UNIVERSITEIT GENT

Group project Databasesystemen en Algoritmen en datastructuren

Seppe vanden Broucke/Frederik Gailly

Dendooven Tristan - 01909417

Denys Thomas - 01907645

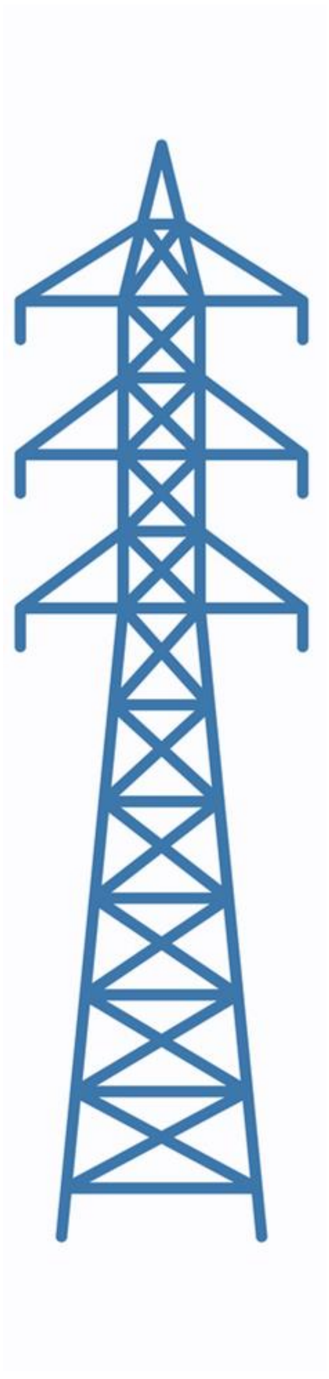
Vanhooren Leon - 01909802

Vanneste Emiel - 01902563

Vissers Milan - 01910269

Willemkens Hanne - 01906875

CONTENT



INTRODUCTION

The past weeks, we, six commercial engineering students, worked on the common project "Student energy conservation" for the courses "Algorithms and data structures" and "Database systems". For this, we had to develop a platform that allows students and landlords to keep track of the energy consumption in their student dorms. This will ensure that students are aware of their consumption.
The energy consumption values in the student rooms are stored in a central database. This platform aims to reduce students' electricity consumption and will help the students that live in the landlords’ dorms to identify problems and take appropriate actions.
Since the measurements have to be entered manually, we developed an attractive user interface that can be used easily by both students and landlords.
A landlord can draw up a contract for a student who is going to move into a room. The data concerning the lease is entered into the database. Applications in the room and energy-saving actions on it can be added to the database.

In the first part of this report we provided a summary of the project timesheet that lists the accomplished activities for every group member. Next, this report highlights the conceptual data model, followed by assumptions, an explanation, and limitations of your model.

In the second part we illustrate the system design and implementation. We provide the relational model and normalization steps. Next, we give an overview of the database tables and the referential integrity rules. Then, we made a comparison between the conceptual model and the relational model where we clearly note which semantics were lost during the conversion from your conceptual model.

In the third part, we explain the system design. We cover the software architecture, the UML class diagram for each module, the implementation and the algorithmic support.

Finally, this report ends on a critical note by outlining general limitations and opportunities for improvement.

Summary of project timesheet

In the first part of this report we provide a summary of the project timesheet that lists the accomplished activities for every group member.

Date	Subject	Members	Hours (per person)
15/11/2021	Discussion overall approach	Everybody	3
17/11/2021	Database design and implementation • Construction EER diagram	Everybody	3
18/11/2021 – 19/11/2021	System design and implementation • Object-oriented software design • System implementation	Leon	5
23/11/2021 – 29/11/2021	Database design and implementation System design and implementation	Milan, Emiel and Hanne Leon, Tristan and Thomas	9
1/12/2021 – 2/12/2021	Database implementation System implementation	Milan, Emiel and Hanne Leon, Tristan, Thomas and Milan	12
6/12/2021 – 8/12/2021	System implementation	Everybody	29
13/12/2021 – 17/12/2021	Report System implementation	Emiel, Thomas and Hanne Leon, Tristan and Milan	48

Conceptual data model

The conceptual model exists of 7 entity types, namely: student, landlord, building, room, monthly consumption, appliances and actions. These are indicated in our E(E)R diagram by boxes. The entity types are connected to each other by relationship types. These are drawn in our model by a circle. In the text below, we discuss every entity type briefly.

The conceptual model exists of 7 entity types, namely: student, landlord, building, room, monthly consumption, appliances, and actions. These are indicated in our E(E)R diagram by boxes. The entity types are connected to each other by relationship types. These are drawn in our model by a circle. In the text below, we discuss every entity type briefly.

The entity type STUDENT represents a student in the model. This entity type can rent exactly one room.
This relationship type is given by LEASE between STUDENT and ROOM. The STUDENT can lease just one room. If we look at our diagram, we can see that STUDENT has also another relationship type, namely CONTRACT.
This relationship type makes a contract between LANDLORD and STUDENT, whereby a STUDENT can have a contract with just one LANDLORD.
CONTRACT has attribute types, namely: contractNr, startDate, contractDuration, status and contractRoomID. These are all atomic and single-valued attribute types.
The key attribute type of this relationship type is contractNr. This attribute type describes the uniqueness of a CONTRACT between one STUDENT and one LANDLORD.
Status gives the status of the contract which is set on ‘pending’ and will change if the student accepts the contract, then the status will change to ‘accepted’. He can also decline the contract and then the status will be ‘declined’. If the contract is accepted, the relationship type LEASE will be made between STUDENT and ROOM.
About the other attribute types there is nothing special to mention.
The entity type STUDENT has also some attribute types. The single-valued attribute types are email, studentID, password, telephoneNr and name. The key attribute type of this entity type is studentID who defines a STUDENT. The attribute type name is a composite attribute type that exists out of firstname and lastname. All the other attribute types are atomic attribute types.

The second entity type is LANDLORD. This entity type represents a person who owns buildings with rooms. This entity type has also two relationship types.

