**Introduction**

Fun2Learn which is a well-established local company has assigned a task to design an edutainment software which would provide education on one hand and the other provides entertainment. The software includes stories with multiple questions and they have to be stored in database. On selecting a particular question (option), next story is loaded with appropriate questions. The stories are assigned to 2 modules holding 5 stories each. Plus, save and load have to be enabled for users who wish to resume later. This highlights the overall requirement of the system with a user-friendly interface I.E. GUI.

**Concepts used:**

* **Class and objects:** Blueprint of a program is class while real time existence is called an object.
* **Attributes:** The variables or fields used in a program.
* **Functions:** A procedure that may or may not return a value depending upon the definition.
* **Methods:** Defined in a class and procedures associated with object and messages. (Wikipedia,2018)
* **Encapsulation:** Securing attributes in a class using keywords private or protected (access by child classes). This ensures external class cannot directly access the attributes and functions.
* **Constructors:** Used to initialize values in methods of a class.
* **Parameterized constructors:** Used to initialize values in methods by passing parameters automatically during object declaration.
* **Getter and setter:** Setter are used to set values in a private attribute in a class where the value returned by getter could only be accessed.
* **Abstraction:** For the classes and functions required to use mandatorily, abstract keyword is used.
* **Overloading:** Passing different parameters to a function having same name for different functionality based on number of parameters. Mainly constructors are overloaded.
* **Overriding:** Functions having same parameters providing different definition are overridden. Usually inherited class override the base class.
* **Inheritance:** Child class inherits several features from a base class.
* **Polymorphism:** If the inherited child class comprises additional features than in base class, it is termed as polymorphism.

(Stackoverflow,2015)

**Task-1**

The conceptual model SDLC has to be utilized for the project management with stages involved in information systems (TechTarget,2015). The program has to be designed with a proper interaction of database and our UI. We follow bottom-up approach through which we design database first and invoke queries to transfer the values to our UI through queries and data tables. We therefore select our required fields in data table and load them to our UI.

First of all, connection establishment is essential and thereafter only query could be invoked. We require to add two libraries System.Data and System.Data.SqlClient. The function, manipulate (query)executes the query with connection open/close. (MSDN.Microsoft,2013)

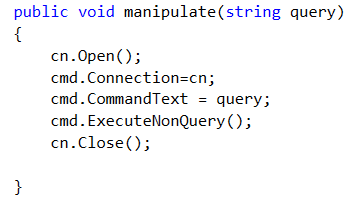
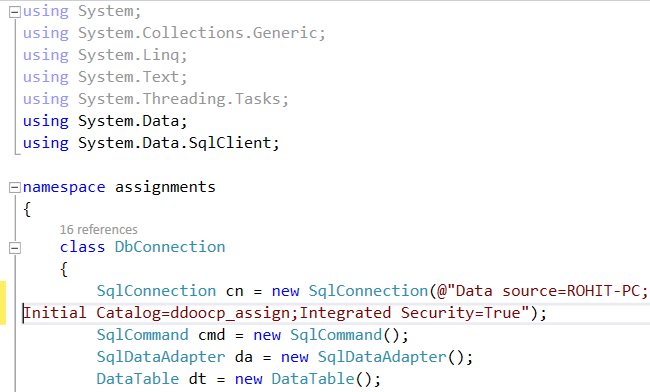


Figure 1: DB Connection

Database connection involves:

* Server connection
* Database selection
* SQL Command insertion
* Query execution
* Connection Exit

The function retrieve (query) meanwhile returns a data table which is mainly used to retrieve values from database to form through data table.

* Dataset: Used for temporarily storing data in table.
* DataAdapter**:** Enables changing, retrieving and displaying data from database.

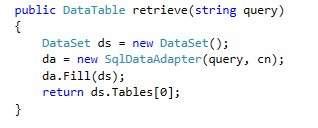


Figure 2: Retrieval through DB connection

The queries are located in a single class **studentcrud** including constructors, methods, functions with varying characteristics. The attributes have been encapsulated so that other class may not access them and their values are set through getter and setter.

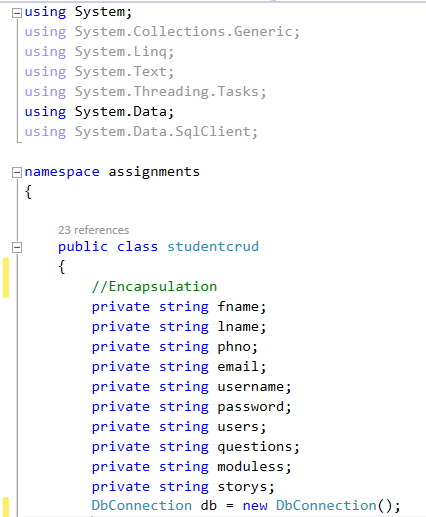


Figure 3: Encapsulation of attributes and connection establishment

Also the DbConnection class has been initialized and through object db, database connection establishment takes place.

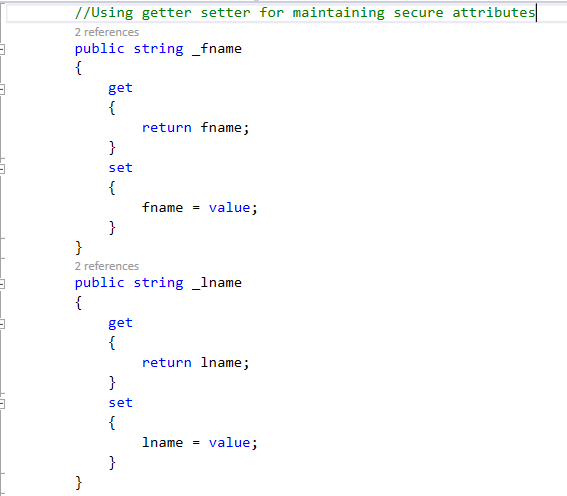


Figure 4: Using getter setter for initialization

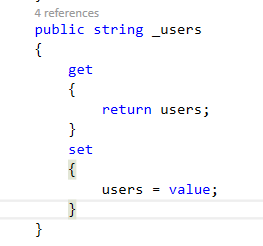
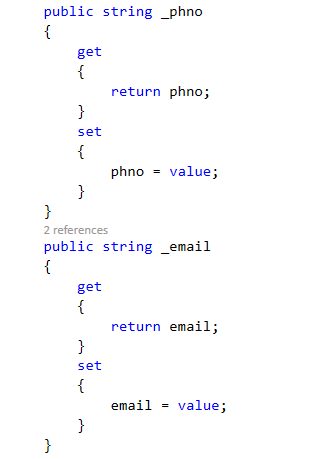


Figure 5:Getter setter



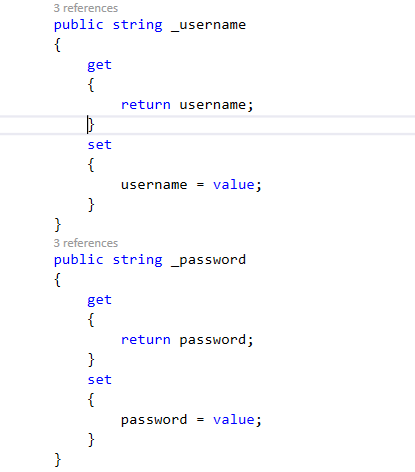


Figure 6 : Initialization using getter setter

The program includes user log in for which registration is a must. The registration form has been constructed as:

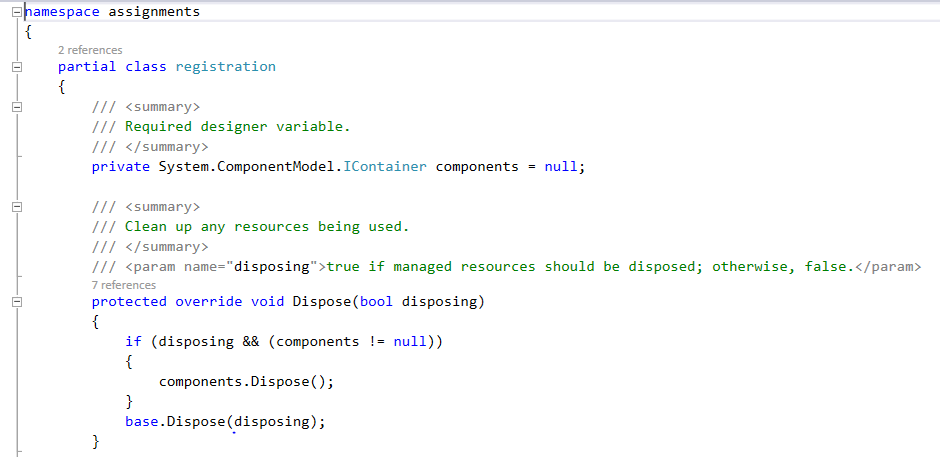


Figure 7: Form registration designer

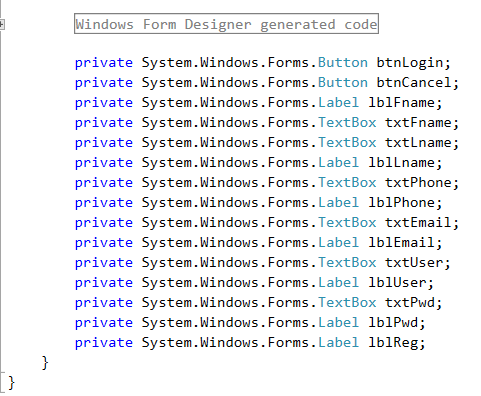


Figure 8: Fields design in the form

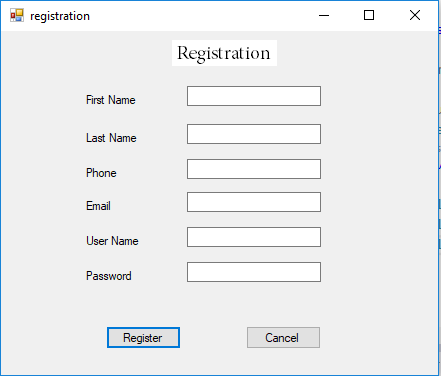


Figure 9: Form layout

First of all, connection establishment is essential which is done through object db. Here, the values are inserted in the text box and those values are set for query fire through getter and setter. Upon calling the function through object StudentCrud, values are inserted through query in function insert ().

