# Project overview

This project is a simulation of an Airport management system, where admins can access the internal data of the system and customers are concerned only with the outer part, like reserving tickets and booking flights. Any changes to the system are automatically flushed to the database correctly using our calls to the DBMS.

Admins can edit their personal information. The admins can add, modify or delete aircrafts. They can set their monitored flights, see reserved tickets and erase them. Finally, they can register admins (a new admin cannot register without the approval of another admin.) Customers can edit their personal information, book flights for them. They can also book flights for their relatives not only for one person. New customers can sign up but with unique usernames.

For our current purposes, we only set the application to work on the localhost. As an added feature, if the database was not found, one is automatically created and a default admin is added. The admins are expected to change the default password of this admin for security purposes.

This project illustrates some of the potential of databases in software development. The project uses the database to store its data and manipulate it. This project was created for educational purposes only.

# GitHub Repository

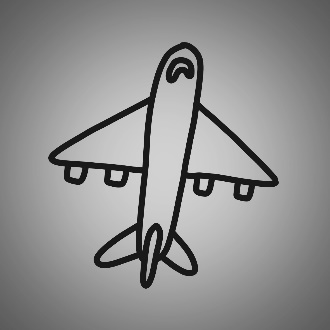
From the first day, the repository was created. Our team used github extensively so we can work on different parts of the project simultaneously. The project went into several phases. We started with the schema design, then we continued with some prototype code for the application itself. We put the SQL code for generating the tables, SQL queries to generate database samples and the code of the project core in separate folders for organization. The GUI and logic of FCIAIR project - written in C# Programming Language – was combined in the folder ‘UI’.

Collectively, there were over 100 commits shared between us. Every team member has contributed to the repository.

GitHub repository mainly offered us a way to sync our project so that all the team members were having the last update of the project at the same time. The repository is private for the time being, but it will be changed later to be public as it is a project that we all are proud of.

# Logo

For Fci Air, many logos have made and considered to be representative of the idea. A poll among the team members were made to choose the most suitable one and in order from best to worst the results were as follows:





The first logo deemed to fit the best, but it had to be slightly altered to fit as the icon for the application.