The first one is also a CONTRACT that makes a contract between LANDLORD and STUDENT. The LANDLORD can have a contract with 0 to N STUDENTS. The attribute types of CONTRACT are just the same as mentioned above. Furthermore, there is nothing to mention about this relationship type.
The second relationship type is OWNERSHIP between LANDLORD and BUILDING. The LANDLORD can have 1 to N BUILDING. The entity type has 5 single valued attribute types. These are name, telephoneNr, email, landlordID and password. We can see that name is a composite attribute type consisting of firstname and lastname. All the others are atomic attribute types. The key attribute type of this entity type is landlordID. This attribute type makes each entity of the entity type LANDLORD unique. The other attribute types need no further explanation.

The third entity type of the conceptual model is BUILDING. This represents the buildings of the landlords who consist of rooms for the students.
This entity type has also two relationship types. The first relationship type is OWNERSHIP. We already discussed this relationship type before, so this one needs no further explanation. You just have to know that each BUILDING is owned by just one LANDLORD.
The second relationships type is BELONGS TO who connects the weak entity type ROOM with the owner entity type BUILDING. Each BUILDING has 0 to N ROOM. The attribute types of BUILDING are all atomic and single valued. They consist of zip, country, address, city and buildingID. The key attribute type is buildingID who represents the uniqueness of each BUILDING. The other attribute types need no further explanation.

The fourth entity type is a weak entity type, namely ROOM. This weak entity type is existence dependent from the owner entity type BUILDING. This is represented in our model using a double bordered rectangle.
This entity type has 3 relationship types. The relationship type BELONGS TO links the rooms to a building. This relationship type is already discussed in the paragraph above, but we have to mention that each ROOM belongs to just one BUILDING.
The second relationship type is REGISTERS between ROOM and MONTHLY CONSUMPTION. This relationship type registers the energy consumption of each room monthly, whereby each ROOM has just one MONTHLY CONSUMPTION. REGISTERS has also one attribute type date who is single valued and atomic. This attribute type represents the day when the monthly consumption of the room is registered.
The third and last relationship type is CONTAINS between ROOM and APPLIANCES, whereby each ROOM has 0 to N APPLIANCES. This relationship type gives the link between all the appliances of a student in his/her room and the room itself, so we can know which appliances are in which room.
The entity type ROOM has 4 attribute types, namely roomID, roomNr, buildingID and characteristics. These are all single-valued and atomic. As we know of the definition of a weak entity type, we know that ROOM borrows buildingID from BUILDING to make up a key attribute type for ROOM. The borrowed attribute type buildingID is represented in our model by a dashed line. So, the key atttibute type of ROOM consists of

buildingID and roomID. The value of roomID is equal to 'roomNr.buildingID'. This makes the entity type ROOM unique. The attribute type characteristics gives certain characteristics of the room. Examples of these characteristics are double glass, isolated …

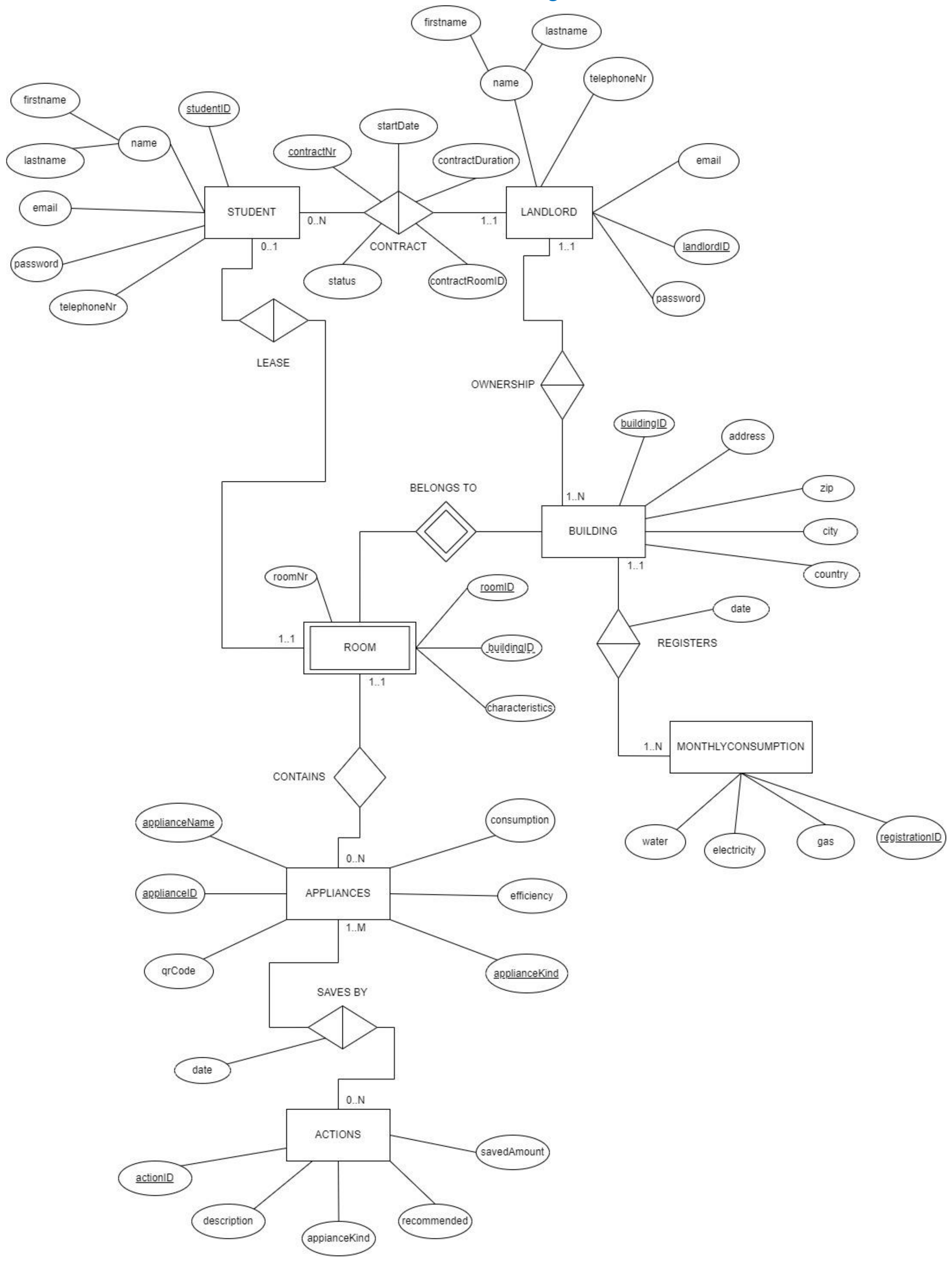
The next entity type we will discuss is MONTHLY CONSUMPTION. This entity type keeps track of the energy consumption of water, electricity, and gas of each room. This is registered each first day of the month for each room. This relationship type is given by REGISTERS. This relationship type is defined as stated in the above paragraph, whereby MONTHLY CONSUMPTION is registered for 1 to N ROOM.
The attribute types of MONTHLY CONSUMPTION are water, electricity, gas and registrationID. These are all atomic and single-valued attribute types. The key attribute type of this model is registrationID. This makes MONTHLY CONSUMPTION unique. The other attribute types need no further explanation.

