Apollo

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

// Import the ApolloServer class
const { ApolloServer } = require('apollo-server-express');

// Import the two parts of a GraphQL schema
const { typeDefs, resolvers } = require('./schemas');
const db = require('./config/connection');

const PORT = process.env.PORT || 3001;
const app = express();

// Create a new instance of an Apollo server with the GraphQL schema
const server = new ApolloServer({
  typeDefs,
  resolvers
});

// Update Express.js to use Apollo server features
server.applyMiddleware({ app });

app.use(express.urlencoded({ extended: false }));
app.use(express.json());

db.once('open', () => {
  app.listen(PORT, () => {
    console.log(`API server running on port ${PORT}!`);
    console.log(`Use GraphQL at http://localhost:${PORT}${server.graphqlPath}`);
  });
});
```

```
const { Class } = require('../models');

// Create the functions that fulfill the queries defined in `typeDefs.js`
const resolvers = {
  Query: {
    classes: async () => {
      // Get and return all documents from the classes collection
      return await Class.find({});
    }
  }
};
module.exports = resolvers;
```

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

mongoose.connect(
  process.env.MONGODB_URI || 'mongodb://localhost/schools-db',
  {
    useNewUrlParser: true,
    useUnifiedTopology: true,
    useCreateIndex: true,
    useFindAndModify: false,
  }
);
module.exports = mongoose.connection;
```

```
const db = require('../config/connection');
const { School, Class, Professor } = require('../models');

const schoolData = require('./schoolData.json');
const classData = require('./classData.json');
const professorData = require('./professorData.json');

db.once('open', async () => {
  // clean database
  await School.deleteMany({});
  await Class.deleteMany({});
  await Professor.deleteMany({});

  // bulk create each model
  const schools = await School.insertMany(schoolData);
  const classes = await Class.insertMany(classData);
  const professors = await Professor.insertMany(professorData);

  for (newClass of classes) {
    // randomly add each class to a school
    const tempSchool = schools[Math.floor(Math.random() * schools.length)];
    tempSchool.classes.push(newClass._id);
    await tempSchool.save();

    // randomly add a professor to each class
    const tempProfessor = professors[Math.floor(Math.random() * professors.length)];
    newClass.professor = tempProfessor._id;
    await newClass.save();

    // reference class on professor model, too
    tempProfessor.classes.push(newClass._id);
    await tempProfessor.save();
  }

  console.log('all done!');
  process.exit(0);
});
```

The following query returns all schools, classes, and professors:

```graphql
query schools {
  schools {
    name
    location
    studentCount
    classes {
      name
      professor {
        name
      }
    }
  }
}
```

The following query returns all classes and professors:

```graphql
query classes {
  classes {
    name
    creditHours
    building
    professor {
      name
      studentScore
    }
  }
}
```

The following query returns all professors:

```graphql
query professors {
  professors {
    _id
    name
    studentScore
    officeHours
    officeLocation
  }
}
```

Queries


```
[
  {
    "name": "Foundations of Data Science",
    "building": "CHEM",
    "creditHours": 3
  },
  {
    "name": "Introduction to Computational Thinking",
    "building": "LS",
    "creditHours": 3
  },
  {
    "name": "C for Programmers",
    "building": "SCI",
    "creditHours": 4
  },
  {
    "name": "C++ for Programmers",
    "building": "SCI",
    "creditHours": 4
  },
  {
    "name": "Discrete Mathematics",
    "building": "SCI",
    "creditHours": 3
  },
  {
    "name": "Computer Architecture and Engineering",
    "building": "SCI",
    "creditHours": 3
  },
  {
    "name": "User Interface Design",
    "building": "HH",
    "creditHours": 4
  },
  {
    "name": "Computer Security",
    "building": "JH",
    "creditHours": 3
  },
  {
    "name": "Internet Architecture and Protocols",
    "building": "JH",
    "creditHours": 2
  },
  {
    "name": "Algorithms for Computational Biology",
    "building": "LS",
    "creditHours": 3
  }
]
```

```
[
  {
    "name": "UseEffect University",
    "location": "Downtown",
    "studentCount": 6000
  },
  {
    "name": "Component College",
    "location": "Art District",
    "studentCount": 8000
  },
  {
    "name": "JSX Technical School",
    "location": "React Park",
    "studentCount": 5000
  }
]
```

```
[
  {
    "name": "Alberto Harrison",
    "studentScore": 7.6,
    "officeHours": "10:00 AM",
    "officeLocation": "HH3"
  },
  {
    "name": "Peyton Olson",
    "studentScore": 9.0,
    "officeHours": "10:30 AM",
    "officeLocation": "JH2"
  },
  {
    "name": "Virgil Parker",
    "studentScore": 8.1,
    "officeHours": "1:00 PM",
    "officeLocation": "LIB1"
  },
  {
    "name": "Rebecca Watts",
    "studentScore": 8.8,
    "officeHours": "4:00 PM",
    "officeLocation": "LIB2"
  },
  {
    "name": "Colleen Moore",
    "studentScore": 7.8,
    "officeHours": "4:30 PM",
    "officeLocation": "KS3"
  }
]
```

```
const { gql } = require('apollo-server-express');

const typeDefs = gql`
  type School {
    _id: ID
    name: String
    location: String
    studentCount: Int
    classes: [Class]
  }
  type Class {
    _id: ID
    name: String
    building: String
    creditHours: Int
    professor: Professor
  }
  type Professor {
    _id: ID
    name: String
    officeHours: String
    officeLocation: String
    studentScore: Float
    # Add a field that will return an array of Class instances
    classes: [Class]
  }
  type Query {
    schools: [School]
    classes: [Class]
    professors: [Professor]
  }
`;

module.exports = typeDefs;
```

