Team 7

William Dickinson -

Design and Deployment Documentation

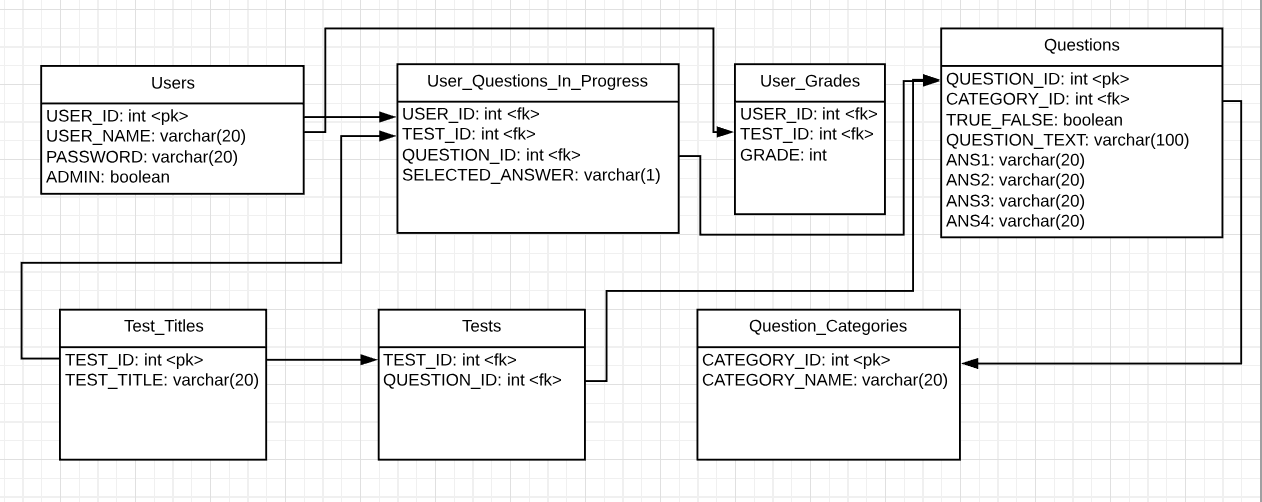
1. **Overview**

Our goal for this project was to create a web application that could be used to take and create tests similar to the blackboard website. The application would grant users a variety of abilities dependent on whether or not they were admins. For example, non-admins choose tests to take, check their past grades, etc. Admins could create and edit tests, change regular user accounts into admins, etc.

For our overall design of the program we chose to use Azure SQL Server to create a database. We used C# with .NET with an MVC model arrangement in order to handle the logic and front end of the application. We then used Azure to host the application online.

1. **Database**

For creating and debugging the database we used the built in Azure query editor. Below is a UML diagram of the completed database(See Appendix for source code).



The Users table held a list of all the user names and passwords required to log into the program, and it also kept a boolean for admin that was used for keeping track of whether or not the user was an admin. The USER\_ID was created later to make adding new users easier later.

Next would be the Question\_Categories table which keeps track of each type of question. The purpose of this table was to make searching for questions based on their type easier.

Then there’s the Questions table which is where the actual question rows are stored. Each row of this table holds a different question with columns for its answers, category, and whether or not it is a true or false.

The Test\_Titles Table was a list of test names each associated with a unique Test\_ID. This provided an easy means of creating a list of Tests that could be used later to look through and print out for further use.

The Test Table should probably have been named Test\_Questions instead, as it is really a list of question ids which are all associated with the test\_id they have been assigned to. So each test that the application creates could have many rows in the Test table associated with it, each representing a test question.

The User\_Questions\_In\_Progress table works similarly to the Test table in that is is essentially just a list of questions. In addition to the question ids being associated with a test though they are also associated with a user id and there is a column in each row specifically for storing their unsubmitted answers. The idea for this table is that upon starting up a new test a row would be created in this table for each question in the given test. Then as the user progressed each row would be updated with their answers until they submitted it. The rows associated with the user taking the test once they had submitted their test for grading.

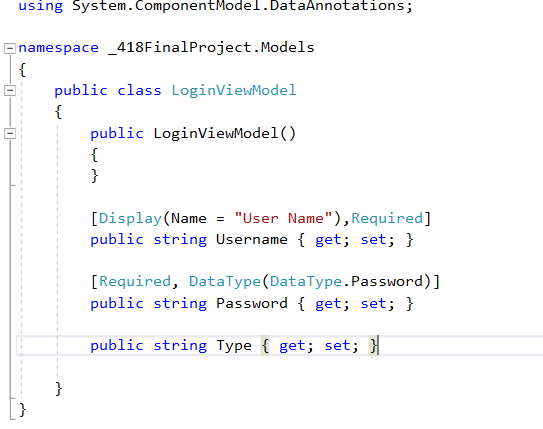
Finally there was the User\_Grades table which was just a table for storing the grades of completed tests for each user. There was only one of these tables which means that for each grade each user took there would be an additional row. For example if user\_a had 3 graded tests and user\_b had 2 graded tests there would be 5 rows in the table in total.

1. **Business Logic/Front End**

For the business/front end portion of the application we used an MVC(Model View Controller) design. This allowed us to manage the database pretty easily while also creating a nice looking front end for the application. The general idea behind the MVC design is to divide the code for a .NET application up into models which determine how data is stored and handled, views which determine how the application will look to a user, and controllers which handle the main bulk of the business logic of the application.



In order to design the user login we created the view above for the login page that would take two string input fields, the Username and the Password. Upon the user clicking enter or the Login button the Username and the Password would be contained in the form of the model in the picture below and sent to the controller(see appendix).



