Notes: FreeCodeCamp Backend Development & APIs - Microservices Projects

These are the final 5 projects in the FreeCodeCamp Backend Development & APIs certification. Each microservice demonstrates specific backend concepts and APIs in a practical, deployable format.

Project 1: Timestamp Microservice

Purpose: Convert between Unix timestamps and human-readable dates, handling various date formats and edge cases.

API Endpoint: GET /api/:date?

What it does: Accepts a date parameter (optional) and returns both Unix timestamp and UTC string format. If no date provided, uses current date/time.

Complete Code:

```
// index.js - Timestamp Microservice
var express = require("express");
var app = express();
var cors = require("cors");

// Enable CORS for FreeCodeCamp testing
app.use(cors({ optionsSuccessStatus: 200 }));

// Serve static files
app.use(express.static("public"));

// Home route
app.get("/", function (req, res) {
  res.sendFile(__dirname + "/views/index.html");
});

// Main API endpoint with optional date parameter
app.get("/api/:date?", function (req, res) {
  let date = req.params.date;
```

```
  let unixDate;
  let dateObj;
  let utcDate;

  // Check if input is Unix timestamp (all digits)
  let isUnix = /^\d+$/.test(date);

  // Handle different input scenarios
  if (!date) {
    // No date provided - use current date
    dateObj = new Date();
  } else if (date && isUnix) {
    // Unix timestamp provided - convert to integer first
    unixDate = parseInt(date);
    dateObj = new Date(unixDate);
  } else if (date && !isUnix) {
    // Regular date string provided
    dateObj = new Date(date);
  }

  // Validate if date is valid
  if (dateObj.toString() === "Invalid Date") {
    res.json({ error: "Invalid Date" });
    return;
  }

  // Convert to both formats
  unixDate = dateObj.getTime();  // Get Unix timestamp in milliseconds
  utcDate = dateObj.toUTCString();  // Get human-readable UTC string

  // Return both formats
  res.json({ unix: unixDate, utc: utcDate });
});

// Start server
var listener = app.listen(process.env.PORT || 3000, function () {
  console.log("Your app is listening on port " + listener.address().port);
});
```

Key Concepts Explained:

1. Regular Expression for Unix Detection:

```
let isUnix = /^\d+$/.test(date);
```

- `^` = start of string
- `\d+` = one or more digits
- `$` = end of string
- Tests if entire string contains only digits

2. parseInt() Function:

```
unixDate = parseInt(date);
```

- Converts string to integer
- Stops at first non-numeric character
- Examples: `parseInt("42")` → 42, `parseInt("3.14")` → 3

3. Date Object Methods:

```
dateObj.getTime()      // Returns Unix timestamp in milliseconds
dateObj.toUTCString()  // Returns formatted UTC string
```

Example API Responses:

Request	Response
`/api/`	`{"unix": 1696435200000, "utc": "Thu, 04 Oct 2025 12:00:00 GMT"}`
`/api/1451001600000`	`{"unix": 1451001600000, "utc": "Fri, 25 Dec 2015 00:00:00 GMT"}`
`/api/2015-12-25`	`{"unix": 1451001600000, "utc": "Fri, 25 Dec 2015 00:00:00 GMT"}`
`/api/invalid`	`{"error": "Invalid Date"}`

Project 2: Request Header Parser Microservice

Purpose: Extract and return client information from HTTP request headers including IP address, language preferences, and browser/OS details.

Complete Code:

```
// index.js - Request Header Parser Microservice
require('dotenv').config();
var express = require('express');
var app = express();
var cors = require('cors');

// Enable CORS
app.use(cors({ optionsSuccessStatus: 200 }));

// Serve static files
app.use(express.static('public'));

// Home route
app.get('/', function (req, res) {
  res.sendFile(__dirname + '/views/index.html');
});

// Main API endpoint - extracts client information
app.get('/api/whoami', function (req, res) {
  res.json({
    ipaddress: req.ip,                        // Client IP address
    language: req.headers['accept-language'], // Language preferences
    software: req.headers['user-agent']       // Browser/OS information
   });
});

// Start server
var listener = app.listen(process.env.PORT || 3000, function () {
  console.log('Your app is listening on port ' + listener.address().port);
});
```

Header Breakdown:

1. req.ip - Client IP Address:

```
ipaddress: req.ip
```

- Express extracts client IP from request headers
- May show proxy IP if behind services like Replit/Glitch
- Example: `"123.45.67.89"`

2. req.headers['accept-language'] - Language Preferences:

```
language: req.headers['accept-language']
```

- Browser sends preferred languages with priority weights
- Example: `"en-US,en;q=0.9,es;q=0.8"`
- Meaning: English (US) primary, English secondary (0.9 priority), Spanish tertiary (0.8 priority)

3. req.headers['user-agent'] - Browser/OS Information:

```
software: req.headers['user-agent']
```

- Identifies browser, version, operating system, and rendering engine
- Example: `"Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/120.0.0.0 Safari/537.36"`

Example Response:

```
{
  "ipaddress": "192.168.1.100",
  "language": "en-US,en;q=0.9",
  "software": "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36"
}
```

Project 3: URL Shortener Microservice

Purpose: Create shortened URLs similar to bit.ly, with URL validation and redirect functionality using MongoDB for storage.

