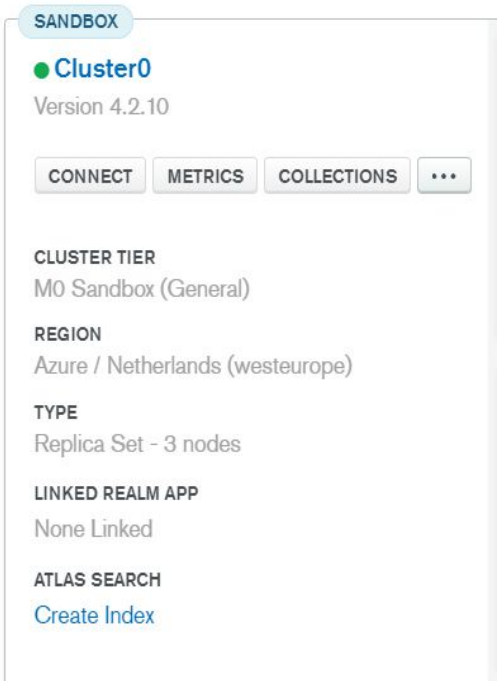
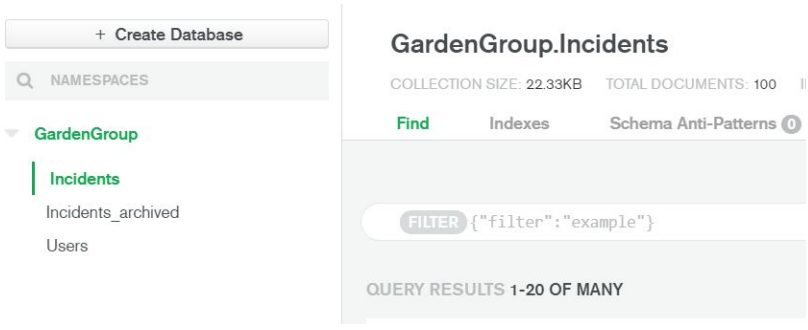
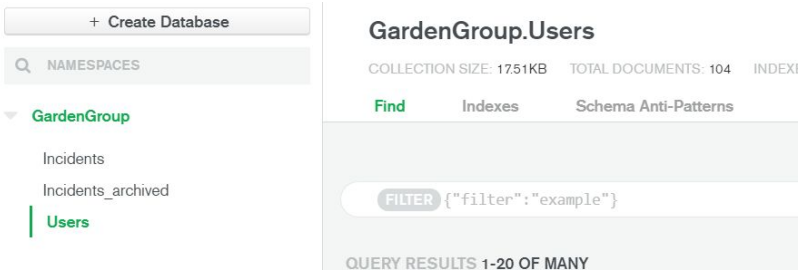
Group: IT2B-PG3, Dewi Becu, Mayron van Leenden, Sjoerd Klooster, Tamanna Abbas

A. MongoDB was used the as database
B. C# was used, with Visual Studio
C. Design patterns used:
    a. Singleton pattern for the database, and the logged in user
D. Authentication made by *Tamanna*. For: end-users & service desk employees
E. General clustered queries were made by *Mayron*
F. Implemented user flow the client wanted
G. Aesthetic design for better UX
H. CRUD functionality implemented by *Sjoerd*
    - Service-desk employee can do all CRUD functionalities
    - End-users can only Create new tickets
I. Current tickets list implemented by *Dewi*
    - Service-desk employees can see all tickets created
    - End-users can only see their own tickets
J. MongoDB database was filled with: 2 collections, each with 100 documents







K. Everyone contributed, as listed above

Note: the reason why end-users have less access and authorization is due to security.

ERD, EER and UML were never used, since MongoDB is a non-relational database, and these diagrams would not have helped in creating the database. Thus we have structured our data to the fields of a user and an incident ticket, the same as in the source code.

Dewi Becu 656552

I was responsible for part. I (*The application has the functionality to show a dashboard of current tickets (personalized to the roles of 'users' and 'employees'*)).

When the person logs in, the user is saved in the class as a private field. This is so that the role of the user can be determined, and hence show the proper features for either an end-user or service desk employee. Then, the tickets at first are automatically gathered from the database while sorted by ID and stored locally, and then displayed in a listview with method **DisplayAllTickets().**

This listview shows: ID, subject, person who reported the incident, date it was reported, priority. Then if you click a row in the listview, all the details are shown in the text boxes. This includes: The above details, the deadline for the incident, the incident types, if the incident is solved

The listview is a clear overview of everything down in the application, and one of the center pieces of the purpose of this application. Thus, this is one of my contributions to the implementation of this project. In the code source, some of my comments show what I have written.

