**Cumulative Project: Codestrips**

Let’s build an API for Codestrips! Codestrips is a small application to allow users to create and save small one-pane comic strips. You can select a head, body, background location, thought/speech bubble type, thought/speech bubble text, and caption for a strip. By the end of this project, you’ll be able to save these to a SQLite database and retrieve them so that they persist even when you restart your server.

You’ll use your knowledge of Express, SQLite, and the sqlite3 node module in order to create a Strip table and then set up a POST route for creating new strips and a GET route to retrieve all strips from the database.

You can start your server from the terminal window with node app.js and stop it with the Ctrl + C key command.

As you progress through the steps, you can test your work by interacting with the front-end web browser component or by running a test suite. To run the test suite in the terminal, enter the mocha command. This will run a test suite for all steps of the project. You won’t be able to run the server and test script in the same terminal window at the same time, so either stop your server before running the test script or open a second terminal window to run mocha.

Note: The files you edit in the code editor will auto-save as you work on them. To see updates from changes you make to your server code, make sure you stop and restart the server from the terminal and then refresh the browser if necessary. You can run the test suite in the terminal with the mocha command.

If you get stuck during this project or would like to see an experienced developer work through it, click “**Get Help**“ to see a **project walkthrough video**.

**Tasks**

**0/22Complete**

Mark the tasks as complete by checking them off

**Setting up the project**

**1.**

In your terminal window, use npm to install express.

Hint

The full command to install Express is npm install express.

**2.**

Inside **app.js**, import Express and create an instance of an Express server called app. Use module.exports to export app.

Hint

Create a server by invoking the express function that you import with require. Once you’ve created a server, you can export it with module.exports = app;.

**3.**

Create a const PORT and set it equal to process.env.PORT || 4001.

**4.**

npm install, import and app.use() body-parsing middleware to parse JSON bodies. Add logging middleware as well (use whatever format of logging that you want).

Hint

You should npm install body-parser in the terminal and require the body-parser package in **app.js** Use its .json() method. If you want a reminder on the body-parser usage, check out the [docs](https://github.com/expressjs/body-parser#examples).

You should require the morgan package for logging. When use morgan, you can invoke it with any of the [predefined formats](https://github.com/expressjs/morgan#predefined-formats) such as 'dev'.

**5.**

Serve the Codestrips site with app.use(express.static('public'));

Start the server listening on the correct PORT with app.listen().

At this point, you can start your server and load the Codestrips site by reloading the browser component. You will be able to interact with the interface for customizing strips, setting the head, body, background, speech bubble, and text fields, but you won’t be able to save or retrieve strips until you complete the steps below.

**Creating the database**

**6.**

Install sqlite3. We’re going to need to use an older version of sqlite3, so use the following command:

npm install sqlite3@^3

Inside **sql.js**, import the sqlite3 library.

**7.**

Still in **sql.js**, create a const db and create a new sqlite3.Database at './db.sqlite'.

Hint

Remember the syntax for making a new sqlite3 database:

const sqlite3 = require('sqlite3');

const db = new sqlite3.Database('path\_to\_file');

**8.**

Use your db to create a table called Strip with the following schema:

* id, an integer as the primary key.
* head, a not-null text column.
* body, a not-null text column.
* background, a not-null text column.
* bubble\_type, a non-null text column.
* bubble\_text, a not-null text column that defaults to an empty string.
* caption, a not-null text column that defaults to an empty string.

Hint

Don’t forget that if you create a table, and try to change the schema, you will need to drop that table before re-creating it. You can do this by always running a [DROP TABLE IF EXISTS](https://sqlite.org/lang_droptable.html) query before running a [CREATE TABLE](http://www.sqlitetutorial.net/sqlite-create-table/) statement.

**9.**

Run the file to create an empty Strip table by running the node sql.js command in your terminal.

Hint

Don’t forget that if you create a table, and try to change the schema, you will need to drop that table before re-creating it. You can do this by always running a [DROP TABLE IF EXISTS](https://sqlite.org/lang_droptable.html) query before running CREATE TABLE.

**10.**

After creating the database, you can add two test strips to the database by running node seed.js in the terminal. If your database has been set up correctly, you will see a message logging that rows have been inserted into Strip.

**Implement Get Strips**

**11.**

Import your SQLite database into **app.js**. You’ll need to require the sqlite3 package, assign it to a constant variable named sqlite3, and create a database variable named db.

Be sure to use

new sqlite3.Database(process.env.TEST\_DATABASE || './db.sqlite');

for your database variable in order for tests to run correctly.

**12.**

In app.js, add a new route to your application. This new route should monitor the /strips endpoint for GET requests.

**13.**

When a GET request is sent to /strips, get an array of all strips from the Strip table.

Hint

* Import your SQLite database into **app.js**. You’ll need to require the sqlite3 package and assign it to a constant variable named sqlite3.
* Create a database variable, const db, and assign it to a new sqlite3.Database at ./db.sqlite.
* Then inside the GET handler, use db.all() to retrieve an array of all strips.

**14.**

Send back the array of all strips in the db.all() callback. Create an object to send in the response and set its strips property equal to the rows returned from the database.

Hint

Your response might look something like this:

res.send({ strips: stripsFromDatabase })

**15.**

If you ran the seed.js script, make sure your GET route behaves as expected by restarting your server and refreshing the web browser component. At the bottom of the page, you should see and be able to click on the names and view the strips from your Strip table!

**Implement Create Strip**

**16.**

In **app.js**, add a new route to your application. The new route should monitor the /strips endpoint for POST requests.

Hint

Use app.post() to set up POST route handlers.

**17.**

When a POST /strips request arrives, the application should validate the strip and send a 400 response if it is invalid.

* The new Strip will arrive as a strip property on the request body. Here is an example req.body.strip:

{

head: 'happy',

body: 'plus',

background: 'boat'

bubbleType: 'statement',

bubbleText: 'Hello, world!',

caption: 'Test strip'

}

* head, body, background, and bubbleType are required. bubbleText and caption have default values (empty string), so they do not need to be validated by the server before being sent to the database in this case. Send a 400 status code if any of the required values are not present in the request.

**18.**

Your POST /strips route should INSERT a new strip into the database using the req.body.strip values.

Hint

Use db.run() to execute an INSERT query with placeholders.

Remember, the column names in the database are snake\_case, commonly used for SQL column names, but the columns in the strip from the request body are camelCase, so you’ll need to take this into account when [interpolating placeholders](https://github.com/mapbox/node-sqlite3/wiki/API#databaserunsql-param--callback).

**19.**

In your INSERT callback, if an error occurs, send back a 500 response status.

Hint

Remember that in the sqlte3 callback functions, the error object is always the first argument, and it is null if it doesn’t exist.

**20.**

Find the newly-created strip if no error occurred. You’ll have to get the proper row from the database with another query.

Hint

Remember that in the db.run() callback, you can use this.lastID to find the id of the newly-created strip. Then, you can use a db.get() call to find the correct strip row.

**21.**

Set a 201 status code and the send the created strip inside the callback of your db.get(). Create an object to send in the response and set its strip property equal to the strip returned from the database. Send this object in the response.

Hint

Your response might look something like this:

res.status(201).send({ strip: stripFromDatabase })

**Finishing up and Solution code**

**22.**

Great work! Verify that everything works as anticipated in the web browser (don’t forget to restart your server), and run the test script in the terminal if you haven’t.

If you want to see code for a possible solution, open the hint below and download the zip file.

Hint

[Solution Code](https://s3.amazonaws.com/codecademy-content/programs/build-apis/project-solutions/mini-capstone-codestrips.zip)