

- **Goal 2:** Create Custom Boiler
  **Status:** Completed. The user of the program can now create a fully customizable boiler.

- **Goal 3:** Neural Network Optimizer
  **Status:** Completed. The program is written for a neural engine to find the optimal solution.

- **Goal 4:** Create Graphs
  **Status:** Completed. All vital and comparable data is displayed for the user to put the optimized scenario in context.

- **Goal 5:** Choosing Boilers for the Optimization
  **Status:** Completed. We made it possible for the user to choose which boilers they want to use to get the optimized results.

- **Goal 6:** Save to External Files
  **Status:** Completed. The user is now able to save the boiler usage and the optimized data to external CSV and Microsoft Excel files.

- **Goal 7:** Heat Demand and Electricity Price Line Chart
  **Status:** To-Do. We want to display additional information about the provided data to give a deeper understanding to the user.

## 2. Completed Work

Many things have been completed during this sprint, all big steps towards the end goal. The biggest things are the new UI of the application, saving to files, and making custom boilers.

## 3. Unfinished Work

Not that many things remain, essentially all the must-have features from the MoSCoW breakdown have been created.

## 4. Quality and Technical Issues

The graphs don't display correctly from time to time, but the bug can be resolved by reloading the page.

## 5. Team Dynamics and Collaboration

Work has been mostly divided equally, with everyone doing their part. Communication was clear and to the point. We had weekly meetings for scrum.

## 6. Processes and Tools

Jira helps keep track of the backlog and manage the sprint. Razor pages, Bootstrap, and JavaScript have been used for UI.

## 7. Stakeholder Feedback

There hasn't been much feedback after the presentation.

## 8. Obstacles and Impediments

None noted.

## 9. Successes and Wins

The presentation was a great success, and the program is almost finished.

## 10. Action Items for Improvement

None noted. 22/05/2024

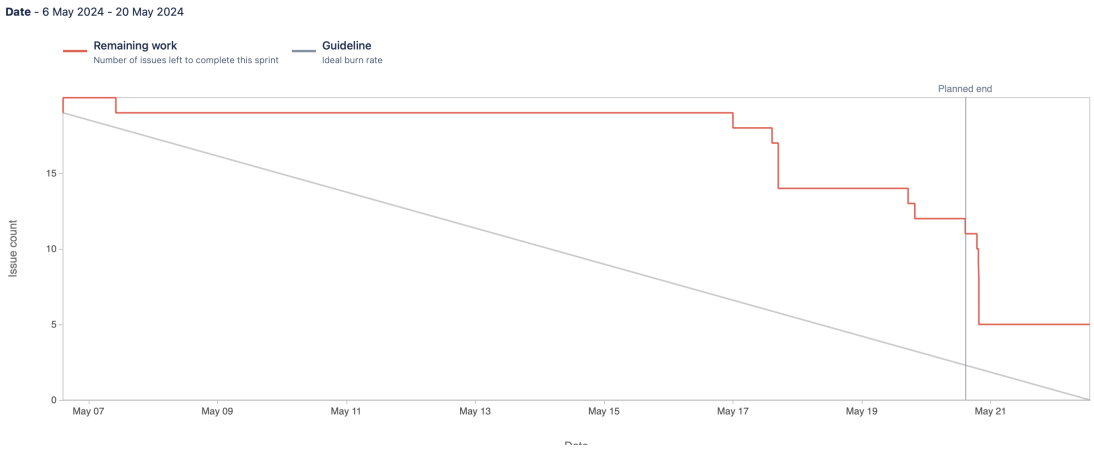



Figure 3.16: Sprint 5 Burndown Chart

# Notes

The rest of the project was finished before submission in a 6th Jira Sprint, that was outside of project requirements and Itslearning Sprint submission tasks, so we decided to omit it's inclusion here.

# Chapter 4

# Technical Details

## 4a Software Architecture and Design

### Tools

The team decided to use the ASP.NET Core framework's Razor Pages for building the project for reasons such as:

- The most important argument for using ASP.NET Core is it's cross-platform functionality. The developers in the team use both Macintosh and Windows based systems, therefore a framework that could switch seamlessly between them was crucial.
- As the name suggests, the framework runs in the .NET ecosystem, which is what the team has been taught in the course so far therefore it is what the team is most experienced and most comfortable working in.
- With Razor Pages being a web-page based solution, the UI is mostly composed of HTML and CSS, which some team members already had experience in and the rest was eager to learn. A light-weight, web-based solution allowed the team to be more flexible, and develop the app at a more rapid pace compared to if the team chose a Model-View-Controler (or a Model-View-ViewModel) solution. This freedom allowed for more features and made the process of adjusting to changing the project requirements easier.
- Additionally, Razor Pages allows the project to be published and deployed straight away, which is a nice bonus.

As for other tools, the team used:

- Github: Source and Version Control, Collaborative Development of the App
- Jira: Task Management and Planning as well as adhearing to Agile which was one of the requirements for the project
- Discord: Communication and Resource Sharing
- diagrams.net—draw.io: Creating UML Diagrams
- Figma: Prototyping and General UI Design
- Visual Studio and Visual Studio Code: Code Editors

## Application Flow

The user journey begins in the Homepage, which is the project requirements' Asset Manager component. The user inside of this component has two possible routes:

- Load Heat Demand Data
- Configure Production Units

Let's explore unit configuration first. When the user clicks on that option, they get redirected to the unit configuration page, where they can change the properties of the pre-loaded production units provided by the project requirements (Gas Boiler, Oil Boiler, Gas Motor and Electric Boiler), while also being able to create and configure thier own custom production units. From this page the only option is to go back to the Homepage. Next, loading the heat demand data can be done in one of three ways: choosing the preloaded data provided by Danfoss Deliveries (Winter Period Heat Demand 08.02.2023-14.02.2023 and Summer Period Heat Demand 08.07.2023-14.07.2023), loading the data from a custom CSV file or loading the data from a custom Excel Workbook file. Once heat demand data has been loaded, more options open up for the user in the navbar:

- Database Preview
- Result Data Manager

The Database Preview component was not initially intended to stay in the final product. At first it was used solely for debugging purposes, when the team was implementing the data loading options. But due to supervisor and project presentation feedback, we decided to keep this component to also allow the user to double-check if their data was correctly loaded and recognized by our application. From this page the user has the option to either go back to the Homepage or go to the Result Data Manager.

The Result Data Manager is where the magic happens. It hosts the main functionality of the program which is the Optimizer Component, Data Visualization and saving optimized data. By default, the Optimizer runs automatically whenever the user get redirected to this page with the settings: Use all production units, optimize for costs, use the standard optimizer. We made this choice so that the user is immediately greeted by the results, and not just a blank page. Once on the page, the user can tune the optimizer to produce data that suits their individual needs. Here are the checkbox options given:

- Choose which production units to include in the optimization process
- Choose which parameters to optimize for (Costs, CO2 Emissions)
- Choose which optimizer to use (Standard or Neural Network)

Then the user can rerun the optimization process with the selection options to obtain their results. The results themselves are displayed in the same page, both in text and graph format, which covers the data visualization requirement. Having the optimization outcomes opens up some more paths to take:

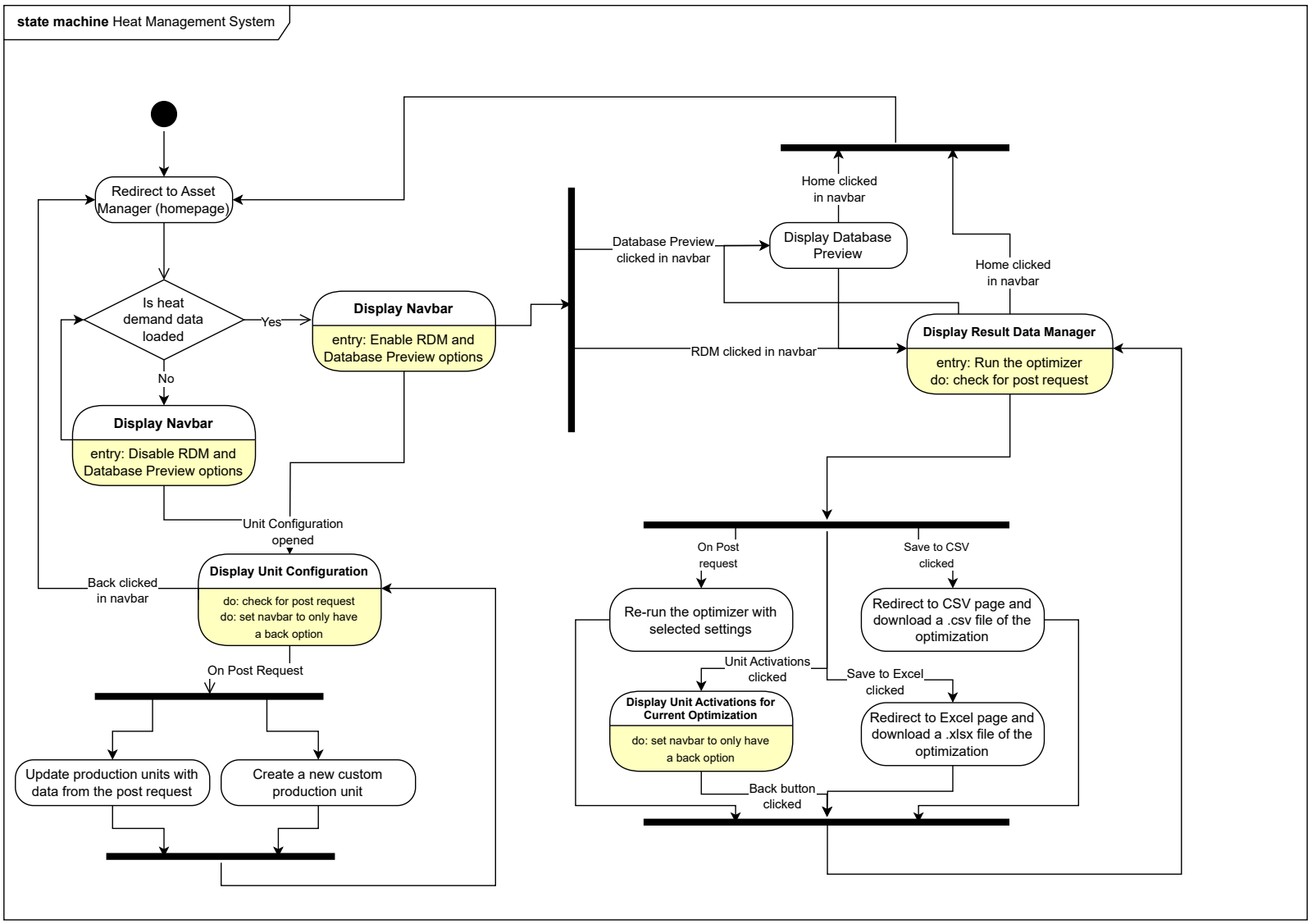
- Unit Activations Page
- Save Optimized Data to CSV
- Save Optimized Data to an Excel Workbook

