

*TMM4275 Knowledge Based Engineering, project  
Hallvard Bjørgen, Johanne Glende, Sigve Sjøvold*

*Assignment 2: Construction Knowledge Base*

Link to Github: <https://github.com/Hallvaeb/kbe-a2>

Link to Diagrams:

<https://app.diagrams.net/#HHallvaeb%2Fkbe-a2%2Fmaster%2FDiagrams%2FDiagrams.drawio>

**Product description and selected parameters.**

Our solution consists of a web server, *Automated Building*, a knowledge base, a class for generating .dfa-files as well as other smaller classes. When opening the website there are four options presented to the end user. One can either “Add space” to the knowledge base, “Construct” a building, use SPARQL queries to ask the knowledge base for different data, or autogenerate buildings if one does not wish to choose the parameters themselves.

There are four different zones that are being constructed: Site, Building, Storey and Space. They have the same parameters as well as some additional ones to add more information to the different zones. The five parameters they have in common are:

- *zone\_id (string)*
- *length (int)*
- *type (string)*
- *width (int)*
- *height (int)*

In addition, the classes Site, Building and Storey each have a list containing the “contained zones”. This means that Site has the list *hasBuilding* containing all the buildings on that specific site, Building has a list *hasStorey* of all the storeys in that building and Storey has a list *hasSpaces* containing all the spaces in that Storey. This is to easily have control over the connections between the different zones. The connections saved in the lists are also saved in the knowledge base.

Building and Space also have the parameter *energy\_consumption*, which is an integer given by the end user. It is used to find the energy efficiency for a building or space, and is calculated in the code to give a character on how efficient it is.

The last parameter is *role* in Space, which says something about what the use of the space is, e.g. kitchen, bathroom or bedroom. It is saved as a string and is used to generate the id for the different spaces in the knowledge base.

All parameters can be found in the Design Class Diagram further down in the report.

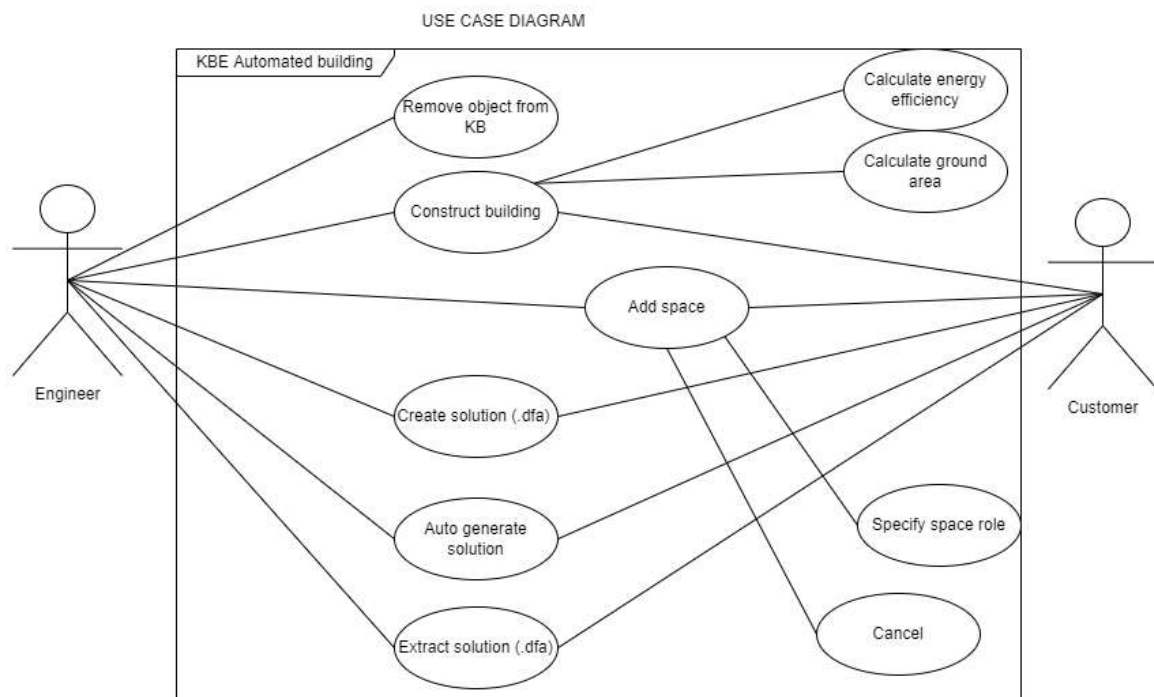
### KBE application architecture (main blocks and interconnections between those described). + UML class diagrams.

Our application is spread across multiple classes, and the interaction between the classes can almost be seen as a “pipe-and-filter” pattern, where the user gives input and the information is then sent through HTTPWebServer, then Controller, and from there on wherever it needs to go. Our architecture can also be seen as representing a “model-view-controller” design pattern, where our HTTPServer alone is the view, Controller is the controller, and model would be the logic in the Zones-classes and their helper-classes.

Clarifying and making the folder structure reflect a model-view-controller design could make the code more clear and easily understood.

We could have implemented the Singleton pattern for our locally HTTPServer ensuring only one instance of the server is ran as having multiple servers running is to no use. We did not, however, spend time making this tweak.

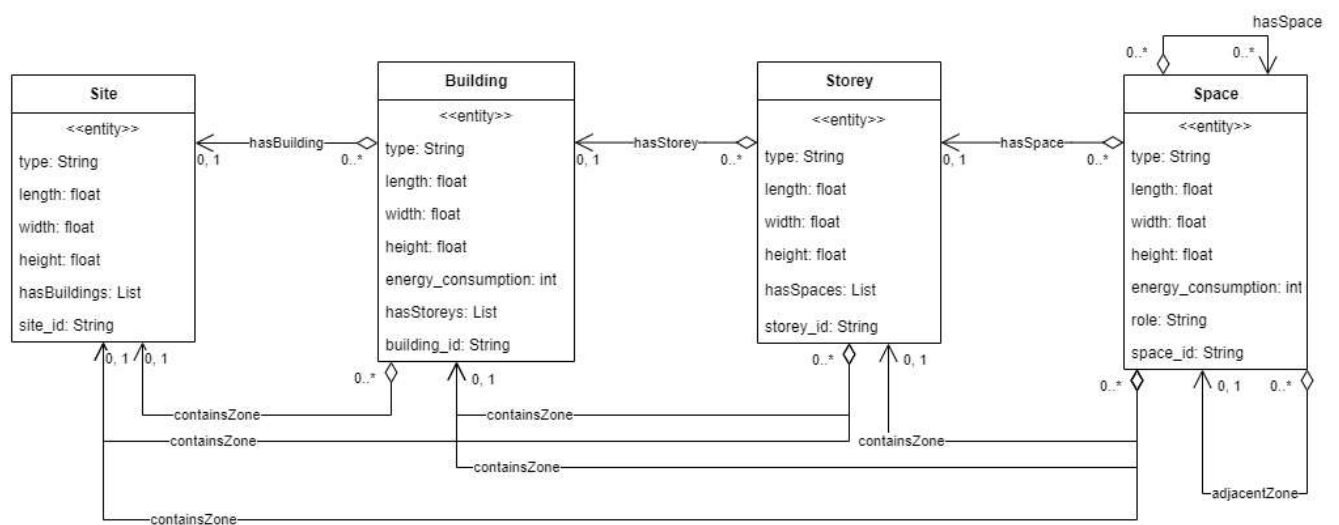
### Different UML diagrams:



**Use Case Diagram**

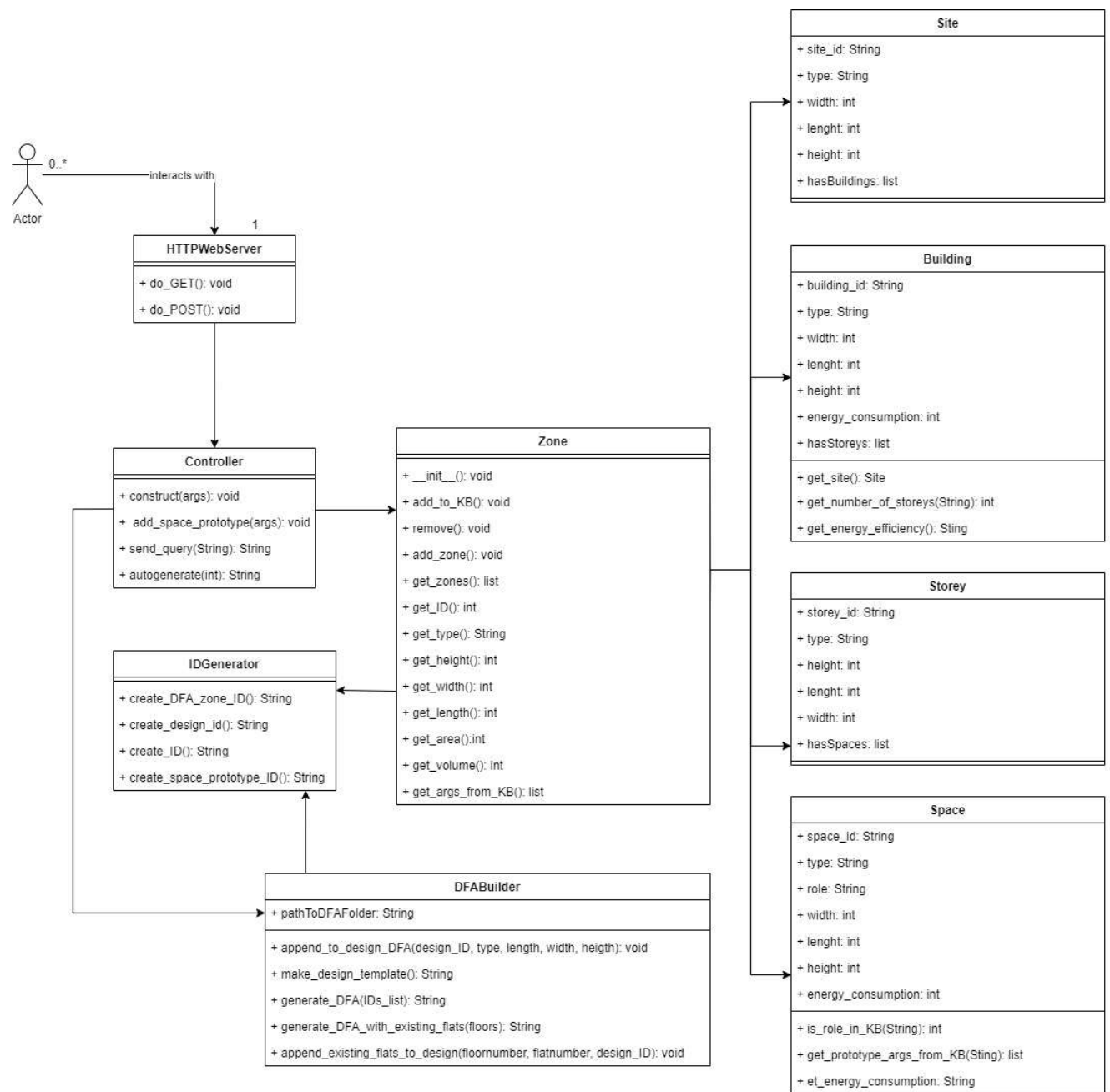
User interface is demonstrated through the result examples, supplemented with the videos in the Diagrams/UI\_videos.zip folder.

