Compass Surveys

Technologies Used

Front-end: The requirements specify that the code components should be reusable therefore for this feature the front end makes use of React JS framework. React JS supports code reusability by creating and reusing generic components. Also, the use of Virtual DOM which can track only the parts of DOM that are changed and render it also makes it a preferred choice as it significantly improves the performance of the web application compared to other frameworks.

Dependencies: Node Js, npm

Back-end: The application backend is written using C# ASP.NET MVC entity core framework. C# is a type-safe language which prevents data loss and entity core framework is a cross-platform lightweight framework which supports Object relation mapping (ORM) to visualize data in the form of classes and objects and interact with data in an object-oriented way.

Dependencies: dotnet sdk, Visual Studio, Entity Core framework package

Sql & Back-end Design

The following feature consists of three entities:

1. Surveys which are represented by survey names
2. Questions where each survey can have 1 or many questions to answer
3. Options where each question can have 1 or many options to select from

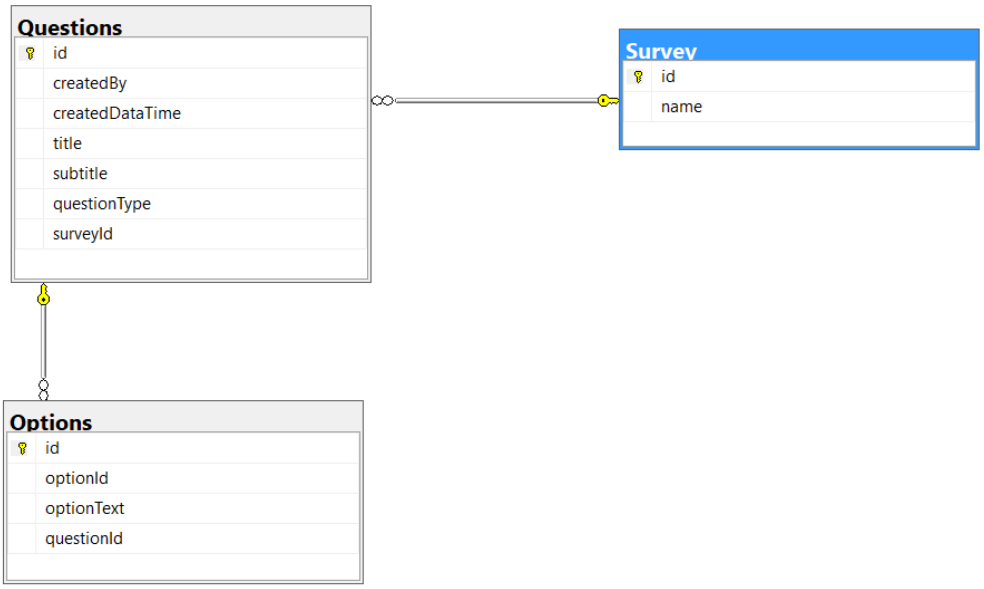

Fig 1: Database structure diagram

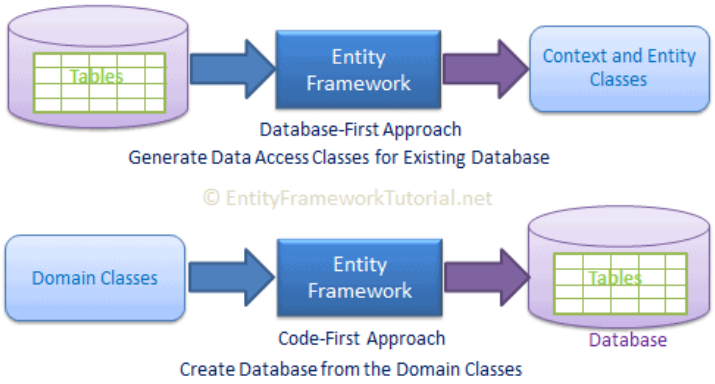

Fig 2: Entity framework design

Entity Relationships

1. 1: many relationship between Survey and Question
2. 1: many relationship between Question and Option

Considering these 1: many relationships each question must be associated to a survey and each option must be associated with a question. Therefore, the question entity has a foreign key which is the survey Id and the option entity has a foreign key which is the question Id field.