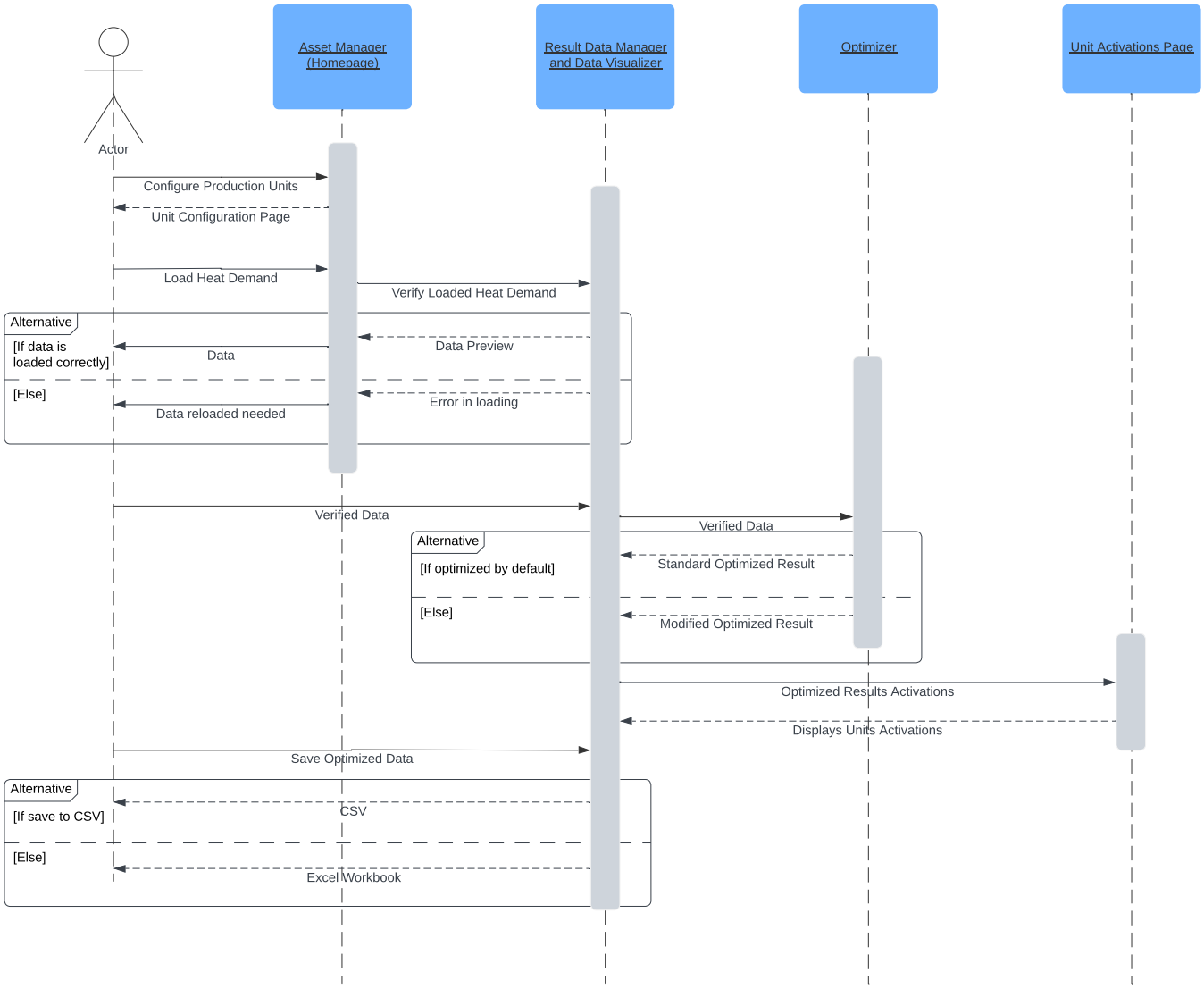
The Unit Activations Page is the last component the user gets access to in their journey. It shows the activation percentages for each production unit at each timeframe. They can be treated as instructions for the user on how to manage the boilers to achieve the optimal Produced Heat, Electricity Consumption, Production Costs, Fuel Consumption and CO2 Emissions proposed by the current optimizer configuration. The last options allow the user to save all of the optimizer output data mentioned previously into a single CSV or Excel Workbook file, which gets downloaded to the users Downloads folder.

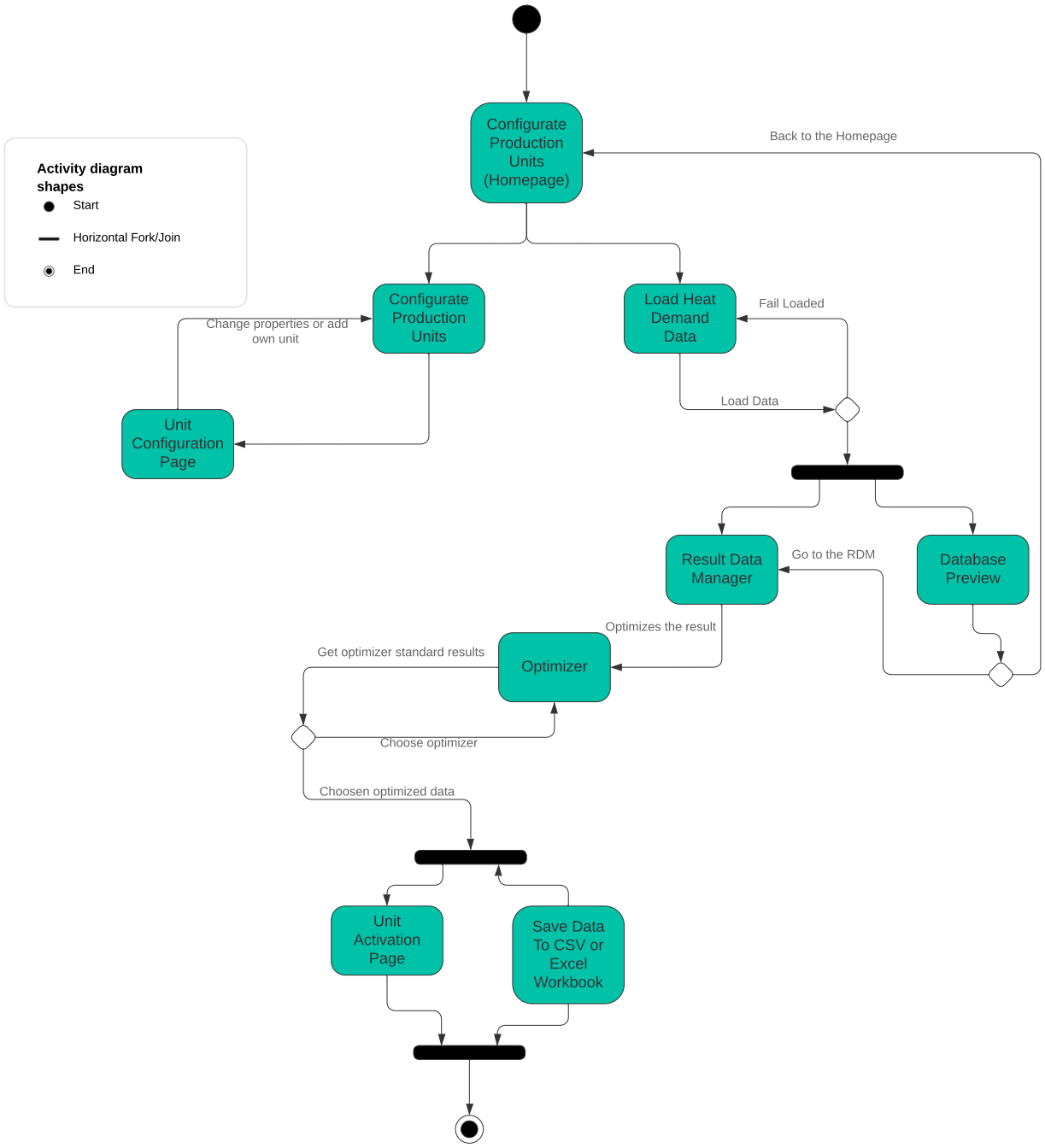
The project in itself does not operate on many states, therefore it does not need nor have a complex state machine diagram. Most of the application's functionality happens automatically and/or instantly, therefore creating custom states and switching between them would have created unnecessary complexity and slowed down the program. With that in mind, during the creation of the state machine diagram, we decided to focus on how Razor Pages functions in general, where the main states are waiting for post requests which is, in other words, user input. Despite that, we still thought that a state machine diagram was one of the best ways to represent the application flow and user journey therefore it can be seen below.

In order to bring more focus to the perspective of the user, we also created a sequence diagram, which contextualizes all of the possible interactions the user can perform. We think that this compliments the rest of the diagrams in this subsection greatly, as it essentially represents the same concept in a different way, giving the reader an easier time understanding our application flow.