TypeDefs

```
const { School, Class, Professor } = require('../models');

const resolvers = {
  Query: {
    schools: async () => {
      // Populate the classes and professor subdocuments when querying for schools
      return await School.find({}).populate('classes').populate({
        path: 'classes',
        populate: 'professor'
      });
    },
    classes: async () => {
      // Populate the professor subdocument when querying for classes
      return await Class.find({}).populate('professor');
    },
    professors: async () => {
      // Populate the classes subdocument on every instance of Professor
      return await Professor.find({}).populate('classes');    }
  }
};

module.exports = resolvers;
```

```
const { School, Class, Professor } = require('../models');

const resolvers = {
  Query: {
    schools: async () => {
      return await School.find({}).populate('classes').populate({
        path: 'classes',
        populate: 'professor'
      });
    },
    classes: async () => {
      return await Class.find({}).populate('professor');
    },
    // Define a resolver to retrieve individual classes
    class: async (parent, args) => {
      // Use the parameter to find the matching class in the collection
      return await Class.findById(args.id).populate('professor');
    },
    professors: async () => {
      return await Professor.find({}).populate('classes');
    }
  }
};

module.exports = resolvers;
```

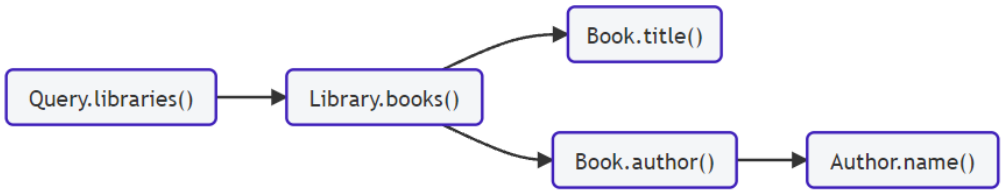
```gql
query class($id: ID!) {
  class(id: $id) {
    name
    professor {
      name
    }
  }
}
```

Query Arguments

```
# A library has a branch and books
type Library {
  branch: String!
  books: [Book!]
}

# A book has a title and author
type Book {
  title: String!
  author: Author!
}

# An author has a name
type Author {
  name: String!
}

type Query {
  libraries: [Library]
}
```

```
type User {
  id: ID!
  name: String
}

type Query {
  user(id: ID!): User
}
```

```
const resolvers = {
  Query: {
    user(parent, args, context, info) {
      return users.find(user => user.id === args.id);
    }
  }
}
```



```
const { School, Class, Professor } = require('../models');

const resolvers = {
  Query: {
    schools: async () => {
      return await School.find({}).populate('classes').populate({
        path: 'classes',
        populate: 'professor'
      });
    },
    classes: async () => {
      return await Class.find({}).populate('professor');
    },
    class: async (parent, args) => {
      return await Class.findById(args.id);
    },
    professors: async () => {
      return await Professor.find({}).populate('classes');
    }
  },
  Mutation: {
    addSchool: async (parent, { name, location, studentCount }) => {
      return await School.create({ name, location, studentCount });
    },
    updateClass: async (parent, { id, building }) => {
      // Find and update the matching class using the destructured args
      return await Class.findOneAndUpdate(
        { _id: id },
        { building },
        // Return the newly updated object instead of the original
        { new: true }
      );
    }
  }
};
module.exports = resolvers;
```

Mutations

```
const { gql } = require('apollo-server-express');

const typeDefs = gql`
  type School {
    _id: ID
    name: String
    location: String
    studentCount: Int
    classes: [Class]
  }
  type Class {
    _id: ID
    name: String
    building: String
    creditHours: Int
    professor: Professor
  }
  type Professor {
    _id: ID
    name: String
    officeHours: String
    officeLocation: String
    studentScore: Float
    classes: [Class]
  }
  type Query {
    schools: [School]
    classes: [Class]
    professors: [Professor]
    class(id: ID!): Class
  }
  type Mutation {
    addSchool(name: String!, location: String!, studentCount: Int!): School
    updateClass(id: ID!, building: String!): Class
  }
`;

module.exports = typeDefs;
```

```json
{
  "id": "<insert ID of a class here>",
  "building": "AA"
}
```

Run the following mutation:

```gql
mutation updateClass($id: ID!, $building: String!) {
  updateClass(id: $id, building: $building) {
    name
    building
  }
}
```

