Recitation 6

MongoDB and Mongoose

Need for a database

- How can we store data **persistently?**
  - Robust against server shutdowns/crashes, user clearing of browser cache, etc.
- Other considerations for data storage:
  - Modifiability: create/modify/delete data concurrently
  - Extensibility: easily redefine data attributes for organization
  - Searchability: quickly find and deliver data in a usable format
- Databases are designed to satisfy all of these qualities!

Document-based databases

- Subset of NoSQL databases
- Stores all information for a given object in a single **document** in the database
- Important feature: data formats are often not predefined
- Database's API or query language allows you to retrieve documents based on content or metadata

Object Relational Models (ORMs)

- Alternative way of interfacing with a database system besides native query language
- ORM **models** (represents) the website's data as JavaScript objects and maps them into the underlying database
  - Benefits:
    - Familiarity
    - Easy validation
  - Detriments:
    - Performance

MongoDB + Mongoose

- MongoDB: document-oriented database
  - Querying
  - Indexing
- Mongoose: ORM for MongoDB
  - Abstracting database connections
  - Schemas (for structure)
  - Data validation
  - Virtual properties (not stored; "dynamic")
- Very popular combo in the Node community due to JSON familiarity

Recitation Code

- https://github.com/61040-fa22/rec6
- Three types of data:
  - Assignment
  - Submission
  - Student

MongoDB Atlas Setup

- Follow instructions: https://docs.google.com/presentation/d/1HJ4Lz1a2IH5oKu21fQGYgs8G2irtMqnVI9vWDheGfKM
- Add copied connection string and password to `.env` file
- Run `npm start` to see if you can connect

Code Review: `Assignment` Schema

- **Introduction to Schema and Model**
  - **Schemas define models**
  - Schema = defining fields on document with types + validation
    - Example: `dueDate` attribute contains Date value (when assignment should be submitted by)
  - Models = interface for documents (finding, creating, modifying)
    - Example: Assignment.findOne(), new Assignment()

Code Review: `Submission` Schema

- **New feature: Data validation**
  - Purpose: Validate data for a field before it gets inserted/updated in the database
  - Examples:
    - `score` attribute rejects all negative values with error message 'Score cannot be negative'
  - Other examples:
    - `Assignment.name` (must start with Fritter)
    - `Student.year` (no alumni or prefrosh)

Code Review: `Assignment` Schema

- **New feature: Virtuals and virtual population**
  - Virtuals: Contain values computed from other attributes that aren't actually stored in the document itself
    - Great way to do synchronizations!
    - Example: `submissions` field contains all Submissions associated with this Assignment

Code Review: `Submission` Schema

- **New feature: Object references and population**
  - Purpose: Denote that attribute contains value of some type we have a schema for
  - Examples:
    - `assignment` attribute stores the Assignment this submission is for
    - `author` attribute stores the Student who made this submission
  - OP: Reference a document by its ID. If developer needs to fetch its value, **populate** it to retrieve its data
    - Also see `findAll()` in `index.ts`

Code Review: `Student` Schema

- **New feature: Getters and setters**
  - Purpose: Execute custom logic when getting or setting a property on an object
  - Examples:
    - `name.first`, `name.last`, `name.middle` setters set value to capitalize first letter of a given name part, regardless of case of input
    - `name.middle` getter always outputs capitalized initial only despite storing full middle name

Exercise: `Student` Schema

- Make a new **virtual** `name.full` on the Student schema computing a student's full name
  - Setter to split parameter into first, middle, last name parts
  - Getter to return conjoined string of first, middle, last name parts

Searching for documents

- How? `<Model>.find()`
  - `<Model>` should be replaced with Assignment, Student, etc.
  - Provide object properties to filter by attribute criteria
  - `{}` = find everything
  - `{attr: val}` = find only documents with value `val` in attribute `attr`
- **Exercise: Find all upperclassmen students**
  - Hint: seniors = 2023, juniors = 2024

Creating new documents

- How? Instantiate a new class of `<Model>`
  - See `repopulate()`, we do a lot of that there...
- Call `save()` on the instantiated model to insert it into the database
- **Exercises:**
  - Add a student named Daniel Nicholas Jackson Jr. with class year 3000 (what happens?)
  - Add a submission by author Daniel Nicholas Jackson Jr. for assignment Fritter Diverge, current date, with no score.
    - Hint: Find appropriate assignment and student documents so you can reference them!

Modifying documents

- **Exercise:** Grade Daniel Nicholas Jackson Jr.'s submission for Fritter Converge with score 10 by modifying his previous submission.
  - Multiple approaches:
    - findOneAndUpdate()
    - find the right submission document, set its properties correctly, call `.save()` on it

If we have time

- Suggest your own document queries to livecode
- Anything you guys want to implement but aren't sure how to do?

Resources

- https://mongoosejs.com/docs/guide.html
- https://mongoosejs.com/docs/api.html
- https://mongoosejs.com/docs/typescript.html
  - Tip: Use the search bar
- Very nice guide: https://developer.mozilla.org/en-US/docs/Learn/Server-side/Express_Nodejs/mongoose