Lastly, we also decided to include a sequence diagram here, for the same reasoning as the state machine diagram.

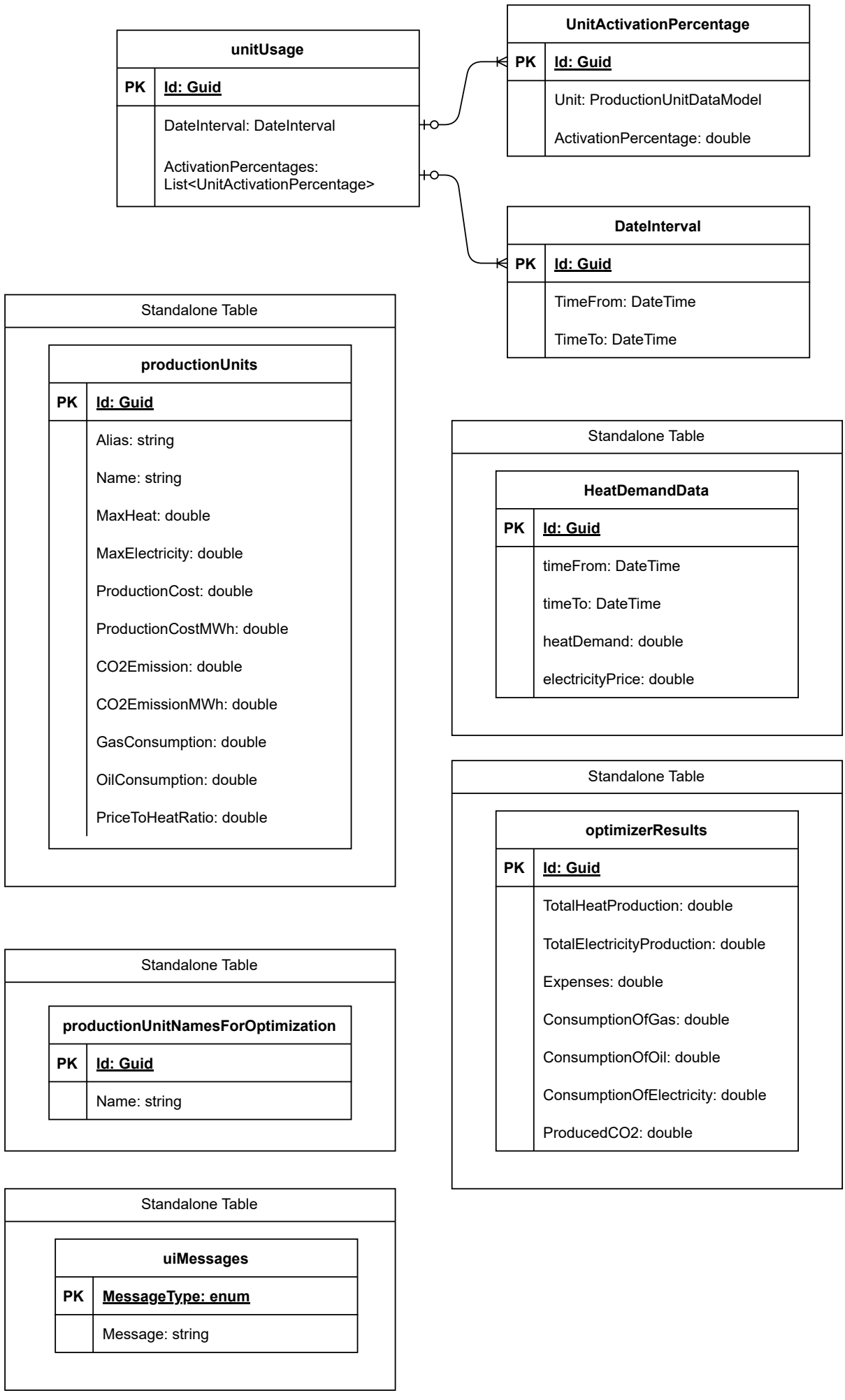

state machine Heat Management System
Redirect to Asset Manager (homepage)
Is heat demand data loaded
Yes
No
Display Navbar
entry: Enable RDM and Database Preview options
Display Navbar
entry: Disable RDM and Database Preview options
Unit Configuration opened
Display Unit Configuration
do: check for post request
do: set navbar to only have a back option
Back clicked in navbar
On Post Request
Update production units with data from the post request
Create a new custom production unit
Home clicked in navbar
Database Preview clicked in navbar
Display Database Preview
Home clicked in navbar
RDM clicked in navbar
Display Result Data Manager
entry: Run the optimizer
do: check for post request
On Post request
Re-run the optimizer with selected settings
Unit Activations clicked
Display Unit Activations for Current Optimization
do: set navbar to only have a back option
Save to CSV clicked
Redirect to CSV page and download a .csv file of the optimization
Save to Excel clicked
Redirect to Excel page and download a .xlsx file of the optimization
Back button clicked


Asset Manager (Homepage)
Result Data Manager and Data Visualizer
Optimizer
Unit Activations Page
Actor
Configure Production Units
Unit Configuration Page
Load Heat Demand
Verify Loaded Heat Demand
Alternative
[If data is loaded correctly]
Data
Data Preview
[Else]
Data reloaded needed
Error in loading
Verified Data
Verified Data
Alternative
[If optimized by default]
Standard Optimized Result
[Else]
Modified Optimized Result
Optimized Results Activations
Displays Units Activations
Save Optimized Data
Alternative
[If save to CSV]
CSV
[Else]
Excel Workbook


Activity diagram shapes
Start
Horizontal Fork/Join
End
Configurate Production Units (Homepage)
Back to the Homepage
Configurate Production Units
Change properties or add own unit
Unit Configuration Page
Load Heat Demand Data
Fail Loaded
Load Data
Result Data Manager
Go to the RDM
Database Preview
Optimizes the result
Optimizer
Get optimizer standard results
Choose optimizer
Choosen optimized data
Unit Activation Page
Save Data To CSV or Excel Workbook

## Database Design

Before diving into the program architecture the in-memory database solution offered by Razor Pages must first be mentioned. In order to achieve data persistance while switching in between pages in a Razor Pages project, a database must be used. Since we did not want to go too far out of the scope of the project, we decided not to use a dedicated database solution like a MySQL or MSSQL Server for this project, expecially because the team has not had any database modeling courses yet. Instead we chose a middle-ground, which is the before mentioned in-memory database. This soultion offers similar functionality to a real database, with the comfort of running in the program's memory, which eliminates potential connection, authentication privilege and/or security risks and issues connected with using databases. To represent the applications data storage sytstem the group devised an improvised custom uml diagram inspired by other standardized database diagrams. It can be seen below, together with explanation for each of the tables.