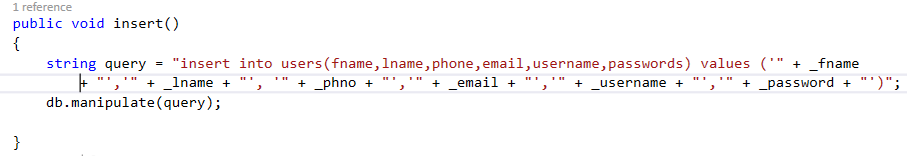


Figure 10: Insert query

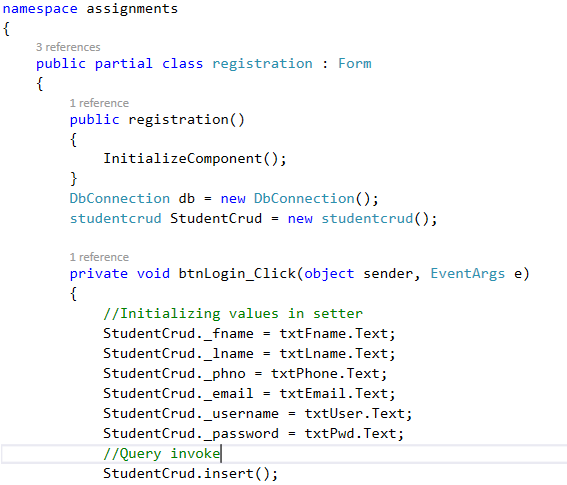


Figure 11: Setting values and firing insert query

Also for save and load purpose, a function savei(string a) is used which automatically sets value of user\_id which is provided during registration, module\_id and story\_id. For save, these values are updated as per user interest.

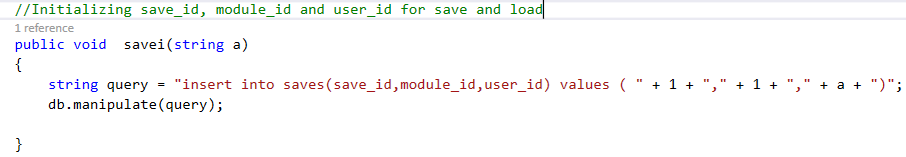


Figure 12: Initializing story\_id and module\_id

Here, user\_id is sent as an argument for initializing story\_id and module\_id as per user entry.

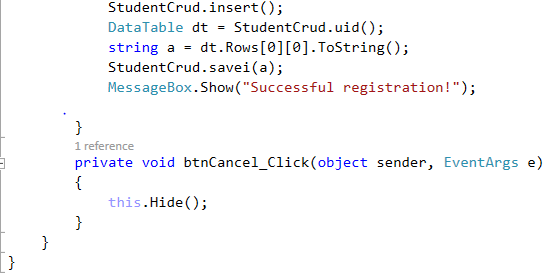


Figure 13: Setting values as per user\_id

After registration, the user is enabled to play the game after filling a login form inserting user name and password they just set. Also, they can login after a while using it.

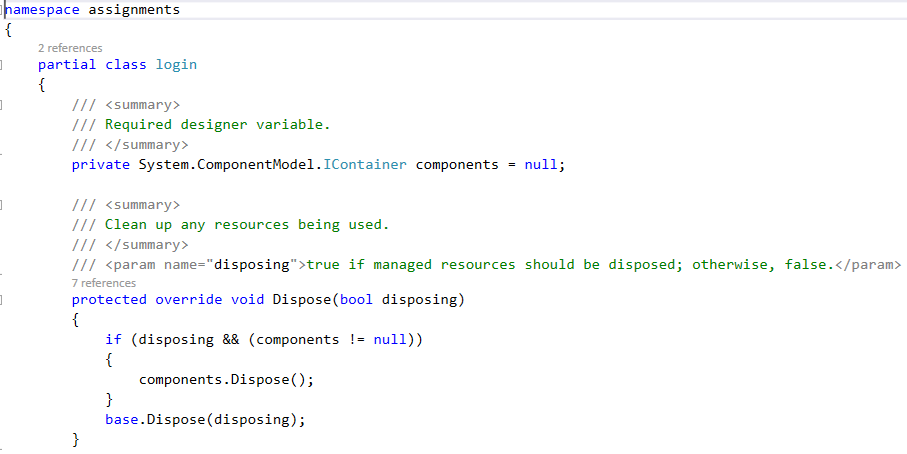


Figure 14: Designing login form

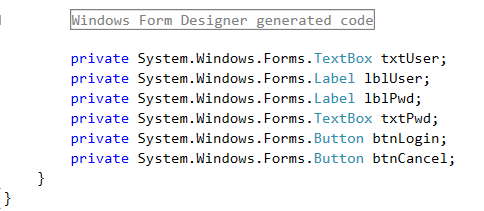


Figure 15: Fields in login form

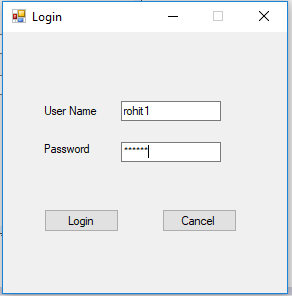


Figure 16: Login form appearance

The username and password need to be validated as mentioned by Fun2Learn so that user information would be secured in the back end.

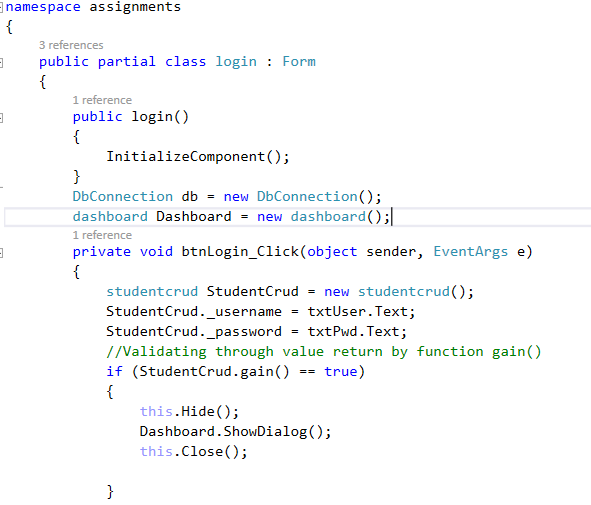


Figure 17: Validating username and password

Through getter and setter, username and password are initialized and values are selected through query1 which are stored in data table inside try. If the username and password does exist, the number of rows in data table would be one and true value is returned which states that user does exist. Upon any exception, Exception ex is thrown and if the no. of rows is 0, false value is returned.

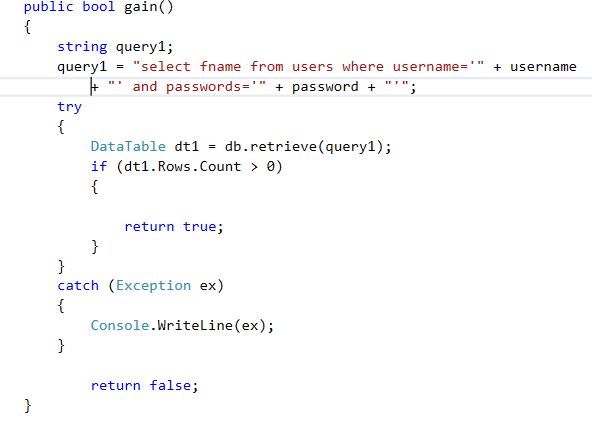


Figure 18: Validation using try catch

If an incorrect username or password is entered, false value is returned showing message box. Function clearit() clears the textbox value upon any invalid insertion. Meanwhile. button cancel leads us back to starting of the program.

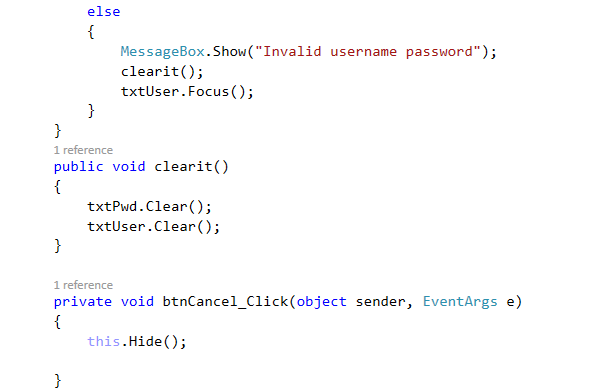


Figure 19: Validation

For gaming purpose, after login we enter dashboard where all the working exists.

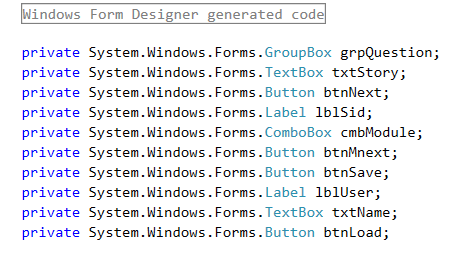


Figure 20: Fields In dashboard form

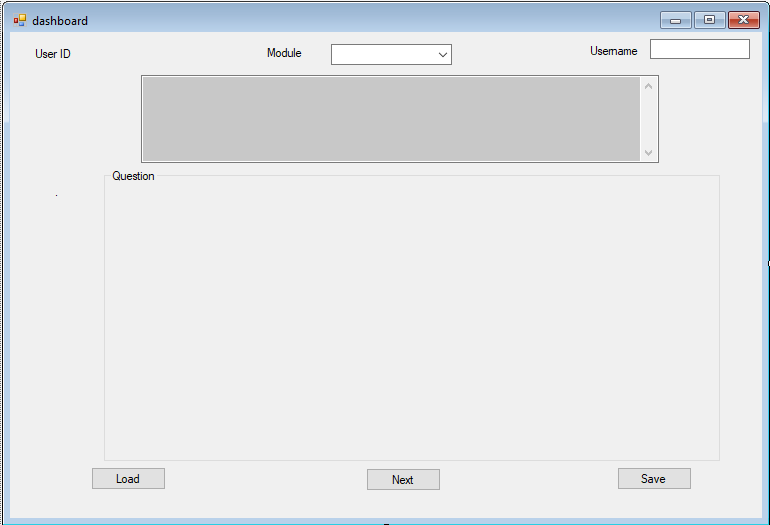


Figure 21: Dashboard form

The dashboard composes the above features. The group-box holds the questions, combo-box holds the module and the multiline textbox displays the stories. The saved values are loaded via button Load while on clicking Save, we can save our appropriate module and story. By clicking next, we move to another story with different questions. Also for verification we tend to enter username just to make sure accurate values are returned.

