Recitation 6 - MongoDB and Mongoose

Introduction

- As seen in previous recitations, we need for a mechanism to store data that can **persist**
    - What does it mean to persist? Answer: Preserve state and retain data even after access to it is lost
    - Needs to be unaffected by server shutdowns/crashes, user clearing of browser cache, etc.
- Databases are designed to efficiently store data in an easily extensible, modifiable, and searchable way
    - Qualities
        - Extensible: can easily define and redefine data attributes for organization
        - Modifiable: stored data can be created/deleted/edited concurrently
        - Searchable: optimized to quickly query and deliver usable data
    - Tons of research into database design to optimize for these qualities; take 6.814 for more info
- Technologies used for persistent storage in this class:
    - MongoDB: document-oriented database
    - Mongoose: library abstracting MongoDB interactions into a familiar JSON interface

Document-oriented Databases

- Subset of NoSQL databases
- Stores all information for a given object in a single **document** in the database
    - Documents are addressed in the database via a unique **key** (ID) representing that document for faster search and retrieval
    - Relies on internal structure in the document in order to extract metadata for optimization
- Since info is all stored in same document, no additional work is needed to find related data when you query for a specific attribute of that document
- Important feature: data formats are often not predefined

    - Any sort of document can be stored in any database and can change in type and form at any time
    - Example: A new field can be added to new documents as they are inserted, but will not be added to documents already stored
    - This allows for greater flexibility and rapid development, but compromises on consistency
- Database's API or query language allows you to retrieve documents based on content or metadata

ORMs

- Object Relational Model (ORM) is one way of interfacing with a database system
- An ORM **models** (represents) the website's data as JavaScript objects and maps them into the underlying database
    - Benefits:
        - Can write code that virtually looks like native code and offers more familiarity to programmers than a query language
        - Provide an easy way for you to validate data schema (structure)
    - Detriments:
        - Performance is generally worse since they use translation code to map between objects and the database format, which may not use the most efficient database queries

MongoDB and Mongoose

- MongoDB: document-oriented database
    - Features:
        - Querying: Can search according to a specific field, a range of values, and even regular expressions!
        - Indexing: Any field can be set at the primary or secondary key(s); even ones containing objects, not just primitive values
            - Can utilize this to mark combinations of fields for uniqueness!
- Mongoose: ORM for MongoDB
    - Features:
        - Database connections: Abstracts a lot of the work behind connecting to a MongoDB server and maintaining that connection

        - Predefining structure: Can preemptively define the structure of inserted documents with **schemas** by declaring fields and their **types,** and create **models** (searchable collection of documents) out of them
        - Data validation: Enforces the acceptable range of field values in documents, and can define error message in event of validation failure
        - Virtual properties: Allows you to construct/deconstruct a new property from stored fields, instead of storing them every time
- Very popular combo in the Node community because document storage and query system looks very much like JSON and is thus familiar

In-class activity (40 min)

Explain schemas/models given in starter code (10 min)

- Assignment
    - name (string)
    - points (number)
    - due date (date)
    - submissions (virtual population)
- Submission
    - assignment (Assignment)
    - date (date)
    - score (number)
    - author (Student)
- Student
    - First name (string)
    - Last name (string)
    - Middle name with:
        - Setter to capitalize first letter of middle name only
        - Getter to only return the capitalized first initial (e.g. Nicholas -> N.)
    - Full name
        - Setter to split parameter into first, middle, last name parts
        - Getter to return conjoined string of first, middle, last name parts
- Key points to cover:
    - Getters/setters
    - Validation

    - Virtual properties
    - References and population (references are IDs by default)

Live exercises (30 min)

- **Repo link:** https://github.com/61040-fa22/rec6
- Setting Up MongoDB Atlas
    - Connect to a MongoDB server with Mongoose
- Walk through each of the existing schemas/models (Assignment, Student, Submission)
- Make modifications to the following schemas:
    - Full name (virtual property computed from first, middle, last name)
- Searching for records
    - Find upperclassmen (hint: current seniors are 2023, juniors are 2024)
- Creating and modifying documents
    - Add a student named Daniel Nicholas Jackson Jr. with class year 3000.
    - Add a submission by author Daniel Nicholas Jackson Jr. for assignment Fritter Diverge, current date, with no score.
    - Grade Daniel N. Jackson's submission for Fritter Converge with score 10 by modifying his previous submission.
- Time permitting, ask students to suggest exercises to live code