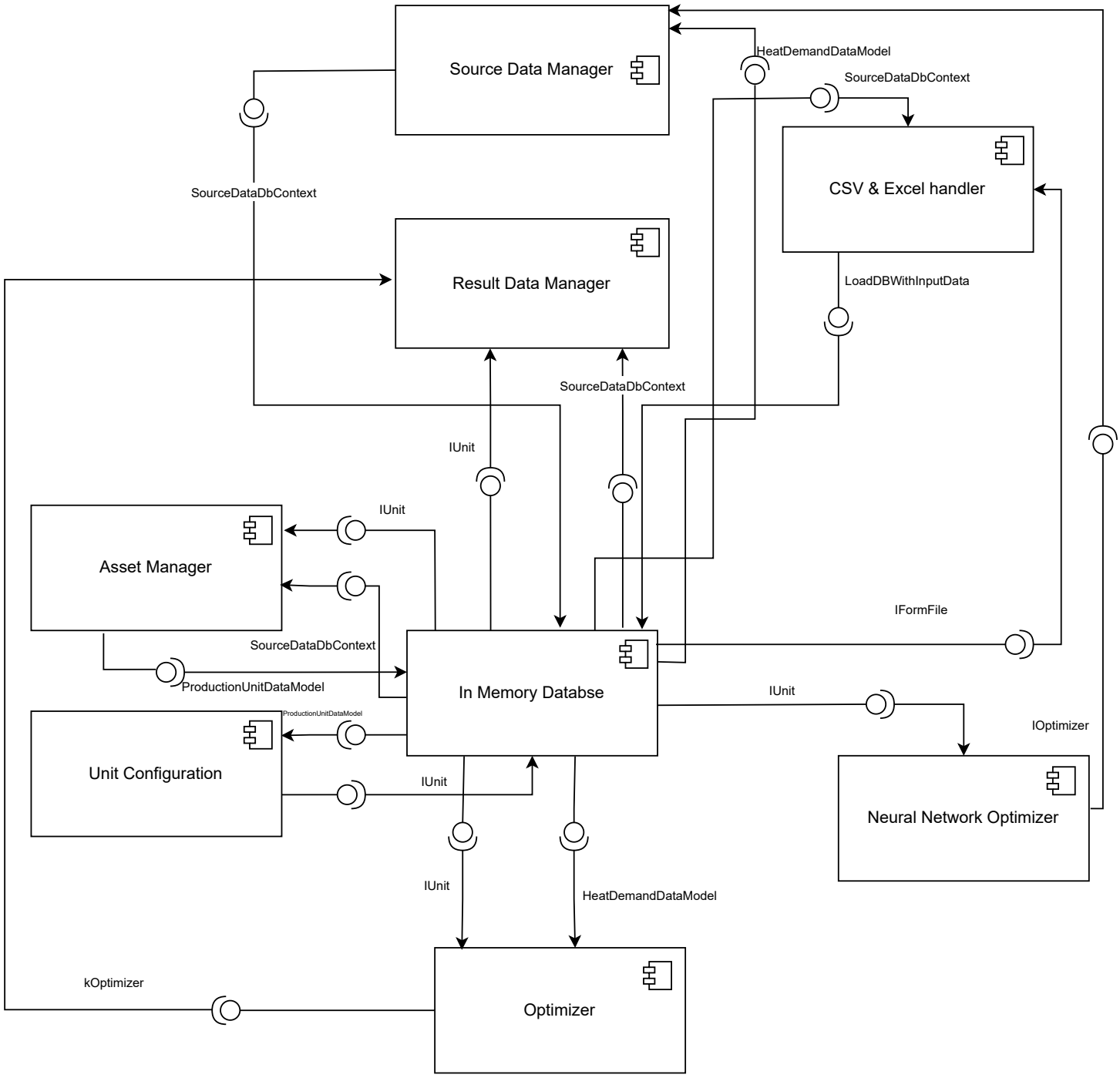

unitUsage
PK
Id: Guid
DateInterval: DateInterval
ActivationPercentages: List<UnitActivationPercentage>
UnitActivationPercentage
PK
Id: Guid
Unit: ProductionUnitDataModel
ActivationPercentage: double
DateInterval
PK
Id: Guid
TimeFrom: DateTime
TimeTo: DateTime
Standalone Table
productionUnits
PK
Id: Guid
Alias: string
Name: string
MaxHeat: double
MaxElectricity: double
ProductionCost: double
ProductionCostMWh: double
CO2Emission: double
CO2EmissionMWh: double
GasConsumption: double
OilConsumption: double
PriceToHeatRatio: double
Standalone Table
HeatDemandData
PK
Id: Guid
timeFrom: DateTime
timeTo: DateTime
heatDemand: double
electricityPrice: double
Standalone Table
optimizerResults
PK
Id: Guid
TotalHeatProduction: double
TotalElectricityProduction: double
Expenses: double
ConsumptionOfGas: double
ConsumptionOfOil: double
ConsumptionOfElectricity: double
ProducedCO2: double
Standalone Table
productionUnitNamesForOptimization
PK
Id: Guid
Name: string
Standalone Table
uiMessages
PK
MessageType: enum
Message: string

The database does not fully comply to database system standards. We do see the limitations and flaws of this design choice, such as the lack of connections between tables and the lack of foreign keys inside of each table. Instead some fields in the tables use custom class data types for ease of use. Despite the flaws of using DbSets, we found that their functionality was sufficient for the scope of the project. It is also worth to mention, that due to our lack of experience with Razor Pages, we were unsure of how DbSets and in-memory data structures function. Because of that in the middle of development the database had to be refactored from a more json-like data storage structure to one that complies with standards enforced by DbSets, such as using Primary Keys. It is because of this process that we ended up having a mixture of both structures. The most crucial tables found within the programs database are the *productionUnits* and *HeatDemandData*. As the names suggests they contain data that is needed for the application's main functionality - calculating optimized production unit usage results for the inputted heat demand for each timeframe. Some of the properties inside of *productionUnits* are additions from the team, such as *ProductionCostMWh* and *CO2EmissionMWh*, which made it easier to remember how to calculate final optimization results, and *PriceToHeatRatio*, which also was a helper variable in optimization calculations. When it comes to storing the results from the optimizer for later display or download, the *unitUsage* and *optimizerResults* tables were used. We think their names explain their functionality sufficiently. Lastly, the *uiMessages*, as the name suggests stores messages for the user that may appear in the user interface under certain conditions, such as confirmation of a successful data upload or an error during the optimization process. These messages have to be stored in a database, because of how user input works in Razor Pages. The only way the team found to obtain user input is through post requests, which only make impact on the interface after a page redirect, which would reset the pages memory and lose the message intended for display. That is why for these messages to persist it is important to have a place to save them before the redirect.

## Code Architecture

With the database covered, we can shift our focus towards the rest of the codebase. Any issues, bugs or problems encountered by the group during the development/implementation phase of the project will be underlined in this subsection of the report. We shall start in the same place as the program's structure: *Program.cs*. This file mostly contains automatically generated Razor Pages setup code for every built-in feature to work correctly, but it is also where one of the first problems started. Culture Info was a common problem across all groups working on

this project in this semester, and it definitely caused some issues for us as well, like incorrect DateTime variable parsing, incorrect decimal separator detection, errors with reading data from files and incorrect data display. Fortunately manually setting the user's DefaultThreadCurrentCulture and DefaultThreadCurrentUICulture to the Danish CultureInfo was an easy fix that solved most of these problems. From this code file, the Razor Pages applications gets built and the user gets redirected to the *Index.cshtml* page with the Navbar and DataUpload ViewComponents overlayed on top of it, which we will come back to shortly. The group chose to create both component and class diagrams to explain how the code is designed. The component diagram helps us break down the program into the components from the project's requirements, and clarify in which location each component is implemented. This diagram also plays a part in explaining the interactions between each component, and the class diagram is able to complete this explanation, as well as go more in depth for each component, breaking them down into individual classes. The component diagram can be seen below.

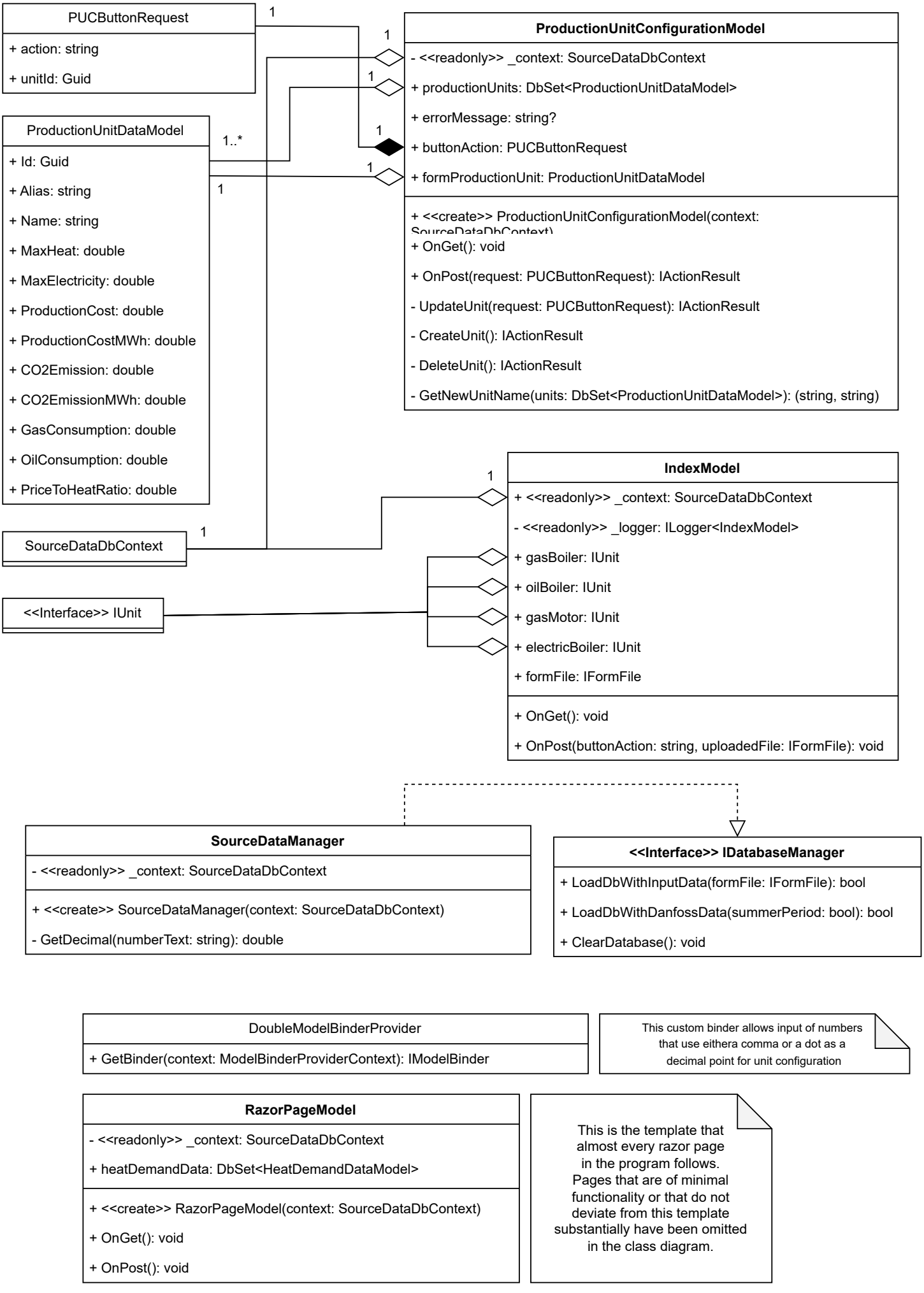

Source Data Manager
HeatDemandDataModel
SourceDataDbContext
CSV & Excel handler
SourceDataDbContext
Result Data Manager
LoadDBWithInputData
SourceDataDbContext
IUnit
IUnit
Asset Manager
IFormFile
SourceDataDbContext
In Memory Databse
ProductionUnitDataModel
IUnit
IOptimizer
ProductionUnitDataModel
Unit Configuration
IUnit
Neural Network Optimizer
IUnit
HeatDemandDataModel
kOptimizer
Optimizer