The sixth entity type is APPLIANCES which are all appliances a student has in his/her room.
This entity type has two relationship types.
The first one we already discussed if we talked about the entity type ROOM. This is the relationship type CONTAINS, whereby APPLIANCES can exist on just one ROOM.
The second relationship type is SAVES BY between APPLIANCES and ACTIONS, whereby APPLIANCES can have 0 to N ACTIONS. This relationship type has also an attribute type, namely date. This attribute type is atomic and single-values and it tells us when the action is executed.
The attribute types of APPLIANCES are applianceID, consumption, efficiency, qrCode, applianceName and applianceKind. The key attribute type which makes this entity type unique is applianceID. This gives an ID to every appliance. ApplianceKind also needs some further explanation. Appliances can work on electricity, water, or gas. ApplianceKind assigns each appliance to one of these three. The attribute type consumption gives the amount of energy an appliance consumes. The qrCcode is a link where you go to a database where you can find information about the appliance. The attribute type efficiency gives the efficiency of the appliance. the last attribute type (applianceName) needs no further explanation.

The last entity type we will discuss is the entity type ACTIONS. This entity type describes the energy conservation actions an appliance can execute to reduce his energy consumption. This entity type has one relationship type SAVES BY who is already discussed above, whereby each ACTION can be used for 0 to M APPLIANCES.
ACTIONS has three attribute types, namely actionsID, recommended, savedAmount, applianceKind and description. These are all atomic and single-valued. The key attribute type that provides the uniqueness of the entity type is actionID. The attribute type description gives a description of what the action does. The attribute type recommend gives a ranking of the actions. The higher the action stands in the ranking, the more popular it is. The attribute type applianceKind are used to divide the different actions over the different types of energy sources (water, electricity, gas) The last attribute type (savedAmount) are the amounts that are saves because of the actions.

(E)ER diagram

Because we don't use specialisation, generalisation, categorisation and aggregation, the diagram below is a ER diagram



Assumptions, explanation and limitations of our model

It is possible to continuously improve and expand the program. However, we must stay within the deadline. In this section, we describe the assumptions made, an explanation and limitations of our model.

Firstly, we make some assumptions regarding the student. Only one student can stay in each room. It is therefore not permitted to live in a room as a couple or with a few friends. Next, it is not possible for students to sublet their room to other students. Next, we assume that the student who rents the room is also the contact person. In other words, we are not adding a guardian of the student who will be contacted by the landlord. So we assume that the students are of age. This can be seen as a limitation of our program. We could improve it in the future by making it possible to add contact persons of student.

Secondly, we make some assumptions about the landlord. There is only one landlord per room. So ownership cannot be shared. The landlord must register the consumption of a room on the first day of each month. In our program, the landlord can choose the date on which he enters the month's consumption.

Thirdly, we assume that a date is entered as day/month/year.

Finally, we make some assumptions about energy consumption. An appliance can work on three kinds of energy source. An appliance can work on electricity, gas, and water. The attribute type “applianceKind” will assign each appliance to one of these three.
Electricity consumption is measured in kWh. Gas consumption is measured in m3. Water consumption is measured in m3.
The landlord can compare the consumption of a certain month with some average values. The average electricity consumption is 153 kWh per month. The average gas consumption is 78 m3 per month. The average water consumption per month is 4 m3.

Our programme also has the following limitation:

In our database you can see that ACTIONS and APPLIANCES are not connected via the relationship SAVESBY. We know that theoretically they should be, but we did not do this for the following reason: an appliance can use different actions every month. So because of the uniqueness constraint that a primary key must satisfy, we cannot make ApplianceID and actionID a primary key. we would like to add that APPLIANCES and ACTIONS are connected by methods in Intellij but not in our database.

DATABASE DESIGN AND IMPLEMENTATION

In this part you can find the relational model and normalization steps. Next to this you also find the logical model, an overview of the database tables and relations, referential integrity rules. We conclude this part with a comparison between relational and conceptual model

Relational model and normalization steps

In the lines below we discuss the relational model. This model is constructed based upon the conceptual model (see above)

Relational model

STUDENT (studentID, firstname, lastname, email, password, telephoneNr)

LANDLORD (landlordID, firstname, lastname, email, telephoneNr, password)

BUILDING (buildingID, country, city, address, zip)

ROOM (roomNr, roomID, *buildingID*, characteristics)

- buildingID: foreign key refers to buildingID in BUILDING, NOT NULL

APPLIANCES (applianceID, consumption, efficiency, qrCode, applianceName, applianceKind)

MONTHLYCONSUMPTION (registrationID, water, electricity, gas)

ACTIONS (actionID, executed, recommended, description)

SAVES_BY (*applianceID*, *actionID*, date)

- applianceID: foreign key refers to applianceID in APPLIANCES, NOT NULL
- actionID: foreign key refers to actionID in ACTIONS, NOT NULL

Normalisation steps

To construct out relational model correctly, we have to take normalization into account. The different steps are listed below.

First normal form 1 NF

The first normal form 1 NF states that each attribute type of a relation must be atomic and single-valued and for each relation a primary key is defined.

We have carried out the first normalization step with student and landlord:

- STUDENT

  STUDENT (<u>studentID</u>, name(firstname, lastname), email, password, telephoneNr)

  ↓ (composite not allowed)

  STUDENT (<u>studentID</u>, firstname, lastname, email, password, telephoneNr)

- LANDLORD

  LANDLORD (<u>landlordID</u>, name(firstname, lastname), email, telephoneNr, password)

  ↓ (composite not allowed)

  LANDLORD (<u>landlordID</u>, firstname, lastname, email, telephoneNr, password)

Further normalization steps are not necessary here as the model already satisfies these. Below we briefly explain why.