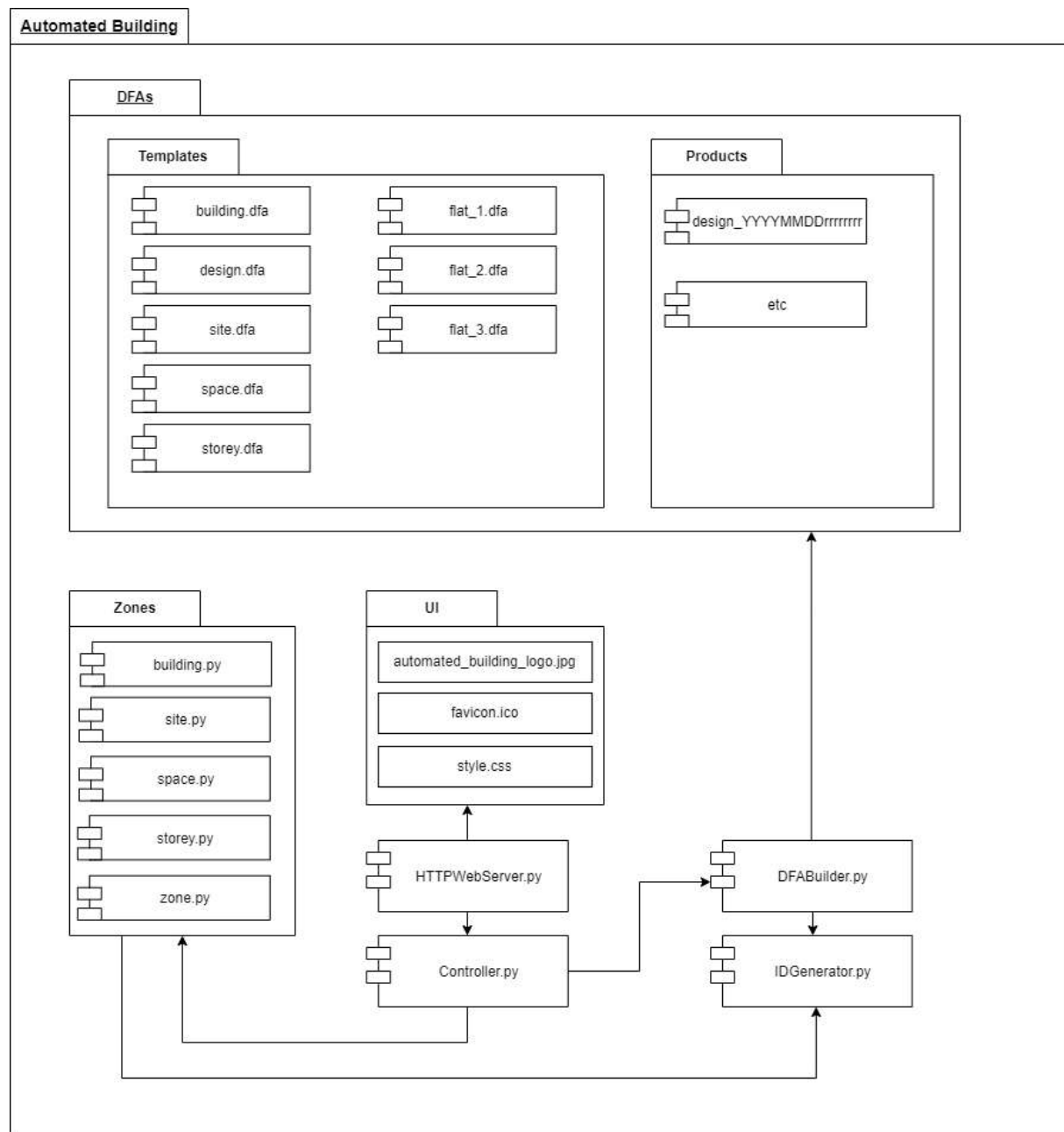
### ***User Interface Diagram***

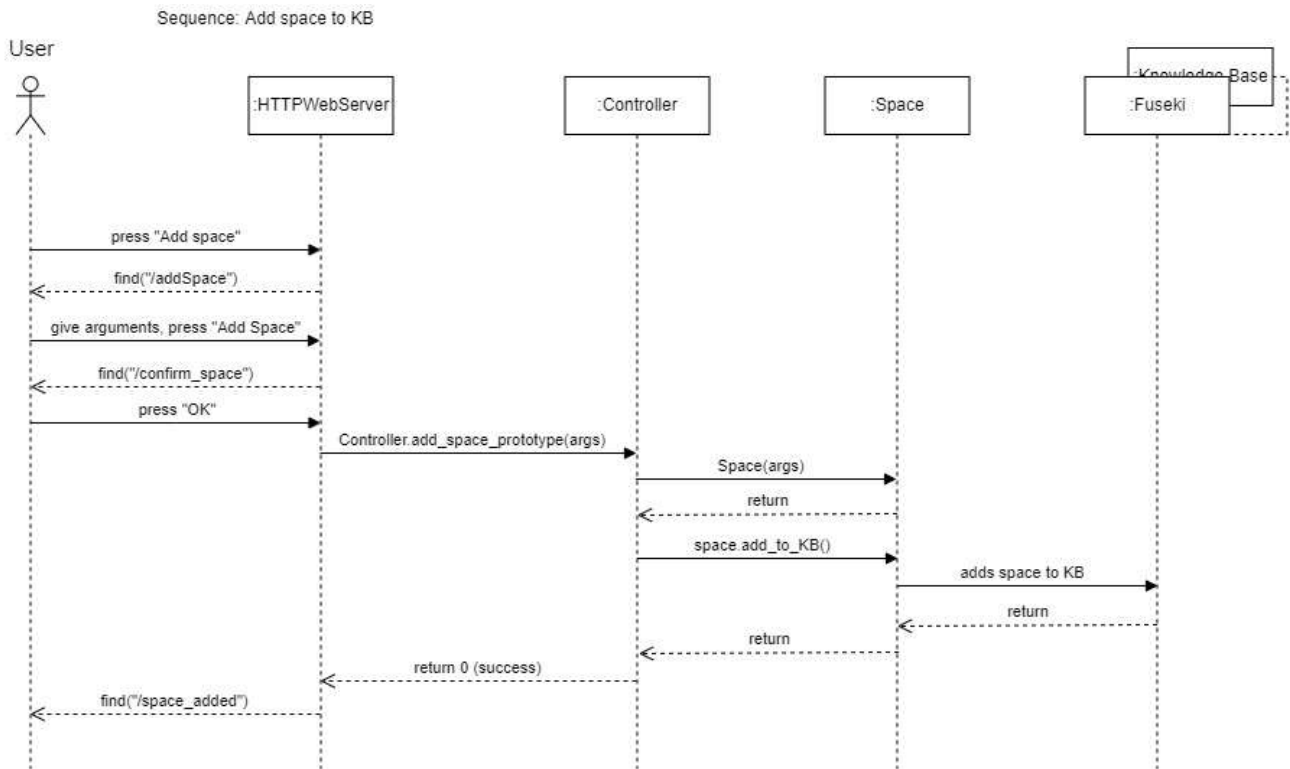
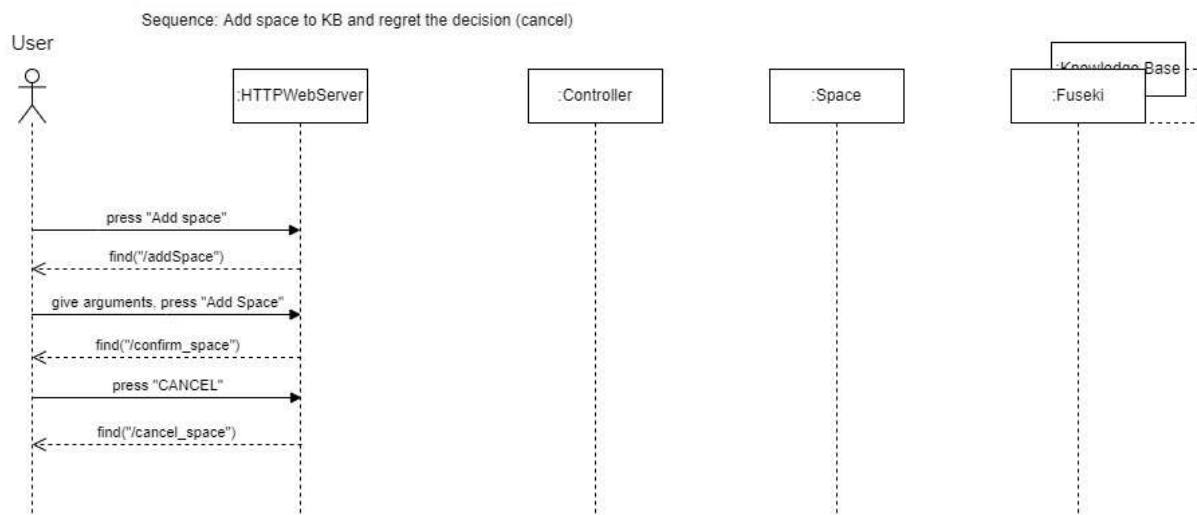


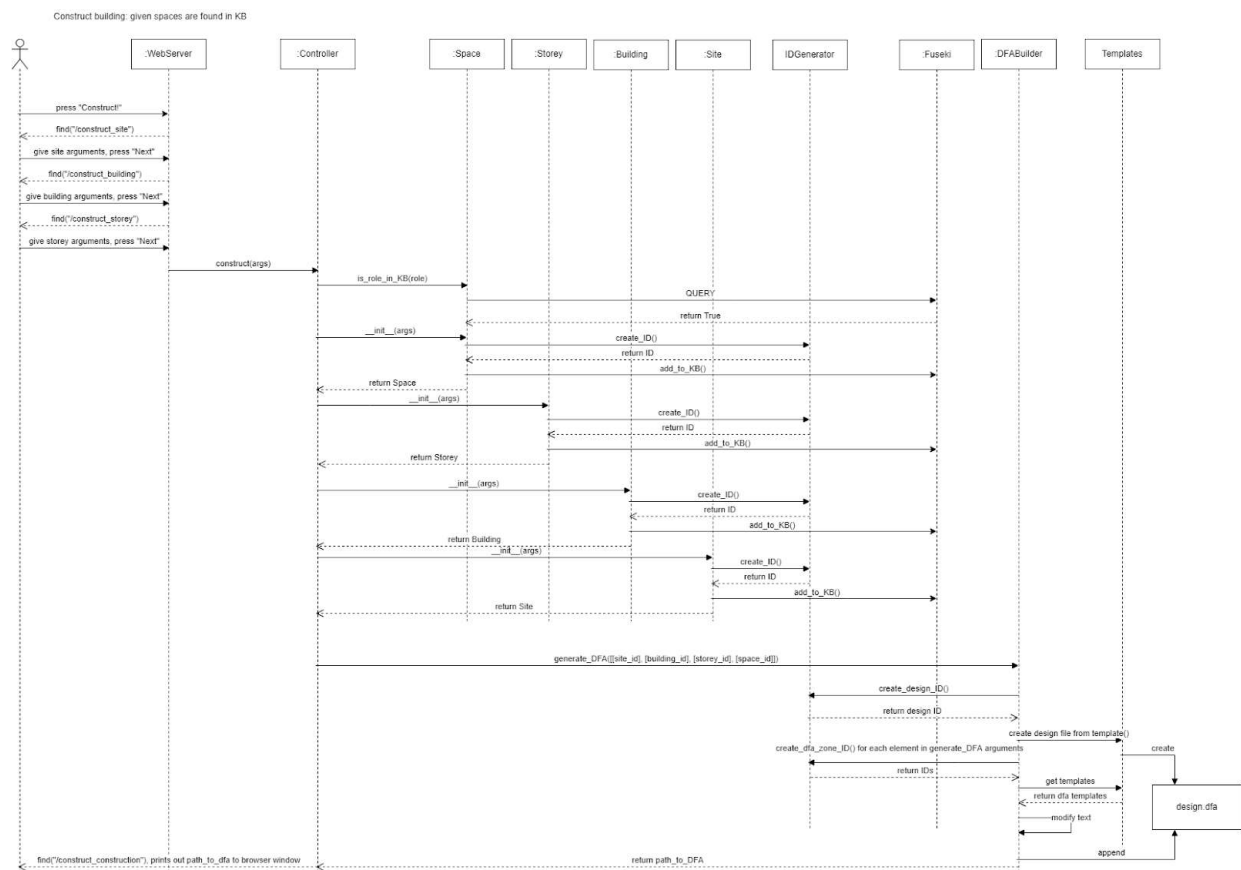
setter getters are line  
Diamond is owner relationship

### ***Problem Domain Class***

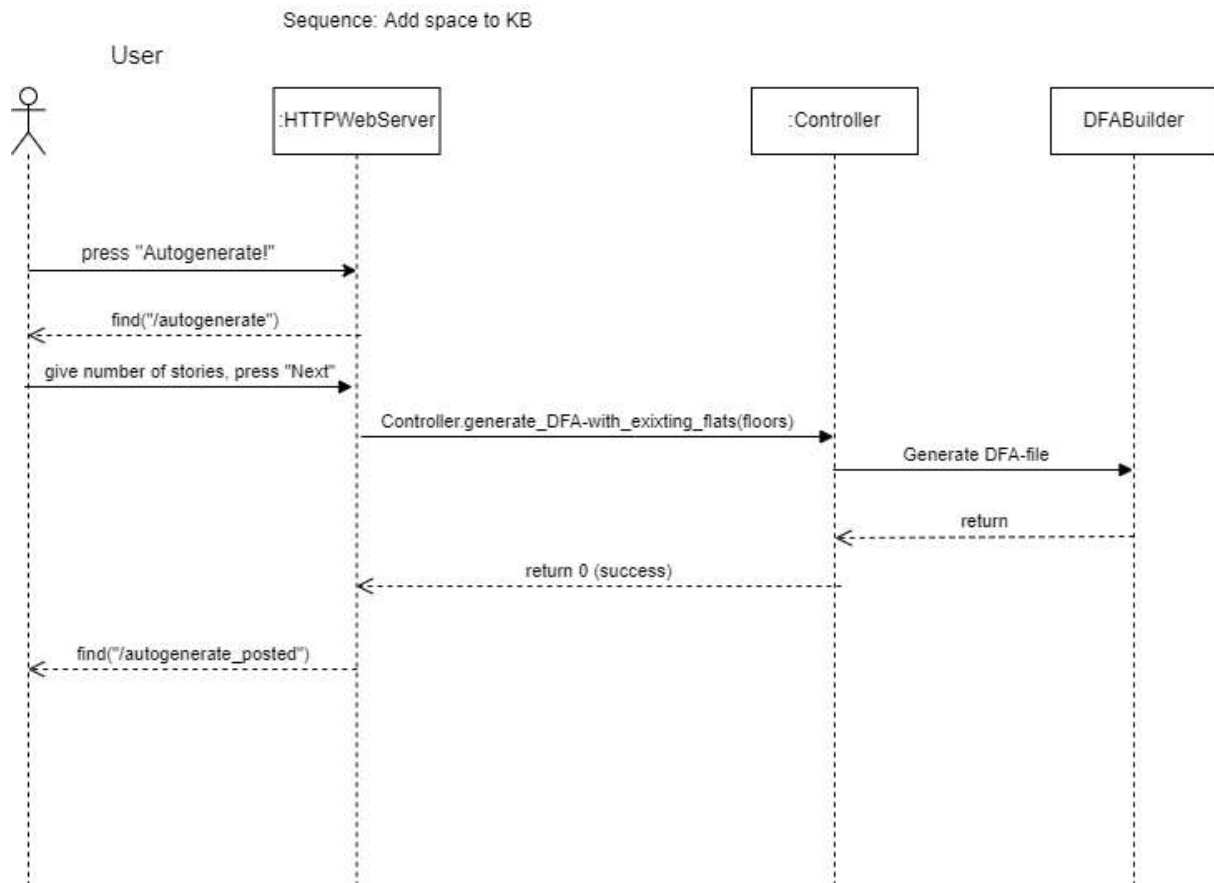
**Design Class Diagram**

**Package Diagram**

**Sequence Diagrams:****Sequence: Add space****Sequence: Add space, cancel before its added**

**Sequence: Construct building**



**Sequence: Autogenerate building**

**Description of the code you have developed - module names, their purpose and main functions.**

Our code consists of 9 modules:

- *HTTPWebServer.py*  
Contains the class HTTPWebServer which hosts a local server for the project's user interface. Communicates with Controller.py and sends all input from the user to the controller-class.
- *Controller.py*  
Contains the class Controller, which is the class that connects the project together. It communicates with all the other modules and makes instances of Site, Building, Storey and Space when a new .dfa-file is to be made. Contains the functions construct() which is used to construct new .dfa-files, add\_space\_prototype() which is used when the functionality "Add space to knowledge base" on the web site is used. It also contains the functions send\_query() which is used when the user wants to ask the knowledge base about something as well as autogenerate() which autogenerates a building in a new .dfa-file.
- *IDGenerator.py*  
Contains the class IDGenerator. Used to generate specific ids for the different zones built in DFABuilder.
- *DFABuilder.py*  
Used to build the .dfa-files the user is presented at the end of its order. Contains the class DFABuilder and the functions append\_to\_design\_DFA(), make\_design\_template() and generate\_DFA(), generate\_DFA\_with\_existing\_flats(), and append\_existing\_flat\_to\_design().
- *Zone.py*  
Contains the abstract class Zone, which Site, Building, Storey and Space inherits from. Contains functions like \_\_init\_\_(), add\_to\_KB(), remove(), add\_zone(), get\_height(), get\_length(), get\_width() such that Site, Building, Storey and Space must contain it as well.
- *Site.py*  
Inherits from Zone. Contains the class Site as well as small helper functions. Communicates with the fuseki server, and uses the functions inherited from Zone to add and remove instances to the knowledge base, as well as getters to get parameters like id and area.

- *Building.py*  
Inherits from Zone. Contains the class Building. Communicates with the fuseki server to add and remove instances of Building to the knowledge base. contains getters and other smaller functions like `get_energy_efficiency()`, `get_number_of_storeys()`, `get_site()` and `get_args_from_KB()`.
- *Storey.py*  
Inherits from Zone. Contains the class Storey. Communicates with the fuseki server, used to make instances of "storey" in the knowledge base with the help of different functions as `__init__()`, `add_to_KB()` and `get_args_from_KB()`.
- *Space.py*  
Inherits from Zone. Contains the class Space. Communicates with the fuseki server to add or remove instances or space from the KB. Contains different functions inherited from Zone as well as small helper functions.

Our .dfa-files does not contain the parameters given such as energy consumption and materials used. This could be an issue as the .dfa-files are thought to be used for sending a solution to coworkers. For visualization not relying on energy consumption, this is fine.

The .dfa-file is thought of as a way of extracting information from the KB. In the future, other applications apart from NX (which uses .dfa-files) could make their own "interface" for extracting information from the KB: our DFABuilder serves an example of how this task can be achieved.

### **Main challenges:**

Our biggest challenge with this assignment was time. It is a big, almost limitless assignment where several functionalities we couldn't implement would be great to implement. We will comment on this later on in the report.

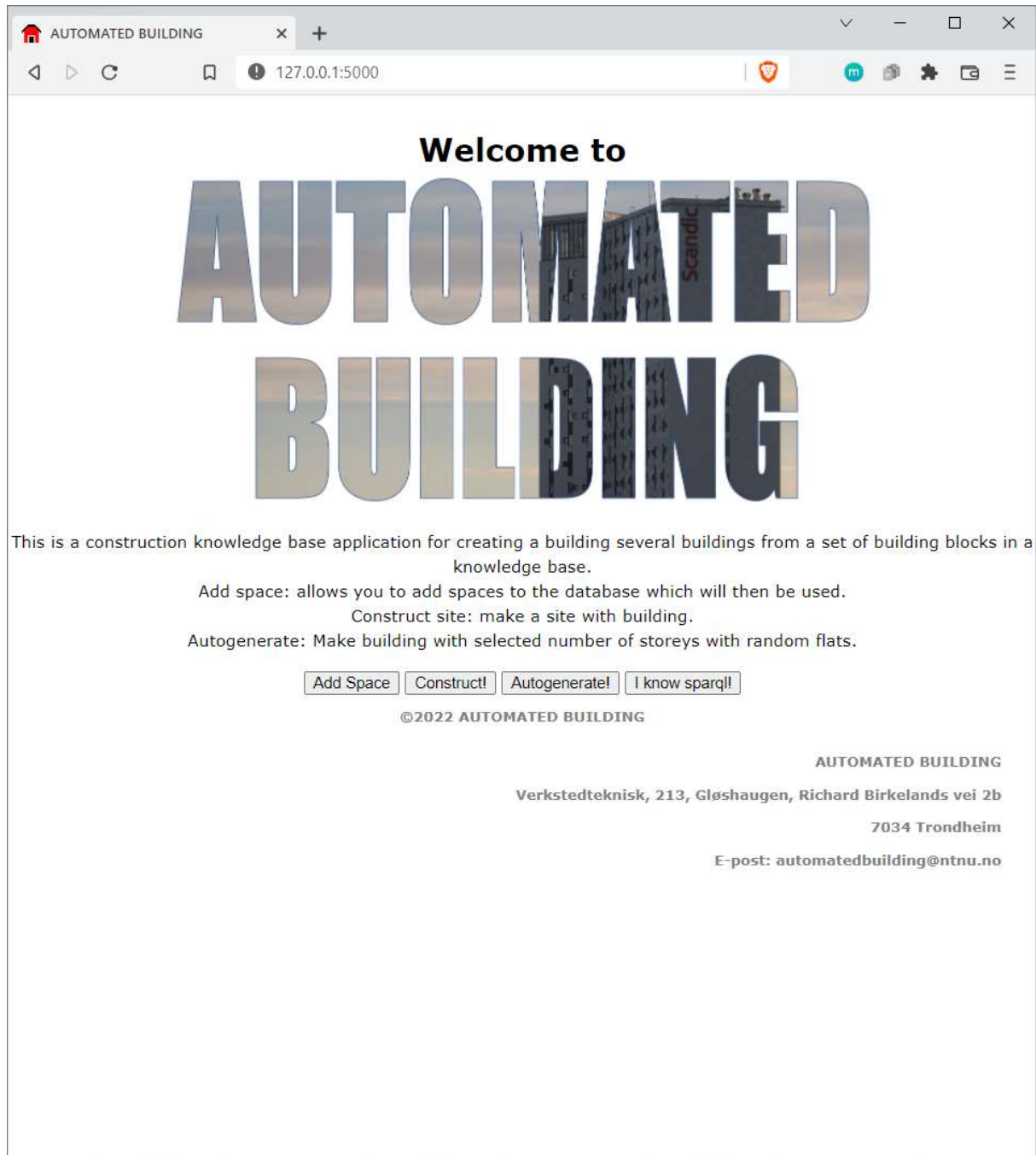
From a technical point of view, we had some issues putting the zones into the fuseki knowledge base correctly. We also had trouble with the web pages loading slowly and sometimes not at all. The way we created or web page is not a recommended way to go about it, and in future projects we will try a different tool than python HTTP Web Server.

