|  |  |
| --- | --- |
| A picture of a winding road and trees  Garden group  NoSQL Project | Group 2    Petros Simonyan 69815 Anthony Shoshi Gomes 695220 Stepan Shupletsov 697570 Maike Pierick 699725  IT2A |

Contents

[1. Contribution from Anthony 2](#_Toc150625070)

[2. Contribution from Petros 4](#_Toc150625071)

[3. Contribution from Stepan 5](#_Toc150625072)

[4. Contribution from Maike 7](#_Toc150625073)

[Rubric A + J 7](#_Toc150625074)

[Rubric F 7](#_Toc150625075)

[Rubric G 7](#_Toc150625076)

[Rubric H 7](#_Toc150625077)

[Rubric I 7](#_Toc150625078)

[Rubric K + L 7](#_Toc150625079)

[Rubric M + N 7](#_Toc150625080)

[Rubric O 8](#_Toc150625081)

# Contribution from Anthony

In our collaborative project, I actively contributed to various aspects of the application's development, ensuring its functionality and usability. My contributions can be outlined as follows:

**1. MongoDB Cluster Setup:**

* I played a pivotal role in setting up the MongoDB cluster, establishing a strong foundation for data storage and retrieval.
* This laid the groundwork for the efficient management of data within our application.

**2. Database Abstraction:**

* To streamline database operations, I designed a database abstraction layer.
* This layer includes a base database class, enabling dynamic collection management without the need for static collection name definitions.

**3. Implementation of Login Functionality:**

* I was responsible for the development of the application's login functionality.
* This involved:
* Creating an Employee model with relevant properties for user authentication.
* Designing an intuitive and user-friendly login form.
* Implementing the underlying logic for user authentication and login.

**4. Ticket Closure Feature:**

* Another key contribution was the implementation of a ticket closure feature.
* Service employees can now effortlessly close tickets from the list with just one click.
* This enhancement significantly improved the service desk's workflow efficiency.

**5. Web API Development:**

* As an additional task, I developed a robust RESTful Web API for our application.
* This API provides essential endpoints, including GET, POST, PUT, and DELETE queries.
* It ensures the application's readiness for future integration with mobile applications and microservices, adding a layer of flexibility and extensibility to our project.

**Additional Task: TheGardenGroupAPI.zip**

* To access my additional task, you can find the project folder named "**TheGardenGroupAPI.zip**" within the project repository.
* After unzipping the folder, you can open it using the Visual Studio IDE.
* Simply click the "Run" button within the IDE, and it will launch the application in a web browser.

My contributions to the project have been geared towards enhancing the overall functionality, user experience, and potential for future expansion. I believe these efforts have played a significant role in achieving the project's objectives. My work has not only contributed to the core functionality but also ensured our application's scalability and readiness for future developments.

# Contribution from Petros

* Created the queries for joining employees and tickets, thus displaying the Service desk overview. In addition to that the queries different who is which by matching checking the role of the logged in user.

The joined of tickets displays the following function if an employee works for the service desk, he gets all the tickets that are being done and are not past the deadline, they also get colored. In Addition to that if the service desk employee click on the dashboard he gets all the tickets that are past deadline for all employees and the tickets that are being done.

If an employee is a regular employee, he gets his own tickets that are not past deadline.

If and regular employee goes to the dashboard, he gets his all the matching tickets that are past the deadline and the tickets he is currently doing.

* Created the pi chart and visualization of how many tickets are done by the employee and gotten all the past tickets that if they are not resolved or past deadline, they go automatically to past deadline.
* Created the forget password (extra functionality) for resetting the password, you enter an email that is valid a code gets sent by the SMTP provider and you enter the code if the code is valid, you create a new password that gets hashed and sent to the database.

(For the person checking the 'forget password' functionality, please email me so I can add your email to the database. For now, I have only added Dan's Inholland email.)

* Create the MVC pattern for the ticket overview.

For the logic of how the tickets are being displayed we used one – many, this was the most logical reasoning to store tickets and this is how the display method is working by getting the employee and retrieving all their tickets

A screenshot of a computer

Description automatically generated

# Contribution from Stepan

My contribution was to make a working form for creating a new ticket from that is used not by service desk employee and to make a form for adding a new employee, which is accessible only for Service desk Employee.

**Regarding to Creating Ticket form:**

**In Ticket class:**

I adjusted some properties to be able to set, such as IncidentType, Priority and Status.

**In CreateTicketView:**

Firstly, I made a second method of ReadUserInput, addTicketToDatabase like what Maika did, but with several changes because I was making create ticket form for non-service desk employee and structure of the forms is a bit different. Also, a second constructor and second PopulateComboBoxes() were made. Later I adjusted all ReadUserInput, and addTickettoDatabase methods. I combine me with Maika methods to increase code clarity.