MERN Setup

Root-level Functionality

The npm start script: In production, we only run the back-end server, which will serve the built React application code as its front end.
The npm run develop script: In development, we need to run both a back-end server and the React development server, so we use the concurrently library to execute two separate promises at the same time.
The npm install script: Since our dependencies for the entire application exist in two smaller applications, we use this script to automatically install all of them at once.
The npm run seed script: We can seed our database with data when we run this command.
The npm run build script: When we deploy our application, we instruct the hosting service to execute the build command and build our production-ready React application."

```
"scripts": {
  "start": "node server/server.js",
  "develop": "concurrently \"cd server && npm run watch\" \"cd client && npm start\"",
  "install": "cd server && npm i && cd ../client && npm i",
  "seed": "cd server && npm run seed",
  "build": "cd client && npm run build"
},
```

Client-side Functionality

Since we run a front-end and back-end server for our full-stack application in development, we set it up so all client-side requests to our API server are prefixed with the API server's URL.

```
"proxy": "http://localhost:3001",
```

Server-side Functionality

In production, when we no longer need to use the Create React App development server, we set up our server to serve the built React front-end application that is in the ../client/build directory.

```
if (process.env.NODE_ENV === 'production') {
  app.use(express.static(path.join(__dirname, '../client/build')));
}
```

Since the React front-end application will handle its own routing, we set up a wildcard route on our server that will serve the front end whenever a request for a non-API route is received.

```
app.get('*', (req, res) => {
  res.sendFile(path.join(__dirname, '../client/build/index.html'));
});
```

```
import React from 'react';

// Import the `useQuery()` hook from Apollo Client
import { useQuery } from '@apollo/client';

import ThoughtList from '../components/ThoughtList';

// Import the query we are going to execute from its file
import { QUERY_THOUGHTS } from '../utils/queries';

const Home = () => {
  // Execute the query on component load
  const { loading, data } = useQuery(QUERY_THOUGHTS);

  // Use optional chaining to check if data exists and if it has a thoughts property.
            If not, return an empty array to use.
  const thoughts = data?.thoughts || [];
  return (
    <main>
      <div className="flex-row justify-center">
        <div className="col-12 col-md-8 mb-3">
          {/* If the data is still loading, render a loading message */}
          {loading ? (
            <div>Loading...</div>
          ) : (
            <ThoughtList
              thoughts={thoughts}
              title="Some Feed for Thought(s)..."
            />
          )}
        </div>
      </div>
    </main>
  );
};
export default Home;
```

```
            import { gql } from '@apollo/client';

            export const QUERY_THOUGHTS = gql`
              # create a GraphQL query to be executed by Apollo Client
              query getThoughts {
                thoughts {
                  _id
                  thoughtText
                  thoughtAuthor
                  createdAt
                }
              }
            `;
```

useQuery

```
import React from 'react';

const ThoughtList = ({ thoughts, title }) => {
  if (!thoughts.length) {
    return <h3>No Thoughts Yet</h3>;
  }
  return (
    <div>
      <h3>{title}</h3>
      {thoughts &&
        thoughts.map((thought) => (
          <div key={thought._id} className="card mb-3">
            <h4 className="card-header bg-primary text-light p-2 m-0">
              {thought.thoughtAuthor} <br />
              <span style={{ fontSize: '1rem' }}>
                had this thought on {thought.createdAt}
              </span>
            </h4>
            <div className="card-body bg-light p-2">
              <p>{thought.thoughtText}</p>
            </div>
          </div>
        ))}
    </div>
  );
};
export default ThoughtList;
```

```
import React from 'react';
import { ApolloClient, InMemoryCache, ApolloProvider } from '@apollo/client';
import Home from './pages/Home';
import Header from './components/Header';
import Footer from './components/Footer';

const client = new ApolloClient({
  uri: '/graphql',
  cache: new InMemoryCache(),
});

function App() {
  return (
    <ApolloProvider client={client}>
      <div className="flex-column justify-flex-start min-100-vh">
        <Header />
        <div className="container">
          <Home />
        </div>
        <Footer />
      </div>
    </ApolloProvider>
  );
}
export default App;
```