Second normal form 2 NF

A relation satisfies the second form if it satisfies the first form and if each non-key attribute A in relation R is fully dependent on any key of R. Since we do not work with composite primary keys, our model already satisfies the second form without further steps.

Third normal form 3 NF

A relation satisfies the third normal form if it satisfies the second normal form and if, in addition, all non-key attributes are independent of another non-key attribute. In other words, there may be no transitive dependence.

Boyce-Codd normal form BCNF

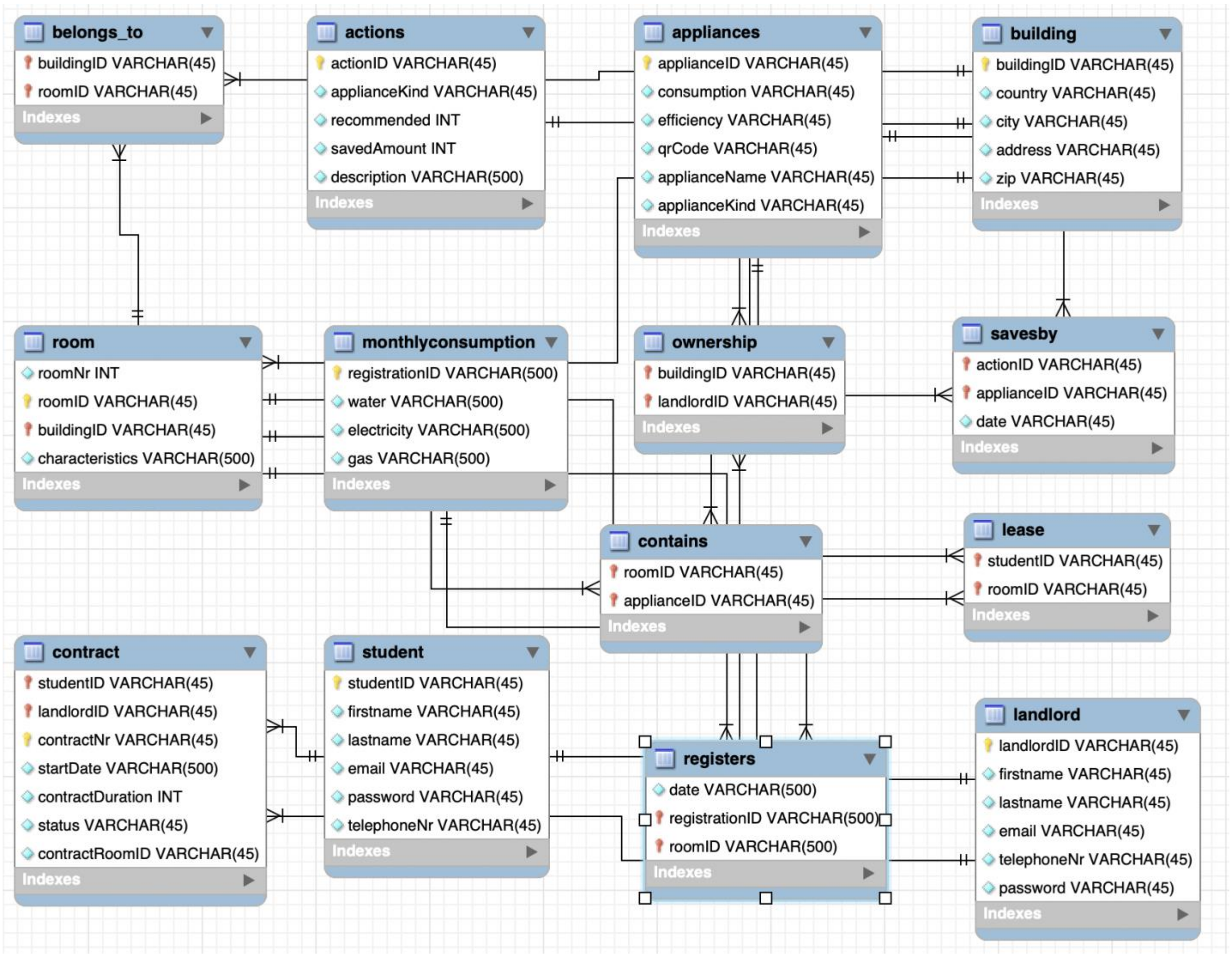
For a relation to be in BCNF, the third normal must already be satisfied and for any dependency A → B, A must not be a non-prime attribute if B is a prime attribute.

Fourth normal form 4 NF

This last form is also already fulfilled. This is because we are not dealing with multivalued dependencies.

Overview of database tables and referential integrity rules

Here, we provide an overview of the database tables and relationships. The referential integrity rules are also highlighted.



Used symbols

Yellow key 🔑

The yellow key is the primary key of the table. This value has to be a unique value and must comply with the **entity integrity constraint**. The entity integrity constraint states that attribute types that form the primary key must always comply with a NOT NULL constraint.

Blue diamond ◈

These values do not have to be unique but may never contain the value NULL.

Red key

The red key is a foreign key of the table. A foreign key must comply with the **referential integrity constraint**. This constraint states that a foreign key has the same domain as the primary key attribute types it refers to and either occurs as a value of the primary key or NULL if allowed by the minimum cardinality. For example, in the relationship SAVES_BY the foreign key applianceID refers to the primary key applianceID in APPLIANCE. The minimum cardinality is 1, so the not null constraint must be set to applianceID.

Comparison relational model and conceptual model

During the transition from the conceptual model to the relational model, there may be loss of semantics. So certain things who were evident in the conceptual model will be lost. The cause may be the cardinalities, the use of specialization/generalization and aggregation, or with ternary relationships. So, applied to the model under consideration, the aggregation and the cardinalities must be checked.

The cardinalities where we have to look at to detect loss of semantics, 1 to N and N to M relations. If the minimum of a cardinality of an entity type is one by one, then this can’t be enforced in the relational model. We can see that this sometimes occurs in our conceptual model.