Connection establishment is a must along with class initialization. Upon loading form, a message-box appears stating if we wish to continue save game or not. Upon clicking yes, we head towards our saved value while upon clicking no, we start from the very beginning if we don’t intend to load our saved game. Also values of module are retrieved from database and displayed inside the combo-box.

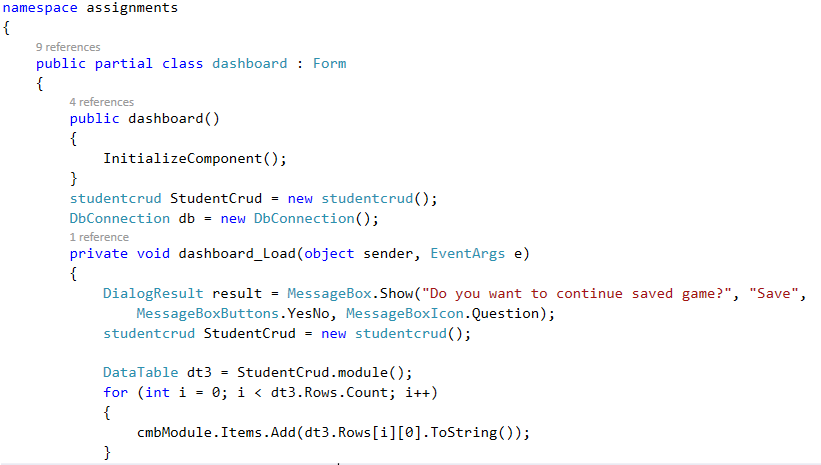


Figure 22: Combo box load

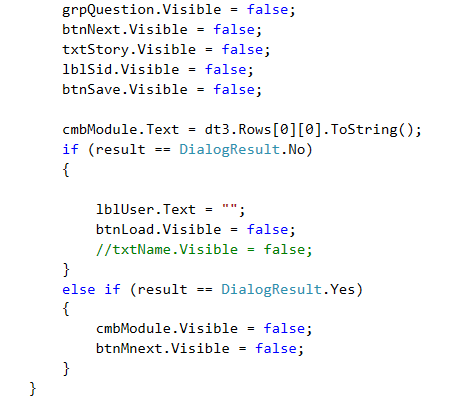


Figure 23: Loading according to user choice

Here, the best way of user friendliness has been opted. As per user choice, the form attributes like textbox, combo box, label and so on are displayed. If a user opts to continue a saved game, combo box is not displayed as s/he would have already selected a desired module. Also, since the button btnMnext selects stories and questions based on module, it is also hidden as well. Also, if the user doesn’t want to continue saved game, displaying load button wouldn’t be ideal as well.

For security purpose, first of all username is verified via setter and if value is returned via data table, game is proceeded otherwise the following message box is displayed. Also for exception handling, try catch is used.

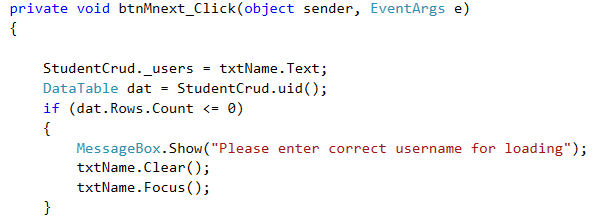


Figure 24: Security maintenance

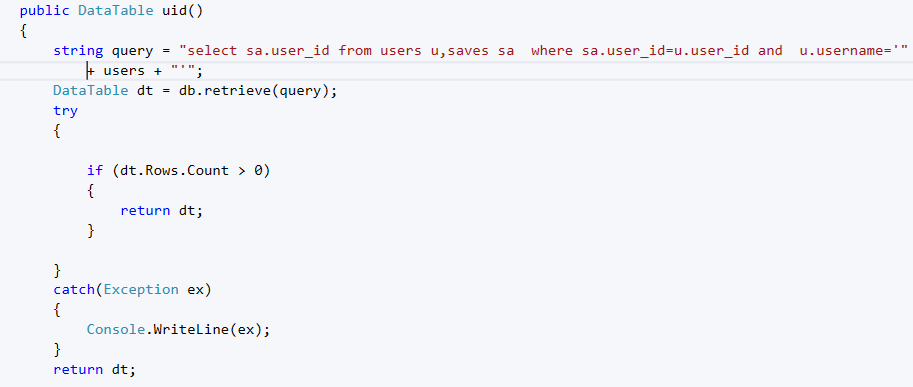


Figure 25: Checking existence of user

After successful verification as per user interest, the game is loaded. If user tends to begin from the core, module selection would be possible while continuing saved game wouldn’t require that.

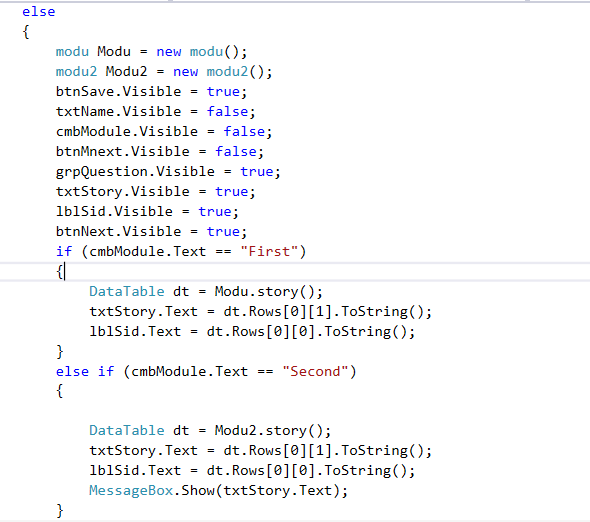


Figure 26: Story load as per module

Since, two modules exist comprising separate stories, function overriding is essential by which essential story would be display as per selected module. Two classes modu and modu2 inherited from studentcrud are created with objects Modu and Modu2 respectively. Upon selecting a module, appropriate function is called firing the appropriate query.

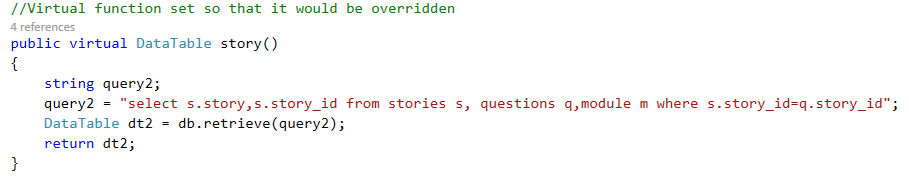


Figure 27: Using virtual keyword for being overridden

Also, polymorphism enables us to add unique feature to a inherited class (query being unique).

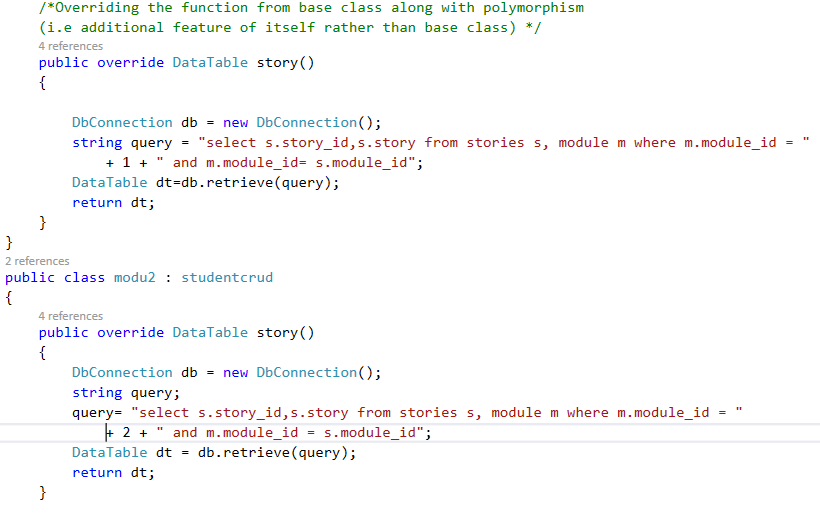


Figure 28: Overriding function story()

By this story selection would be successful for beginners or users that wished to start from the scratch.

After story selection, question load is a must as per assignment. For this, story\_id is sent through an argument to select questions via foreign key I.E. story\_id (f.k) in table questions. We cannot predict the exact number of questions which could be changed and therefore dynamically, questions are loaded and kept in radio-button as we have to move towards next story. The total number of rows in the retrieved data table would be the total number of required radiobutton.

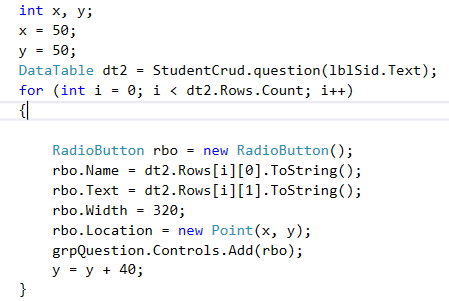


Figure 29: Dynamically creating radio button

Here, data table is returned and the number of rows is accounted.

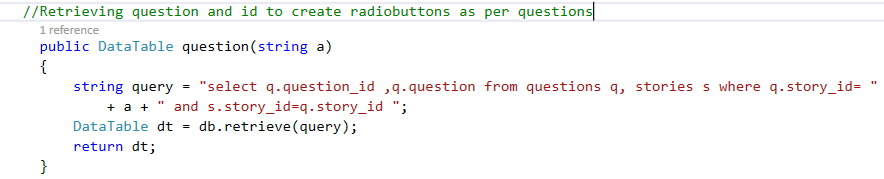


Figure 30: Retrieving questions

On selecting a question, users tend to see a new story. Upon clicking Next after selecting an interested question, next story is loaded. Also, we tend to remain in the same module and module\_id is selected reusing the function savemodule(). DataTable is returned from function nextque() with arguments question\_id respectively and module\_id. The column consisting story is displayed in the multiline textbox. All of this is possible due to the column next\_phrase in table questions in the database.