useMutation pt1

```
import React, { useState } from 'react';

// Import the `useMutation()` hook from Apollo Client
import { useMutation } from '@apollo/client';

// Import the GraphQL mutation
import { ADD_PROFILE } from '../../utils/mutations';

const ProfileForm = () => {
  const [name, setName] = useState('');

  // Invoke `useMutation()` hook to return a Promise-based function and data about the ADD_PROFILE mutation
  const [addProfile, { error }] = useMutation(ADD_PROFILE);

  const handleFormSubmit = async (event) => {
    event.preventDefault();
    // Since mutation function is async, wrap in a `try...catch` to catch any network errors
          from throwing due to a failed request.
    try {
      // Execute mutation and pass in defined parameter data as variables
      const { data } = await addProfile({
        variables: { name },
      });
      window.location.reload();
    } catch (err) {
      console.error(err);
    }
  };
  return (
    <div>
      <h3>Add yourself to the list...</h3>
      <form
        className="flex-row justify-center justify-space-between-md align-center"
        onSubmit={handleFormSubmit}
      >
        <div className="col-12 col-lg-9">
          <input
            placeholder="Add your profile name..."
            value={name}
            className="form-input w-100"
            onChange={(event) => setName(event.target.value)}
          />
        </div>
        <div className="col-12 col-lg-3">
          <button className="btn btn-info btn-block py-3" type="submit">
            Add Profile
          </button>
        </div>
        {error && (
          <div className="col-12 my-3 bg-danger text-white p-3">
            Something went wrong...
          </div>
        )}
      </form>
    </div>
  );
};
export default ProfileForm;
```

```
import React from 'react';

import { useQuery } from '@apollo/client';

import ProfileList from '../components/ProfileList';
import ProfileForm from '../components/ProfileForm';

import { QUERY_PROFILES } from '../utils/queries';

const Home = () => {
  const { loading, data } = useQuery(QUERY_PROFILES);

  const profiles = data?.profiles || [];
  return (
    <main>
      <div className="flex-row justify-center">
        <div
          className="col-12 col-md-10 mb-3 p-3"
          style={{ border: '1px dotted #1a1a1a' }}
        >
          <ProfileForm />
        </div>
        <div className="col-12 col-md-10 my-3">
          {loading ? (
            <div>Loading...</div>
          ) : (
            <ProfileList
              profiles={profiles}
              title="Here's the current roster of friends..."
            />
          )}
        </div>
      </div>
    </main>
  );
};
export default Home;
```

```
import { gql } from '@apollo/client';

export const ADD_PROFILE = gql`
  mutation addProfile($name: String!) {
    addProfile(name: $name) {
      _id
      name
      skills
    }
  }
`;
```

useMutation pt2

```
import React, { useState } from 'react';
import { useMutation } from '@apollo/client';
import { ADD_THOUGHT } from '../../utils/mutations';

const ThoughtForm = () => {
  const [formState, setFormState] = useState({
    thoughtText: '',
    thoughtAuthor: '',
  });
  const [characterCount, setCharacterCount] = useState(0);

  // Set up our mutation with an option to handle errors
  const [addThought, { error }] = useMutation(ADD_THOUGHT);

  const handleFormSubmit = async (event) => {
    event.preventDefault();
    // On form submit, perform mutation and pass in form data object as arguments
    // It is important that the object fields are match the defined parameters in `ADD_THOUGHT` mutation
    try {
      const { data } = addThought({
        variables: { ...formState },
      });
      window.location.reload();
    } catch (err) {
      console.error(err);
    }
  };

  const handleChange = (event) => {
    const { name, value } = event.target;
    if (name === 'thoughtText' && value.length <= 280) {
      setFormState({ ...formState, [name]: value });
      setCharacterCount(value.length);
    } else if (name !== 'thoughtText') {
      setFormState({ ...formState, [name]: value });
    }
  };

  return (
    <div>
      <h3>What's on your techy mind?</h3>
      <p
        className={`m-0 ${
          characterCount === 280 || error ? 'text-danger' : ''
        }`}
      >
        Character Count: {characterCount}/280
        {error && <span className="ml-2">Something went wrong...</span>}
      </p>
      <form
        className="flex-row justify-center justify-space-between-md align-center"
        onSubmit={handleFormSubmit}
      >
        <div className="col-12">
          <textarea
            name="thoughtText"
            placeholder="Here's a new thought..."
            value={formState.thoughtText}
            className="form-input w-100"
            onChange={handleChange}
          ></textarea>
        </div>
        <div className="col-12 col-lg-9">
          <input
            name="thoughtAuthor"
            placeholder="Add your name to get credit for the thought..."
            value={formState.thoughtAuthor}
            className="form-input w-100"
            onChange={handleChange}
          />
        </div>
        <div className="col-12 col-lg-3">
          <button className="btn btn-primary btn-block py-3" type="submit">
            Add Thought
          </button>
        </div>
        {error && (
          <div className="col-12 my-3 bg-danger text-white p-3">
            Something went wrong...
          </div>
        )}
      </form>
    </div>
  );
};
export default ThoughtForm;
```

```
import { gql } from '@apollo/client';

export const ADD_THOUGHT = gql`
  mutation addThought($thoughtText: String!, $thoughtAuthor: String!) {
    addThought(thoughtText: $thoughtText, thoughtAuthor: $thoughtAuthor) {
      _id
      thoughtText
      thoughtAuthor
      createdAt
      comments {
        _id
        commentText
      }
    }
  }
`;
```

