*Santarossa Riccardo* *5BIA 01/01/2022*

**ITST “J.F. Kennedy”**

**SQLite e Flutter**

**Abstract**

The application is a normal flutter application, made up of four separate screens, which visualize the different operations that the user can perform using it. The app is connected to a SQLite database, to store the data that is inserted by the user.

**General Technical Data**

Describe features, components, modules, libraries and plugins included in your project:

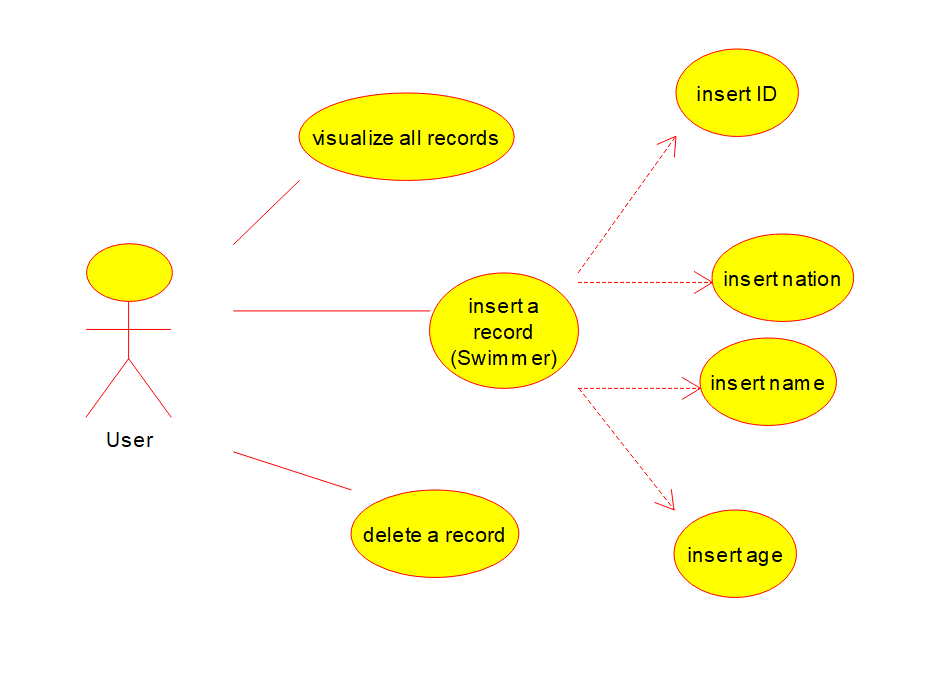
* Technology (es. Node, Flutter, Java, Python, Ionic, etc.. specify the version)  
  ………………………Flutter and SQLite…………………………………………………
* Project core features  
  ………Inserting, deleting and displaying data from a database in SQLite……………
* Type of project (i.e. server, web app, mobile app, batch program, GUI app, ..)  
  …………………Mobile app with integrated database ………………………………………………
* Development environment  
  …………………Visual Studio Code IDE……………………………………………………………………
* Operative system  
  ………………Windows 11 home………………………………………………………………………………
* Target Devices and O.S.  
  ………All Android devices………………………………………………………………………
* Internal database engine (if any)  
  ………………………………SQLite…………………………………………………………………………………
* Multiple device layout and resolution support (tablet, smartphone, ..specify)  
  ………….The application is supported by all devices, but is optimized for smartphone use…………………………………………………

**Key Features**The application can perform the most basic tasks of a database interface, which are inserting, deleting and showing records from a table. The app has 3 separate screens, each one has the ability of performing one of the previously mentioned tasks: in the first one the user can insert a record into the table, compiling all the textfields displayed; in the second screen the user can visualize all the records of the table, which are arranged in a list; in the last screen the user can type in the ID if the record he wants to delete and then delete it from the table.