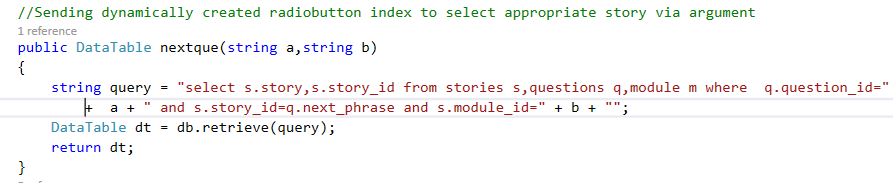
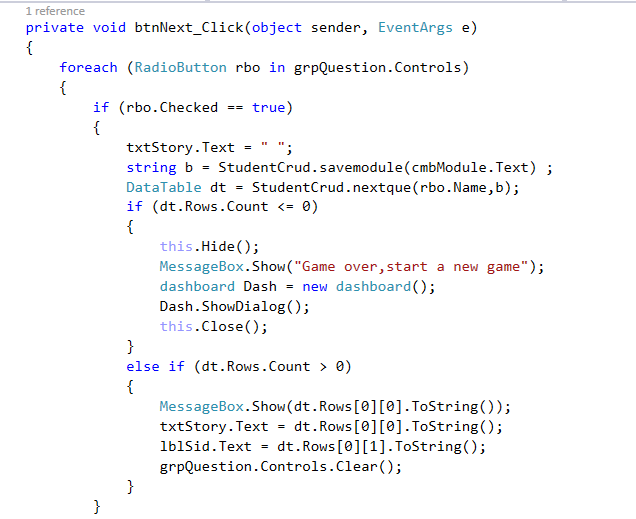


Figure 31: Selecting next story from question via next\_phrase

Also as per requirement, users must be able to save in each and every step. During registration, automatically their values are inserted in default where module\_id=1 and save\_id=1. Then as per their choices, these values would be updated enabling them to save the values and load them accordingly. User\_id is selected through username set via setter.

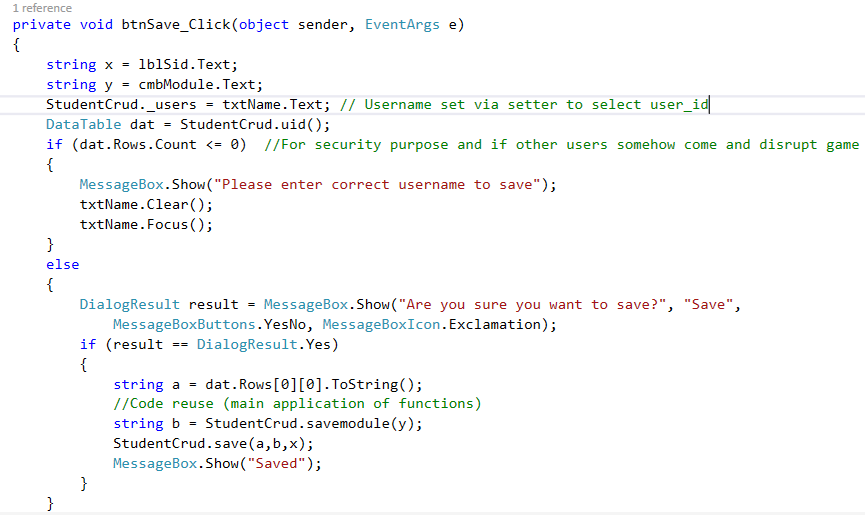


Figure 32: Algorithm for saving

Selected module, story and current user id is sent through argument and updated on clicking save.

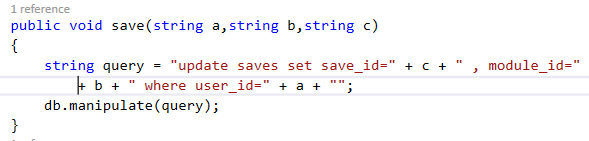


Figure 33: Save backend query

Similarly, after saving load of the saved game is mandatory. Upon pressing button Load, the saved game is loaded. This button appears only after the user clicks Yes to saved game load.

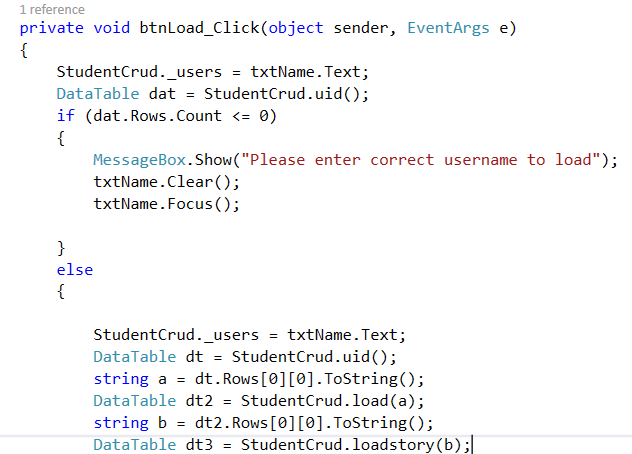


Figure 34: Loading saved game

First of all, user\_id is a must since the program has to identify which user is tending to load the save game. This is set via setter and user\_id is passed via argument and save\_id (representing story\_id) is selected.

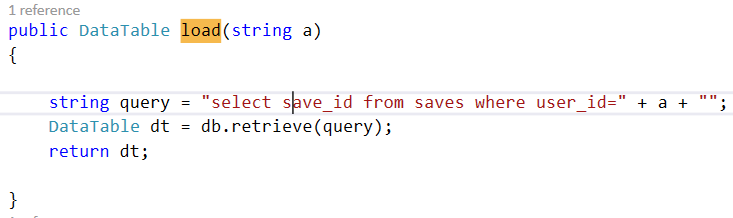


Figure 35: Selecting saved save\_id

Basically save table relates entities stories, modules and questions. Save\_id is the saved value of story\_id and selecting through story\_id and save\_id is just the same. So, function loadstory(a) takes an argument I.E. save\_id returned from function load () and selects corresponding story as per save\_id which is displayed under the text box. Upon selecting a story, questions are automatically loaded via One to Many relationship and story\_id being foreign key.

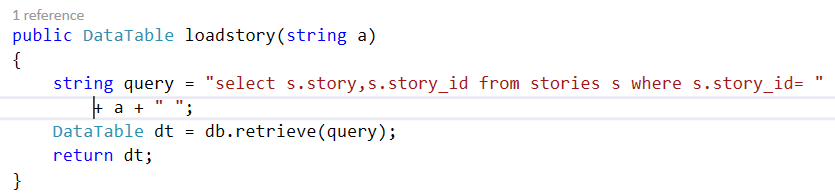


Figure 36: Selected story accordance to save\_id

After loading, button load and username textbox is not displayed since it’s not required. Also, story\_id is displayed in the label.

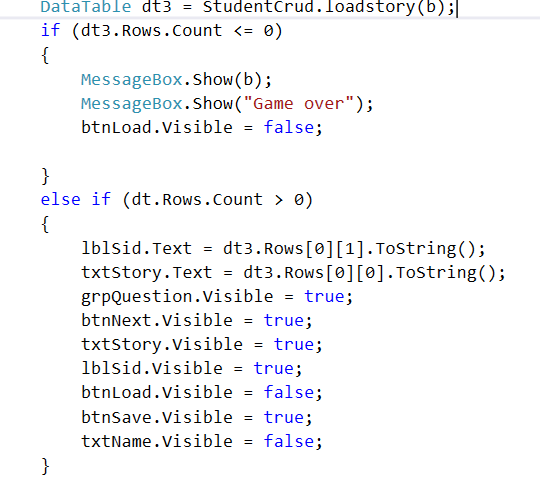


Figure 37: Loading saved game

Also we tend to create admin panel for inserting further stories and questions. Also, update and delete would be essential as admin would be granted such rights.



Figure 38: Admin form fields

The fields question and story appear as per user interest. On selecting the checkbox Question, text box of question appears. Similarly, on selecting story module and story appear respectively. On choosing both, both of the fields appear correspondingly.

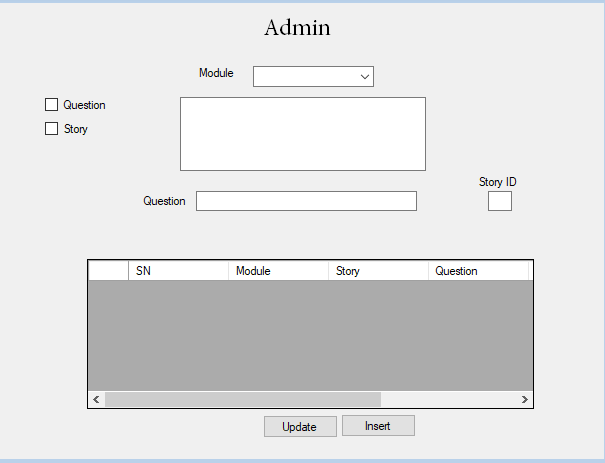


Figure 39: Admin form

Required values such as modules, questions and stories are selected and inserted into the data grid for admin benefits and the data tables being data source for the data grid. Also on clicking the links **Edit** and **Delete**, update and delete procedure can be held correspondingly.

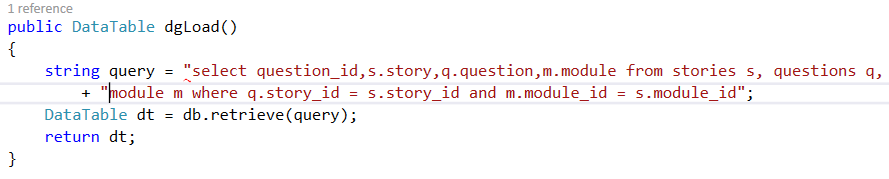


Figure 40: Selecting appropriate fields

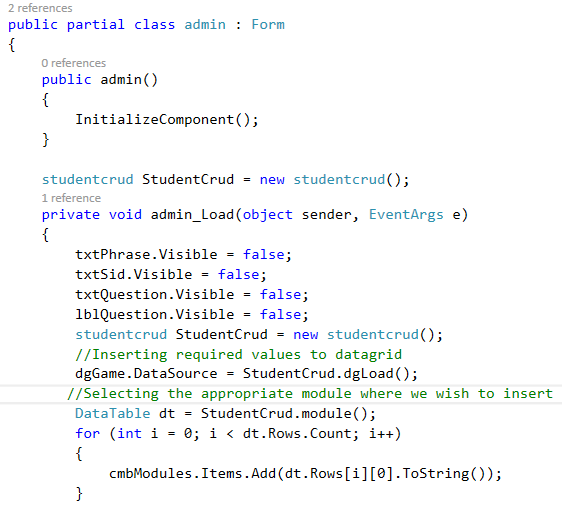


Figure 41: Inserting values in data grid for admin

On clicking **Edit**, the values displayed on the data grid are displayed under the corresponding text box. For update, the loaded values are changed as per our wish and update query is fired respectively. Also, the values to be updated are initialized via parametrized constructor.

