King Fahd University of Petroleum & Minerals

College of Computing and Mathematics

SWE 363: Web Engineering and Development (Term 232)

Homework Assignment #4

Submission Guidelines

- It is an individual assignment.

1. Use your work on Assignment 3 and add server-side scripts using Node.js to produce a dynamic webapp like the one in this video.
2. Install node and MongoDB on your machine.
3. Create directory structure in the backend directory as shown in Figure 1.
4. Move your CSS file, frontendJS.js, and all the images to the public folder.
5. Install the following packages: express, nunjucks, mongoose, bcrypt, and cors.
6. Run MongoDB and connect to it using mongoose. Export the connection object from the db.js module to use it in the model in the next step.
7. Model:
    a. Create a database named as your KFUPM username i.e. s20xxxxxxx then create three collections: questions, answers, and users. Refer to Figure 2 for hints on the database schema. Store that in the `db_schema.js` module.
8. Controller:
    a. In `questions_controller` module, import the `db_schema` models then create and export the following asynchronous functions: `getAllQuestions()`, `createQuestion(questionData)`, `getQuestionById(questioId)`, `addAnswer(questionId, answerData)`. These functions should use the database models to do their task. For addAnswer, create an answer object in the database and append it to the answers array of the question with the given id.
    b. In the `answers_controller` module implementation and export the following asynchronous function `updateAnswer(answerId, method, questionId)` where the `method` can be: `'upvote'`, `'downvote'`, or `'accept'`. For upvote and downvote increment/decrement the `votes` field of the answer. For the accept method update the `accepted` field of the answer and set the `answered` field of the question using their ids.
9. Routers:
    a. In `routers/questions.js` module export an express `router` that handles the following routes using the functions implemented in the `questions_controller`.

Route	Handling

Get /	Respond by a JSON representation of the getAllQuestions() return.
Post /	Respond by a JSON representation of the createQuestion return.
Get /:id	Respond by rendering question.njk passing the return of getQuestionById function.
Post /:id	Call addAnswer then redirect the user to /questions/:id

   b. In the `answers_router` module, export an express router that responds to Get /:answerId by calling the `updateAnswer(answerId, method, questionId)` and redirect the user to `/questions/:questionId`. Note both the `method` and `questionId` are query parameters.

10. Views
    a. The layout.njk is the parent of all other templates, it holds the common parts between the templates and creates three placeholder blocks: content block for the main content of the page, aside block, and scripts block for in-html JavaScript if needed.
    b. The `index.njk`: extends the layout template and overrides the content and aside blocks. You may override the scripts block to call the `getQuestions` of the frontendJs depending on how you implemented it.
    c. The `question.njk`: extends the layout template and overrides the content and aside blocks. It receives the `question` object and use it to populate the template. It uses loops to display the question tags and answers. It checks if there are answers, otherwise displays No answers yet. It creates forms for upvoting, downvoting, and accepting the answer. The upvote form submits using Get to `/answer/{{ answer._id }}?method=upvote` and the others follow a similar approach. The acceptance form is only rendered if the question answered property is false. It also creates a form for adding a new answer to the question. this form contains a hidden field that sets the `answeredBy` to `Anonymous` and submits a post request to the `/questions/:questionId`.
11. Server.js: import the modules and create an `express` app, configure it to use the `cors` middleware, to use `nunjucks` as a template engine to render the templates in the `views` directory, to serve `static` files in the `public` directory, accept and parse JSON and url-encoded forms, render the index template for Get /, use the answers_router for any request to '/answer', use the questions router for requests to 'questions', and listen on port 3000.
12. In frontendJs in displayQuestions make the question title links to /questions/:questionId.

Deliverables:

- A compressed file containing all the content of the backend folder EXCEPT node_modules

Useful Links

- https://www.geeksforgeeks.org/mongoose-populate-method/
- https://www.geeksforgeeks.org/mongoose-virtuals/?ref=ml_lbp
- https://www.geeksforgeeks.org/how-to-use-findoneandupdate-in-mongoose/?ref=ml_lbp

- https://attacomsian.com/blog/mongoose-increment-decrement-number
-

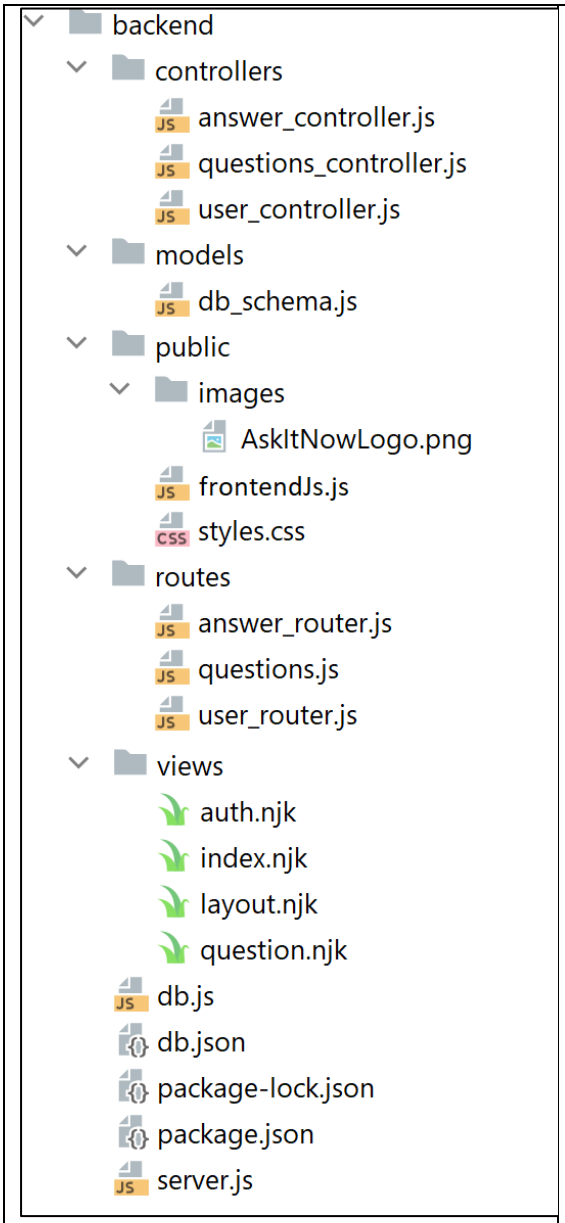

Figure 1 Directory Structure

```
const answerSchema =  mongoose.Schema({
  text: String,
  addedAt: {type: Date, default: Date.now()},
  answeredBy: String,
  votes: {type: Number, default: 0},
  accepted: {type: Boolean, default: false}
});
const questionSchema =  mongoose.Schema({
  title: String,
  description: String,
  tags: [String],
  createdAt: Date,
  answers: [{type: mongoose.Schema.Types.ObjectId, ref: 'Answer'}],
  answered: Boolean,
  author: {
    id: Number,
    username: String
  }
});
const User = mongoose.model('User', userSchema);
const Question = mongoose.model('Question', questionSchema);
```

Figure 2 Database Schema