Complete Code:

```
// index.js - URL Shortener Microservice
require("dotenv").config();
const express = require("express");
const cors = require("cors");
const { MongoClient } = require("mongodb");
const dns = require("dns");
const urlParser = require("url");

const app = express();
const port = process.env.PORT || 3000;

// MongoDB connection
const uri = process.env.DB_URL;
const client = new MongoClient(uri);

async function connectDB() {
  try {
    await client.connect();
    console.log("☑ Connected to MongoDB");

    // Middleware setup
    app.use(cors());
    app.use(express.json());
    app.use(express.urlencoded({ extended: true }));
    app.use("/public", express.static(`${process.cwd()}/public`));

    // Home route
    app.get("/", (req, res) => {
      res.sendFile(process.cwd() + "/views/index.html");
    });

    // POST endpoint - Create short URL
    app.post("/api/shorturl", async (req, res) => {
```

```
try {
  const originalUrl = req.body.url;
  let hostname;

  // Step 1: Validate URL format
  try {
    hostname = new URL(originalUrl).hostname;
  } catch (e) {
    return res.json({ error: "invalid url" });
  }

  // Step 2: DNS lookup to verify domain exists
  dns.lookup(hostname, async (err, address) => {
    if (err || !address) {
      return res.json({ error: "invalid url" });
    }

    // Step 3: Save valid URL to database
    const db = client.db("fcc-urlshortner");
    const urls = db.collection("urlshortner");

    // Generate incremental short URL number
    const count = await urls.countDocuments({});
    const urlDoc = {
      original_url: originalUrl,
      short_url: count + 1,
    };

    await urls.insertOne(urlDoc);
    console.log("☑ Inserted:", urlDoc);

    // Step 4: Return success response
    res.json({
      original_url: originalUrl,
      short_url: urlDoc.short_url,
    });
  });
} catch (err) {
  console.error(err);
  res.status(500).json({ error: "Server error" });
```

```
    }
  });

  // GET endpoint - Redirect to original URL
  app.get("/api/shorturl/:short_url", async (req, res) => {
    try {
      // Step 1: Validate short URL is numeric
      const shortUrl = parseInt(req.params.short_url);
      if (isNaN(shortUrl)) {
        return res.json({ error: "Wrong format" });
      }

      // Step 2: Find original URL in database
      const db = client.db("fcc-urlshortner");
      const urls = db.collection("urlshortner");
      const urlDoc = await urls.findOne({ short_url: shortUrl });

      if (!urlDoc) {
        return res.json({ error: "No short URL found" });
      }

      // Step 3: Redirect to original URL
      res.redirect(urlDoc.original_url);
    } catch (err) {
      console.error(err);
      res.status(500).json({ error: "Server error" });
    }
  });

  // Start server after database connection
  app.listen(port, () => {
    console.log(`🚀 Listening on port ${port}`);
  });
 } catch (err) {
  console.error("X Could not connect to MongoDB:", err);
 }
}

connectDB();
```

Key Implementation Details:

1. URL Validation Process:

```
// Extract hostname from URL
hostname = new URL(originalUrl).hostname;

// Verify domain exists via DNS
dns.lookup(hostname, async (err, address) => {
  // If DNS fails, URL is invalid
});
```

2. Short URL Generation:

```
const count = await urls.countDocuments({});
const urlDoc = {
  original_url: originalUrl,
  short_url: count + 1,  // Incremental numbering
};
```

3. Database Structure:

```
// Document stored in MongoDB
{
  "original_url": "https://freecodecamp.org",
  "short_url": 1
}
```

API Flow Summary:

Endpoint	Method	Purpose	Example
`/api/shorturl`	POST	Create short URL	`{"original_url": "https://google.com", "short_url": 1}`
`/api/shorturl/1`	GET	Redirect to original	Redirects to `https://google.com`

Project 4: Exercise Tracker

Purpose: Track users' exercise activities with date filtering and logging capabilities using MongoDB with Mongoose ODM.