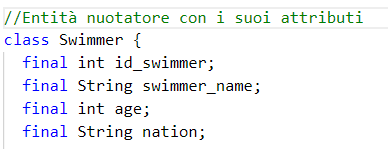
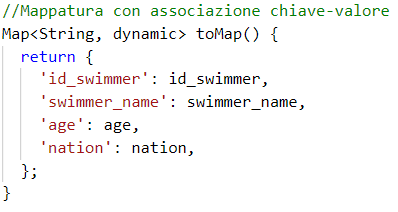
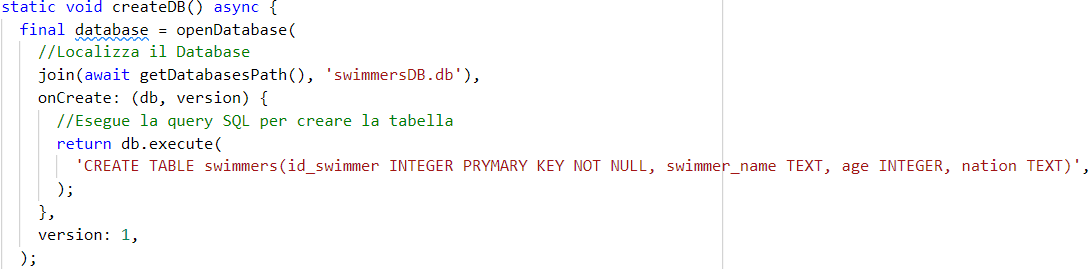
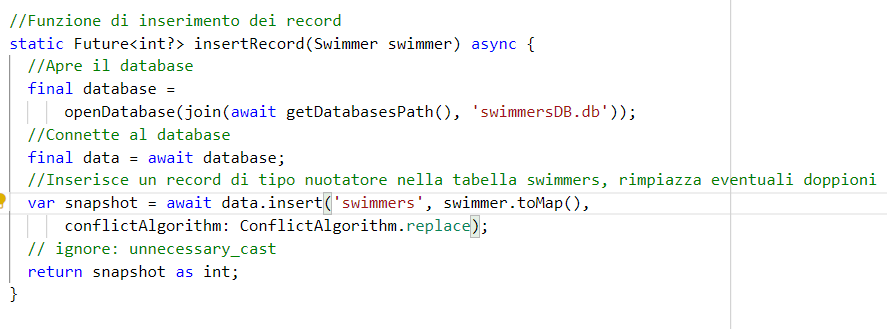
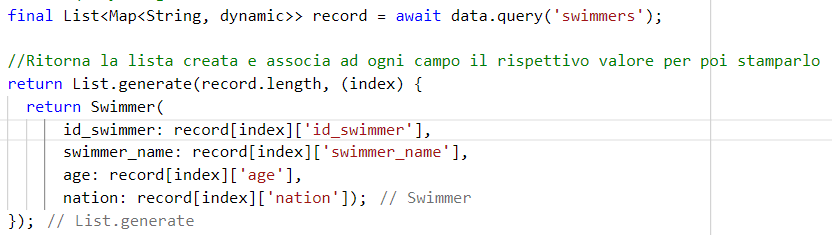
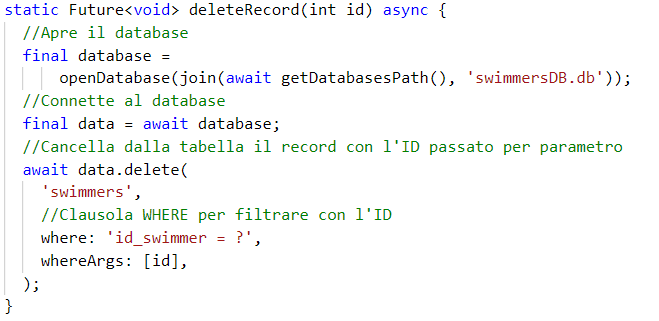
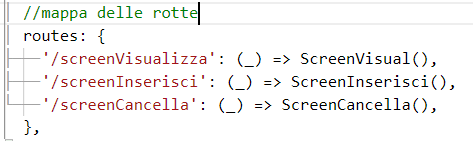
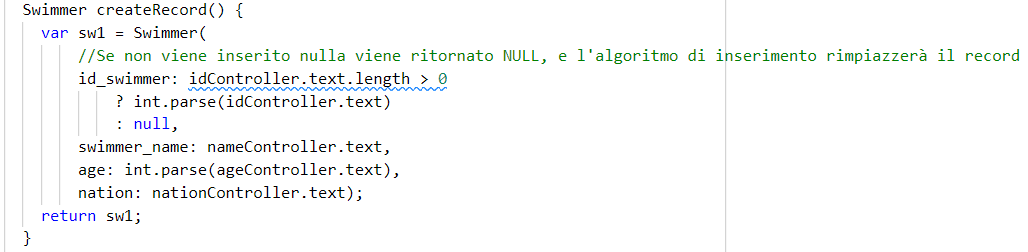
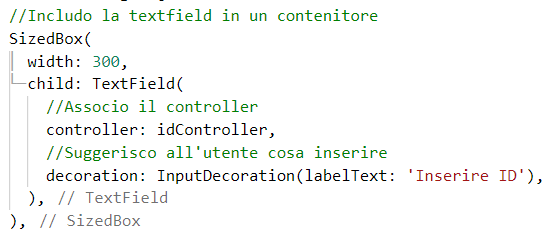
**Project structure overview**