Apollo Cache

```
import React, { useState } from 'react';
import { useMutation } from '@apollo/client';

import { ADD_PROFILE } from '../../utils/mutations';
import { QUERY_PROFILES } from '../../utils/queries';

const ProfileForm = () => {
  const [name, setName] = useState('');

  const [addProfile, { error }] = useMutation(ADD_PROFILE, {
    // The update method allows us to access and update the local cache
    update(cache, { data: { addProfile } }) {
      try {
        // First we retrieve existing profile data that is stored in the cache
            under the `QUERY_PROFILES` query
        // Could potentially not exist yet, so wrap in a try/catch
        const { profiles } = cache.readQuery({ query: QUERY_PROFILES });
        // Then we update the cache by combining existing profile data
            with the newly created data returned from the mutation
        cache.writeQuery({
          query: QUERY_PROFILES,
          // If we want new data to show up before or after existing data,
            adjust the order of this array
          data: { profiles: [...profiles, addProfile] },
        });
      } catch (e) {
        console.error(e);
      }
    },
  });

  const handleFormSubmit = async (event) => {
    event.preventDefault();
    try {
      const { data } = addProfile({
        variables: { name },
      });
      setName('');
    } catch (err) {
      console.error(err);
    }
  };

  return (
    <div>
      <h3>Add yourself to the list...</h3>
      <form
        className="flex-row justify-center justify-space-between-md align-center"
        onSubmit={handleFormSubmit}
      >
        <div className="col-12 col-lg-9">
          <input
            placeholder="Add your profile name..."
            value={name}
            className="form-input w-100"
            onChange={(event) => setName(event.target.value)}
          />
        </div>
        <div className="col-12 col-lg-3">
          <button className="btn btn-info btn-block py-3" type="submit">
            Add Profile
          </button>
        </div>
        {error && (
          <div className="col-12 my-3 bg-danger text-white p-3">
            Something went wrong...
          </div>
        )}
      </form>
    </div>
  );
};
export default ProfileForm;
```

```
import { gql } from '@apollo/client';

export const ADD_PROFILE = gql`
  mutation addProfile($name: String!) {
    addProfile(name: $name) {
      _id
      name
      skills
    }
  }
`;
```

```
import { gql } from '@apollo/client';

export const QUERY_PROFILES = gql`
  query allProfiles {
    profiles {
      _id
      name
      skills
    }
  }
`;
```

React Router

```
import React from 'react';

import { ApolloClient, InMemoryCache, ApolloProvider } from '@apollo/client';
import { BrowserRouter as Router, Route } from 'react-router-dom';

import Home from './pages/Home';
import Profile from './pages/Profile';

import Header from './components/Header';
import Footer from './components/Footer';

const client = new ApolloClient({
  uri: '/graphql',
  cache: new InMemoryCache(),
});

function App() {
  return (
    <ApolloProvider client={client}>
      {/* Wrap page elements in Router component to keep track of location state */}
      <Router>
        <div className="flex-column justify-flex-start min-100-vh">
          <Header />
          <div className="container">
            {/* Define routes to render different page components at different paths */}
            <Route exact path="/">
              <Home />
            </Route>
            {/* Define a route that will take in variable data */}
            <Route exact path="/profiles/:profileId">
              <Profile />
            </Route>
          </div>
          <Footer />
        </div>
      </Router>
    </ApolloProvider>
  );
}
export default App;
```

```
import React from 'react';

// Import the `useParams()` hook
import { useParams } from 'react-router-dom';
import { useQuery } from '@apollo/client';
import SkillsList from '../components/SkillsList';
import SkillForm from '../components/SkillForm';

import { QUERY_SINGLE_PROFILE } from '../utils/queries';

const Profile = () => {
  // Use `useParams()` to retrieve value of the route parameter `:profileId`
  const { profileId } = useParams();

  const { loading, data } = useQuery(QUERY_SINGLE_PROFILE, {
    // pass URL parameter
    variables: { profileId: profileId },
  });
  const profile = data?.profile || {};
  if (loading) {
    return <div>Loading...</div>;
  }
  return (
    <div>
      <h2 className="card-header">
        {profile.name}'s friends have endorsed these skills...
      </h2>
      {profile.skills?.length > 0 && <SkillsList skills={profile.skills} />}
      <div className="my-4 p-4" style={{ border: '1px dotted #1a1a1a' }}>
        <SkillForm profileId={profile._id} />
      </div>
    </div>
  );
};
export default Profile;
```

```
import React from 'react';

// Import Link component for all internal application hyperlinks
import { Link } from 'react-router-dom';

const ProfileList = ({ profiles, title }) => {
  if (!profiles.length) {
    return <h3>No Profiles Yet</h3>;
  }
  return (
    <div>
      <h3 className="text-primary">{title}</h3>
      <div className="flex-row justify-space-between my-4">
        {profiles &&
          profiles.map((profile) => (
            <div key={profile._id} className="col-12 col-xl-6">
              <div className="card mb-3">
                <h4 className="card-header bg-dark text-light p-2 m-0">
                  {profile.name} <br />
                  <span className="text-white" style={{ fontSize: '1rem' }}>
                    currently has {profile.skills ? profile.skills.length : 0}{' '}
                    endorsed skill
                    {profile.skills && profile.skills.length === 1 ? '' : 's'}
                  </span>
                </h4>
                {/* Use <Link> component to create an internal hyperlink reference.
                        Use `to` prop to set the path instead of `href` */}
                <Link
                  className="btn btn-block btn-squared btn-light text-dark"
                  to={`/profiles/${profile._id}`}
                >
                  View and endorse their skills.
                </Link>
              </div>
            </div>
          ))}
      </div>
    </div>
  );
};

export default ProfileList;
```

```
const { Schema, model } = require('mongoose');
const bcrypt = require('bcrypt');

const profileSchema = new Schema({
  name: {
    type: String,
    required: true,
    unique: true,
    trim: true,
  },
  email: {
    type: String,
    required: true,
    unique: true,
    match: [/.+@.+\..+/, 'Must match an email address!'],
  },
  password: {
    type: String,
    required: true,
    minlength: 5,
  },
  skills: [
    {
      type: String,
      trim: true,
    },
  ],
});

// set up pre-save middleware to create password
profileSchema.pre('save', async function (next) {
  if (this.isNew || this.isModified('password')) {
    const saltRounds = 10;
    this.password = await bcrypt.hash(this.password, saltRounds);
  }
  next();
});

// compare the incoming password with the hashed password
profileSchema.methods.isCorrectPassword = async function (password) {
  return bcrypt.compare(password, this.password);
};

const Profile = model('Profile', profileSchema);

module.exports = Profile;
```