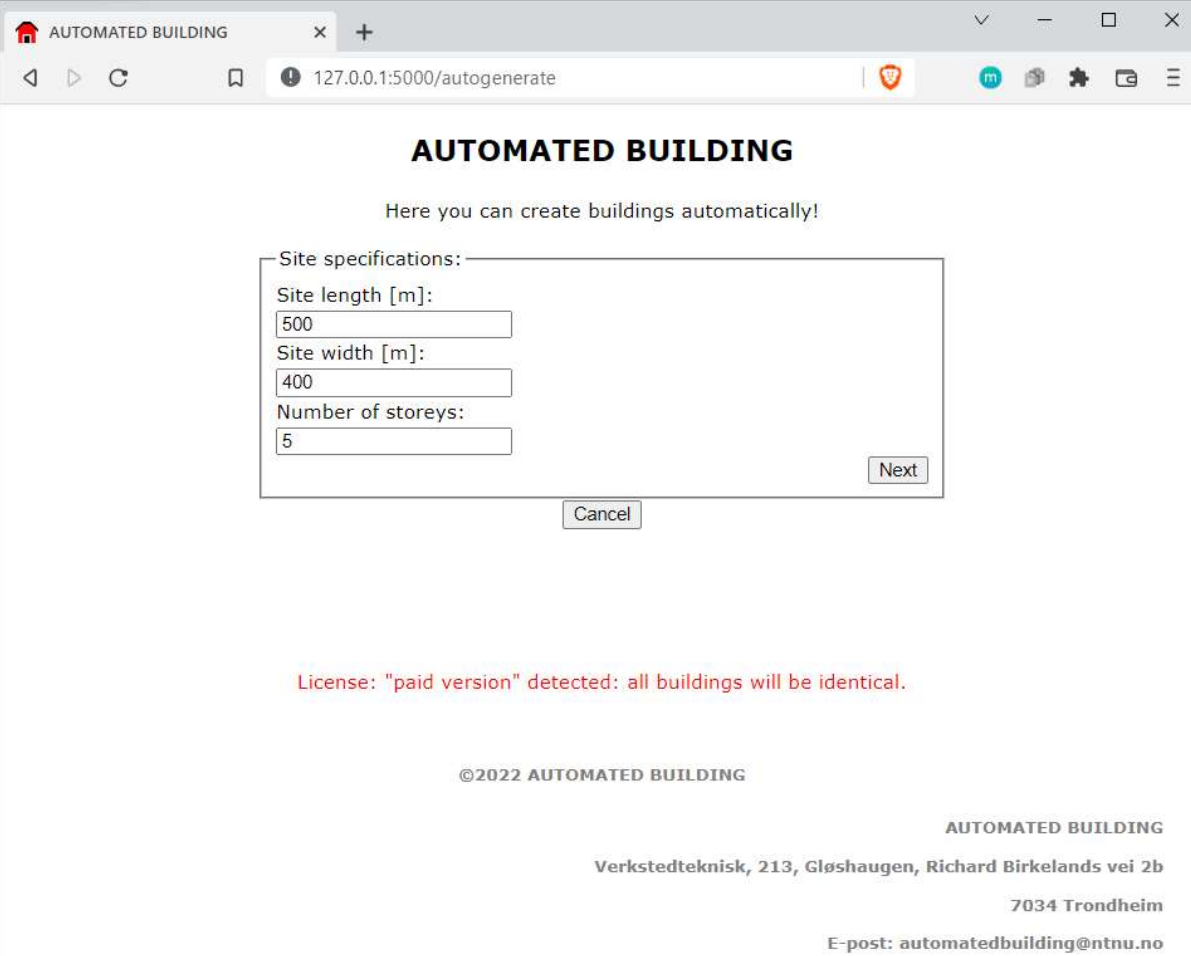
In addition, we also struggled when we attempted to write queries directly to the knowledge base from the website, as there were some errors with the encoding/decoding. Time would most likely have solved this issue.

**Results: 3 examples of the product ordered with different and arbitrary values of the parameters.**

Example 1: Placing an order for an autogenerated building.

Using the web server, on the front page, press "Autogenerate!".





**AUTOMATED BUILDING**

Here you can create buildings automatically!

Site specifications:

Site length [m]:

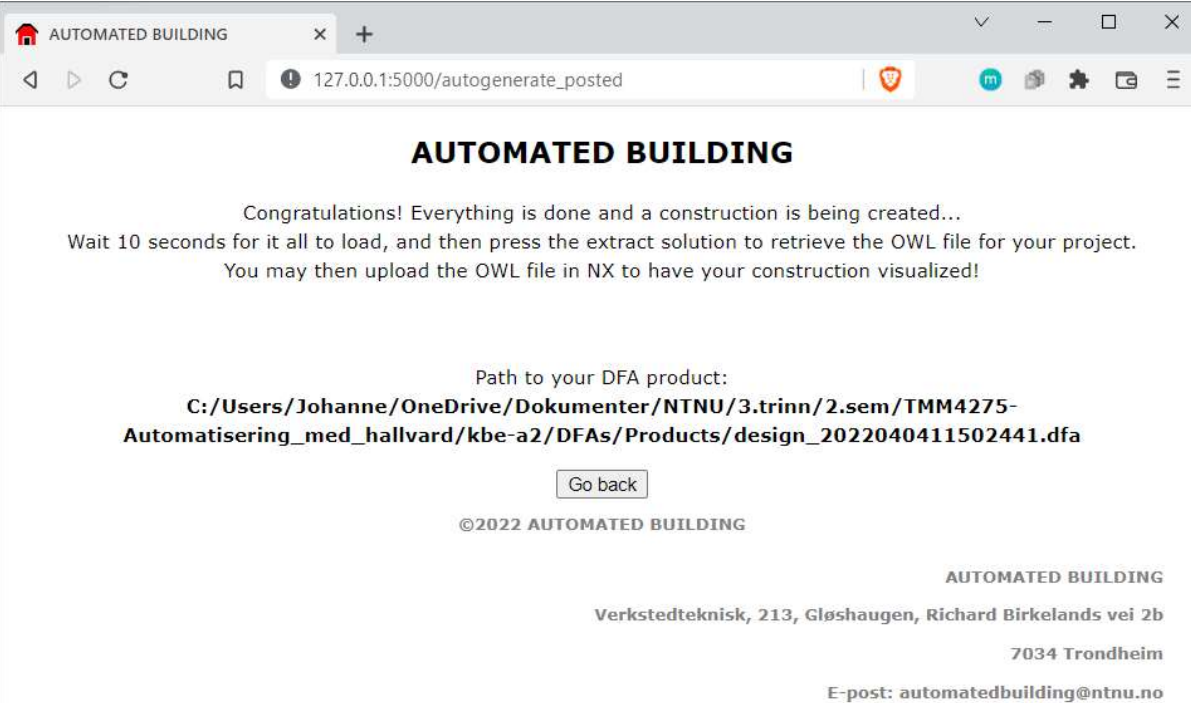
Site width [m]:

Number of storeys:

License: "paid version" detected: all buildings will be identical.

©2022 AUTOMATED BUILDING

**AUTOMATED BUILDING**  
Verkstedteknisk, 213, Gløshaugen, Richard Birkelands vei 2b  
7034 Trondheim  
E-post: automatedbuilding@ntnu.no



**AUTOMATED BUILDING**

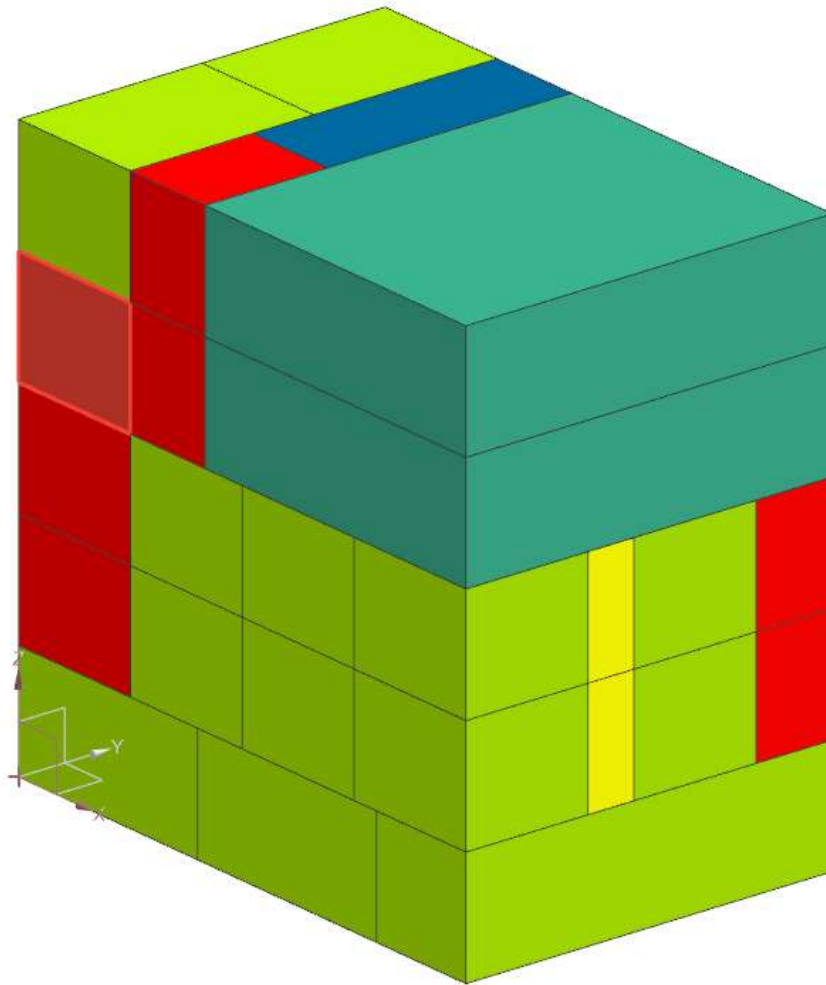
Congratulations! Everything is done and a construction is being created...  
Wait 10 seconds for it all to load, and then press the extract solution to retrieve the OWL file for your project.  
You may then upload the OWL file in NX to have your construction visualized!