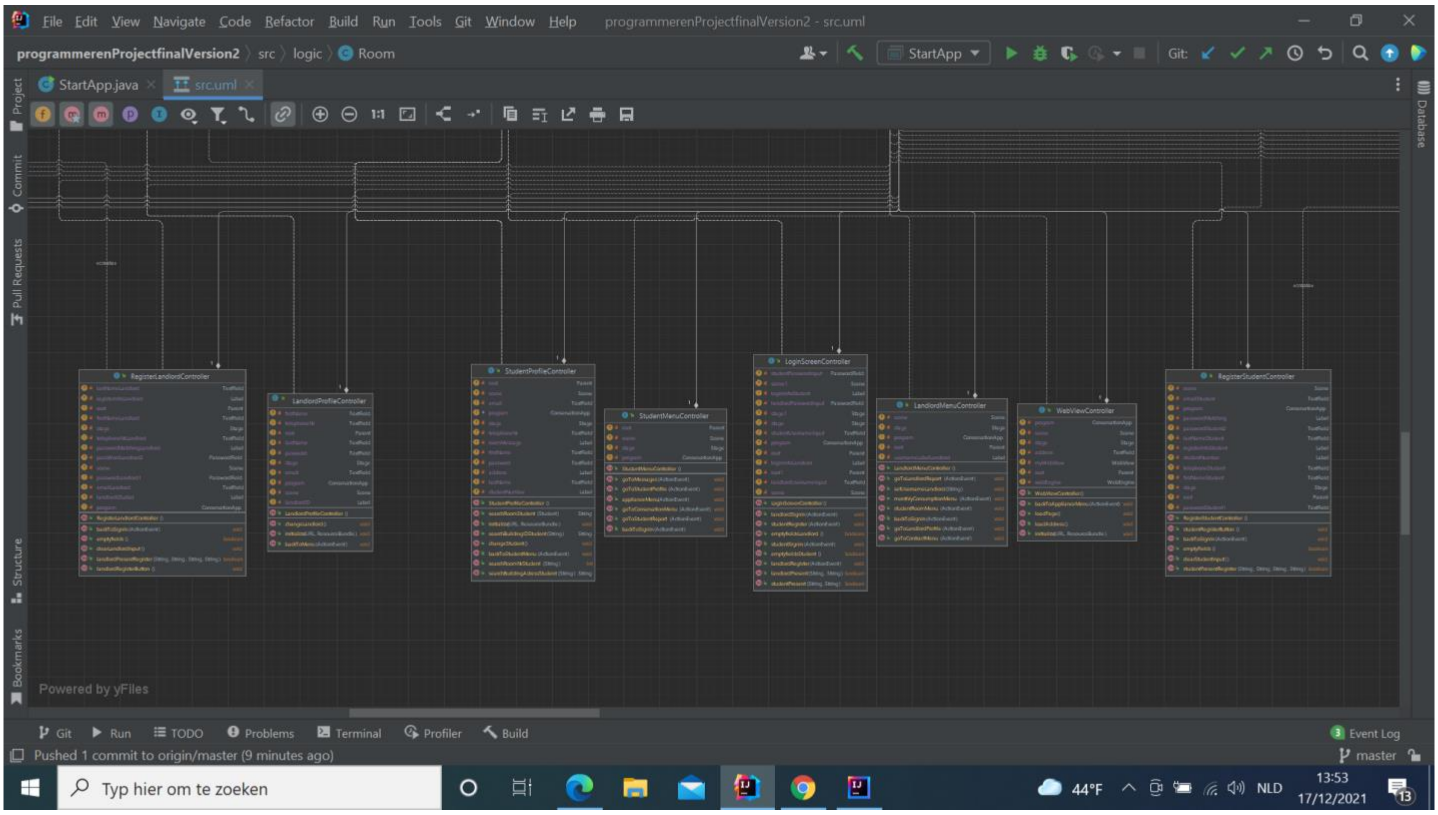
In addition, we have to look at the aggregation, of which we have two elements in our conceptual model, namely the name of landlord and student. The name contains several attribute types, namely firstname and lastname. Again, however, this does not present any problems since the landlord and student are clearly exposed in the relational model.

SYSTEM DESIGN AND IMPLEMENTATION

In this section, you will find a UML class diagram. Furthermore, aspects such as ‘Implementation’ and ‘Algorithmic support’ are discussed. This includes a clear description of our implementation, how the functional requirements have been implemented and how we have tackled this in our application.

UML class diagram for every module

For a full overview, see our UML class diagram in IntelliJ


programmerenProjectfinalVersion2 - src.uml
programmerenProjectfinalVersion2 › src › logic › Room
StartApp.java
src.uml
StartApp
RegisterLandlordController
LandlordProfileController
StudentProfileController
StudentMenuController
LoginScreenController
LandlordMenuController
WebViewController
RegisterStudentController
Powered by yFiles
Pushed 1 commit to origin/master (9 minutes ago)
Typ hier om te zoeken
13:53
17/12/2021

Implementation

When starting the application, the login screen opens, and a distinction is made between a student and a landlord. On the left side, the student must enter his/her student number and password. If the student is not yet in the database, he/she must register. On the right side, the landlord has to enter the same information and must also register, if he/she has not done so already. When the user is finally logged in, the content of the screen that follows depends on whether this person is a student or a landlord. Both the student and the landlord obtain a menu where they can choose which action to take. In what follows, these different actions and screens are discussed in more detail.

In the GUI, we have created 17 controllers and FXML classes and 1 main class. These classes are: AddContactPerson, AddContract, AddContract, AddStudentRoom, ApplianceMenu, EnergyConservationActions, LandlordMenu, LandlordProfile, LandlordProfile, LandlordReport, LoginScreen, Messages, RegisterEnergyConsumption, RegisterLandlord, RegisterStudent, StudentMenu, StudentProfile, StudentReport, WebView en de StartApp.

Login screen

We have decided to select a login screen called 'Login'. This GUI is divided into two sides, namely the 'Student login' and the 'Landlord login'. If the user is a student who is not yet registered, one can log in by clicking on the 'Register here (student)' label in the left corner of the ‘Student Login’. This takes the user to a register menu which allows the student to register with a first name, a last name, an email, a telephone number, and a password by entering these in the corresponding text fields. We have included a safety mechanism in our application by asking the user to fill his/her password twice. If the passwords do not match, a text in red will appear above the lower password text field with the message "The passwords do not match". In addition, we have ensured that every text field must be filled in, otherwise the message "Please fill in every field" will appear.