Complete Code:

```
// index.js - Exercise Tracker Microservice
const express = require("express");
const app = express();
const cors = require("cors");
require("dotenv").config();
const mongoose = require("mongoose");
const { Schema } = mongoose;

// Connect to MongoDB
mongoose.connect(process.env.DB_URL);

// User schema - stores basic user information
const userSchema = new Schema({
  username: String,
});
const User = mongoose.model("User", userSchema);

// Exercise schema - stores exercise data linked to users
const ExerciseSchema = new Schema({
  userId: { type: Schema.Types.ObjectId, ref: "User" },  // Reference to User
  description: String,
  duration: Number,  // Duration in minutes
  date: Date,
});
const Exercise = mongoose.model("Exercise", ExerciseSchema);

// Middleware
app.use(cors());
app.use(express.static("public"));
app.use(express.json());
app.use(express.urlencoded({ extended: true }));

// Home route
```

```
app.get("/", (req, res) => {
  res.sendFile(__dirname + "/views/index.html");
});

// API Endpoint 1: Create new user
app.post("/api/users", async (req, res) => {
  const { username } = req.body;

  if (!username) {
    return res.status(400).json({ error: "Username is required" });
  }

  try {
    const newUser = new User({ username });
    const savedUser = await newUser.save();

    res.json({
      username: savedUser.username,
      _id: savedUser._id
    });
  } catch (err) {
    res.status(500).json({ error: "Failed to create user" });
  }
});

// API Endpoint 2: Get all users
app.get("/api/users", async (req, res) => {
  try {
    // Only return _id and username fields
    const users = await User.find({}).select("_id username");
    res.json(users);
  } catch (err) {
    res.status(500).json({ error: "Failed to retrieve users" });
  }
});

// API Endpoint 3: Add exercise for specific user
app.post("/api/users/:_id/exercises", async (req, res) => {
  const userId = req.params._id;
  const { description, duration, date } = req.body;
```

```
  try {
    // Verify user exists
    const user = await User.findById(userId);
    if (!user) {
      return res.status(404).json({ error: "User not found" });
    }

    // Create exercise object
    const exerciseObj = new Exercise({
      userId: user._id,
      description,
      duration: parseInt(duration),  // Ensure duration is number
      date: date ? new Date(date) : new Date(),  // Use current date if none provided
    });

    const exercise = await exerciseObj.save();

    // Return user info with exercise details
    res.json({
      _id: user._id,
      username: user.username,
      description: exercise.description,
      duration: exercise.duration,
      date: new Date(exercise.date).toDateString(),  // Convert to readable format
    });
  } catch (err) {
    console.log(err);
    res.status(500).json({ error: "Failed to add exercise" });
  }
});

// API Endpoint 4: Get user's exercise log with optional filters
app.get("/api/users/:_id/logs", async (req, res) => {
  const id = req.params._id;
  const { from, to, limit } = req.query;

  try {
    // Verify user exists
    const user = await User.findById(id);
```

```
  if (!user) {
    return res.status(404).json({ error: "User not found" });
  }

  // Build date filter object
  let dateObj = {};
  if (from) {
    dateObj["$gte"] = new Date(from);  // Greater than or equal to 'from' date
  }
  if (to) {
    dateObj["$lte"] = new Date(to);    // Less than or equal to 'to' date
  }

  // Build complete filter
  let filter = { userId: id };
  if (from || to) {
    filter.date = dateObj;
  }

  // Execute query with optional limit
  const exercises = await Exercise.find(filter).limit(+limit || 500);

  // Format exercises for response
  const log = exercises.map((e) => ({
    description: e.description,
    duration: e.duration,
    date: e.date.toDateString(),  // Convert to readable date string
  }));

  // Return user info with exercise log
  res.json({
    username: user.username,
    count: exercises.length,
    _id: user._id,
    log
  });
} catch (err) {
  console.log(err);
  res.status(500).json({ error: "Failed to retrieve logs" });
}
```

```
});

// Start server
const listener = app.listen(process.env.PORT || 3000, () => {
  console.log("Your app is listening on port " + listener.address().port);
});
```

API Endpoints Breakdown:

1. POST /api/users - Create User:

```
// Request: { "username": "Chethan" }
// Response: { "username": "Chethan", "_id": "6724ad86fcd91bbf81f2fa43" }
```

2. GET /api/users - List All Users:

```
// Response: [{"_id": "6724ad86fcd91bbf81f2fa43", "username": "Chethan"}]
```

3. POST /api/users/:_id/exercises - Add Exercise:

```
// Request: { "description": "Running", "duration": "60", "date": "2024-10-04" }
// Response: {
//   "_id": "6724ad86fcd91bbf81f2fa43",
//   "username": "Chethan",
//   "description": "Running",
//   "duration": 60,
//   "date": "Fri Oct 04 2024"
// }
```