The diagram below explains how the entire application works: in the first page the user can decide what to do, depending if he wants to insert, delete or visualize some records. The visualizing and the deletion page are made up of a very simple interface, whit a text field and a button for the delete screen and only a listview page for the visualizing page. The most complicated page is the insertion one, which is made up of four separate textfields in which the user inputs all the information needed to create a record.

***Use case diagram***

******

**Code fragments**

**-Entity-  
**This code fragment represents the Swimmer entity, which is the type of record the user can insert in the database, which is a Swimmers type database. In the next code fragment the swimmer entity is mapped to a hash-code so it can be inserted in the database.  
  
  
**-Database operations-**This first code fragment shows how the table is created in the database: the database itself is opened with the openDatabase() method and then an SQL query is executed to create the table.  
  
This second code fragment shows how a record is inserted: firstly the database is opened in the same way as before, then the swimmer which is passed as a parameter is mapped using the toMap() method shown before and then the method .insert() is executed on that hash-map.  
  
The most complicated code fragment is the one which is used to generate the list of records that have to be displayed in the display screen. In this fragment the “swimmers” table is queried with the .query() method, which executes a “SELECT \* FROM swimmers” query to select all fields and records in the table. Then the result of this query is converted in a dynamic map and stored in a List type variable called “record”. Then a List is generated using record’s length, and this list is made up of swimmers: note that each swimmer returned has a key to value structure, defined by the index of the list.  
  
The last code fragment of the dbOperations class is the one used to delete a record from the table. The user, in the appropriate screen, inserts the ID of the record that wants to delete an then this ID is passed to the deleteRecord() function. This function executes an SQL query that has the ID passed by the used as a filter, to delete the record using the .delete() method.   
  
**-User interface-  
**In this code fragment the main.dart file is displayed: in it all the routes for the different screens are stored, they are called named routes.  
  
  
The most important screen of the three is the insert screen, in which all the swimmer (entity) parameters are entered by the user. The first of the two fragments shows how the record is created using all the data retrieved from the textfields.  
The second one shows how each textfield works: each one of them is paired with a controller which retrieves the text and it’s inserted in a container called SizedBox.  
  
**Development**

Project repository: https://github.com/riccardosantarossa/SQLite\_e\_Flutter

IDE: …………Visual Studio Code…..

Man-hours: …………20……………

***For mobile apps***

Target environment: ……………Android…………

Target API level: …………Any…….……………

Minimum API level: …………Any………….………  
  
***Problems and difficulties***The main problems encountered during the development were about the use of the database instance. At the beginning I couldn’t find an efficient way of passing the database between screens, to perform the various tasks, so I ended up using static methods which can access the database from any screen.  
  
***Further development***The project could be improved by adding some graphical items, for example a nicer interface, and a different way of visualizing the records. Now all the records in the swimmer table are displayed one under the other without fixed rows or columns, the idea of adding a table could be an interesting improvement.

***Self-rating  
3.5***

**References**

<https://www.youtube.com/watch?v=nyvwx7o277U&t=430s&ab_channel=ResoCoder>  
<https://docs.flutter.dev/cookbook/persistence/sqlite>  
<https://dart.dev/codelabs/async-await>  
<https://medium.com/nonstopio/flutter-future-builder-with-list-view-builder-d7212314e8c9>