Constructors are used to initialize values in a certain class and if parameters are used, such constructors are called parameterized constructors. (Techopedia,2014)

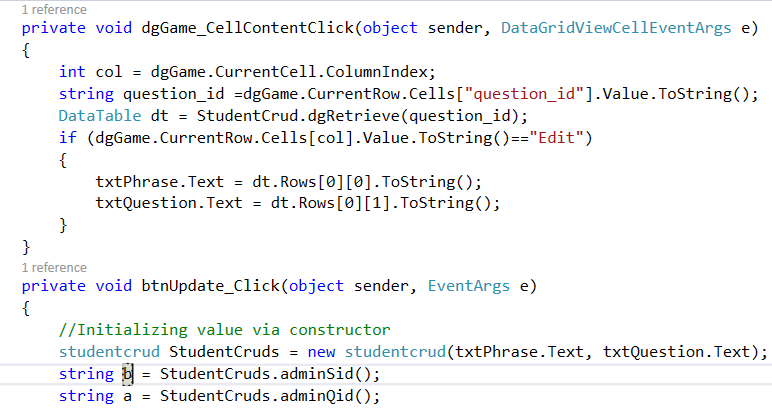


Figure 42: Data Grid functionality

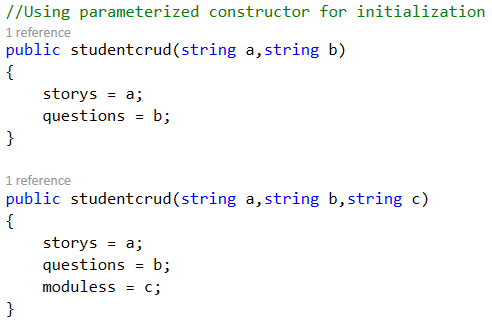


Figure 43: Constructors for value initialization

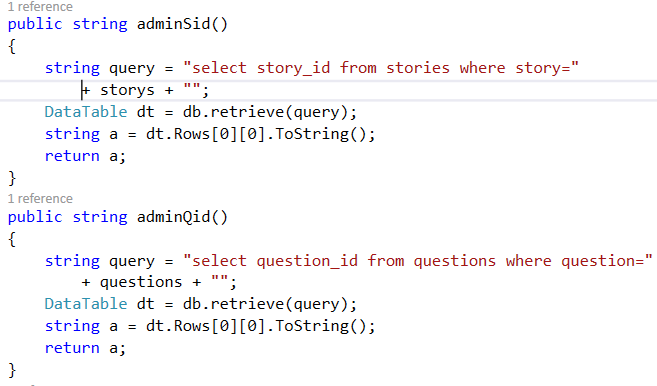


Figure 44:Functions for retrieving ID's

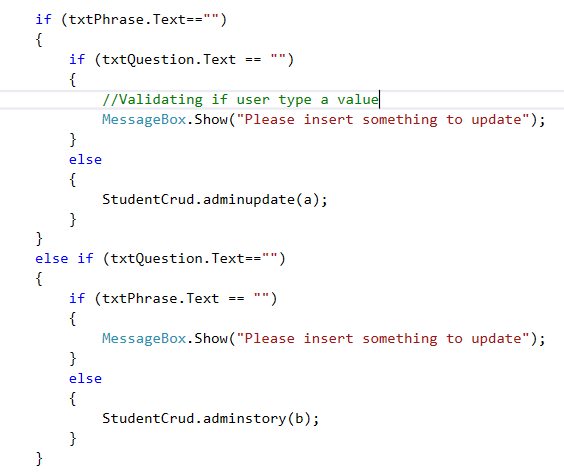


Figure 45: Update algorithms

According to user entries, queries are accordingly fired and hence update procedure completes. Upon a blank value, the program understands the blank nature and discards it accordingly. As per ID’s update queries initiate as:

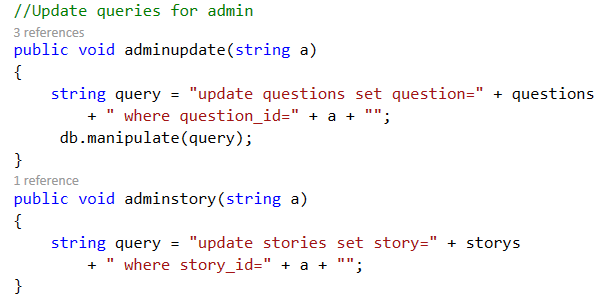


Figure 46: Update queries

For insertion, values are written accordingly into the corresponding text box and the insert queries are fired. Values are initialized via constructor.

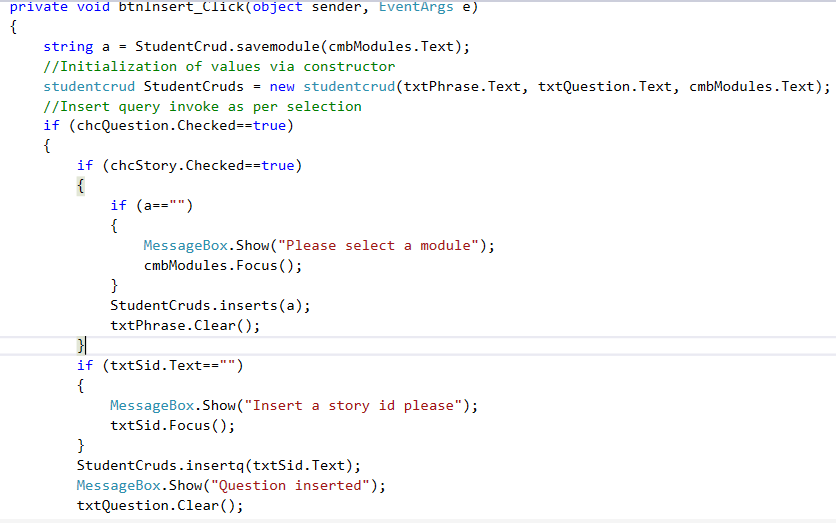


Figure 47: Insertion of values

.

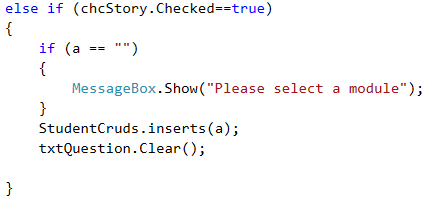


Figure 48: Insert as per checkbox

Through these functions, insert query is fired by passing arguments & constructor initialization.

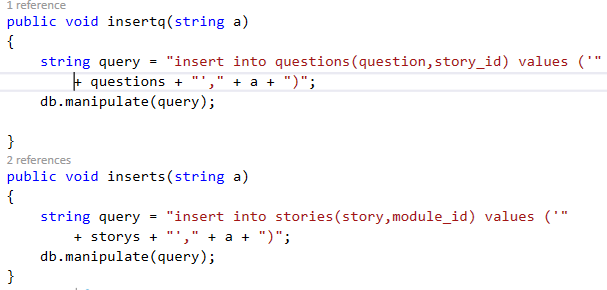


Figure 49: Insert queries

Delete function is mandatory for admin and for this through **Delete** link, delete queries are fired. But for assuring admin command, message box is displayed just for the case if admin clicks it mistakenly. Upon clicking Yes, delete query is fired as per textbox values.

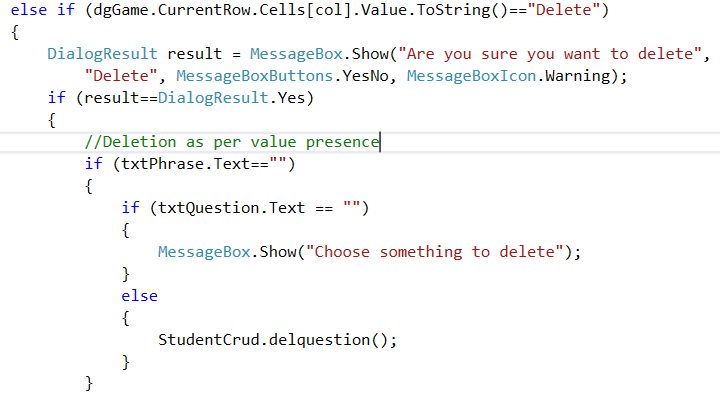


Figure 50: Deletion

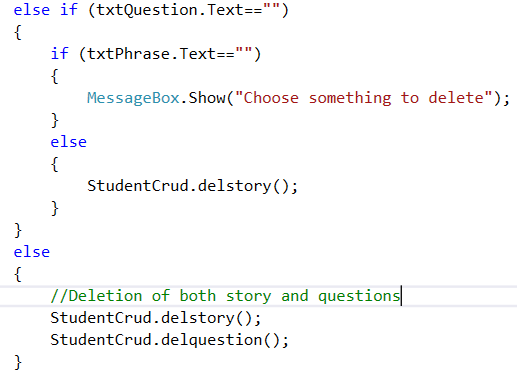


Figure 51: Deletion

The following queries have been invoked for deletion.