4. GET /api/users/:_id/logs - Get Exercise Log:

```
// Query: /api/users/6724ad86fcd91bbf81f2fa43/logs?from=2024-09-01&to=2024-10-01&limit=2
// Response: {
//   "username": "Chethan",
//   "count": 2,
//   "_id": "6724ad86fcd91bbf81f2fa43",
//   "log": [
//     {"description": "Running", "duration": 60, "date": "Mon Sep 30 2024"},
//     {"description": "Cycling", "duration": 30, "date": "Wed Oct 02 2024"}
```

```
//   ]
// }
```

MongoDB Query Concepts:

Date Range Filtering:

```
// Build MongoDB date query
let dateObj = {};
if (from) dateObj["$gte"] = new Date(from);  // Greater than or equal
if (to) dateObj["$lte"] = new Date(to);      // Less than or equal

// Results in query like: { date: { $gte: Date, $lte: Date } }
```

Field Selection:

```
User.find({}).select("_id username")  // Only return specified fields
```

Project 5: File Metadata Microservice

Purpose: Analyze uploaded files and return metadata information (name, type, size) using Multer middleware for file handling.

Complete Code:

```
// index.js - File Metadata Microservice
var express = require('express');
var cors = require('cors');
require('dotenv').config()
const multer = require('multer');

// Configure multer to store files in memory (not disk)
const upload = multer();

var app = express();

// Middleware
app.use(cors());
app.use('/public', express.static(process.cwd() + '/public'));

// Home route - serves upload form
app.get('/', function (req, res) {
  res.sendFile(process.cwd() + '/views/index.html');
});

// File analysis endpoint
app.post('/api/fileanalyse', upload.single('upfile'), (req, res) => {
  // Check if file was uploaded
  if (!req.file) {
    return res.status(400).json({ error: 'No file uploaded' });
  }

  // Return file metadata
  res.json({
    name: req.file.originalname,  // Original filename from user's computer
    type: req.file.mimetype,      // File MIME type (e.g., image/png)
    size: req.file.size           // File size in bytes
  });
```

```
});

// Start server
const port = process.env.PORT || 3000;
app.listen(port, function () {
  console.log('Your app is listening on port ' + port)
});
```

Multer Breakdown:

What Multer Does:

- Handles `multipart/form-data` requests (file uploads)
- Parses file data from HTTP requests
- Makes file information accessible via `req.file`

Key Multer Concepts:

1. Configuration:

```
const upload = multer();  // Store in memory, no disk storage
```

2. Middleware Usage:

```
upload.single('upfile')  // Expect single file with field name 'upfile'
```

3. File Object Structure:

```
req.file = {
  fieldname: 'upfile',           // Form field name
  originalname: 'resume.pdf',    // Original filename
  encoding: '7bit',              // File encoding
  mimetype: 'application/pdf',   // MIME type
  buffer: <Buffer 25 50 44...>,  // File data in memory
  size: 23456                    // File size in bytes
}
```

HTML Form Requirements:

```
<!-- Form must have specific attributes for file uploads -->
<form action="/api/fileanalyse" method="POST" enctype="multipart/form-data">
  <input type="file" name="upfile" />  <!-- Name must match upload.single() -->
  <button type="submit">Upload</button>
</form>
```

Example Responses:

Successful Upload:

```
{
  "name": "my-document.pdf",
  "type": "application/pdf",
  "size": 245760
}
```

No File Uploaded:

```
{
  "error": "No file uploaded"
}
```

File Type Examples:

File Extension	MIME Type	Example Size
`.jpg`	`image/jpeg`	1,245,760 bytes
`.png`	`image/png`	567,890 bytes
`.pdf`	`application/pdf`	2,345,678 bytes
`.txt`	`text/plain`	1,024 bytes
`.docx`	`application/vnd.openxmlformats-officedocument.wordprocessingml.document`	45,678 bytes

Summary

These 5 microservices demonstrate essential backend development concepts:

1. **Timestamp Service**: Date/time manipulation and format conversion
2. **Header Parser**: HTTP request analysis and client information extraction
3. **URL Shortener**: Database operations, URL validation, and redirect handling
4. **Exercise Tracker**: Complex CRUD operations with filtering and data relationships
5. **File Metadata**: File upload handling and metadata extraction

Each project showcases different aspects of API development, from simple data transformation to complex database operations and file handling. They collectively cover the core skills needed for backend development using Node.js, Express, and MongoDB.