Use case scenarios

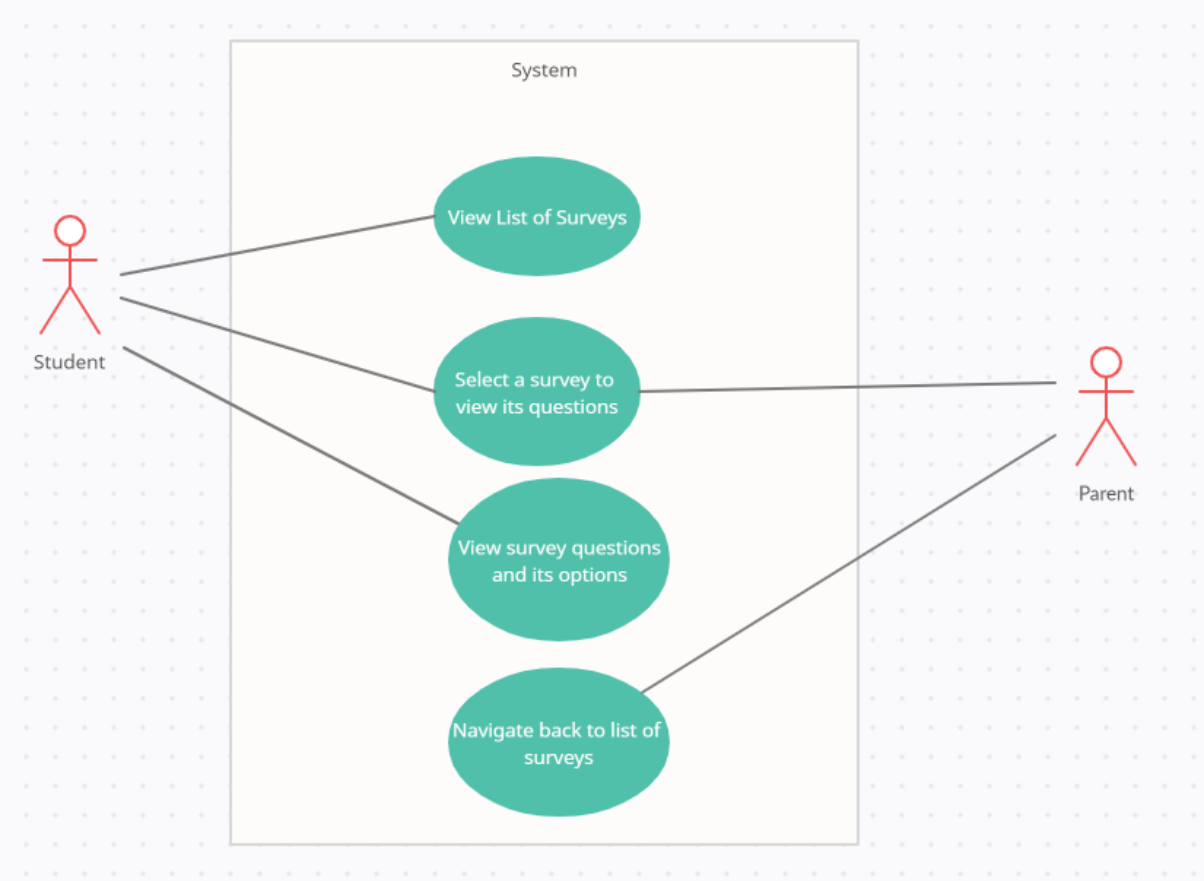

Fig 3: Summary of user stories

For the use case scenarios listed above, the main functionality of the compass survey feature is to fetch and display survey related data. When users navigate to the home page data they should first be able to see the list of surveys they can respond to. They should be able to select a survey and view the questions of that survey along with its options. Thus, on navigating to a survey the system should provide the question and its options data. This data must be listed in a new page. Finally, they should have an option to navigate back to view the list of surveys by clicking the back button.