JSON Web Token pt1

```
const { gql } = require('apollo-server-express');

const typeDefs = gql`
  type Profile {
    _id: ID
    name: String
    email: String
    # There is now a field to store the user's password
    password: String
    skills: [String]!
  }
  # Set up an Auth type to handle returning data from a profile creating or user login
  type Auth {
    token: ID!
    profile: Profile
  }
  type Query {
    profiles: [Profile]!
    profile(profileId: ID!): Profile
  }
  type Mutation {
    # Set up mutations to handle creating a profile or logging into a profile and return Auth type
    addProfile(name: String!, email: String!, password: String!): Auth
    login(email: String!, password: String!): Auth
    addSkill(profileId: ID!, skill: String!): Profile
    removeProfile(profileId: ID!): Profile
    removeSkill(profileId: ID!, skill: String!): Profile
  }
`;

module.exports = typeDefs;
```

```
const jwt = require('jsonwebtoken');

const secret = 'mysecretssshhhhhhh';
const expiration = '2h';

module.exports = {
  signToken: function ({ email, name, _id }) {
    const payload = { email, name, _id };
    return jwt.sign({ data: payload }, secret, { expiresIn: expiration });
  },
};
```

JSON Web Token pt2

```
const { AuthenticationError } = require('apollo-server-express');

const { User, Thought } = require('../models');
const { signToken } = require('../utils/auth');

const resolvers = {
  Query: {
    users: async () => {
      return User.find().populate('thoughts');
    },
    user: async (parent, { username }) => {
      return User.findOne({ username }).populate('thoughts');
    },
    thoughts: async (parent, { username }) => {
      const params = username ? { username } : {};
      return Thought.find(params).sort({ createdAt: -1 });
    },
    thought: async (parent, { thoughtId }) => {
      return Thought.findOne({ _id: thoughtId });
    },
  },
  Mutation: {
    addUser: async (parent, { username, email, password }) => {
      // First we create the user
      const user = await User.create({ username, email, password });
      // To reduce friction for the user, we immediately sign a JSON Web Token and
            log the user in after they are created
      const token = signToken(user);
      // Return an `Auth` object that consists of the signed token and user's information
      return { token, user };
    },
    login: async (parent, { email, password }) => {
      // Look up the user by the provided email address. Since the `email` field is unique,
            we know that only one person will exist with that email
      const user = await User.findOne({ email });
      // If there is no user with that email address, return an Authentication error stating so
      if (!user) {
        throw new AuthenticationError('No user found with this email address');
      }
      // If there is a user found, execute the `isCorrectPassword` instance method
            and check if the correct password was provided
      const correctPw = await user.isCorrectPassword(password);
      // If the password is incorrect, return an Authentication error stating so
      if (!correctPw) {
        throw new AuthenticationError('Incorrect credentials');
      }
      // If email and password are correct, sign user into the application with a JWT
      const token = signToken(user);
      // Return an `Auth` object that consists of the signed token and user's information
      return { token, user };
    },
    addThought: async (parent, { thoughtText, thoughtAuthor }) => {
      const thought = await Thought.create({ thoughtText, thoughtAuthor });
      await User.findOneAndUpdate(
        { username: thoughtAuthor },
        { $addToSet: { thoughts: thought._id } }
      );
      return thought;
    },
    addComment: async (parent, { thoughtId, commentText, commentAuthor }) => {
      return Thought.findOneAndUpdate(
        { _id: thoughtId },
        {
          $addToSet: { comments: { commentText, commentAuthor } },
        },
        {
          new: true,
          runValidators: true,
        }
      );
    },
    removeThought: async (parent, { thoughtId }) => {
      return Thought.findOneAndDelete({ _id: thoughtId });
    },
    removeComment: async (parent, { thoughtId, commentId }) => {
      return Thought.findOneAndUpdate(
        { _id: thoughtId },
        { $pull: { comments: { _id: commentId } } },
        { new: true }
      );
    },
  },
};

module.exports = resolvers;
```