Path to your DFA product:  
**C:/Users/Johanne/OneDrive/Dokumenter/NTNU/3.trinn/2.sem/TMM4275-Automatisering\_med\_hallvard/kbe-a2/DFA/Products/design\_2022040411502441.dfa**

©2022 AUTOMATED BUILDING

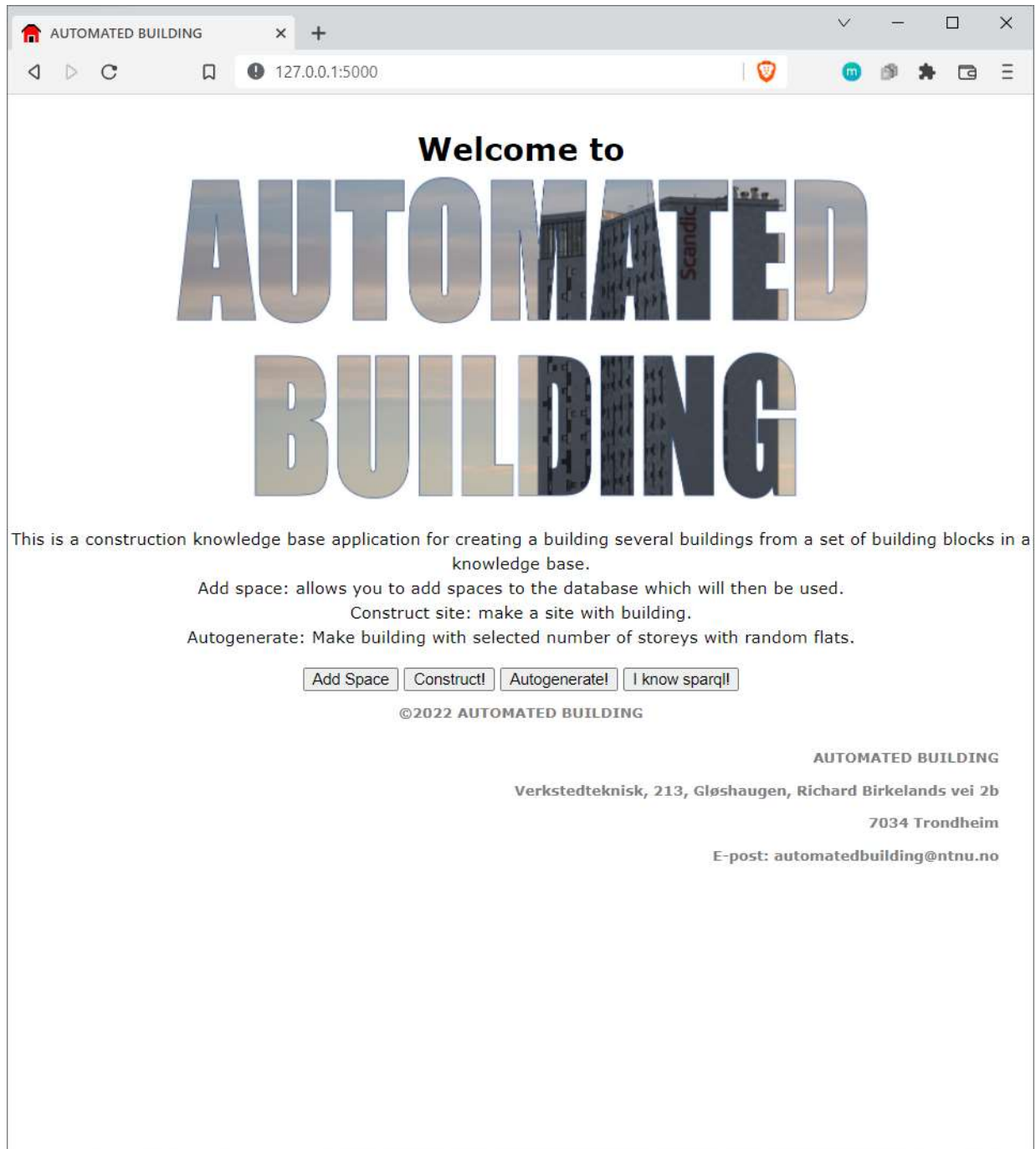
**AUTOMATED BUILDING**  
Verkstedteknisk, 213, Gløshaugen, Richard Birkelands vei 2b  
7034 Trondheim  
E-post: automatedbuilding@ntnu.no

Opening the .dfa-file in NX:



Presented: a building with five storeys. The storeys are autogenerated with different spaces in them. The different colors define the roles of the spaces. The green ones are bedrooms, the red spaces are bathrooms, the blue are kitchens, the small yellow ones are hallways and the dark green are living rooms.

Example 2: A customer wants to add a space with the role “kitchen” to the knowledge base. On the front page, press “Add space”. Note: This is also demonstrated in one of the videos in Diagrams/UI\_videos



The screenshot shows a web browser window with the title 'AUTOMATED BUILDING' and a tab for 'Apache Jena Fuseki - inspect dataset'. The address bar shows '127.0.0.1:5000/add\_space'. The main heading is 'AUTOMATED BUILDING'. Below it, a paragraph explains that users can add a space to a knowledge base skeleton and use the role field to specify the space type. It suggests 'kitchen', 'bedroom', or 'chair' as good names. A 'Space specification' form contains input fields for Length (20), Width (30), Height (30), Energy consumption (60000), and Role (Kitchen). An 'Add Space' button is at the bottom right of the form, and a 'Go back' button is below it. The footer includes the copyright notice '©2022 AUTOMATED BUILDING' and contact information for 'AUTOMATED BUILDING' at Verktedteknisk, 213, Gløshaugen, Richard Birkelands vei 2b, 7034 Trondheim, with email 'automatedbuilding@ntnu.no'.

**AUTOMATED BUILDING**

Here you can add a space to the knowledge base skeleton for later use in a construction of the building.  
Use the role field to specify if your space is a "flat" or a specific room or object type.  
A good name would be: "kitchen", "bedroom" or "chair".

Space specification:

Length:

Width:

Height:

Energy consumption:

Role:

©2022 AUTOMATED BUILDING

**AUTOMATED BUILDING**  
Verktedteknisk, 213, Gløshaugen, Richard Birkelands vei 2b  
7034 Trondheim  
E-post: automatedbuilding@ntnu.no

The screenshot shows a web browser window with the title 'AUTOMATED BUILDING' and a tab for 'Apache Jena Fuseki - inspect dataset'. The address bar shows '127.0.0.1:5000/confirm\_space'. The main heading is 'AUTOMATED BUILDING'. Below it, a paragraph states 'Your space is being added... Click "OK" to confirm or CANCEL IF YOU REGRET YOUR DECISION.' A 'Review input' form displays the same data as the previous screen: Length (20), Width (30), Height (30), Energy efficiency (60000), and Role (Kitchen). It includes 'OK' and 'CANCEL' buttons. The footer is identical to the previous screen, showing the copyright notice and contact information for 'AUTOMATED BUILDING'.

**AUTOMATED BUILDING**

Your space is being added... Click "OK" to confirm or CANCEL IF YOU REGRET YOUR DECISION.

Review input

This is your input for space, you can still modify...

Lenght:

Width:

Height:

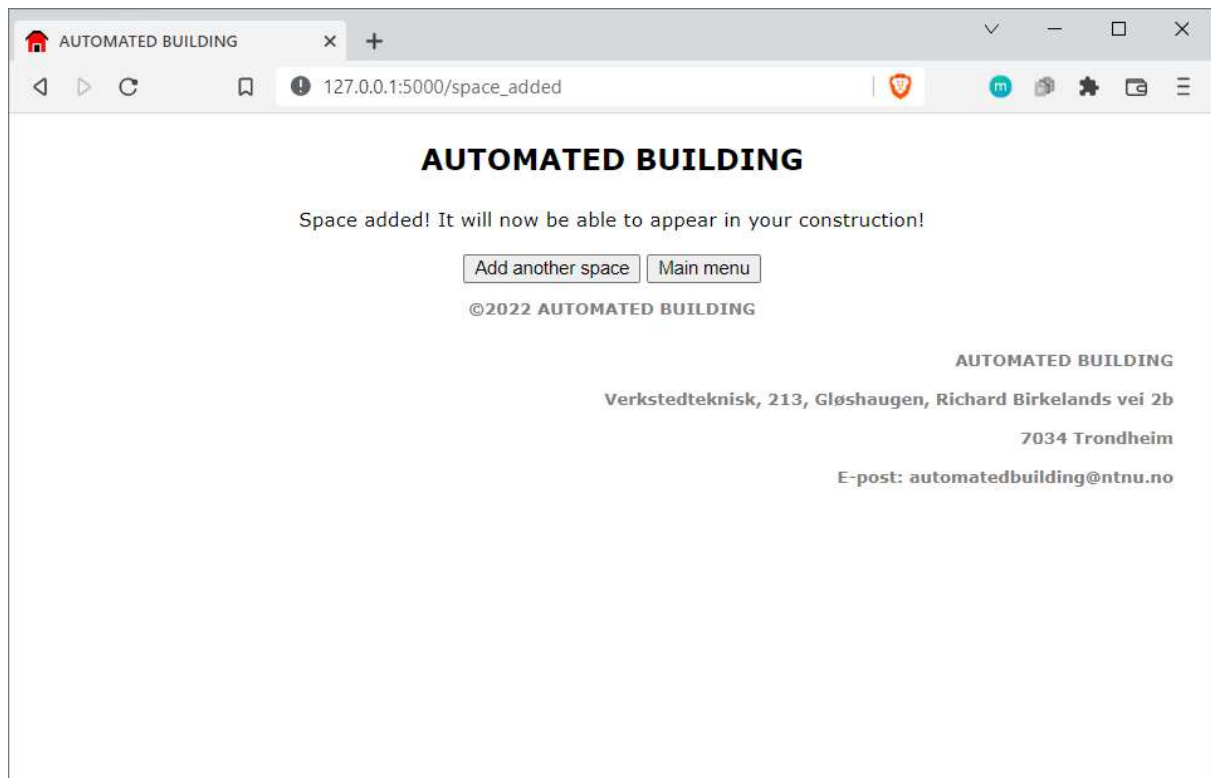
Energy efficiency:

Role:

©2022 AUTOMATED BUILDING

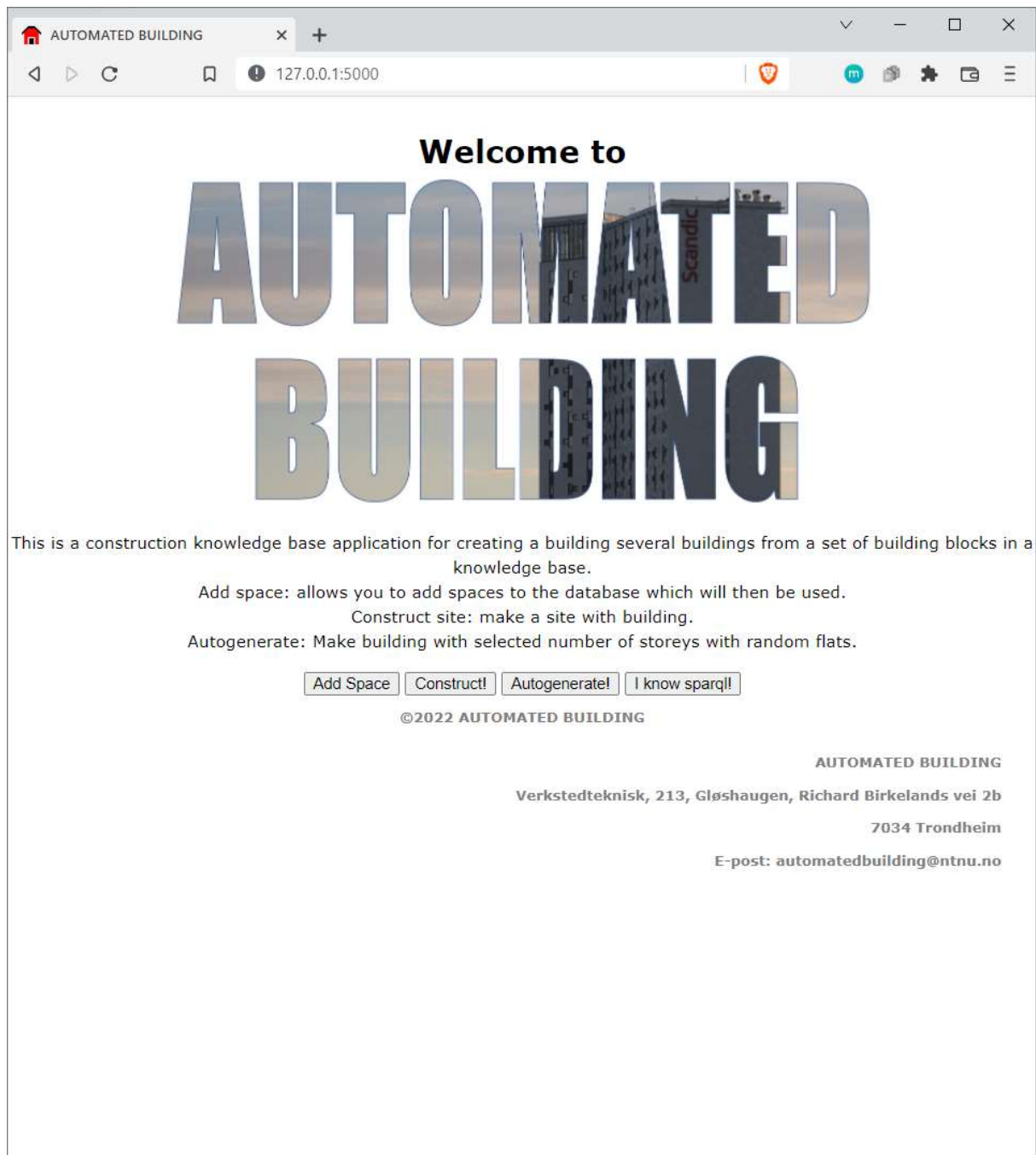
**AUTOMATED BUILDING**  
Verktedteknisk, 213, Gløshaugen, Richard Birkelands vei 2b  
7034 Trondheim  
E-post: automatedbuilding@ntnu.no





Example 3: Placing an order for a site with one building, which contains the space added in example 2.

On the front page, press “Construct!”.



**AUTOMATED BUILDING**

Here you can create buildings automatically!  
Does it require specific spaces not in the knowledge base?  
Contact an engineer or add it through the 'add space' option.

Site specifications:

Site length [m]:  
500

Site width [m]:  
400

Number of buildings:  
1

Buildings identical: ☒

Next

Cancel

License: "free version" detected: all buildings will be identical.

©2022 AUTOMATED BUILDING

**AUTOMATED BUILDING**  
Verkstедteknisk, 213, Gløshaugen, Richard Birkelands vei 2b  
7034 Trondheim  
E-post: automatedbuilding@ntnu.no

**AUTOMATED BUILDING**

Here you can create buildings automatically!  
Does it require specific blocks not in the knowledge base?  
Contact an engineer or add it through the 'add space' option.

License: "free version" detected: all storeys will be identical.

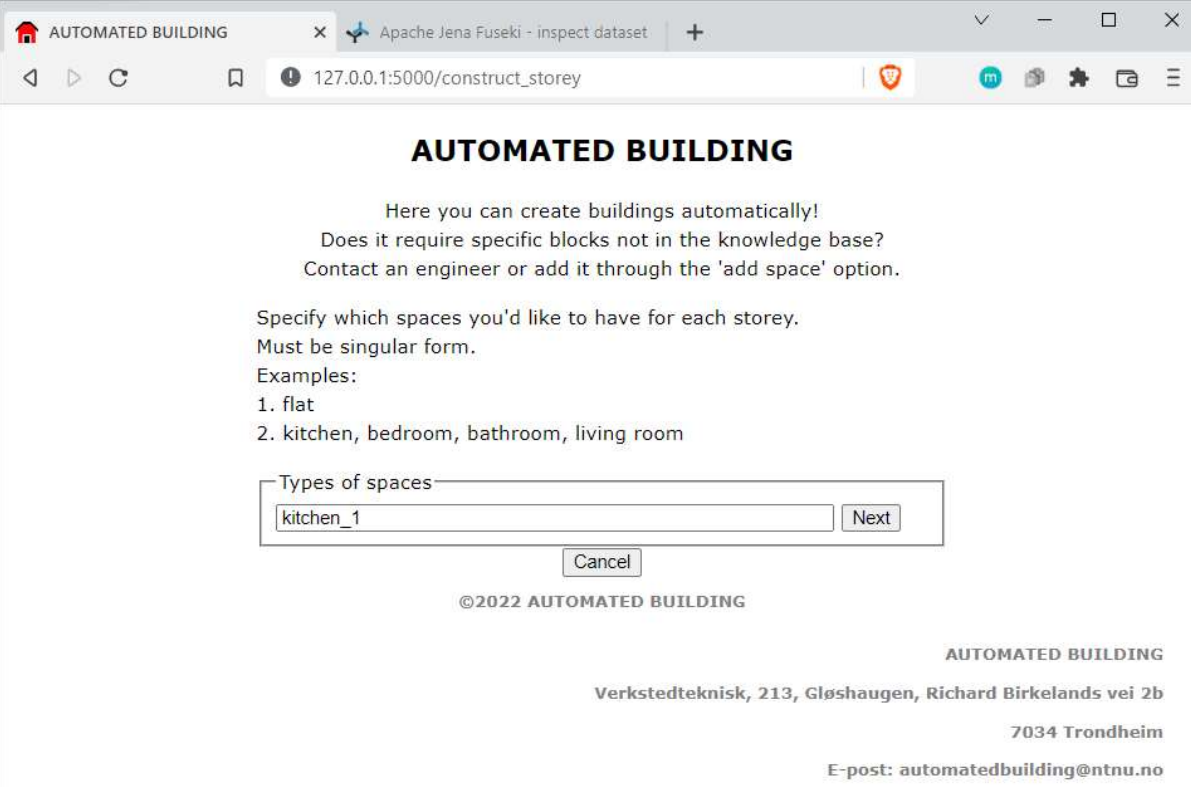
Building Number	Length [m]	Width [m]	Height [m]	Energy Consumption [kWh]	Storeys:	Storeys identical:
1	40	30	30	60000	1	<input checked="" type="checkbox"/>