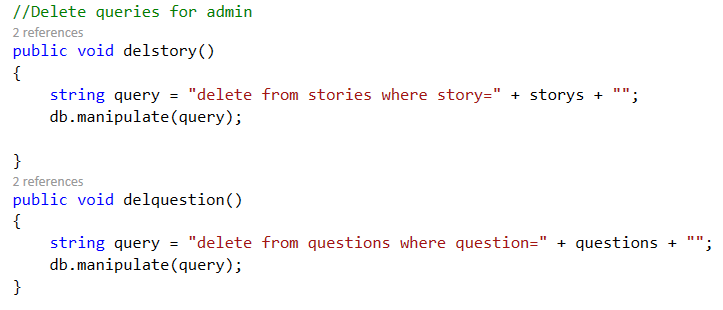


Figure 52: Delete queries

The above algorithms are involved in a program specialized in edutainment purpose. Modelling of static view of an application is explained by class diagram. (TutorialsPoint,2015)

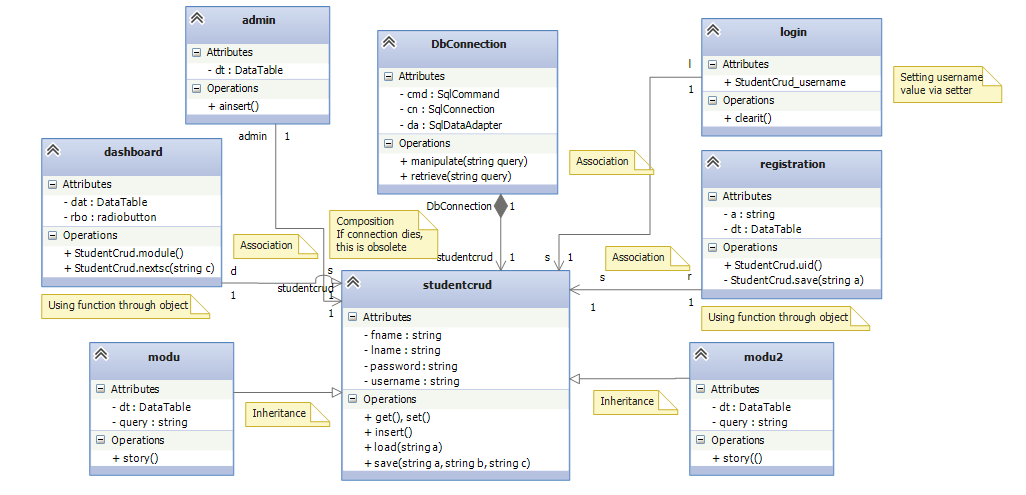


Figure 53: Class diagram explaining the program

**Task-2: Testing**

Evaluation of software such that it meets the essential requirements is called testing. (TutorialsPoint,2013). For the identification of errors, left requirements, gaps and false values, testing is a must. It is one of the core stage of Software Development Life Cycle(SDLC). During testing, every steps are documented and software designing is done as per requirement. First of all, test plan is carried out.

The resources, objectives and processes are documented in overall for a specific testing purpose which is termed as test plan. (Wikipedia,2018)

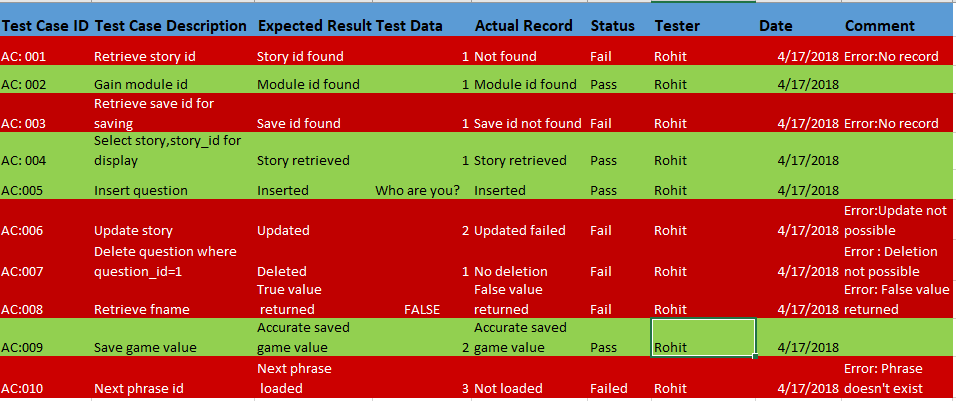


Figure 54: Test plan

**Unit testing**: The analyzation of units of codes in a program along with working, data and usage & determination of validity.