Decode JWT pt1

```
import React, { useState } from 'react';
import { Link } from 'react-router-dom';

import { useMutation } from '@apollo/client';
import { LOGIN_USER } from '../utils/mutations';
import Auth from '../utils/auth';

const Login = (props) => {
  const [formState, setFormState] = useState({ email: '', password: '' });
  const [login, { error, data }] = useMutation(LOGIN_USER);

  // update state based on form input changes
  const handleChange = (event) => {
    const { name, value } = event.target;
    setFormState({
      ...formState,
      [name]: value,
    });
  };

  // submit form
  const handleFormSubmit = async (event) => {
    event.preventDefault();
    console.log(formState);
    try {
      const { data } = await login({
        variables: { ...formState },
      });
      Auth.login(data.login.token);
    } catch (e) {
      console.error(e);
    }
    // clear form values
    setFormState({
      email: '',
      password: '',
    });
  };
```

```
  return (
    <main className="flex-row justify-center mb-4">
      <div className="col-12 col-lg-10">
        <div className="card">
          <h4 className="card-header bg-dark text-light p-2">Login</h4>
          <div className="card-body">
            {data ? (
              <p>
                Success! You may now head{' '}
                <Link to="/">back to the homepage.</Link>
              </p>
            ) : (
              <form onSubmit={handleFormSubmit}>
                <input
                  className="form-input"
                  placeholder="Your email"
                  name="email"
                  type="email"
                  value={formState.email}
                  onChange={handleChange}
                />
                <input
                  className="form-input"
                  placeholder="******"
                  name="password"
                  type="password"
                  value={formState.password}
                  onChange={handleChange}
                />
                <button
                  className="btn btn-block btn-info"
                  style={{ cursor: 'pointer' }}
                  type="submit"
                >
                  Submit
                </button>
              </form>
            )}
            {error && (
              <div className="my-3 p-3 bg-danger text-white">
                {error.message}
              </div>
            )}
          </div>
        </div>
      </div>
    </main>
  );
};
export default Login;
```

```
import { gql } from '@apollo/client';
export const LOGIN_USER = gql`
  mutation login($email: String!, $password: String!) {
    login(email: $email, password: $password) {
      token
      user {
        _id
        username
      }
    }
  }
`;
export const ADD_USER = gql`
  mutation addUser($username: String!, $email: String!, $password: String!) {
    addUser(username: $username, email: $email, password: $password) {
      token
      user {
        _id
        username
      }
    }
  }
`;
export const ADD_THOUGHT = gql`
  mutation addThought($thoughtText: String!, $thoughtAuthor: String!) {
    addThought(thoughtText: $thoughtText, thoughtAuthor: $thoughtAuthor) {
      _id
      thoughtText
      thoughtAuthor
      createdAt
      comments {
        _id
        commentText
      }
    }
  }
`;
export const ADD_COMMENT = gql`
  mutation addComment(
    $thoughtId: ID!
    $commentText: String!
    $commentAuthor: String!
  ) {
    addComment(
      thoughtId: $thoughtId
      commentText: $commentText
      commentAuthor: $commentAuthor
    ) {
      _id
      thoughtText
      thoughtAuthor
      createdAt
      comments {
        _id
        commentText
        createdAt
      }
    }
  }
`;
```

Decode JWT pt2

```
import React from 'react';

import {
  ApolloClient,
  InMemoryCache,
  ApolloProvider,
  createHttpLink,
} from '@apollo/client';

import { setContext } from '@apollo/client/link/context';
import { BrowserRouter as Router, Route } from 'react-router-dom';
import Home from './pages/Home';
import Profile from './pages/Profile';
import Signup from './pages/Signup';
import Login from './pages/Login';
import Header from './components/Header';
import Footer from './components/Footer';

// Construct our main GraphQL API endpoint
const httpLink = createHttpLink({
  uri: '/graphql',
});

// Construct request middleware that will attach the JWT token
            to every request as an `authorization` header
const authLink = setContext((_, { headers }) => {
  // get the authentication token from local storage if it exists
  const token = localStorage.getItem('id_token');
  // return the headers to the context so httpLink can read them
  return {
    headers: {
      ...headers,
      authorization: token ? `Bearer ${token}` : '',
    },
  };
});
const client = new ApolloClient({
  // Set up our client to execute the `authLink` middleware prior
            to making the request to our GraphQL API
  link: authLink.concat(httpLink),
  cache: new InMemoryCache(),
});

function App() {
  return (
    <ApolloProvider client={client}>
      <Router>
        <div className="flex-column justify-flex-start min-100-vh">
          <Header />
          <div className="container">
            <Route exact path="/">
              <Home />
            </Route>
            <Route exact path="/login">
              <Login />
            </Route>
            <Route exact path="/signup">
              <Signup />
            </Route>
            <Route exact path="/profiles/:profileId">
              <Profile />
            </Route>
          </div>
          <Footer />
        </div>
      </Router>
    </ApolloProvider>
  );
}
export default App;
```

```
// use this to decode a token and get the user's information out of it
import decode from 'jwt-decode';

// create a new class to instantiate for a user
class AuthService {
  // get user data from JSON web token by decoding it
  getProfile() {
    return decode(this.getToken());
  }
  // return `true` or `false` if token exists (does not verify if it's expired yet)
  loggedIn() {
    const token = this.getToken();
    // If there is a token and it's not expired, return `true`
    return token && !this.isTokenExpired(token) ? true : false;
  }
  isTokenExpired(token) {
    // Decode the token to get its expiration time that was set by the server
    const decoded = decode(token);
    // If the expiration time is less than the current time (in seconds),
            the token is expired and we return `true`
    if (decoded.exp < Date.now() / 1000) {
      localStorage.removeItem('id_token');
      return true;
    }
    // If token hasn't passed its expiration time, return `false`
    return false;
  }  getToken() {
    // Retrieves the user token from localStorage
    return localStorage.getItem('id_token');
  }
  login(idToken) {
    // Saves user token to localStorage and reloads the application
            for logged in status to take effect
    localStorage.setItem('id_token', idToken);
    window.location.assign('/');
  }
  logout() {
    // Clear user token and profile data from localStorage
    localStorage.removeItem('id_token');
    // this will reload the page and reset the state of the application
    window.location.reload();
  }
}
export default new AuthService();
```