At the bottom, you will find three buttons: a ‘Sign in’ button, a ‘Register’ button and an ‘Empty’ button. The ‘Register’ button ensures that the data of a particular user is stored in the database and that the user creates an account in the application, while the ‘Sign in’ button returns the user to the 'Login menu’. First the register button must be pressed and only then the ‘Sign in’ button, otherwise the user has not created an account. The ‘Empty’ button is added in case of incorrect information being entered and deletes all entered data. Once registered, the student will receive a student number at the top of the menu, which he/she can use to log in to the 'Login' menu together with the chosen password. The user can also log in with his/her email. The same procedure applies to a user who wants to log in as a landlord, except that the landlord has a landlord ID/ email. In the case of a wrong combination of student number / landlord ID (email) and password in the GUI ‘LoginScreen’, a text in red will appear next to the ‘Sign in’ button with the message "This account does not match an account in the DB!". After logging in, the user is redirected to the 'StudentMenu' GUI, if one is a student, or the 'LandlordMenu' GUI, if one is a landlord. The user can always sign out again from these two screens, as there are 'Sign out' buttons provided.

Student

In the GUI 'StudentMenu', the student must select one of these three options: ‘Appliance menu’, ‘Energy conservation menu’ or ‘Energy conservation/ consumption report’.

Before discussing the actions behind these three options, we have added two links to the menu that the user can consult to find and change information. We have added a link in the left corner that refers to the student user's profile. This GUI, called 'StudentProfile', displays the student number, first name, last name, email, phone number and password. One can change his/her data by typing the new information in the text fields and then pressing the 'Change’ button. The room number and the address of the student room are also given. The other link 'Messages' in the GUI 'StudentMenu' indicates whether the student has a contract open. The landlord sends a contract proposal with all the details to the student, who can either accept it by pressing the 'Accept’ button or decline it by pressing the 'Decline’ button. If the student does not have an open contract with a landlord, the message “You have no open contract!” will appear at the top.

After selecting the ‘Appliance menu’ option, the user is directed to the GUI 'ApplianceMenu'. In 'ApplianceMenu', the user can add, remove, and modify new appliances from a room. The 'Add appliance' tab allows the student to add a new appliance to his/her room with the necessary information about the appliance. The window displays four text fields in which the name, the consumption, the efficiency and a QR code must be entered. The label ‘show’ in blue brings the user to the QR-code one enters. Finally, the user has to choose the basic facility, such as electricity, water or gas, on which the appliance operates. Once the 'Add appliance' button is pressed, the appliance is added to the user's room and the message “The appliance has been added to the database!” appears at the top, along with the ID of the specific appliance.

The 'Modify appliance' tab displays a list of all appliances that this user has already added to a room. The user can look in the list and click on the appliance he/she wants to delete or change. If the user wants to remove an appliance from his/her room, then one must push the ‘Delete’ button. If one wants to change some of the appliance's data, the student can do so by entering the new information in the appropriate text area. When the "Change" button is pressed, the information of the appliance will change. There is clear feedback when adding appliances or adjusting existing appliances. Here, too, you can return to the previous screen at any time using the ‘Back’ button.

In ‘Energy conservation menu’, the user can register energy conservation actions by selecting an appliance name and a day and next registering the energy conservation action. This action can be an old action or a new added action in the tab ‘Add conservation action. Here, we have made a distinction between electricity, water, and gas appliances. For example, in the 'Electricity appliance' tab, one must first choose the name of the appliance that works on electricity, then the day of consumption and an energy conservation action. The action is saved by pushing the ‘Submit’ button. The bar chart shows how many times the conservation action is already used. This way, the student can see which energy conservation action is popular among other students for the same appliances (recommendations). Here, too, the student can return to the previous screen at any time using the ‘Back’ button. In the tab ‘Add conservation action’, users have the possibility to add new actions by giving a description of the action, the saved amount and the kind of appliance. One can also delete an action in the listview by pushing on the ‘Delete’ button or change it when typing the new information in the text fields and push the ‘Add action’ button.

The third option, the student can choose, is to request a relevant report of his/her registered appliances and energy conservation actions. The GUI 'StudentReport' gives the user an overview of all his/her added appliances and consumptions in his/her room. If one then selects a date in the date picker and presses the 'show' button, one sees the actions that have been added before that date. This report focuses only on one student room. The next three tabs show a graph of the monthly consumption of every kind of appliance. For example, in the tab ‘Electricity graph’, the user needs to choose a month from the date picker and then push the button ‘Add month’. One needs to do this for every month he/she wants to get a visual overview from. At the bottom, one sees a path of all added months separated by an arrow. After adding all the months the user wants an overview from, one must push the button ‘Show graph’. This action leads to a line chart with a x-axis that shows the month and a y-axis that shows the consumption. Visually, the student sees three lines; the red line refers to the monthly consumption, the yellow line refers to the monthly conservation and the green line is the difference between the red and yellow line and is called the netto-consumption. If the user wants to remove the graph, he/she can do this by pushing the ‘Clear graph’ button. This explanation is the same in the tab's ‘Gas graph’ and ‘Water graph’.

Landlord

After signing in, a landlord will see the ‘Landlord’ window. Here, he/she will see an overview of all possible actions one can take as a landlord.

The ‘Student room and building menu’ option will direct the person to the GUI 'AddStudentRoom'. In this menu, the user will see four tabs called: ‘Add student room’, ‘Modify student room’, ‘Add building’, and ‘Modify building’. Before a landlord can add a room to a building, he must first add a building. The user is alerted by the fact that he/she must add a building ID to add a student room.

To start with, there is an 'Add student room' window. This window is used for adding a student room, which clearly follows from the name of the tab. The window will display 3 text fields, in which one must insert a self-chosen room number, a building ID and the characteristics of the new room. Finally, you must click on the 'Add student room' button. If he/she clicks on it, the room will be added to the building and the room ID of the specific room will appear at the top. However, this window also contains a few controls. The program will check whether all text boxes have been filled in. If not, a little text will appear to remind the user that all text boxes must be filled in. If saving the new room was successful, the person will also see an appropriate text.