Furthermore, without my contribution, the application would not work/fulfill its purpose, because then a person could not see the list of tickets, and thus could not delete or update tickets. The purpose of this application is to record incidents and eventually solve them, so seeing the tickets is crucial. I have also edited and updated the UI so that it looks coherent and presentable for the user. And so that there is a better UX.

My additional functionality was the following: "*Sorting a ticket list: be able to sort the ticket list based on PRIORITY LEVEL of the tickets*."

My implementation of this feature went to the database layer. I made some mongoDB queries that would fetch the collection of documents as a sorted list to the application.

Thus, a drop-down menu was put as a UI (UserControlls.CurrentTickets), and when a box was selected, the method "**comboBoxSortBy_SelectedIndexChanged()**" from the TicketService (in GardenGroupLogic) would be calls "**ChangedListSort()".** This method uses a switch case to find which option was picked, and a sorted list is collected from the database through a specialized sorting mongoDB query. **DisplayAllTickets()** is called to display the changed tickets.

I have added other ways to sort the lists (reported date, deadline, and putting the unsolved cases on top). And since end-users only gather lists for them on tickets, I made similar DAO methods in the same classes.

Mayron van Leenden (560859)

Group work

I was responsible for the DAO with the CRUD functionality for Tickets and Users by creating MongoDB Queries. My responsibility was also to make sure the software connects with MongoDB. In order to make that connection, I created a separate class called MongoDB. In order to make sure the entire app had only one connection to the database, we decided to use Singleton Pattern. I also created the Models as well with some MongoDB Annotations. I think without the connection to MongoDB and the DAO to fetch and save data, the application would not work.

Individual work

My additional functionality was to archive all tickets that are older than 2 years.

Sjoerd Klooster 654848

I was in charge of creating the Create Update and Delete functions of the GarderGroup Ticket application. This is crucial to the application otherwise the users would not be able to create or update any tickets preventing the application from fulfilling its purpose. Without my contributions the application would need a database expert for every ticket created which is very prone to mistakes.

What I created

I created 2 separate User Controls Which panels, one for the ticket creation and one for the ticket updating. I chose User Controls because this allows all the relevant buttons and inputs to be separate from the rest of the application making for easier use because the relevant concerns are separated but also allows repeated use of the panel if that would be needed. The rest is pretty self explanatory, I added all the relevant fields that were wanted for each ticket as input for both panels and added update and create buttons.
For the delete functionality I added a button to Dewi's panel where you can overlook all the tickets.
Furthermore I removed the buttons that update and delete the tickets and removed the button that allows you to select the user that is reporting the issue and made it the logged in user by default.

Individual functionality

The Individual functionality I have chosen was to allow the ticket to be assigned to another user, this was one of the pre-approved functionalities.
I implemented this through creating another User Control which allows you to select a user from a list of all users. I thought it would be nice to make it multipurpose and allow it to work with selecting the user when creating it as well so I made the callback to an interface called UserSelect and used that in both the new ticket panel and the update ticket panel. This UserSelect is now also useful for anywhere in the application that a user would need to be selected if more functionalities would be added in the future.

Tamanna Abbas 648956
Part I worked on:I created the login/authentication part in the application and as an extra individual functionality I worked on the forget-password functionality.

Description about the login system:

I created the login system/authentication of users by having a windows forms panel which contains two input boxes for the user i.e. one for the username (which in this case is the email that the user used registered with) and the other one is for password.The credentials are then cross-checked with the database and depending on the state , access is granted (depending on the type of user) or denied.A user being a service desk employee is able to perform every function provided but for the end users the updation or deletion of tickets is not possible. With the login form there comes an additional option of "Remember Me" which gives the privilege to remember the credentials for next time.

Description about the individual part:

The individual functionality was about the forget-password. For this purpose,a method SendEmail() which gets in the user's email as a parameter.The method also call to other methods i.e. GetNewPassword() and ResetPassword().The GetNewPassword() is a method used to generate a valid random password for the user.The ResetPassword() resets/updates the password in the database.This method further uses a method UpdateDocumentbyEmail() which takes in the email and password as parameters.The email to filter and the password to change.Further,I used smtp client to set up a connection with a client which was hosted out of "smtp gmail". A variation was made in the functionality due to some factors and that it's more reliable.

Important Note:

To test the functionality I created two real gmail accounts for two of the users,so that I could actually send emails.
I am providing you with the credentials , in case, you might want to check if it really works..

USER1:
Email: ***williamkMckay.89@gmail.com***
Password: **William@89**

USER2:
Email: ***mildred.whippel@gmail.com***
Password: **mildred@123**

The emails can be found in the spam folder.