Software Design - HW4
Group 2
GitHub: github.com/yoshadmun/VolunteerActivities

Design and Development Methodology:
Our methodology for this assignment has remained Agile Development. We continued using this development methodology because the modular design and incremental development of features helped us with collaboration by outlining and working on different modules simultaneously. This also allows us to gradually incorporate our frontend, backend, and database, and iterative development allows for continuous improvement as we go along with the project.

Database Technologies:

- MongoDB: The primary database technology used is MongoDB which is a NoSQL database. We used this because it is popular for its flexibility, scalability, and performance. Our code employs Mongoose which is an Object Data Modeling (ODM) library for MongoDB and Node.js. This provides a straightforward, schema-based solution.
    - Schema Definition: For example, the UserProfileSchema defines the structure of a volunteer's profile, including fields such as userId, fullName, location, skills, availability, assignedEvents, completedEvents, and active. Each field has a specified type and additional constraints, such as required or unique.
    - Connecting to MongoDB: Our code connects to a MongoDB instance using Mongoose. During testing, an in-memory MongoDB server is used to isolate tests from external factors and pre-existing data.

- Validations: Mongoose schema validations were used to ensure that the data being saved to the database adheres to specific rules and constraints. Validations are used for maintaining data integrity and preventing invalid data from being stored. These include required fields and unique fields where certain fields must have a value and be unique before a document can be saved. An example for this in our code is how userID and fullName are required, while userId which must be unique in UserProfile Schema.
    - Application level validations: Additional validations are performed in the controllers to ensure that operations are only carried out on valid and existing entities. An example of this is the isValidObjectId function which

checks if the eventId is a valid MongoDB ObjectId, and the existence of volunteer and event is verified before proceeding.

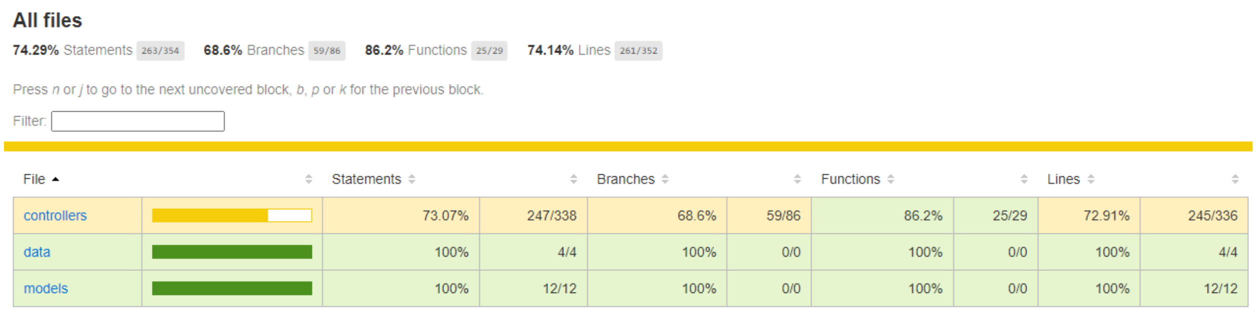
- Data Retrieval: This is implemented using Mongoose queries which allows for data fetching. For example, volunteers are fetched from the database based on search criteria and pagination parameters. The populate method is used to include related events.

- Data Persistence: This involves storing data to the MongoDB database so that it can be accessed. Events, volunteers, etc. are created or updated using Mongoose model methods such as save, findByIdAndUpdate, and findOneAndUpdate, and are then saved to the database.

Document Structure (NoSQL):

```
const mongoose = require('mongoose');

const eventSchema = new mongoose.Schema({
   eventName: {type: String, required: true, maxlength: 100},
   eventDescription: {type: String, required:true},
   location:{
      streetAddress: {type: String, required: true},
      city: {type: String, required: true},
      state: {type: String, require: true},
      zipCode: {type: String, required: true},
   },
   requiredSkills: {type: [String], required: true},
   urgency: {type: String, required: true},
   date: {type: Date, required: true},
   assignedVolunteers: {type: [String], default: []},
   active: {type: Boolean, default: true},
});

const Event = mongoose.model('Event', eventSchema);

module.exports = Event;

const mongoose = require('mongoose')

const notificationSchema = new mongoose.Schema({
   volunteerId: {type: String, required: true},
    message: {type: String, required: true},
    type: {type: String, required: true},
    date: {type: Date, default: Date.now},
});

const Notification = mongoose.model('Notification', notificationSchema);

module.exports = Notification;

const mongoose = require('mongoose');

const userProfileSchema = new mongoose.Schema({
    userId: {type: String, required: true, unique: true},
    fullName: {type: String, required: true},
    location:{
        address1: {type: String, required: true},
        city: {type: String, required: true},
        state: {type: String, required: true},
        zipCode: {type: String, required:true}
    },
    skills: {type: [String], required:true},
    availability: {type: String, required:true},
    assignedEvents: {type: [mongoose.Schema.Types.ObjectId], ref:'Event', default:[]},
    completedEvents: {type: [mongoose.Schema.Types.ObjectId],ref:'Event', default:[]},
    active: {type: Boolean, default:true}
});

const UserProfile = mongoose.model('UserProfile', userProfileSchema);

module.exports = UserProfile;
```

Code Coverage Report:



All files

74.29% Statements 263/354 **68.6%** Branches 59/86 **86.2%** Functions 25/29 **74.14%** Lines 261/352

Press *n* or *j* to go to the next uncovered block, *b*, *p* or *k* for the previous block.

Filter:

File ▲		Statements		Branches		Functions		Lines	
controllers		73.07%	247/338	68.6%	59/86	86.2%	25/29	72.91%	245/336
data		100%	4/4	100%	0/0	100%	0/0	100%	4/4
models		100%	12/12	100%	0/0	100%	0/0	100%	12/12

<u>Team Contributions:</u>

Group Member Name	**What is your contribution?**
Matin Amoozadeh	Require functions for mongoose and mongodb-memory-server
Thinh Pham	Majority implementation of MongoDB and connecting backend with database, and ran code coverage report
Shadmun Talukder Shahed	Users and Events populate method, and wrote the documentation

Note: Despite working on this assignment, we were unable to conduct any in-person meetings before the deadline. Thinh Pham handled the majority of the coding and implementation for this assignment.