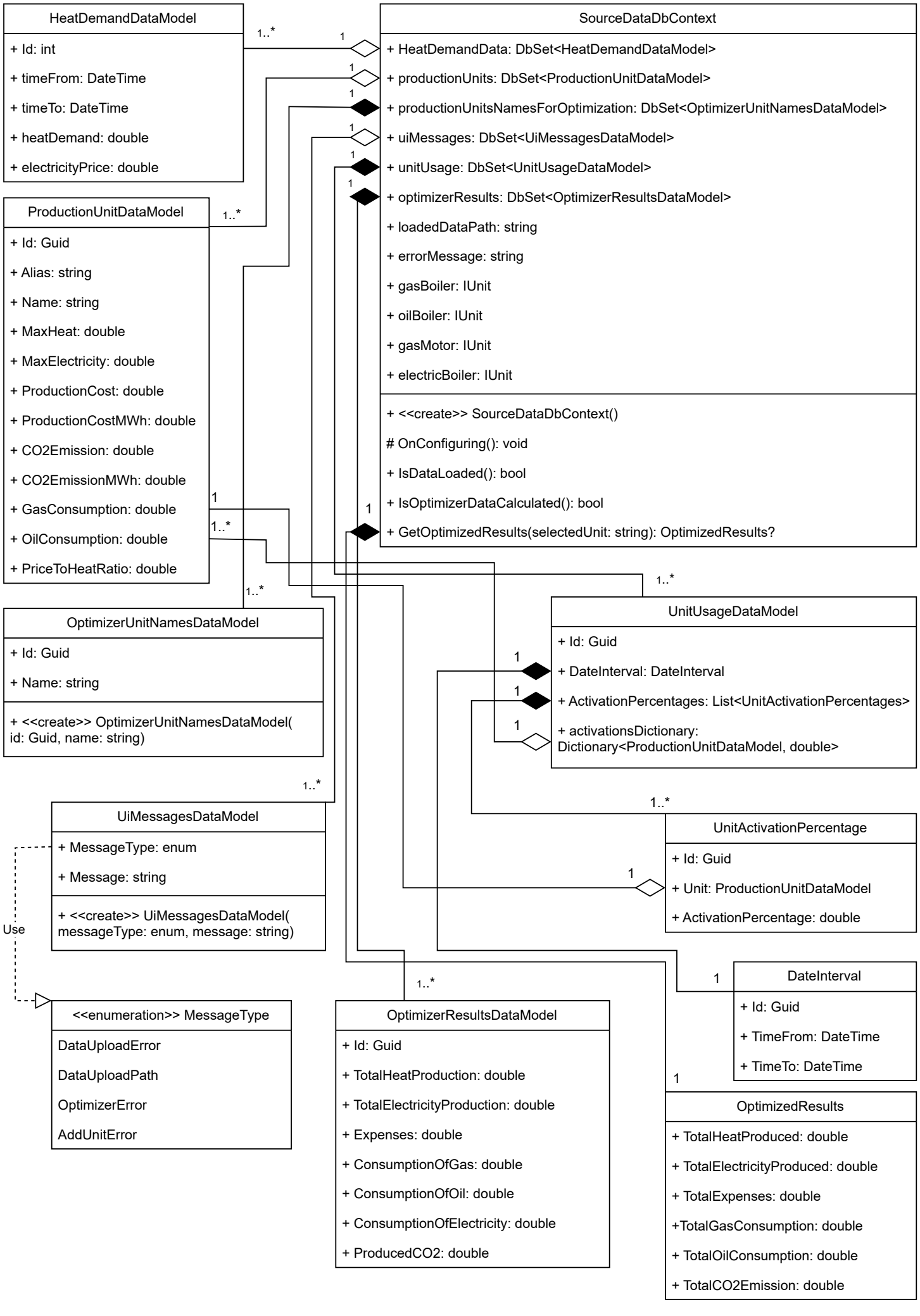
Following the component hierarchy we will cover the **Source Data Manager** next. The class diagrams facilitating this explanation can be seen below this paragraph. Our implementation of this component makes it handle the most important interaction between the program and the database, which is loading user input or preloaded data and deleting that data. The component itself is an implementation of the *IDatabaseManager* interface, which was used to reinforce the SOLID principles within our codebase, decouple dependencies, make the code less brittle and more expandable. The main property of **Source Data Manager** is *_context*, which will appear in most other components, and is essentially a reference to the in-memory database. *SourceDataDbContext*, which implements the *DbContext* Interface is the class that contains the in-memory database. It uses class data models such as *HeatDemandDataModel*, *ProductionUnitDataModel*, *OptimizerUnitNamesDataModel*, *UnitUsageDataModel*, *UiMessagesDataModel* and *OptimizerResultsDataModel* for data storage. The other properties it contains have custom get and set methods defined which interact with their respective DbSets, and serve as quick references to specific items from the database that are widely used throughout the program. This is a result of major database refactor, before which these properties were preset in the **Source Data Manager** component. In order to avoid merge conflicts and errors these properties were kept but puerly as references to the database, which is where C#'s feature of custom property getters and setters came in really handy. Moving onto the **Asset Manager** component, which uses the *IndexModel.cshtml.cs* class as it's codebehind. As mentioned previously, this is the Homepage of the program, it contains a reference to the database, OnGet and OnPost methods for handling user input. In this component, we also made use of Razor Pages' View Component feature, which essentially allowed us to display a page within another page, which helped break up the code into more manageable chunks and work on different parts of the page without interfering with each other. The only two View Components in the program are:

- DataUploadView - a page that either calls **Source Data Manager** to load in pre-provided heat demand data, or accepts a file uploaded by the user, while checking if its format and extension is correct.
- PageNavigationView - as the name suggests, it is an overlay that allows the user to navigate through the program, namely to three components: **Asset Manager**, **Result Data Manager** and **Database Preview**. The navigation bar is displayed in each mentioned component.

The **Asset Manager** also gives access to the **Production Unit Configuration** component. This is where the user can configure properties of pre-existing or cus-

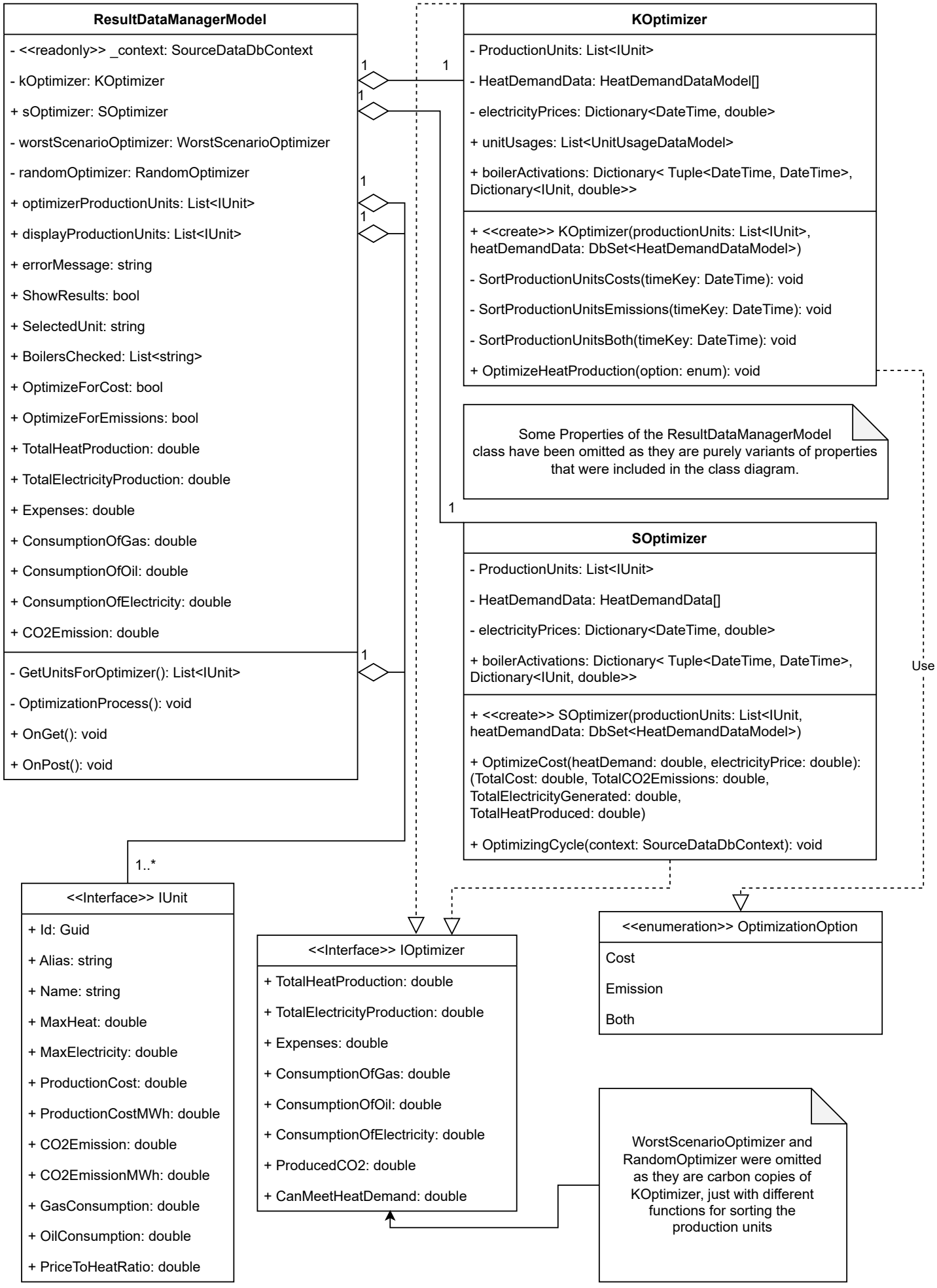
tom added production units. All of the user input is again handled in the OnGet and OnPost methods with the aid of the *PUCButtonRequest* class, which allows the OnPost request to recognize which button in the UI was pressed and to react accordingly. Unit creation, updating and deletion processes use unit IDs from the *ProductionUnitDataModel* class to make sure changes occur to the right unit, thanks to Guid ensuring each Production Unit record in the database has a unique ID. Obtaining the new parameters for chaning a picked production unit is handled thorugh binding the *formProductionUnit* property, which is a technique used also in other pages when handling user input. This is also where an issue came up connected with Culture Info, that was not solved in *Program.cs*, which were the decimal separators for number inputs. This is why a custom *DoubleModelBinder* was written, to convert HTML form input's values to variables of type *double* no matter if they used a dot or a comma as their decimal separator. A safety check was also implemented, so that a production unit's Alias property has to be unique, to eliminate potential errors with the optimization process, which will be covered soon.