Next to the tab ‘Add student room’ in the menu, one can click on the tab ‘Modify student room’. The window that is now displayed has the same structure as the ‘Modify appliance’ window. The window contains a listview with all student rooms that have been added via the 'Add student room' window. If one clicks on a room in the listview, all data of the clicked room will be loaded in the text boxes. After this, one can make the adjustments one wishes and finally press the button ‘Save student room’ to confirm. If the student clicks on a room in the listview and then clicks on remove, the room is permanently deleted.

In addition to the 'Student room' tabs on the 'Student room and building menu’, there are also windows about buildings. Here too, you can choose between add and modify. It is based on the same principle as creating and modifying a student room, which we think increases the user-friendliness. The only difference is logically the things that are requested. For example, ‘Add building’ asks for the address, country, city, and ZIP. The same check is built in here, namely checking whether everything is filled in. The ID of the building is automatically generated by the program itself, making double IDs impossible. Regarding the ‘Modify building’ screen, four different elements of a building can be modified. These are the address, country, city, and ZIP of the building. We have made this editable to correct any typing errors made when creating the building. The user can also remove a building from this screen by clicking on the 'Remove' button.

The second option that the landlord can choose from the landlord menu is the ‘Contract menu’. The purpose of this menu is to give the landlord the opportunity to present a contract proposal to a particular student. As previously explained in the 'Student menu’, the student user can accept or decline this contract proposal. How does a landlord add a contract? This can be done by filling in the following text fields in the 'Add a contract' window: student ID of the student who rents the room, the room ID of the rented room, indicating the day on which the contract was concluded and the duration of the contract. The contract with all its details is then sent to the student by pressing the 'Send to student' button. To give the user an overview of the contracts he/she has sent to the students, we have added a 'Contract status' tab to the ‘Contract menu’. In the listview, the landlord can see the contracts that have been sent. If one wants to see the details of the contract, the user must click on a contract and the details will appear. In this window, it is also possible for the landlord to cancel a contract with immediate effect. He/she can do this by putting the contract number in the text field and pressing the 'Terminate contract and lease' button.

The window also contains a few controls. The program will check whether all text boxes have been filled in. If not, a little text will appear to remind the user that all text boxes must be filled in. In addition, the program also checks that the Student ID and room ID that are provided are not used by another contract. In this case, too, a small text message appears informing the user that the student ID or room ID are already in use. It is also not possible to draw up a contract for a student or room that does not exist, here too, a message will appear in the program. If the saving of the new contract was successful, the person will also see an appropriate text message.

The Energy consumption menu is the third option a user can click on if the landlord wants to register the energy consumption for a room. In this menu, one will find the same ‘Add’ and ‘Change’ window as with the student room and contract but applied to energy consumption. In the ‘Add consumption’ screen, the landlord is asked to indicate the date of consumption, the room ID and the values of electricity, water, and gas. When the user presses the ‘Submit’ button, the information in the completed text fields is saved and a certain message appears if the submission was successful or not. The 'Change consumption' tab, like most ‘Change’ windows, contains a listview from which the user can select a registration and the additional data is placed in the text fields. The user then has the choice to delete or change the details of the added consumption. Just like all other derived menus, each tab contains a 'Back' button back to the original 'Appliance menu’.

The fourth and last option is the ‘landlord report’ option. In this window, the landlord can get an overview of the monthly energy consumption per month and per room. In the date selector, the landlord needs to select a month for which he/she wants to see the energy consumption. When the button ‘show report’ is pressed, a bar chart appears that indicates the electricity, water and gas consumption of the selected month. The user can also add several months and compare the consumption per month and per room. Furthermore, one can press the ‘clear report’ button, the program will reset the page and an empty bar chart will appear again. The last button that can be indicated is the ‘back’ button whereby the landlord menu reappears.

Algorithmic support

Database layer

In the database layer we created 14 classes, namely: DBActions, DBAppliance, DBBelongsTo, DBBuilding, DBContains, DBContract, DBLandlord, DBLease, DBMonthlyConsumption, DBOwnership, DBRegisters, DBRoom, DBSavesBy and DBStudent.

Each DB class makes the connection with the database in mySQL Workbench and our program in IntelliJ. In this way, the programme can communicate with the database.

The DBAppliance class contains several methods that are used in our programme. These methods allow the user to add appliances to the database, to change the properties of the appliances already in the database and to remove devices from the database. Finally, there is also a method that reads the devices that belong to the database. Then, this method returns all devices that are in the database.
The following classes also use one or more of these methods. DB Building uses the same methods as the DBAppliance but personalized on the building class. The DBOwnership class is one of the classes that only has a read and add method and no change or remove. This seemed less useful to us, so we left it out.

GUI layer

JavaFX was used for the GUI of the application. The windows were built with Scenebuilder. Each panel that is displayed is represented in the GUI-package by an FXML file and an associated controller class. In the UML diagram, all controller classes are listed with their methods. FXML variables, such as text fields, buttons and others, are not included in the diagram in order to keep an overview, as they are of little semantic value.

We will now discuss why we have chosen different layouts for the different menus.

First, there are the login screens. Here, we have chosen text fields to enter your ID and password fields to enter your password. This seems logical to us. When you log in as a student, you are taken to the student menu. There we chose radio buttons for the simple reason that you can indicate an option quickly and easily.

When you select the first option (Appliance menu), you can see that we have chosen a clean layout. This menu consists of several tabs. We have chosen to do this in order to keep the add and modify functions apart and avoid confusion. Both tabs have text fields. At the bottom is a choice box where you can select electricity, water and gas. We use a choice box in our programme when there are a limited number of options to choose from and we do not need to change anything. This is the case here. There is a list view on the ‘Modify appliance’ tab. We work with a list view if we want to get an overview of the array list of objects that belong to the database. We can then select an object from it and modify or delete it.

The second option in the ‘Student menu’ is the ‘Energy conservation menu’. The special feature of the layout in the various tabs is obviously the datepicker, used to select a date. This seemed to us the easiest and most beautiful method to select a date. Furthermore, choice boxes were used for the same reason as mentioned above. There is also a barchart in the first three tabs. In the barchart you can see on the x-axis the different applianceIDs and on the y-axis the number of times a conservation action has been used for this appliance. On this basis you can see which conservation actions are popular for a particular appliance. The last tab contains a listview for the reason described above.