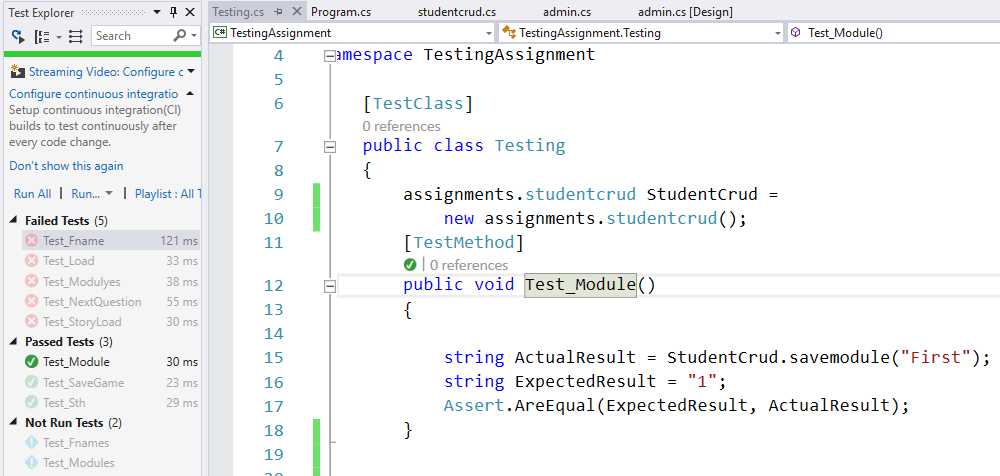


Figure 55: Test-1

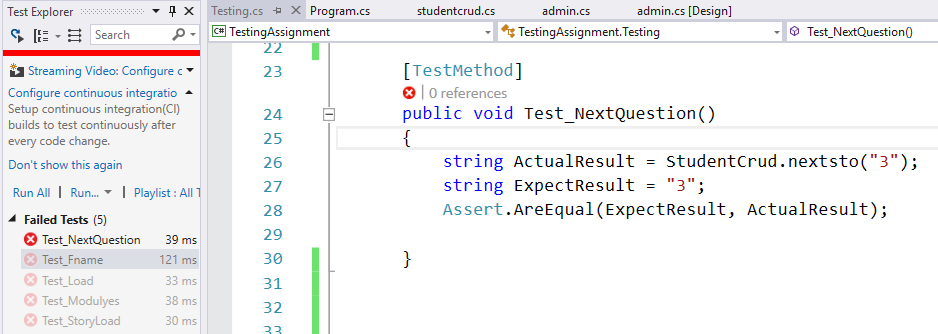


Figure 56: Test-2

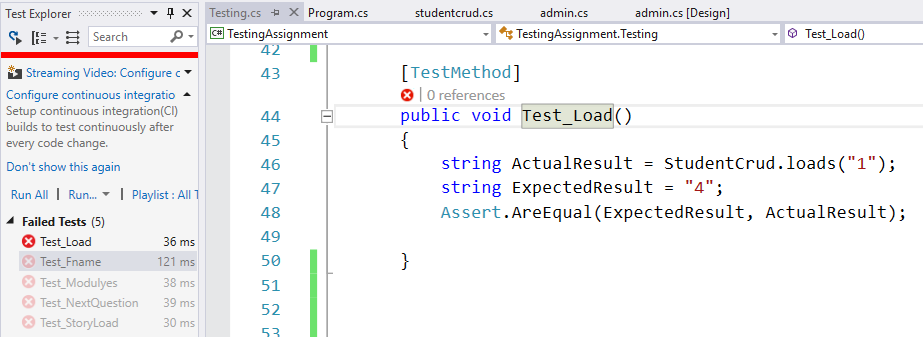


Figure 57: Test-3

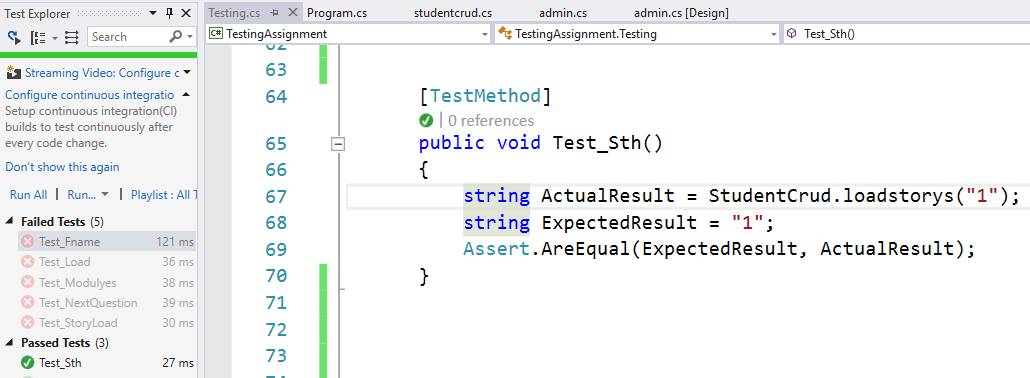


Figure 58: Test-4

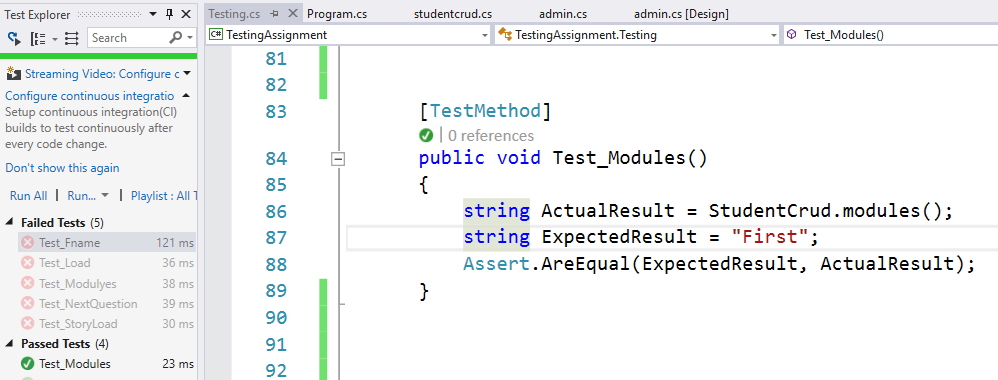


Figure 59: Test-5

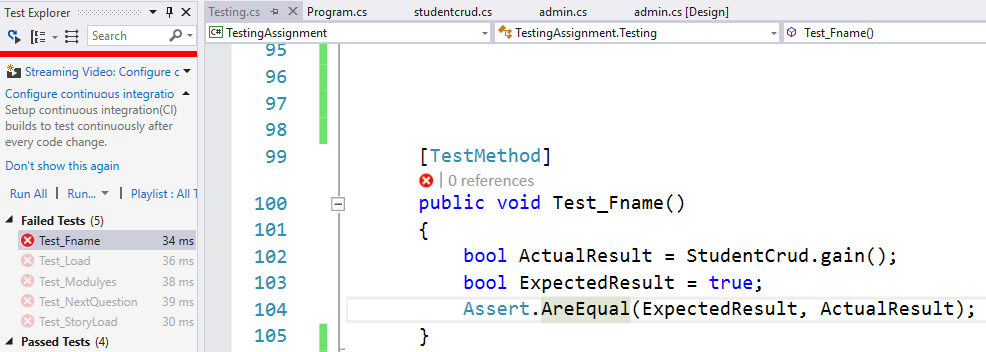


Figure 60: Test-6

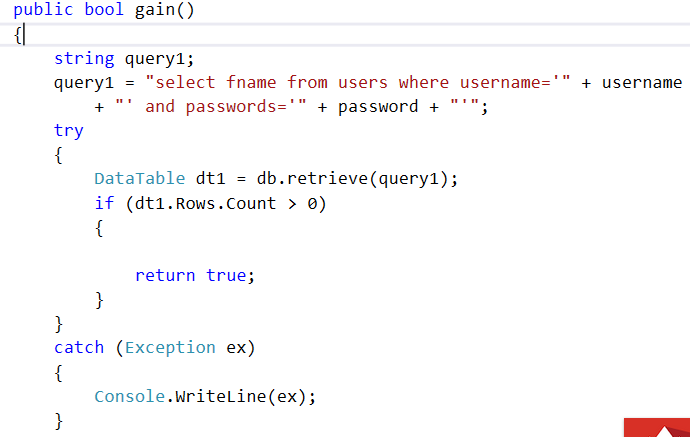


Figure 61: Exception handling through try catch

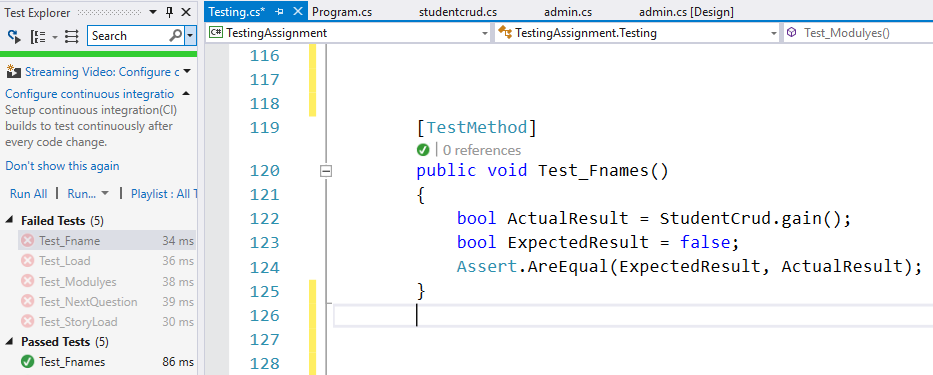


Figure 62: Test-7

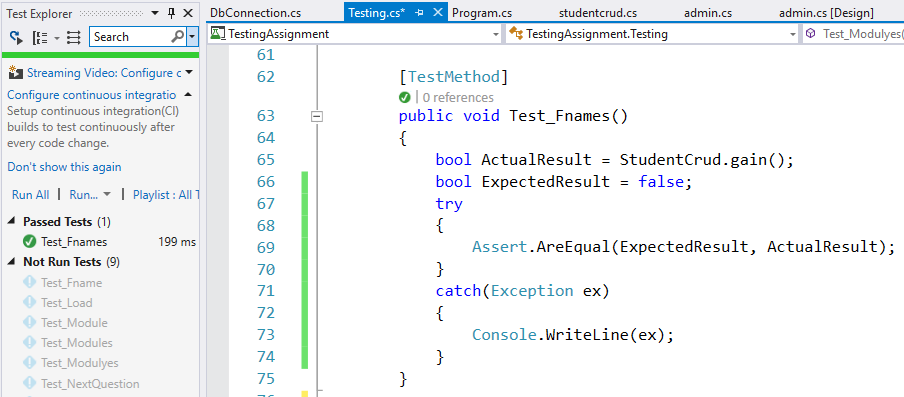


Figure : Testing using exception handling

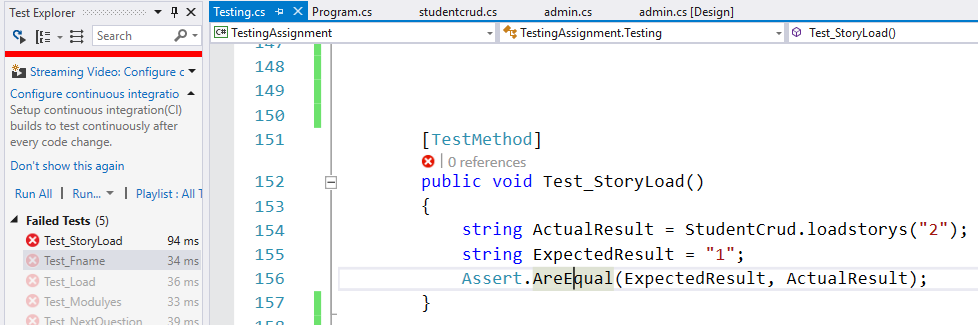


Figure 64: Test-9

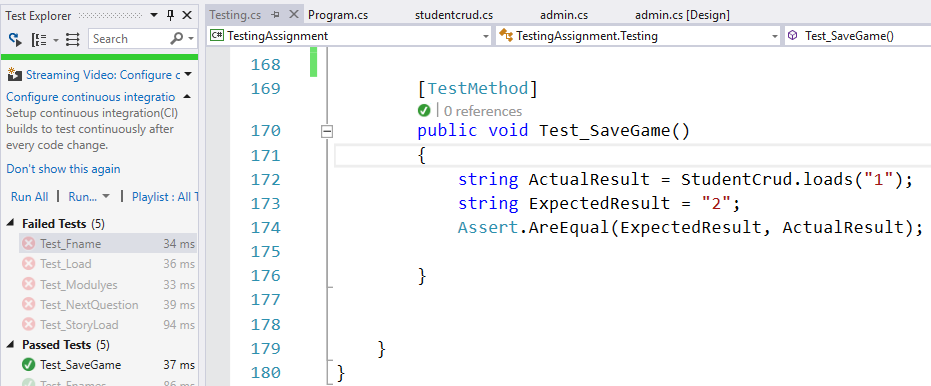


Figure 65: Test-10

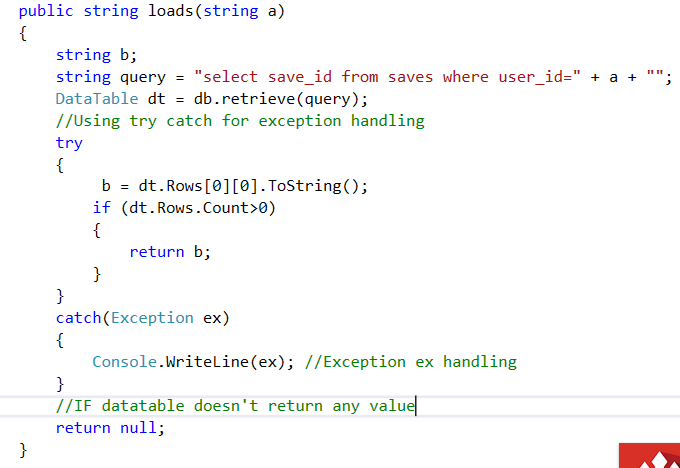
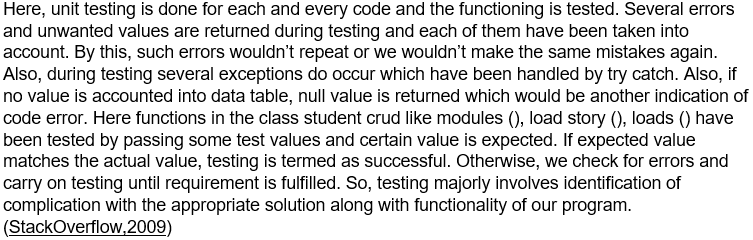


Figure 66: Exception handling-2



**Task-4**

Paper prototypes are hand-made architecture of the interface before actual implementation mainly utilized in the end user development by gathering feedbacks. (Wikipedia,2018)

Designing a paper prototype usually proves to be quite handy nice program can be generated as per feedback with better functionality with simpler approach. Not only this, with the help of paper prototype, we can generate ideas ourselves as well and point out functionality we expect to exist. On one hand, we could directly test the bug areas and design a program in a small interval of time and on the other our stress free mind could gather more ideas and we could design the perfect program we wish to design. The assigned program has been approached through paper prototype as:

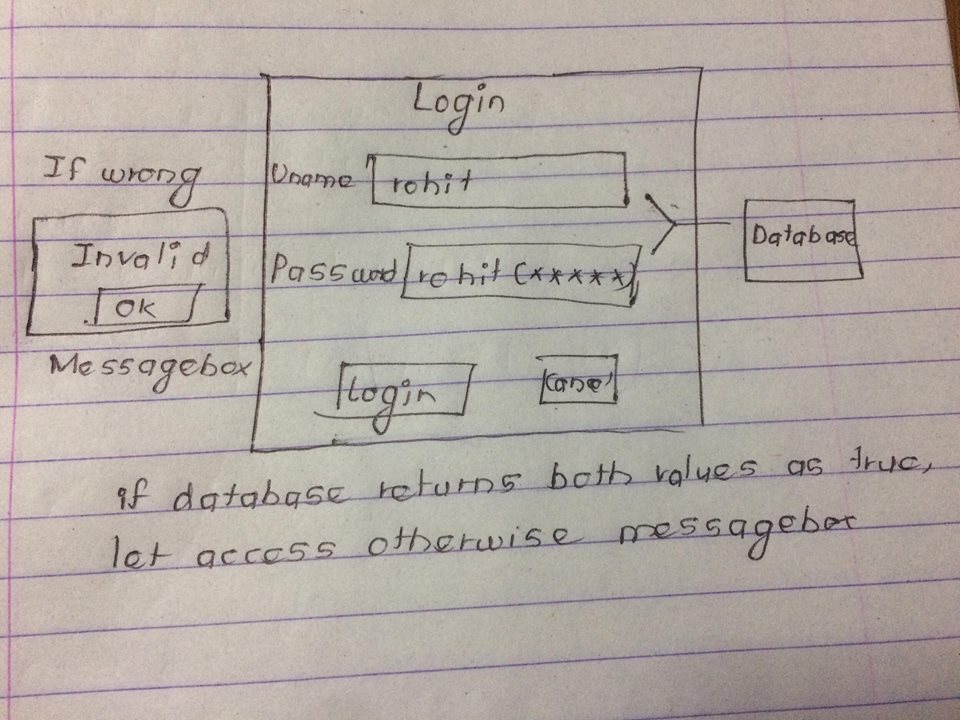


Figure 67: User form

First of all, displaying password as invisible characters wasn’t taken into consideration. But the registration form includes vital information like email which must be secured. Any access to the account grants access to the information. Hence through testing of prototype, password has been provided as hidden character.