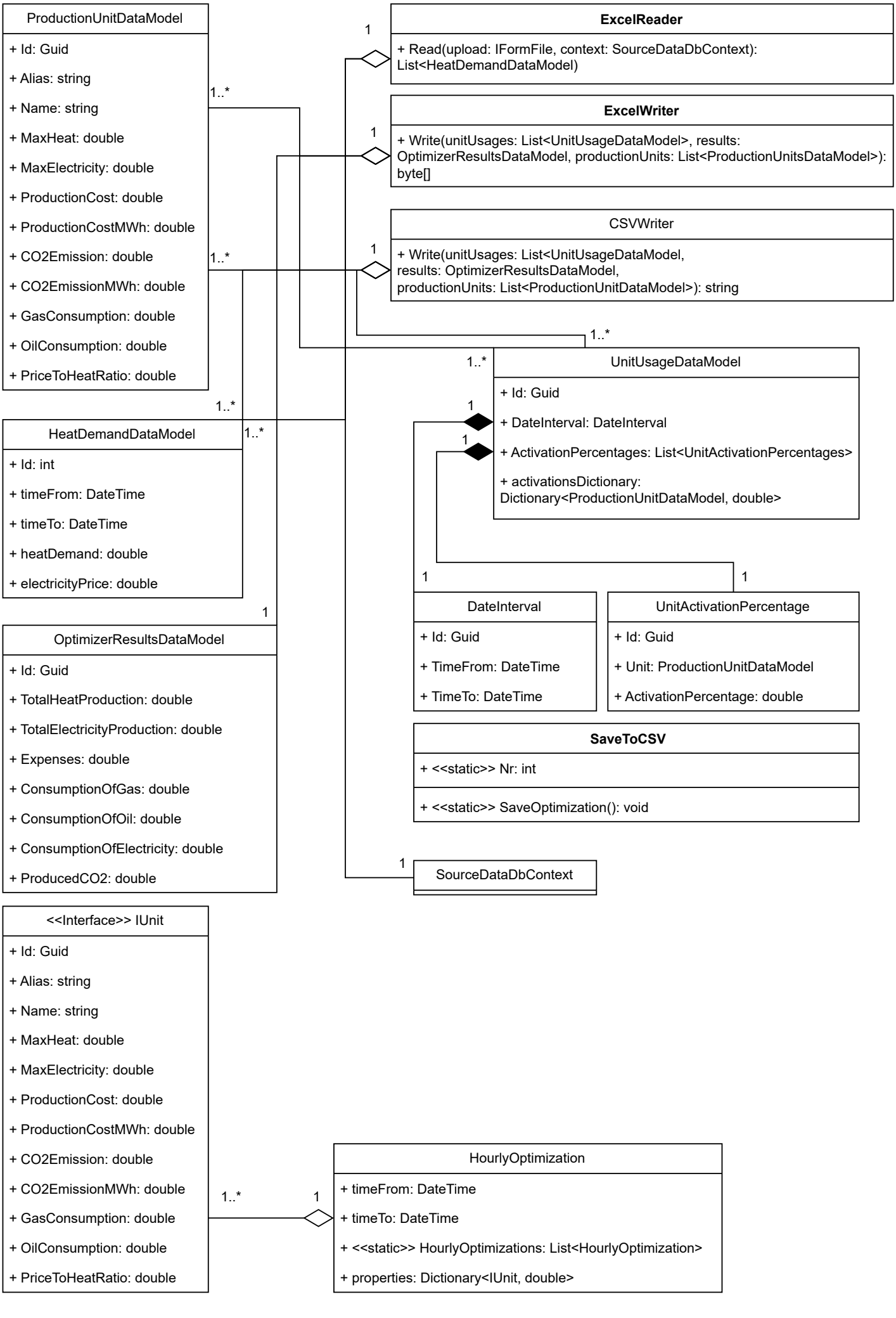

PUCButtonRequest
+ action: string
+ unitId: Guid
ProductionUnitDataModel
+ Id: Guid
+ Alias: string
+ Name: string
+ MaxHeat: double
+ MaxElectricity: double
+ ProductionCost: double
+ ProductionCostMWh: double
+ CO2Emission: double
+ CO2EmissionMWh: double
+ GasConsumption: double
+ OilConsumption: double
+ PriceToHeatRatio: double
ProductionUnitConfigurationModel
- <<readonly>> _context: SourceDataDbContext
+ productionUnits: DbSet<ProductionUnitDataModel>
+ errorMessage: string?
+ buttonAction: PUCButtonRequest
+ formProductionUnit: ProductionUnitDataModel
+ <<create>> ProductionUnitConfigurationModel(context:
+ OnGet(): void
+ OnPost(request: PUCButtonRequest): IActionResult
- UpdateUnit(request: PUCButtonRequest): IActionResult
- CreateUnit(): IActionResult
- DeleteUnit(): IActionResult
- GetNewUnitName(units: DbSet<ProductionUnitDataModel>): (string, string)
IndexModel
+ <<readonly>> _context: SourceDataDbContext
- <<readonly>> _logger: ILogger<IndexModel>
+ gasBoiler: IUnit
+ oilBoiler: IUnit
+ gasMotor: IUnit
+ electricBoiler: IUnit
+ formFile: IFormFile
+ OnGet(): void
+ OnPost(buttonAction: string, uploadedFile: IFormFile): void
SourceDataDbContext
<<Interface>> IUnit
SourceDataManager
- <<readonly>> _context: SourceDataDbContext
+ <<create>> SourceDataManager(context: SourceDataDbContext)
- GetDecimal(numberText: string): double
<<Interface>> IDatabaseManager
+ LoadDbWithInputData(formFile: IFormFile): bool
+ LoadDbWithDanfossData(summerPeriod: bool): bool
+ ClearDatabase(): void
DoubleModelBinderProvider
+ GetBinder(context: ModelBinderProviderContext): IModelBinder
This custom binder allows input of numbers that use eithera comma or a dot as a decimal point for unit configuration
RazorPageModel
- <<readonly>> _context: SourceDataDbContext
+ heatDemandData: DbSet<HeatDemandDataModel>
+ <<create>> RazorPageModel(context: SourceDataDbContext)
+ OnGet(): void
+ OnPost(): void
This is the template that almost every razor page in the program follows. Pages that are of minimal functionality or that do not deviate from this template substantially have been omitted in the class diagram.


HeatDemandDataModel
+ Id: int
+ timeFrom: DateTime
+ timeTo: DateTime
+ heatDemand: double
+ electricityPrice: double
SourceDataDbContext
+ HeatDemandData: DbSet<HeatDemandDataModel>
+ productionUnits: DbSet<ProductionUnitDataModel>
+ productionUnitsNamesForOptimization: DbSet<OptimizerUnitNamesDataModel>
+ uiMessages: DbSet<UiMessagesDataModel>
+ unitUsage: DbSet<UnitUsageDataModel>
+ optimizerResults: DbSet<OptimizerResultsDataModel>
+ loadedDataPath: string
+ errorMessage: string
+ gasBoiler: IUnit
+ oilBoiler: IUnit
+ gasMotor: IUnit
+ electricBoiler: IUnit
+ <<create>> SourceDataDbContext()
# OnConfiguring(): void
+ IsDataLoaded(): bool
+ IsOptimizerDataCalculated(): bool
+ GetOptimizedResults(selectedUnit: string): OptimizedResults?
ProductionUnitDataModel
+ Id: Guid
+ Alias: string
+ Name: string
+ MaxHeat: double
+ MaxElectricity: double
+ ProductionCost: double
+ ProductionCostMWh: double
+ CO2Emission: double
+ CO2EmissionMWh: double
+ GasConsumption: double
+ OilConsumption: double
+ PriceToHeatRatio: double
OptimizerUnitNamesDataModel
+ Id: Guid
+ Name: string
+ <<create>> OptimizerUnitNamesDataModel(
id: Guid, name: string)
UnitUsageDataModel
+ Id: Guid
+ DateInterval: DateInterval
+ ActivationPercentages: List<UnitActivationPercentages>
+ activationsDictionary:
Dictionary<ProductionUnitDataModel, double>
UiMessagesDataModel
+ MessageType: enum
+ Message: string
+ <<create>> UiMessagesDataModel(
messageType: enum, message: string)
UnitActivationPercentage
+ Id: Guid
+ Unit: ProductionUnitDataModel
+ ActivationPercentage: double
Use
<<enumeration>> MessageType
DataUploadError
DataUploadPath
OptimizerError
AddUnitError
OptimizerResultsDataModel
+ Id: Guid
+ TotalHeatProduction: double
+ TotalElectricityProduction: double
+ Expenses: double
+ ConsumptionOfGas: double
+ ConsumptionOfOil: double
+ ConsumptionOfElectricity: double
+ ProducedCO2: double
DateInterval
+ Id: Guid
+ TimeFrom: DateTime
+ TimeTo: DateTime
OptimizedResults
+ TotalHeatProduced: double
+ TotalElectricityProduced: double
+ TotalExpenses: double
+TotalGasConsumption: double
+ TotalOilConsumption: double
+ TotalCO2Emission: double

