Take 2

Backend 104

Databases & Web Servers

Databases

- DB = Database. Stores lots of data (1 kB - 1 TB)
- "NoSQL" DB = **no**t like the **SQL** databases. Advantages and disadvantages
- One item in DB often called:
  - "row" (SQL)
  - "document" (NoSQL)
  - "record" (either)
- Biggest difference in data format.
  - SQL = columnar = every record has the same fields.
  - NoSQL = document-based = every record can have different fields.

How to access data in DB?

- The objects are in the DB. Where do we want them instead?
- **Driver**: package that connects and talks to DB via a specific protocol (not HTTP) from JavaScript.
- **ORM**: Object Relational Mapper. Uses driver to output JavaScript objects and enable convenient access to records in database.

MongoDB & Mongoose

- MongoDB = Database
- Mongoose = ORM

```
bun add mongoose
```

```
const dbName = "my_database_development" // or "my_db_test" ...
const uri = `mongodb://127.0.0.1:27017/${dbName}`
await mongoose.connect(uri);
const userFromDatabase = mongoose.findOne(...);
```

CRUD actions & Mongoose

- For `GET /users` : `find`
- For `GET /users/3d31` : `findOne`
- For `PUT /users/3d31` : `update` - Use `{ returnDocument: 'after' }`
- `.lean()` converts from `mongoose` object to POJO
- POJO: Plain Old Javascript Object

Web Server 101

- What is a web server? What is an API server?
- How to structure our code?

Popular: MVC

- **Model**: interface to data storage, validations, associations
- **View**: output of data in formats like JSON, HTML, ...
- **Controller**: thin layer to coordinate models, views, and side effects like a sign up email

Model

- Schema used to describe how objects look in database.
- Model used to access data as objects in JavaScript.
- Validations, schema, associations are defined through model.

```
const ArticleSchema = new mongoose.Schema({
  title: { type: String, required: true },
  description: String
});

const Article = mongoose.model('Article', ArticleSchema);
```

Associations

- connect to other objects
- one→one or one→many or many→many

```
const CommentSchema = new mongoose.Schema({
  articleId: { type: mongoose.Schema.Types.ObjectId,
    ref: 'Article', required: true },
  text: { type: String, required: true },
});
```

- Here: one Todo required to be connected to each Attachment

View

- If you want to output HTML it's usually a template

```
<html>
  <head></head>
  <body>
    <h1>${todo.title}</h1>
  </body>
</html>
```

```
<html>
  <head></head>
  <body>
    <h1>My first Todo</h1>
  </body>
</html>
```

Content-Type / Accept

- `Accept` = what does receiver want
- `Content-Type` = what did sender send
- Same server, two "views" possible

`application/json`:

```
{
  "title": "My first Todo"
}
```

`text/html`:

```
<html>
  <head></head>
  <body>
    <h1>My first Todo</h1>
  </body>
</html>
```

Controller

- Collection of handler functions
- Often grouped by resource: `ArticlesController`, `CommentsController`
- Decides which view to render
- Might do business logic things like
  - "send the signup email when the User model is saved after signup"
  - "send a notification to Slack when a new support ticket was opened"

Handler example

- Routing parameters
- Find record/document in DB using model
- Render view according to `Accept` header

```
app.get('/article/:id', async (req, res) => {
  const article = await Article.findOneById(req.params.id); // ORM method
  if(req.accept == 'application/json') {
    res.status(200).json(articles);
  } else {
    res.status(200).send(`<html><body>${article.title}...`)
  }
});
```

Incoming `Content-Type`

- We know `application/json`
- How to upload large files (e.g. image) using JSON?

```
{
  "file": {
    "name": "when_are_these_slides_finished.jpg",
    "data": "....<two megabytes of encoded data>"
  }
}
```

- Can be done but cumbersome: have to use an encoding like base64.

Uploading files

- Use `multipart/form-data`!
- Can carry **multi**ple **part**s, one part image file, one part JSON, one part form data, ...

```
const fileInput = document.getElementById('my-file-input');
const nameInput = document.getElementById('my-name-input');
const fd = new FormData(); // Works in bun/node/browser
fd.append(
  'file', fileInput.files[0].file, "this is the filename.txt"
);
fd.append('name', nameInput.value);
fetch('some url', { method: 'POST', body: fd });
```