Upon receiving the model object which contains the Username and Password data the user put in the UserLogin method runs a SQL check with the Azure database to check to see if there is a row in the database which matches the Username and Password. If it does then the method returns a link to the admin or non-admin webpage where they can continue using the application. For the time being they are simply kicked from the application if their username and password aren’t in the database.

The other methods in the business logic work under similar logic. The view collects the data from the user, packages it in the form of a model, and sends it to a controller which usually uses the data to read, create, or remove data from the database, and then switch to a different webpage.

1. **Deployment**

Deploying the application was probably the easiest and simplest part. All it involved was creating a web application resource in Azure and then right clicking on the folder which contained the application and selecting publish. After that it could be accessed by clicking on the hyperlink with any device connected to the internet.

**APPENDIX:**

**Controller Login Method Source Code**

[HttpPost, ValidateAntiForgeryToken]

public IActionResult UserLogin(LoginViewModel model)

{

SqlConnectionStringBuilder builder = new SqlConnectionStringBuilder();

builder.DataSource = "testtaker.database.windows.net";

builder.UserID = "user";

builder.Password = "Password1";

builder.InitialCatalog = "TestTaker";

using (SqlConnection connection = new SqlConnection(builder.ConnectionString))

{

connection.Open();

String sql = "SELECT USER\_NAME, PASSWORD, ADMIN FROM dbo.Users WHERE USER\_NAME = '" + model.Username + "';";

using (SqlCommand command = new SqlCommand(sql, connection))

{

using (SqlDataReader reader = command.ExecuteReader())

{

while (reader.Read())

if (reader.GetString(0) == model.Username && reader.GetString(1) == model.Password)

{

if (reader.GetBoolean(2) == true) return Redirect("/Admin/index");

else return Redirect("/NonAdmin/Index");

}

}

connection.Close();

}

}

return NotFound();

}

**Database Source Code:**

CREATE TABLE Users(

USER\_ID int PRIMARY KEY,

USER\_NAME varchar(20) NOT NULL,

PASSWORD varchar(20) NOT NULL,

ADMIN BIT NOT NULL

);

INSERT INTO Users VALUES(0,'user\_loser','pass',0);

INSERT INTO Users VALUES(1,'user\_admin','pass',1);

//For ADMIN, 0 = regular user, 1 = admin

CREATE TABLE User\_Grades(

USER\_ID int,

TEST\_ID int,

GRADE int,

FOREIGN KEY(USER\_ID) REFERENCES Users(USER\_ID),

FOREIGN KEY(TEST\_ID) REFERENCES Test\_Titles(TEST\_ID)

);

INSERT INTO User\_Grades VALUES(0,1,50);

CREATE TABLE User\_Questions\_In\_Progress(

USER\_ID int,

TEST\_ID int,

QUESTION\_ID int,

SELECTED\_ANSWER varchar(1),

FOREIGN KEY(USER\_ID) REFERENCES Users(USER\_ID),

FOREIGN KEY(TEST\_ID) REFERENCES Test\_Titles(TEST\_ID),

FOREIGN KEY(QUESTION\_ID) REFERENCES Questions(QUESTION\_ID)

);

//Here is where all the in progress test data will be stored. This will be a single table that will keep track of all the questions

//that have been answered on incomplete tests.

//Of important note: the SELECTED\_ANSWER can be set to 'A','B','C', or 'D' for multiple choice or 'T' or 'F' for true or false.

//Also, if it is left null that means that the answer wasn't filled in, although in that case the question won't even need to be stored.

CREATE TABLE Question\_Categories(

CATEGORY\_ID int PRIMARY KEY,

CATEGORY\_NAME varchar(20),

);

INSERT INTO Question\_Categories VALUES (1, 'Ducks');

INSERT INTO Question\_Categories VALUES (2, 'Math');

CREATE TABLE Questions(

QUESTION\_ID int PRIMARY KEY,

CATEGORY\_ID int,

TRUE\_FALSE BIT NOT NULL,

QUESTION\_TEXT varchar(100) NOT NULL,

ANS1 varchar(100) NOT NULL,

ANS2 varchar(100) NOT NULL,

ANS3 varchar(100) NOT NULL,

ANS4 varchar(100) NOT NULL,

FOREIGN KEY(CATEGORY\_ID) REFERENCES Question\_Categories(CATEGORY\_ID)

);

INSERT INTO Questions VALUES(

1,

1,

0,

'What sound does a duck make?',

'The walls have eyes',

'Pikachu!',

'Kermit the frog here!',

'\*Quack!'

);

INSERT INTO Questions VALUES(

2,

1,

1,

'Does Donald Duck wear pants?',

'',

'',

'',

''

);

//If there is no asterix before the question in a true/false question text then the answer is false.

CREATE TABLE Test\_Titles(

TEST\_ID int PRIMARY KEY,

TEST\_TITLE varchar(20),

);

INSERT INTO Test\_Titles VALUES (1, 'Duck Test #1');

INSERT INTO Test\_Titles VALUES (2, 'Math Test');

CREATE TABLE Tests(

TEST\_ID int,

QUESTION\_ID int,

FOREIGN KEY(TEST\_ID) REFERENCES Test\_Titles(TEST\_ID),

FOREIGN KEY(QUESTION\_ID) REFERENCES Questions(QUESTION\_ID)

);

INSERT INTO Tests VALUES (1, 1);

INSERT INTO Tests VALUES (1, 2);