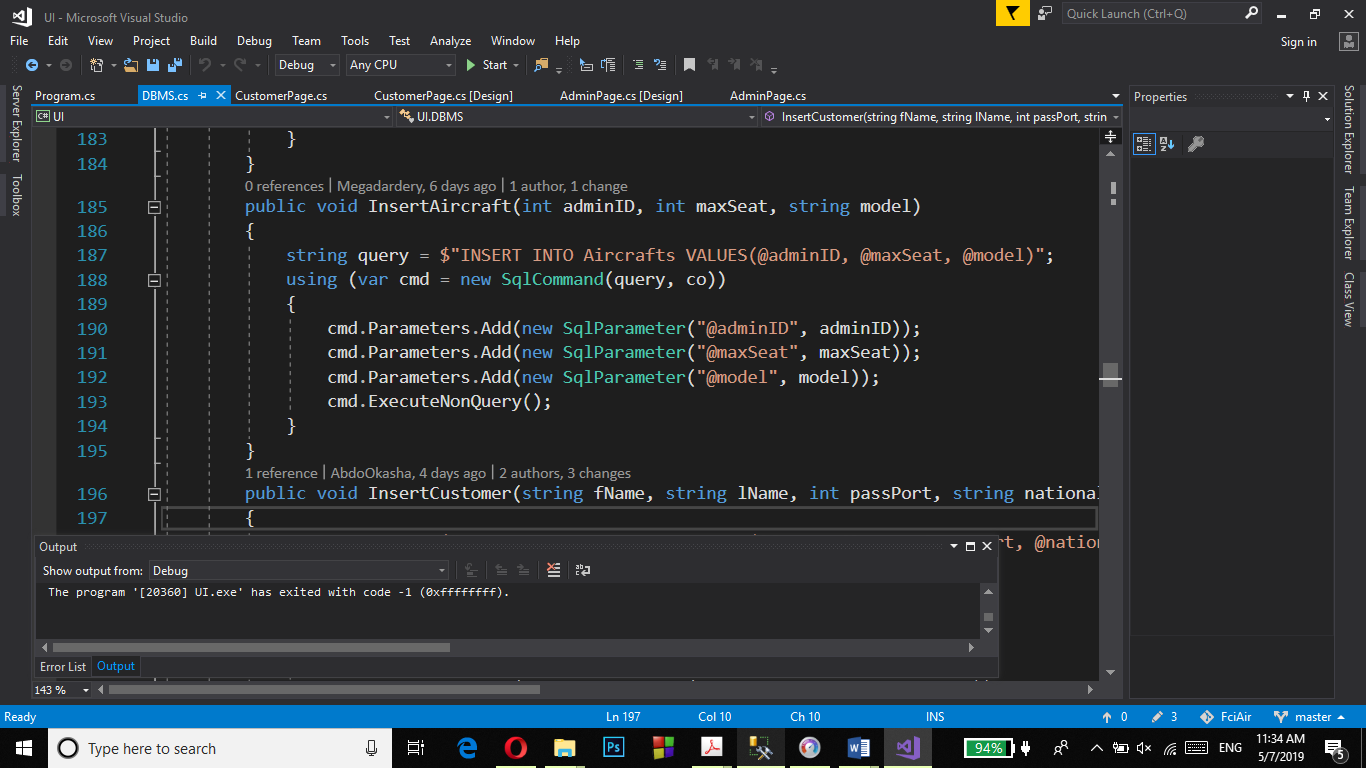
# Code

In our development, we strived as much as possible to maintain the code to be as clean as possible. Several organization styles were considered and we settled on one that is both readable, and efficient.

The project code functions are separated from each other. Every layout (form) has a separate file that manages all of its events. The common functions between several layouts and main code was put inside the ‘Logic’ class. The Database connections and queries were separated from the GUI into a class we named ‘DBMS’. All other classes use ‘DBMS’ for all the queries that are implemented in them, like searching, adding, updating and deleting.

As a proof of concept, we decide to hash passwords before putting them in the database using the MD5 hash. We are well aware that MD5 has major vulnerabilities and other Hashing algorithms like bcrypt and scrypt. Also, in a real application, we would have used other security measures like salting to protect from Rainbow Table attacks.

Another major issue is the possibility of SQL injection. This is the type of attack where the malicious user types in a query inside a textbox to be executed by the server. We prevented this type of attack by using SqlParamters (see following figure) available neatly in C#.



# Database Schema

When we started developing our schema, we used Power Designer to make the Conceptual, Logical and Physical ERD Models. The models are as follows:

## Conceptual Model

It should be noted here that ‘Pilot’ is a weak entity; it cannot exist without Aircraft because in our simplified system, we set the aircraft with the pilot directly, i.e we don’t have accounts for pilots.

Because it is a weak and a one-to-one relation, in the logical model, we decided to integrate Pilot into the Aircraft Entity.

## Logical Model

The logical model added the foreign keys as expected. For the many-to-many relation a relation table was added. And as mentioned before, the Pilot entity was integrated into the Aircraft Entity.

## Physical Model

In the physical model, we just set the ‘Username’ attributes to be unique, and then we ran the SQL generation. Due to some limitations, we had to manually modify the resulting SQL code so it fits the Microsoft SQL standards. The edits are minor and as such, they are not mentioned in this report.

Alongside the query for the creation of the database, we also provided a query to insert some sample data for demonstration purposes.

## Relationships

The admin can perform operations on aircrafts. So, there is relationship (one (mandatory) to many (optional)) between admin and aircrafts. The admin can perform operations on flights. The admin table has a relation (one (mandatory) to many (optional)) with monitor table and monitor table has a relation (one (mandatory) to many (optional)) with flights table. So, the admin can access the flights using monitor table. The aircraft has a relationship with flights (one (Mandatory) to many (optional)). The flights table has a relation (one (mandatory) to many (optional)) with tickets table.

The Customer can reserve a ticket. There is a relationship (one (mandatory) to many (optional)) between Customer table and tickets table. The customer can only access the ticket table.