Back in **Asset Manager**, the navbar allows the user to navigate to the **Result Data Manager** and **Database Preview** components. The latter does not contain any distinguishing code, and was covered entirely in the subsection Application Flow. The **Result Data Manager**, on the other hand, is the most substancial component of the whole application. The code-behind for its page is in *ResultDataManagerModel.cshtml.cs*. It contains the usual reference to the database, but a thing that stands out are the two IUnit Lists and the IUnit Interface itself. Firstly, the program contains both a IUnit Interface and a ProductionUnitDataModel parent class for production units, which are used interchangeably, because of a DbSet issue. At one point in the development when trying to access one of the feature of DbSets, it turned out that a DbSet cannot be of an interface type. At that point, the IUnit interface was used extensively within the codebase and there was not enough time to make the changes necessary, so a band-aid fix was applied, with which the ProductionUnitDataModel class was created. The **Result Data Manager** also hosts the **Optimizer** component, which consists of the KOptimizer, SOptimizer, GeneticAlgorithm (The Neural Network Optimizer) (and WorstScenarioOptimizer with RandomOptimizer, which are used solely as comparison data for some of the graphs). Multiple Optimizer implementations are a result of one of the Sprints the team carried out, where everyone worked on an implementation and at the end the best one was chosen. The main optimizer that the application uses is the KOptimizer, which is also referred to as the Standard Optimizer. It is a simple Greedy Algorithm implementation that judges which production units are the most cost effective for each timeframe, and uses them to produce as much heat as possible until the demand is met. An outlier to the goal of our optimizer sprint was the Neural Network Optimizer, which was too good of an idea to pass up on, while begin to risky to be the only optimization algorithm, so we decided to use both. The **Neural Network Optimizer** is a simple implementation of a Fitness Score Neural Network algorithm. It first creates a population of Individuals, where each Individual is randomly mutated in terms of the production unit activation percentage in order to explore new solutions. Then a Fitness Score is calculated for each Individual based on how close it is to achieving the heat demand whilst keeping the costs low. Once the Individual with the highest Fitness Score is found, the process begins again, this time every Individual in the new population is a mutation of the best Fitness Score individual from the previous generation, and this is repeated a large amount of times until a satisfying result is achieved. Unfortunately there was not enough time for the group to implement optimizer settings for the **Neural Network Optimizer** component, so it is only able to optimize for costs and use the default production units but we think that even at it's most basic it adds value to the application. The Opti-

mization Process has 3 main configuration options to consider before re-running it, which were also mentioned in Application Flow. This configuration is handled in the OnPost method of ResultDataManagerModel. It is important to mention, that due to allowing the user to choose their own production unit scenario, each **Optimizer** implementation had to have a built-in check to make sure that the production units provided can even meet the heat demand. If that is not the case, the uiMessages DbSet is used in the database to display an error message to the user. Additionally, this option meant that we needed to store all of the production units and the production units selected for the optimization process by the user at the same time. This is where a second problem with DbSets came up. Razor Pages does not seem to allow for two DbSets of the same type to exist within one database, instead merging them into one singular DbSet if that was the case, resulting in duplicate key value errors. A workaround had to be implemented. This is why the OptimizerUnitNamesDataModel exists, which stores just the production unit aliases, which are then matched with the prodcution units in the productionUnits DbSet and passed into the **Optimizer**. This is also why the **Unit Configuration** component makes sure that each alias is unique. Once the optimization process is finished, the **Result Data Manager** saves the outputs to the database using the UnitUsageDataModel and OptimizerResultsDataModel classes. This is what later allows the ExcelWriter and CSVWriter classes to access this data, convert it to their own respective formats, and redirect the user to a file download page. Lastly, **Result Data Manager** also hosts the **Unit Activations** component, which in similar fashion as the **Database Preview**, was covered in Application Flow.


ResultDataManagerModel
- <<readonly>> _context: SourceDataDbContext
- kOptimizer: KOptimizer
+ sOptimizer: SOptimizer
- worstScenarioOptimizer: WorstScenarioOptimizer
- randomOptimizer: RandomOptimizer
+ optimizerProductionUnits: List<IUnit>
+ displayProductionUnits: List<IUnit>
+ errorMessage: string
+ ShowResults: bool
+ SelectedUnit: string
+ BoilersChecked: List<string>
+ OptimizeForCost: bool
+ OptimizeForEmissions: bool
+ TotalHeatProduction: double
+ TotalElectricityProduction: double
+ Expenses: double
+ ConsumptionOfGas: double
+ ConsumptionOfOil: double
+ ConsumptionOfElectricity: double
+ CO2Emission: double
- GetUnitsForOptimizer(): List<IUnit>
- OptimizationProcess(): void
+ OnGet(): void
+ OnPost(): void
KOptimizer
- ProductionUnits: List<IUnit>
- HeatDemandData: HeatDemandDataModel[]
- electricityPrices: Dictionary<DateTime, double>
+ unitUsages: List<UnitUsageDataModel>
+ boilerActivations: Dictionary< Tuple<DateTime, DateTime>, Dictionary<IUnit, double>>
+ <<create>> KOptimizer(productionUnits: List<IUnit>, heatDemandData: DbSet<HeatDemandDataModel>)
- SortProductionUnitsCosts(timeKey: DateTime): void
- SortProductionUnitsEmissions(timeKey: DateTime): void
- SortProductionUnitsBoth(timeKey: DateTime): void
+ OptimizeHeatProduction(option: enum): void
Some Properties of the ResultDataManagerModel class have been omitted as they are purely variants of properties that were included in the class diagram.
SOptimizer
- ProductionUnits: List<IUnit>
- HeatDemandData: HeatDemandData[]
- electricityPrices: Dictionary<DateTime, double>
+ boilerActivations: Dictionary< Tuple<DateTime, DateTime>, Dictionary<IUnit, double>>
+ <<create>> SOptimizer(productionUnits: List<IUnit, heatDemandData: DbSet<HeatDemandDataModel>)
+ OptimizeCost(heatDemand: double, electricityPrice: double): (TotalCost: double, TotalCO2Emissions: double, TotalElectricityGenerated: double, TotalHeatProduced: double)
+ OptimizingCycle(context: SourceDataDbContext): void
Use
1
1..*
<<Interface>> IUnit
+ Id: Guid
+ Alias: string
+ Name: string
+ MaxHeat: double
+ MaxElectricity: double
+ ProductionCost: double
+ ProductionCostMWh: double
+ CO2Emission: double
+ CO2EmissionMWh: double
+ GasConsumption: double
+ OilConsumption: double
+ PriceToHeatRatio: double
<<Interface>> IOptimizer
+ TotalHeatProduction: double
+ TotalElectricityProduction: double
+ Expenses: double
+ ConsumptionOfGas: double
+ ConsumptionOfOil: double
+ ConsumptionOfElectricity: double
+ ProducedCO2: double
+ CanMeetHeatDemand: double
<<enumeration>> OptimizationOption
Cost
Emission
Both
WorstScenarioOptimizer and RandomOptimizer were omitted as they are carbon copies of KOptimizer, just with different functions for sorting the production units


ProductionUnitDataModel
+ Id: Guid
+ Alias: string
+ Name: string
+ MaxHeat: double
+ MaxElectricity: double
+ ProductionCost: double
+ ProductionCostMWh: double
+ CO2Emission: double
+ CO2EmissionMWh: double
+ GasConsumption: double
+ OilConsumption: double
+ PriceToHeatRatio: double
ExcelReader
+ Read(upload: IFormFile, context: SourceDataDbContext): List<HeatDemandDataModel)
ExcelWriter
+ Write(unitUsages: List<UnitUsageDataModel>, results: OptimizerResultsDataModel, productionUnits: List<ProductionUnitsDataModel>): byte[]
CSVWriter
+ Write(unitUsages: List<UnitUsageDataModel, results: OptimizerResultsDataModel, productionUnits: List<ProductionUnitDataModel>): string
UnitUsageDataModel
+ Id: Guid
+ DateInterval: DateInterval
+ ActivationPercentages: List<UnitActivationPercentages>
+ activationsDictionary: Dictionary<ProductionUnitDataModel, double>
HeatDemandDataModel
+ Id: int
+ timeFrom: DateTime
+ timeTo: DateTime
+ heatDemand: double
+ electricityPrice: double
OptimizerResultsDataModel
+ Id: Guid
+ TotalHeatProduction: double
+ TotalElectricityProduction: double
+ Expenses: double
+ ConsumptionOfGas: double
+ ConsumptionOfOil: double
+ ConsumptionOfElectricity: double
+ ProducedCO2: double
DateInterval
+ Id: Guid
+ TimeFrom: DateTime
+ TimeTo: DateTime
UnitActivationPercentage
+ Id: Guid
+ Unit: ProductionUnitDataModel
+ ActivationPercentage: double
SaveToCSV
+ <<static>> Nr: int
+ <<static>> SaveOptimization(): void
SourceDataDbContext
<<Interface>> IUnit
+ Id: Guid
+ Alias: string
+ Name: string
+ MaxHeat: double
+ MaxElectricity: double
+ ProductionCost: double
+ ProductionCostMWh: double
+ CO2Emission: double
+ CO2EmissionMWh: double
+ GasConsumption: double
+ OilConsumption: double
+ PriceToHeatRatio: double
HourlyOptimization
+ timeFrom: DateTime
+ timeTo: DateTime
+ <<static>> HourlyOptimizations: List<HourlyOptimization>
+ properties: Dictionary<IUnit, double>
1
1..*

## 4b Simple Design

In software engineering, simple design is a fundamental element to creating a software. The aim of this principle is to make the project as uncomplicated as possible while achieving all the goals and requirements. In the we used agile methodologies which is suitable for this concept. The key aspects of simple design include functionality, clarity, minimalism, flexibility and maintainability. The functionality part was achieved by following the requirements in the first and second iterations. There were different points to follow in each section. The first iteration required a simpler user interface and optimizer with 2 boilers. Later on, in the second iteration we used the same design and added two extra boilers for the optimizer. This method kept the code clear and maintainable. This also means that the we created a flexible code as it could accommodate future changes and extensions. The design had to be written in a self-explanatory way as we were 5 developers working on a project and had to be able to add extra features to the code by ourselves. We made the code easy to understand using meaningful names for variables, functions and classes. We broke down complex functions into smaller, single purpose functions. Moreover, we utilised the 4 principles of object-oriented programming: encapsulation, abstraction, inheritance and polymorphism. While our code includes several elements of inheritance, they are not overcomplicating the code and used in simple understandable ways. We used the DRY (Don't Repeat Yourself) principle to avoid duplicating the code by abstracting common functionality into reusable components.