Next

Cancel

©2022 AUTOMATED BUILDING

**AUTOMATED BUILDING**  
Verkstедteknisk, 213, Gløshaugen, Richard Birkelands vei 2b  
7034 Trondheim  
E-post: automatedbuilding@ntnu.no



**AUTOMATED BUILDING**

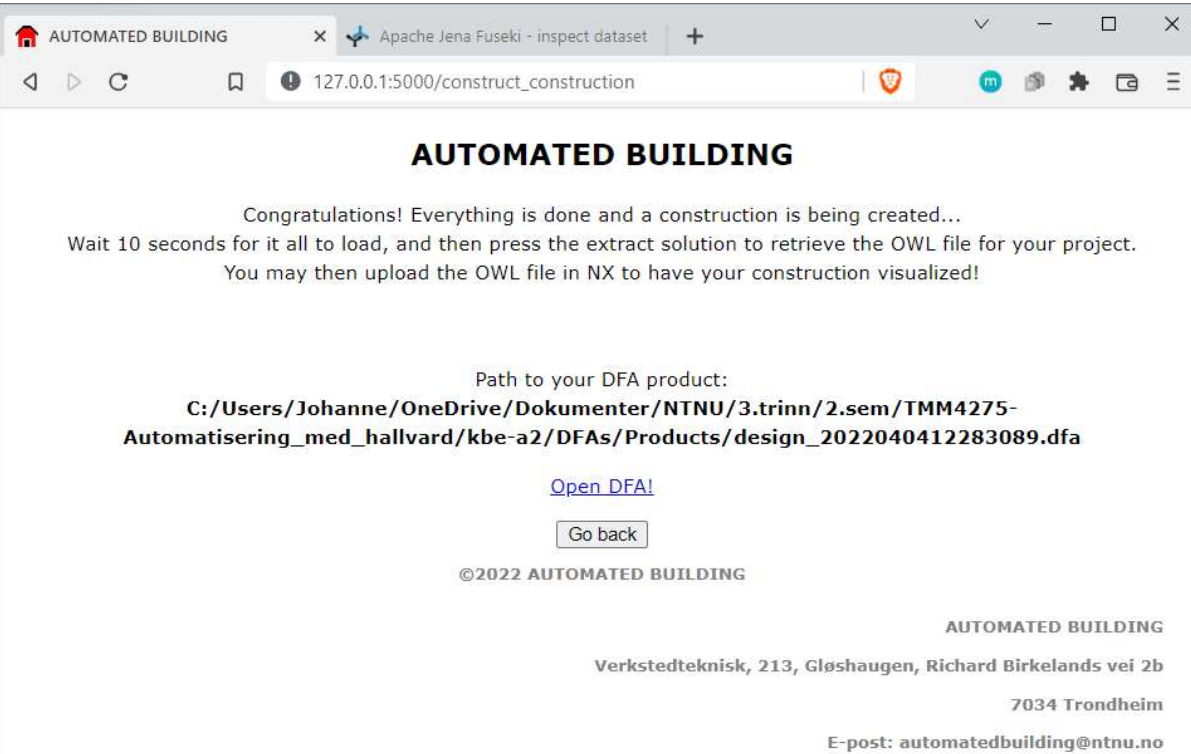
Here you can create buildings automatically!  
Does it require specific blocks not in the knowledge base?  
Contact an engineer or add it through the 'add space' option.

Specify which spaces you'd like to have for each storey.  
Must be singular form.  
Examples:  
1. flat  
2. kitchen, bedroom, bathroom, living room

Types of spaces

©2022 AUTOMATED BUILDING

**AUTOMATED BUILDING**  
Verkstедteknisk, 213, Gløshaugen, Richard Birkelands vei 2b  
7034 Trondheim  
E-post: automatedbuilding@ntnu.no



**AUTOMATED BUILDING**

Congratulations! Everything is done and a construction is being created...  
Wait 10 seconds for it all to load, and then press the extract solution to retrieve the OWL file for your project.  
You may then upload the OWL file in NX to have your construction visualized!

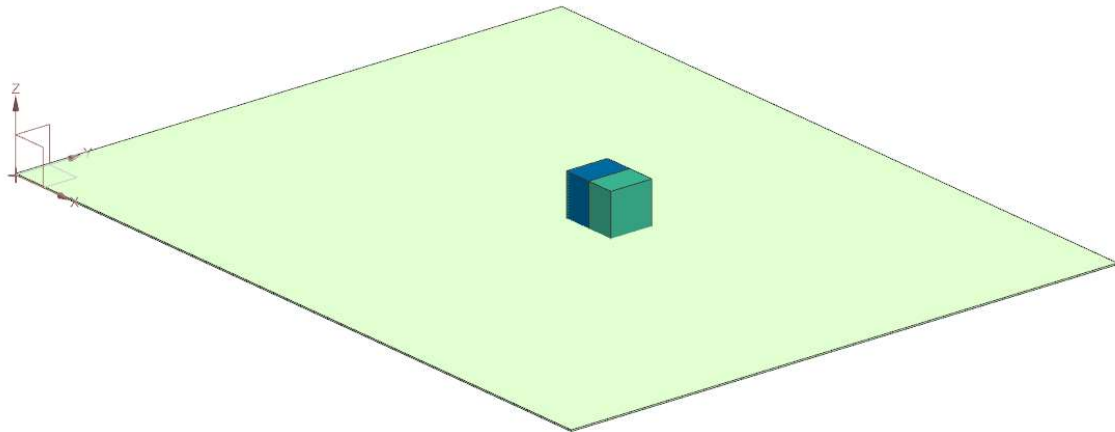
Path to your DFA product:  
**C:/Users/Johanne/OneDrive/Dokumenter/NTNU/3.trinn/2.sem/TMM4275-Automatisering\_med\_hallvard/kbe-a2/DFAs/Products/design\_2022040412283089.dfa**

[Open DFA!](#)

©2022 AUTOMATED BUILDING

**AUTOMATED BUILDING**  
Verkstедteknisk, 213, Gløshaugen, Richard Birkelands vei 2b  
7034 Trondheim  
E-post: automatedbuilding@ntnu.no

Opening the .dfa-file in NX:



Here we can see the building generated in NX. The color codes are: storeys are colored dark green, the spaces (here a kitchen) are blue and the site is light green. The reason we can see both the green and blue color is because the kitchen only takes up half of the storey in the building. If the kitchen had been bigger the blue would have colored the whole building.

**Future implementation:**

This assignment was really enjoyable for all three members, and we all saw the potential to create a usable, sought after system. Unfortunately, mainly due to time restrictions we were not able to implement all the functions we visualized, and some of the functionality is not optimal. Therefore we wanted to talk about future development and improvements to the system. Currently our system does not allow the user to create several buildings on one site. In addition a building can only contain one storey and the storey only contains one space. The exception to this being the “autogenerate”-option. The reason we initially went for this solution was so we could start developing our system, and detect/fix bugs and errors more easily. Our main goal in the beginning was for the system to work with one element of each zone. But, once we got there we had to prioritize other functionalities, and we weren't able to implement a solution for more than one of each zone before our deadline was due.

Another part of the system we wanted to improve but couldn't prioritize is the .dfa-files. As it is now, every element of the construction is constructed as a simple block. Initially we wanted to be able to generate more complex structures with walls, doors, windows, furniture etc. Our system supports a simple implementation of this, but it would be time-consuming to create templates for all these different elements, which is why we haven't done it yet.

An implementation we haven't started yet is user login. We want a functionality where users have to log in to be able to use the system. With this functionality they should have a list of previous solutions belonging to them, and could retrieve these solutions. Login via email would also allow for the option to send dfa-files by email once a solution is created.

We also wanted to add some external parameters, but were not able to begin this process. E.g. amount of sunlight a building receives. We also started adding methods for energy efficiency, but at the moment we don't utilize these methods. This is something we would do in the future.