Because ticket in my case is made by just an employee, then ReportedBy field in the ticket must be defined automatically based on who was logged to send the full information to database, this was done by me. We need this because created tickets are later visible on ticket view.

**In UserInterface class:**

I implemented a method that defines to which form of creating ticket the system should open based on who is logged and functionality for opening, closing and submitting a form.

**Regarding to Adding Employee form:**

**In Employee class:**

I adjusted UserType property.

**I wrote necessary to DAL, Logic layer to support this feature.**

**I made CreateEmployeeView** class that is a part of a User Interface for creating new Employee records. It appears to be a view component responsible for interacting with users, collecting input, and communicating with the Employee Controller. I made that password is hashed using existing Anthony code (in addEmployeeToDatabase() method in CreateEmployeeView class).

**In UserInterface class:**

I made a cleaning method for panel of adding employee.

I made that only service desk employee can add a new employee.

Functionality for opening, closing and submitting a form.

In general, I was also discussing what data types we will have in our database and its structure.

**As an additional feature:**

I made a new Employee type Incident Employee who can manage tickets with new status “SentToIncidentManagement” and made improvements in permission of reading and editing information between user types. For instance, a Service Desk Employee understands that he cannot resolve an incident, so he sends it to the Incident Management Department. This reflects on the results of the statistics on the dashboard screen. When logged user is Incident Employee on the ticket view screen, he sees only tickets with status “SentToIncidentManagement” and no others. The ticket can get this status when service employee edits ticket information. After that Service Employee can only edit Incident Type and Description but not Status, Priority and Deadline and only Incident Employee can adjust ticket information, however, Incident Employee cannot create new employees. Moreover, in the previous version the general Employee could not edit tickets, however now he is able to change Incident type and Description.

**In UserInterface class:**

LoadAndUpdateView() and ticketView\_DoubleClick() methods were adjusted; DisplayTicketBasedOnUserType(List<Ticket> tickets) was made.

**In ChangeTicketView class:**

Two new fields were added:

private UserType currentUserType

private bool canChangeDeadlineStatusPriority;

The constructor was adjusted and methods ConfigureControlsBasedOnUserType() and CanChangeDeadlineStatusPriority(Ticket ticket) were made. ConfigureControlsBasedOnUserType() method limits the editing ticket based on its status, while CanChangeDeadlineStatusPriority(Ticket ticket) is a bool method that is used to determine the status of the ticket.

**In TicketDAO class the same queries are used for listview of Incident Employee**. **Sorting** of tickets happen **within DisplayTicketsForIncidentManagment(List<Ticket> tickets)** method in **TicketView class.**

**To reduce the duplicate code in TicketView class** DisplayTickets(List<Ticket> tickets) was adjusted and private ListViewItem CreateTicketListViewItem(Ticket ticket) method was made.

Also, in TicketView class Colours method was adjusted to highlight new type of tickets, and the PiChartTickets method was changed for opportunity to see new statistic information.

# Contribution from Maike

## Rubric A + J

I have set up and filled the Ticket Collection in the MongoDB database.

## Rubric F

I followed the assignment's design as closely as possible for my parts.

## Rubric G

Enhanced the UI, making the toolbar more appealing and aesthetically pleasing.

## Rubric H

I developed CRUD functionality for ServiceDesk employees, building upon Petros' existing ticket overview. This involved creating panels for ticket addition and updates, as well as implementing a delete button that appears when a ServiceDesk employee selects a ticket. I also wrote the necessary DAL layer methods, service layer, and UI components to support these features.

## Rubric I

I modified Petro’s code to display all tickets for ServiceDesk employees.

## Rubric K + L

The following were part of my contribution:

* CRUD functionality for service desk employees
* Refactoring of the switching between panels to avoid duplicate code
* Improvement of menu strip
* Filling & setting up ticket collection

## Rubric M + N

This was the ERD I worked with, we chose not to embed the tickets inside of the employee as this could have resulted in hitting the maximum size of a document over time. We did not embed the employee inside of the ticket, as it is necessary to be able to access the employee object on its own.

A screenshot of a graph

Description automatically generated

I decided to make a separate panel for editing an existing ticket instead of adding the functionality to the ticket overview. I made that decision because the editing functionality would have overcrowded the overview panel, and it is cleaner and clearer this way. I left the delete button on the overview panel to make it easy and to avoid unnecessary clicks for the task.

## Rubric O

For the additional functionality I chose to add a search bar where users can search based on words that occur in the ticket subject and content. This also supports the use of AND OR search and sorts the results with the most recent at the top of the list.