## 4c Incremental Design

Incremental design is a development approach where a system is designed and tested in small manageable parts, rather than building the entire software system at once. In order to achieve that in each sprint we set out development goals that had to be implemented in the code. The sprints were based on the iterations as those tasks were still too complex to be tested at once. This method also helped us to have an early and continuous feedback which align our agile methodology. Debugging was made easier as we could go over the smaller, newly added parts of the code and see what problems we had to solve from the extended code. Overall, incremental design has helped us to manage complexity and to reduce risk and made sure that the final product is well-aligned with the requirements.

## 4d Refactoring

Most of the group did not have much experience in this process because most of the assignments we were being given were not on a large enough scale to consider restructuring code throughout their development - the tasks and functionality were finished long before poor code design could become a problem. The scope and timespan of this project was much larger, therefore the we decided that refactoring should play a considerable role in our development. Somewhat to our surprise, refactoring felt quite natural, especially since this was the team's first project with Razor Pages, the more we learned about the framework and the more we understood, the more we saw of what could be implemented better, improved or discarded. Some features of the application even required code refactors, because the foundation of the program could not have facilitated them beforehand due to our initial poor understanding of the framework. It is fair to say that a considerable amount of development time was dedicated solely to code refactoring and it very much paid off in the long run. We found that thanks to continuous code maintenance and refactors, the source code became simpler, cleaner, easier to understand and more expandable. Frequent refactoring also strengthened the code's structure, making it less brittle as well as eliminating a sizeable amount of bugs and vulnerabilities the team was struggling with, which could be called a happy accident. A great example of refactoring done by the group are the database refactors described in Datbase Design and Code Architecture. Storing variables the traditional, which is what we have been doing for all our previous assignments just did not work with Razor Pages, thus we needed to shift more to a database mindset. The first 'code smell' that suggested it was a bug with displaying conditional ui messages for the user, they would show up once but after a refresh disappear, simply because they were not stored in a database. Another instance of refactoring in development was handling the output data from the **Optimizer** component. At first every component that wanted to access it had to get the appropriate properties from the object instance itself. This created and issue when it came time to implement downloading this data to the user's device. Since the user needed to be redirected to a different page, we could not pass over the object reference, so a change was made to store the results in the database. With that also came another refactor to the structure of the **Optimizer's** output data, which was simplified and made compatible with storing in the DbSet data structure. This was a prime example of how refactoring made it easier to work with the code and simplified the underlying logic of a component.

Code refactoring was tremendously benefitial in the implementation phase of our application, and we will most definitely keep this practice in the future, maybe even

putting more focus on it.

# 4e Test-Driven Development

## Implementation of Test-Driven Development

In our project, we decided not to implement Test-Driven Development. We developed our project differently, we decided to write first the principal parts of the code and then we did different code reviews to control if all the code was accurate and reliable. We decided this because of the preference of the team in general, and this allowed us to utilize the team's strengths and successfully complete the project.

## Reflection on Test-Driven Development

Test-Driven Development is in general writing tests before the code, to be sure that the code complies with some requirements, we did not implement TDD, but we acknowledge the benefits it could have brought to the project, like more code quality, early bug detection, and refactoring in the code.

# 4f Unit Testing

## Implementation of Unit Testing

In this project, we did not implement unit testing for the different parts of the program. We developed the project thinking about writing first the fundamental parts of the project and then we analyzed the code with manual testing to check the correctness of the code. We decided this because we prioritized fast development and a direct validation of the program.

## Reflection on Unit Testing

The advantages of implementing unit testing in a program are relevant, like ensuring code correctness, easy bug detection, and help with code refactoring, we decided not to incorporate this feature into our project. We acknowledge the benefits of unit testing but decided to use another approach that suits our case better helping us take advantage of our skills and complete the project within the deadline.

# 4g Pair Programming

## Implementation of Pair Programming

In our project, we sometimes used pair programming normally when two team members had to work on the same part of the code. This gave us the opportunity to develop the code more quickly and reduce the number of errors. However, we generally did not use pair programming in a big part of the project. Instead, we divided the part of the program into groups each member had their part to work on. When a team member has a problem with his part, he could fix the error with help from another team member. This method helps with deadlines and each member's strengths.

## Reflection on Pair Programming

Pair programming is basically two developers working together on the same code, one developer who writes the code, and the other who reviews the code in real-time. We did not use Pair Programming that much in this project, but the benefits of Pair programming could have brought like better code quality, real-time error detection, and more efficient knowledge sharing, would have been good.

# 4h Code Review

## Implementation of Code Review

In the project, we have done multiple times code reviews, in the middle of the development process to help the team members develop its part correctly or more aligned with the project in general. Moreover, we have done a code review at the end of the code development to see if everything is well written or can be written in other ways. In general, this examination process has been used a lot throughout the project, to generate the best code and the easiest one to merge.

## Reflection on Code Review

Code review is one of the most important development processes, this process has priceless perks that can not be overlooked, providing great help with bugs, by finding bugs in an early stage of the development and reducing the probability of errors in the final version of the project. Ensuring code quality, and making the written code meet

the established standards and best practices set by the team. Knowledge sharing, the code review process facilitates sharing knowledge with the other team members. Consistency through the code makes the codebase more understandable and easier to organize.

# Chapter 5

# Conclusion and Group's Reflections

The completion of the Heat Production Management Project was far from a simple task. It consisted of many steps, thorough planning, and unpredicted difficulties. The main goal was to create and implement a web-based application to help Heatington improve its heat output. In addressing the challenges and learnings of the project, our team identified several obstacles and experiences crucial to our learning and development. This chapter includes our assessment of our activities, successes and failures, and tips for future projects.

## 5a Working on a Common Project with Other Groups

### Positive Aspects

Working on the same problem alongside other groups was a beneficial experience, especially for early-semester students. It allowed us to learn from each other and exchange valuable insights. For instance, the use of extreme programming and code reviews within the team helped maintain consistency and avoid mistakes. Additionally, observing how other groups dealt with similar problems provided a comparative perspective, aiding our understanding of problem-solving approaches. Adopting the Agile methodology for the first time was an interesting experience for all team members. Learning to spread the work across different sprints made the project more manageable and intuitive once the methodology was grasped.

### Challenges

However, there were some drawbacks to having multiple groups work on the same projects. Meetings were sometimes too loose, lacking the necessary structure and efficiency. More structured and consistent meetings, especially at the beginning of the project, could have eliminated uncertainties among group members. Additionally, clearer project specifications from the outset would have facilitated better execution.

## 5b Experience with the Development of the Group's Specific Set of Tasks

### Positive Aspects

The chosen development process for our specific tasks led the team to achieve all vital aspects of the project, combined with additional components that improved the user experience. The implementation phase went smoothly, and the team divided tasks efficiently, ensuring that each member made significant contributions. Following the Agile methodology was quite helpful, as its flexible nature made responding to new ideas and requirements effortless as the project progressed.

### Challenges

Nevertheless, there were areas for improvement. Meetings could have been better organized, and time could have been utilized more effectively. It was sometimes unclear how various pieces of code interfaced within the project, suggesting that a better understanding of the system architecture was needed. More planning at the project's beginning could have developed a clearer vision and understanding of the project requirements.

## 5c Specific contributions of each team member

### Sebestyén

Sebestyén served as the Scrum Master. His responsibilities included building the original **Asset Manager**, developing different optimizer versions for general purposes and for comparable data, and saving data to external CSV files.

## Ignat

Ignat was part of the developer team and served as the main UI programmer. He enhanced the **Asset Manager** UI, polished the **Result Data Manager**, helped develop the **Boiler Usage Page**, and added optimization features for both costs and emissions. Additionally, Ignat maintained the group log and occasionally provided feedback to the team.

## Leonardo

Leonardo was part of the devleoper team and helped with different parts of the application. His biggest and most important contribution was the role of developing the **Neural Network Optimizer** component for the program. Moreover, he worked on other parts that did not involve the code but helped with the general success of the project.

## Kacper

Kacper served as the Product Owner, making sure the group is alligned on the vision and requirements of the project. He was also responsible for managing the jira backlog as well as the github repository, resolving merge conflicts and reviewing pull requests. Kacper also prepared the second version of the application's Figma prototype. In the codebase, Kacper was responsible for the **Source Data Manager** component, reading from .csv and .xslx files and saving data to .xslx files. He was also responsible for the Standard Optimizer used by the **Result Data Manger** component by default. Kacper is responsible for creating the in-memory database, along with it's many refactors, and the data models for the program. Kacper is also responsible for the **Unit Configuration** and the **Unit Activations** components, as well as providing help with the rest of the components.

## Levente

Levente was part of the developer team and created **Optimizer** component to display the optimized results and the production unit usage(hour by hour) data. He was also responsible for the Oil Boiler subcomponent of **Result Data Manager** in Iteration 1.

## 5d Future actions to prevent problems and difficulties faced during the project

To avoid the problems and difficulties faced during this project, the following actions could be adopted in future projects:

- **Structured and Consistent Meetings:** Implement more structured and frequent meetings to ensure all team members are on the same page. Daily Scrums should be carried out to ensure a clear workflow.
- **Clear Role Definitions:** Reevaluate team roles to ensure they suit each member's strengths. Consider having an external facilitator for meeting structures.
- **Early and Thorough Requirement Analysis:** Dive deeper into the project requirements early on and ensure everyone understands them. This includes reading through documentation and preparing for meetings in advance.
- **Better Integration and Communication:** Foster better communication regarding how different parts of the code and project fit together. Regularly review and discuss the overall system architecture.
- **Increased Preparation:** Encourage team members to prepare for new concepts and tasks before meetings, ensuring everyone is well-informed and ready to contribute effectively.

# Appendix

- Team's Github: `https://github.com/Kacper-Grzyb/heat-production-optimization.git`
- Figma Prototype: `https://www.figma.com/design/mVm6JOdz3luFADmHjPqNgJ/UI-Design?node-id=0-1&t=DokOTTTC6V7hd7sF-1`