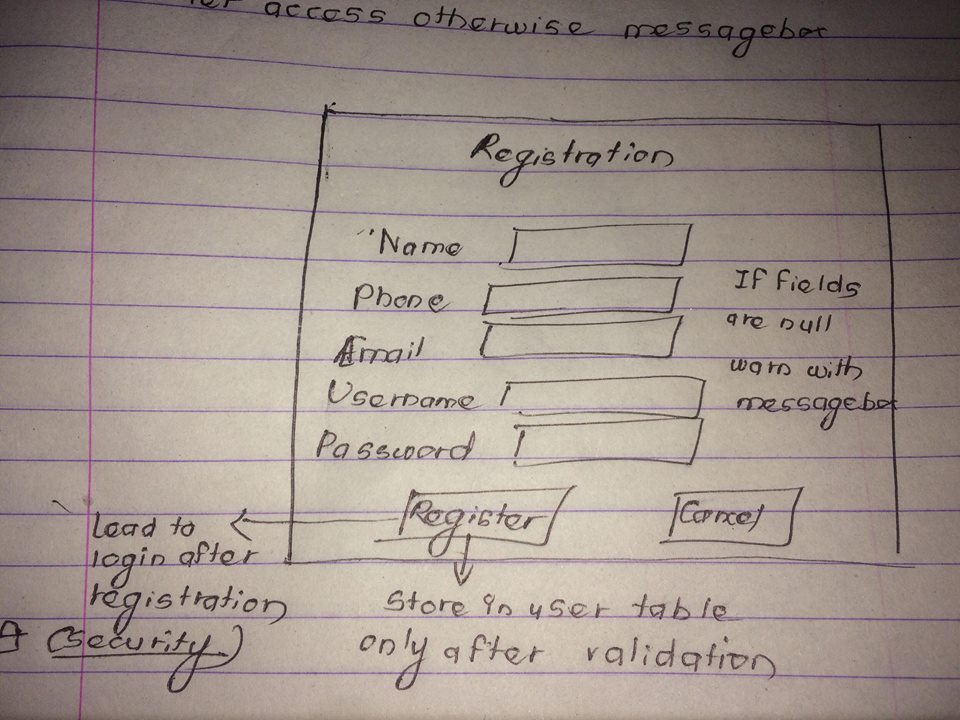


Figure 68: Registration forms

After this, saving a module and story ID automatically after registration looks to be a better choice for providing save features. By pressing button Save, update query would be fired and simultaneously Load would be possible retrieving the saved values. This knowledge was gained only after noting the ideas down to a paper and user recommendation. Hence if software development is a destination, testing is the best part for it.

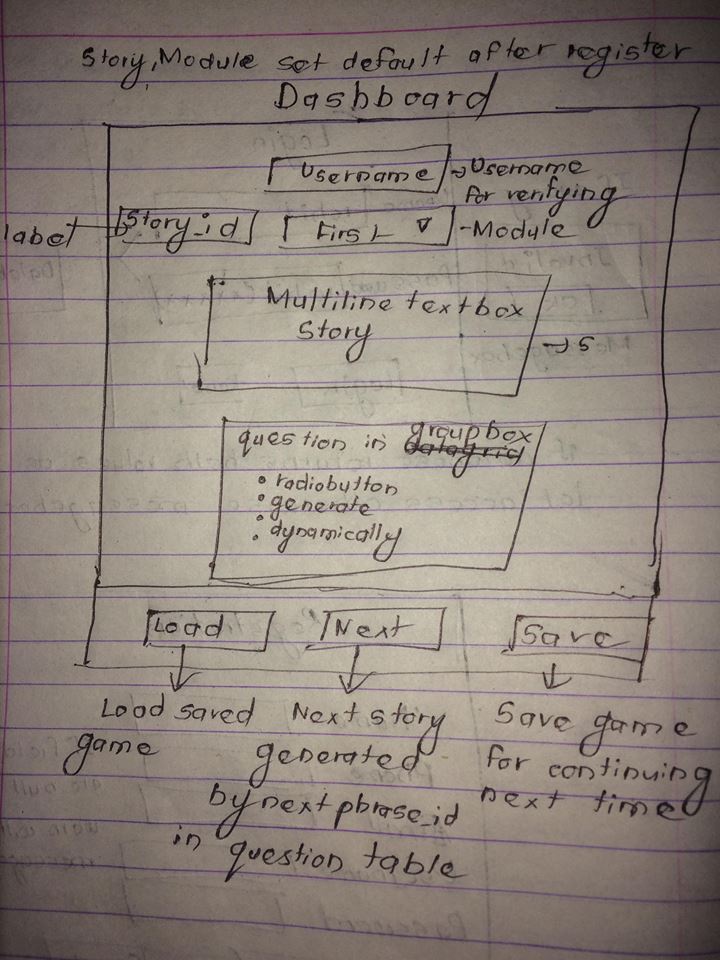


Figure 69: Dashboard prototype

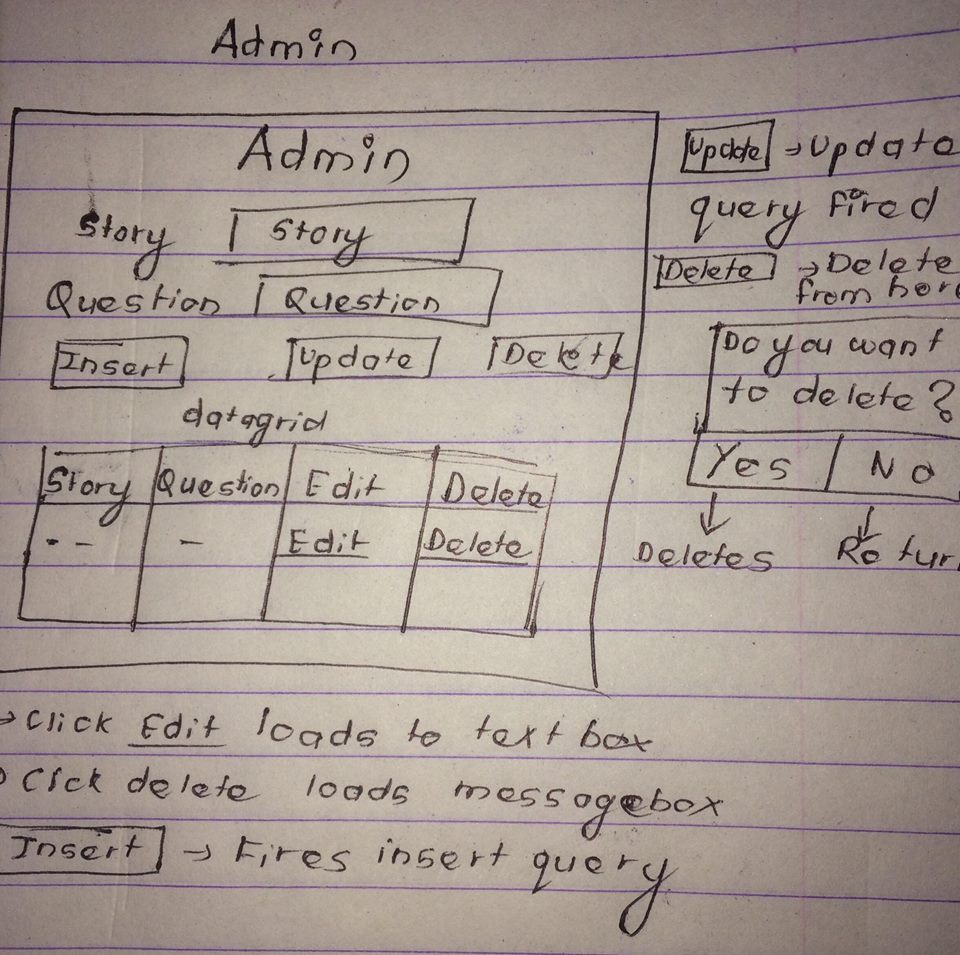


Figure 70: Admin panel

**Methods**

* In the admin panel on clicking **Edit,** the elements of the corresponding rows on data-grid are loaded onto the text box. We can either edit those values or delete it.
* Also, by typing values in the text box, we can easily Insert questions and stories by pressing Insert which invokes the insert query.
* Also, on clicking **Delete,** the corresponding elements in the data-grid would be deleted after clicking Yes on the appeared Message Box.
* The values such as username and password are to be validation for Login and for exception, try-catch is essential. If wrong username or password is provided, the fields are cleared with a message-box stating to insert correct ones.
* None of the fields are to be left blank during registration and this is addressed through message-box. Also, default values of module and story ID (save ID) are inserted.
* Stories with respective questions are loaded via story ID being foreign key with (1-N) relationship. Questions are to be loaded to radio button created dynamically.
* On clicking Next, next story is loaded through column next phrase ID in question entity.
* On clicking Save, story and module ID’s are updated in save table and loaded by clicking Load.

**User experience**

* Paper prototyping ensured vast user complication didn’t occur but every application has a room for improvement in the sooner updates.
* Users felt instead of clicking Next for next story, we could directly load new story on clicking the radio-buttons.
* Further modules might be added with more number of questions so that users could play for a long time and go for other modules simultaneously.
* At the admin level, use of Delete button wouldn’t be essential.
* Users also felt auto-save features might be added onto the program.

**Improvements and Conclusion**

As per user experience, changes would be brought in the upcoming versions of the software. The auto-save feature along with auto-story load features would be the major target. Also, the Delete button in the admin panel would be made obsolete. For user satisfaction, a better graphics for a better gameplay would be introduced in the upcoming version with major transitions providing the cutting edge user interface. Hence the program meets the requirement assigned by Fun2Learn company and is beneficial for edutainment purpose. Improvements would be expected in upcoming versions.