**2.**

* Added a new table to add ratings for a question named ‘question\_rating’ which has a primary key: ‘id’ and two forign keys: ‘user\_id’, ‘question\_id’.
* Added a new table named ‘answer\_rating’ to add ratings for an answer which has a primary key: ‘id’ and two forign keys: ‘user\_id’, ‘answer\_id’.
* Created ‘api/Users.php’, ‘api/Users.php to manage REST API endpoints and there are three controllers named ‘Questions’, ‘Answers’, ‘Users’ to load views in ‘application/controllers’ folder.
* Added a method in ‘api/Users.php’ controller to check whether an username is already in the database.
* Added methods to ‘Questions\_Model’ model and ‘Questions’ controller to accommodate following functionalities,
  + Get all questions in the database.
  + Create a new question
  + Get a question using question id
  + Get all answers for a question
  + Create an answer for a question
  + Add rating to a question
  + Add rating to an answer

In my opinion, the changes made to the UI, database, controllers, models, etc., were useful in the development phase. Adding a new table for question ratings and answer ratings allows for more accurate tracking of user engagement and feedback. The creation of REST API endpoints and controllers for managing them, enables easy access and management of the data. Having a method to check for unique usernames in the database is an important feature for user registration.

The methods added to the 'Questions\_Model' model and 'Questions' controller provide functionality for getting all questions and answers in the database, creating new questions and answers, adding ratings to questions and answers, and getting a question using question id. These functionalities are important for the smooth functioning of a question-answer platform.

If I were to start again, I would have paid more attention to the naming conventions of the tables, controllers and models. I would also have made sure that the controllers and models are in a more modular structure which would have made it easier to scale in the future.

**3. Q1:**

The first reason the author provides for not using web frameworks is that they add unnecessary complexity to the codebase. The author argues that frameworks often come with a lot of features that may not be needed for a specific project, but still need to be understood and navigated by developers. This can make it harder for new developers to understand and work on the code and can also lead to increased maintenance costs.

The second reason the author provides is that frameworks can limit flexibility. The author argues that by using a framework, developers are forced to conform to its conventions and patterns, which can make it difficult to customize the application to specific needs. Additionally, frameworks can make it harder to switch to a different technology stack in the future.

The third reason the author provides is that frameworks can lead to a decrease in performance. The author argues that the overhead of the framework can slow down the application, and that the abstraction provided by the framework can make it harder to optimize specific parts of the application for performance.

The author supports these reasons by citing examples from his personal experience of working with various web frameworks and encountering these issues. He also mentions that frameworks can have their use cases, but it's important to consider if the particular use case is worth the trade-offs of using a framework. The author suggests that it may be better to use a minimalistic, un-opinionated library or build the application from scratch if the project is small and simple, in order to avoid the complexity, inflexibility and performance issues that frameworks can bring.

**3. Q2:**

The most important reason the author gives against using web frameworks is that they add unnecessary complexity to the codebase. The author argues that frameworks often come with a lot of features that may not be needed for a specific project, but still need to be understood and navigated by developers. This can make it harder for new developers to understand and work on the code and can also lead to increased maintenance costs.

I agree with the author's argument that frameworks can add unnecessary complexity to the codebase. The use of a framework can often lead to a larger codebase that is more difficult to navigate and understand, which can make it harder for new developers to get up to speed and can also lead to increased maintenance costs. Additionally, the use of a framework can often lead to a lot of boilerplate code that may not be needed for the specific project, which can also add unnecessary complexity.

However, I also believe that frameworks can have their use cases and can be beneficial in certain situations. For example, frameworks can provide a lot of useful functionality out of the box and can make it easier to develop an application quickly. Additionally, frameworks can provide a consistent structure and a set of conventions which can make it easier for developers to work together and can also make it easier to maintain the codebase.

In conclusion, I agree with the author that frameworks can add unnecessary complexity to the codebase, but I also believe that frameworks can have their use cases, depending on the specific project and the needs of the development team. It's important for developers to weigh the pros and cons of using a framework and to decide if the added complexity is worth it for the specific use case.