We have chosen a layout that we think makes it look clear. Again, we have chosen different tabs.

The first tab contains the appliances with the corresponding monthly consumptions. The third and fourth columns describe the different conservation actions for a certain applianceID and the amount you save with that particular action. We chose a listview because we find it more convenient than a choicebox and the lists are updated several times. In the other tabs, we chose a line chart because there are three lines displayed and the line chart gives a nice view of the interaction between the different lines.

Radio buttons have also been chosen for the landlord menu, for the same reason as for the student menu. In the landlord menu, there are five possible options to choose from.

The first option is the 'Student room and building' menu. In the 'Add' option, text fields have been chosen, which seems logical to us. In the 'Modify' option, we have chosen listviews for the reason described above.

The second option is the 'Contract menu'. We chose a datepicker for the start date because this seemed the perfect option. A date can be chosen quickly and easily. Furthermore, a listview is used again where all contracts in the database are shown and where they can also be deleted. Contracts cannot be changed, because if a contract is signed by both parties it cannot be changed.

The third option, the 'Consumption menu', has the same structure as option two and therefore requires no further explanation.

The last option is the conservation report. A bar chart has been chosen here because it gives a clear comparison between energy consumption in different months.

Logical layer

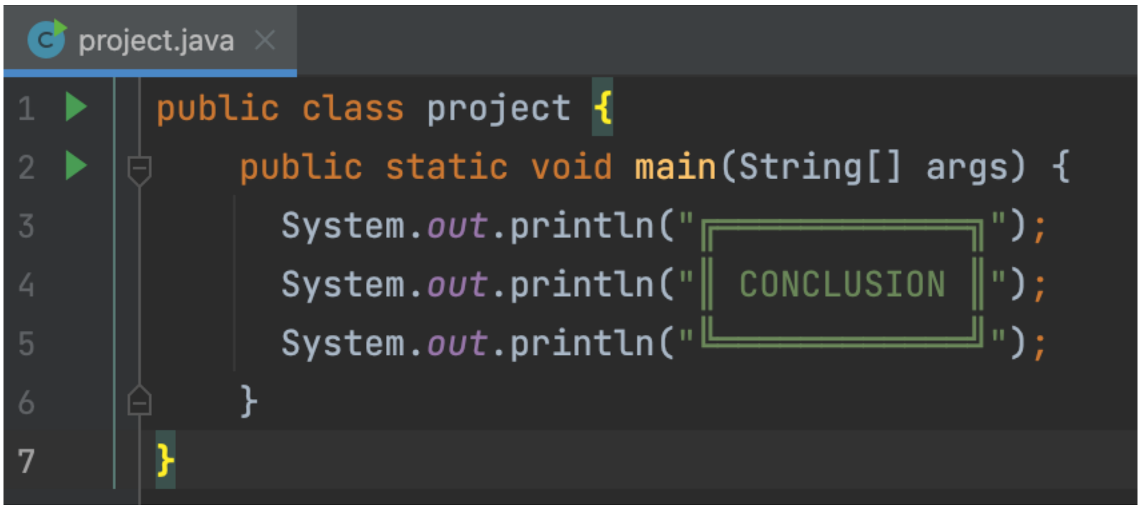
In the logic layer we have created 16 classes, namely: Action, Appliance, BelongsTo, Building, ConservationApp, Contains, Contract, Landlord, Lease, MonthlyConsumption, OpenContract, Ownership, Registers, Room, SavesBy and Student.

The class Appliance consists, like the classes BelongsTo, Building, Contains, Contract, Landlord, Lease, MonthlyConsumption, OpenContract, Ownerships, Registers, Room and Student, of a constructor, getters and setters and a toString method. They also create and declare instance variables in the constructor.

The ConservationApp is an upper class, which mainly contains the array lists by which the GUI layer works with. These arraylists are filled with data from the database, and are updated as changes are made. The methods that fill the array lists are also in this class. Of course, this class also contains a constructor and the necessary getters and setters. Furthermore this class contains a number of methods that, for example, return all roomID's, ApplianceID's, ... that belong to the database. The class also contains the toMonth and the toYear method. These methods require a date to be included. This method then returns the month and year respectively of the given date.

CONCLUSION

In this section, you find opportunities of our projects and a general conclusion.



```java
public class project {
    public static void main(String[] args) {
        System.out.println("╔════════════════╗");
        System.out.println("║   CONCLUSION   ║");
        System.out.println("╚════════════════╝");
    }
}
```

General limitations and possibilities

Possibilities

Password security

To make our project even more secure, the password control could be made better. We had no time to do the implementation and the usage of this more secure system, but we like to share our ideas about it with you.

If we look at the protection of our system we can say that it is already good, but to make it even better we can make use of Regex. Now instead of being in direct contact with the database, our info of the students and landlords is being loaded into local arraylists. This prevents the easy ways to hack into the database with sql statements that are typed into the password field, because these local arraylists in Java code can't be influenced by sql code. Other ways to do this would be to prevent a person, who wants to sign in, to type in sql statements via Regex.

Report of landlord per building

We now represented the report of the landlord per room. We also had the idea to represent the consumption per building so the landlord should have an overview per building. This should give the landlord also the opportunity to compare energy consumption between different buildings instead of only between the rooms.

No possibility to log in if password is forgotten

In our system there is only one possibility to change a password, once you are already logged in. However, if a user forgot his or her password, logging in or changing password is impossible. A solution would be a "forgot password" button, where an email is sent with a link to the given email address that gives the opportunity to change your password.

The attribute type characteristics of room

This attribute type just describes the characteristics of a room which are related to the saving of energy (e.g. isolation, double glass …). The initial purpose of this attribute type was to take these characteristics also into account when calculating the monthly consumption per room. Now it just gives additional information about the room.

General conclusion

The best way we can describe our project experience is by the quote of Richard Branson: “the best way of learning about anything is by doing”. This project was a big challenge for all of us. The fact that we were allowed to work independently and discover things was very enriching. It gave us the opportunity and the experience to look into the world of programming and databases and it taught us how to work as a team.

Working with GUIs was an interesting new feature. After all, we had never looked at this in depth before. It’s so beautiful and shocking which things you can make with this tool.

Especially in the field of database systems, this seemed a very useful project, as it made the material, we had seen much more concrete. It greatly facilitated the understanding of the abstract material.