API Design

Using Entity core framework, the project consists of three models namely:

1. Survey
2. Questions
3. Options

The Compass DB context interacts with the database to return the model data requested by the controller. Considering the use case scenarios described above, the model controller implements the following HTTP methods:

HTTP Method	Method URL	Purpose
GET	api/Surveys	Fetch the list of surveys
GET	api/Surveys/:id	Fetch the question and options data for selected survey
GET	api/getAllSurveyData	Fetch the list of all surveys and its associated data
POST	api/Surveys	Add a new survey to the DB
PUT	api/Surveys/:id	Updates a selected survey in the DB
DELETE	api/Surveys/:id	Deletes a selected survey from the DB

Fig 4: API Design table

In the context of this application, the top three API methods are required for the purpose of retrieval and viewing data. The post, put and delete methods would be used in case if the application requirements are extended to add/update/delete more surveys in the database using forms.

There are two options for retrieval of data:

1. Load data per survey: This method will first retrieve the list of surveys to show on the front end and after user selects a survey it will load the questions and options data for that survey.
   Pros
   - Works relatively better than loading all data at once by only loading requested data. Method can be useful if the data size of all surveys information is high to avoid performance bottlenecks.

   Cons
   - Adds an extra client request so database needs to be accessed again for retrieving information
2. Load all data: This method will retrieve all the data from the database which can be consumed by the application

Pros

- Saves the number of requests from clients by loading everything in one go
- Useful to use if the data to be loaded is has less size. Ideally can be used where number of surveys is less and if surveys contain mostly text related data.

Cons

- This approach can cause performance issues if the data size is high and application is receiving many requests concurrently. Assuming if the application adds a lot more surveys and introduces some image related data to represent the questions/ options.
- Back navigation to load list of surveys may cause performance bottlenecks by loading the same data again.

Tests Design

Automation Testing

The client react application makes use of the Jest and enzyme testing framework to debug testing scenarios related to the components.

Unit Testing – Front end

Unit testing requires to check if the individual components of the application are rendering and working properly. The front end react application consists of the following components:

1. App Component – Parent component that renders all child components inside it.
2. View Survey component – Child component that reads the list of surveys from the database and displays it in a list format
3. View Survey detail component – Child component that reads the questions and options from the database and displays it with a back button to navigate back to surveys page
4. Question component – Child component inside view survey detail component that takes data from its parent which is the list of questions and the options in each question
5. Feedback component – Child component that is reused to display appropriate feedback to the user depending on constraints related to the API data.

The test cases make use of combination of snapshot and DOM testing to verify the working of components.

Unit Testing – Back end

Unit testing requires to check if the controller is working properly. Basic tests are conducted to check if the HTTP API methods of the controller work as expected to verify that the data being sent to the front end as well as stored in the backend database is correct. The backend testing framework uses a combination of Xunit and Moq testing libraries for testing.

Manual Testing

This type of testing requires to manually test the application by running it and doing some manual navigation to check if the basic level integration and unit testing is working. For the compass surveys app, the system is checked for the following scenarios:

Test case	Test Data	Expected Result	Actual Result	Pass/Fail
Start and run the application		Application starts and loads the home page	User sees the home page	Pass
Survey list screen component test	Survey list Survey= [{name:"survey 1"}]	Application should show a list with option to select Survey 1	User sees the list of surveys and can select Survey 1 to view details	Pass
Survey list screen component test-Navigation	Survey list data is loading and not loaded yet State.loading = true	Application should show an interim screen that data is loading to give feedback to the user that application has not crashed	User sees an interim page which shows data is loading	Pass
Survey list screen component test	Survey list data loaded but there are no surveys	Application should give feedback to the user that there are no surveys to view	User can see a feedback page with the appropriate message	Pass
View Survey detail component test-Navigation by clicking survey item	Question list [{title:"1+2=?"}]	On selecting a survey user should see survey questions	Clicking a survey displays a new page with survey details and questions	Pass
View survey detail component – Question interaction	Question list [{title:"1+2=?"}]	User must be able to select one option out of the list of options given for a question	User can select a question and the radio button is highlighted to select a response	Pass
View Survey detail component test-Navigation	Question list = [{}] or no question list data and state.loading = false	Application should give feedback to the user that selected survey doesn't have any questions loaded and give option to navigate back to survey list	User can see a feedback page with the appropriate message	Pass
View survey list component – Click on back button	Survey list = [{}] or no survey data and state.loading = false	If no surveys loaded with feedback screen shown and user clicks back, then the application should go back to	On clicking back, user can see the home screen	Pass

		display the home screen		
View Survey detail component – Click on back button	Question list = [{}] or no question list data and state.loading = false	If no questions loaded with feedback screen shown and user clicks back, then the application should go back to display the page showing list of surveys	On clicking back, user can see the list of surveys	Pass

Fig 5: Manual test cases table

Directory Structure

This solution follows the MVC pattern with views handled by the ClientApp folder. The application contains the CloudDBModels and models folder for storing database related model classes. All related controller classes are stored in the controller folder. The controller interacts with the view and model for passing and receiving data. A separate project is created for unit testing purposes.

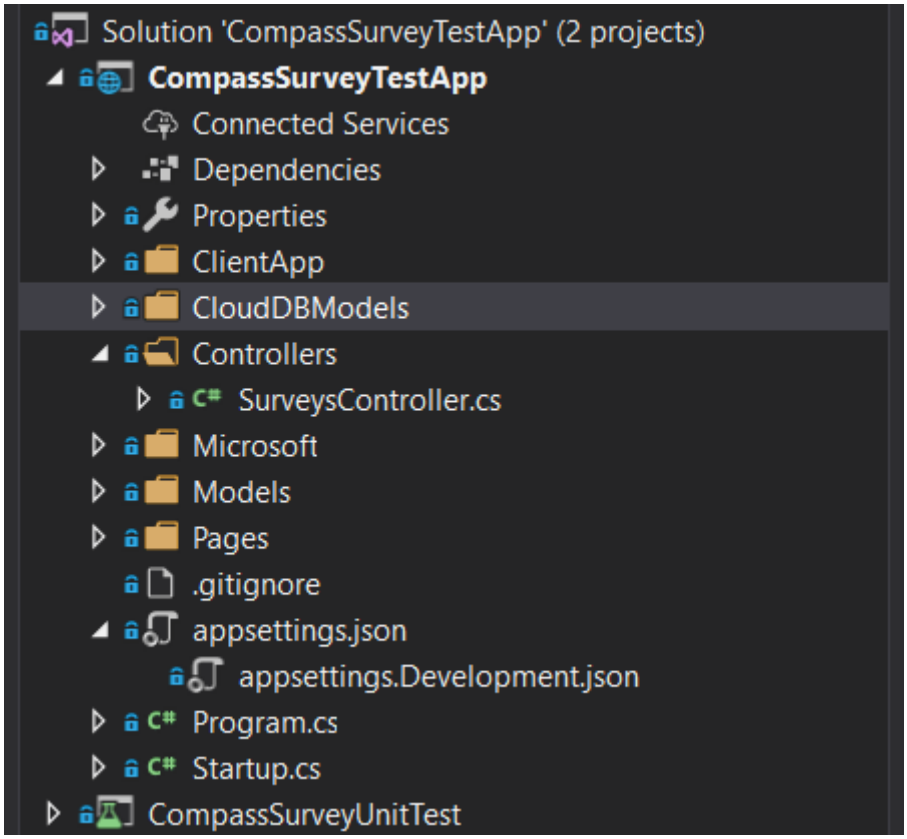

Fig 6: Project directory structure