Resolver Context pt1

```
const jwt = require('jsonwebtoken');

const secret = 'mysecretssshhhhhhh';
const expiration = '2h';

module.exports = {
  authMiddleware: function ({ req }) {
    // allows token to be sent via req.body, req.query, or headers
    let token = req.body.token || req.query.token || req.headers.authorization;
    // We split the token string into an array and return actual token
    if (req.headers.authorization) {
      token = token.split(' ').pop().trim();
    }
    if (!token) {
      return req;
    }
    // if token can be verified, add the decoded user's data to the request
          so it can be accessed in the resolver
    try {
      const { data } = jwt.verify(token, secret, { maxAge: expiration });
      req.user = data;
    } catch {
      console.log('Invalid token');
    }
    // return the request object so it can be passed to the resolver as `context`
    return req;
  },
  signToken: function ({ email, name, _id }) {
    const payload = { email, name, _id };
    return jwt.sign({ data: payload }, secret, { expiresIn: expiration });
  },
};
```

```
const express = require('express');
const { ApolloServer } = require('apollo-server-express');
const path = require('path');

const { typeDefs, resolvers } = require('./schemas');

// Import `authMiddleware()` function to be configured with the Apollo Server
const { authMiddleware } = require('./utils/auth');

const db = require('./config/connection');
const PORT = process.env.PORT || 3001;
const app = express();

const server = new ApolloServer({
  typeDefs,
  resolvers,
  // Add context to our server so data from the `authMiddleware()`
          function can pass data to our resolver functions
  context: authMiddleware,
});

server.applyMiddleware({ app });
app.use(express.urlencoded({ extended: false }));
app.use(express.json());

if (process.env.NODE_ENV === 'production') {
  app.use(express.static(path.join(__dirname, '../client/build')));
}

app.get('*', (req, res) => {
  res.sendFile(path.join(__dirname, '../client/build/index.html'));
});

db.once('open', () => {
  app.listen(PORT, () => {
    console.log(`API server running on port ${PORT}!`);
    console.log(`Use GraphQL at http://localhost:${PORT}${server.graphqlPath}`);
  });
});
```

Resolver Context pt2

```
const { AuthenticationError } = require('apollo-server-express');

const { Profile } = require('../models');
const { signToken } = require('../utils/auth');

const resolvers = {
  Query: {
    profiles: async () => {
      return Profile.find();
    },
    profile: async (parent, { profileId }) => {
      return Profile.findOne({ _id: profileId });
    },
    // By adding context to our query, we can retrieve the logged in user
          without specifically searching for them
    me: async (parent, args, context) => {
      if (context.user) {
        return Profile.findOne({ _id: context.user._id });
      }
      throw new AuthenticationError('You need to be logged in!');
    },
  },
  Mutation: {
    addProfile: async (parent, { name, email, password }) => {
      const profile = await Profile.create({ name, email, password });
      const token = signToken(profile);
      return { token, profile };
    },
    login: async (parent, { email, password }) => {
      const profile = await Profile.findOne({ email });
      if (!profile) {
        throw new AuthenticationError('No profile with this email found!');
      }
      const correctPw = await profile.isCorrectPassword(password);
      if (!correctPw) {
        throw new AuthenticationError('Incorrect password!');
      }
      const token = signToken(profile);
      return { token, profile };
    },
    // Add a third argument to the resolver to access data in our `context`
    addSkill: async (parent, { profileId, skill }, context) => {
      // If context has a `user` property, that means the user executing
            this mutation has a valid JWT and is logged in
      if (context.user) {
        return Profile.findOneAndUpdate(
          { _id: profileId },
          {
            $addToSet: { skills: skill },
          },
          {
            new: true,
            runValidators: true,
          }
        );
      }
      // If user attempts to execute this mutation and isn't logged in, throw an error
      throw new AuthenticationError('You need to be logged in!');
    },
    // Set up mutation so a logged in user can only remove their profile and no one else's
    removeProfile: async (parent, args, context) => {
      if (context.user) {
        return Profile.findOneAndDelete({ _id: context.user._id });
      }
      throw new AuthenticationError('You need to be logged in!');
    },
    // Make it so a logged in user can only remove a skill from their own profile
    removeSkill: async (parent, { skill }, context) => {
      if (context.user) {
        return Profile.findOneAndUpdate(
          { _id: context.user._id },
          { $pull: { skills: skill } },
          { new: true }
        );
      }
      throw new AuthenticationError('You need to be logged in!');
    },
  },
};
module.exports = resolvers;
```