

Generating Tweets from Server

We need to bring back the server to generate varied data again!

WE'LL COVER THE FOLLOWING ^

- Revisiting random text server
- Observations
- Unit tests

Revisiting random text server

Let's bring back our mock server from the autocomplete search and see what we can reuse.

```
// Server

function getRandomString({length}) {
  const characterChoices = "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789 ";
  const characters = [];
  while (characters.length < length) {
    const randomIndex = Math.floor(Math.random() * characterChoices.length);
    characters.push(characterChoices[randomIndex]);
  }
  return characters.join('');
}

function getRandomInteger({min, max}) {
  return Math.floor((Math.random() + min) * (max - min));
}

function generateSuggestion(prefix) {
  const RATIO_EXACT_MATCH = 0.3;
  const RATIO_AUTOCORRECT = 0.1;

  if (Math.random() < RATIO_AUTOCORRECT) {
    return getRandomString({ length: getRandomInteger({min: 1, max: prefix.length}) });
  }

  if (Math.random() < RATIO_EXACT_MATCH) {
    return prefix;
  }

  return prefix + getRandomString({ length: getRandomInteger({min: 1, max: 10}) });
}
```

```

    }

function getAutocompleteHandler(data) {
  const MAX_CHARS = 10;
  const NUM_AUTOCOMPLETE_RESULTS = 10;
  const RATIO_AUXILLERY_DATA = 0.1;

  if (data.length > MAX_CHARS) {
    return [];
  }

  const results = [];
  while (results.length < NUM_AUTOCOMPLETE_RESULTS) {
    const suggestion = generateSuggestion(data)
    if (results.find(result => result.suggestion === suggestion)) {
      continue;
    }

    if (Math.random() < RATIO_AUXILLERY_DATA) {
      for (let i = 0; i < 2; i++) {
        results.push({
          suggestion,
          auxillery: getRandomString({ length: getRandomInteger({min: 5, max: 10}) })
        });
      }
    } else {
      results.push({ suggestion, auxillery: "" });
    }
  }
  return results;
}

const endpoints = {
  "/": {
    "get": () => "hello world"
  },
  "/autocomplete": {
    "get": getAutocompleteHandler
  }
}

// API library

function getFunction(url, data, callback) {
  const domain = url.substring(0, url.indexOf("/"));
  const endpoint = url.substring(url.indexOf("/"), url.length);

  callback(endpoints[endpoint]["get"](data));
}

const api = {
  get: getFunction
};

```

Observations

- We can reuse `getRandomString` to generate gibberish tweets
- We can reuse `getRandomInteger` to generate random IDs

- We'll need to add an endpoint to allow clients to request tweets
- The handler for such a method will need to accept new parameters
- We need to mock a database

```
class Database {
  constructor() {
    this.tweets = [];
  }

  query({lastTweetId, pageSize}) {
    // TODO
  }

  insert(tweet) {
    // TODO
  }
}

const database = new Database();

function getTweetsHandler(data) {
  const pageSize = data.pageSize;
  const sortOrder = data.sortOrder;
  const lastTweetId = data.lastTweetId;

  if (sortOrder !== 'recent') {
    throw new Error('I dont know how to handle that');
  }

  return database.query({lastTweetId, pageSize});
}

function postTweetHandler(data) {
  database.insert(data.tweet);
}

const endpoints = {
  "/tweets": {
    "get": getTweetsHandler,
    "post": postTweetHandler
  }
}
```

We're going to ignore the strictly server-side concerns here like querying databases, distributed servers, etc. Our database will just be an array, and querying will be for-looping.

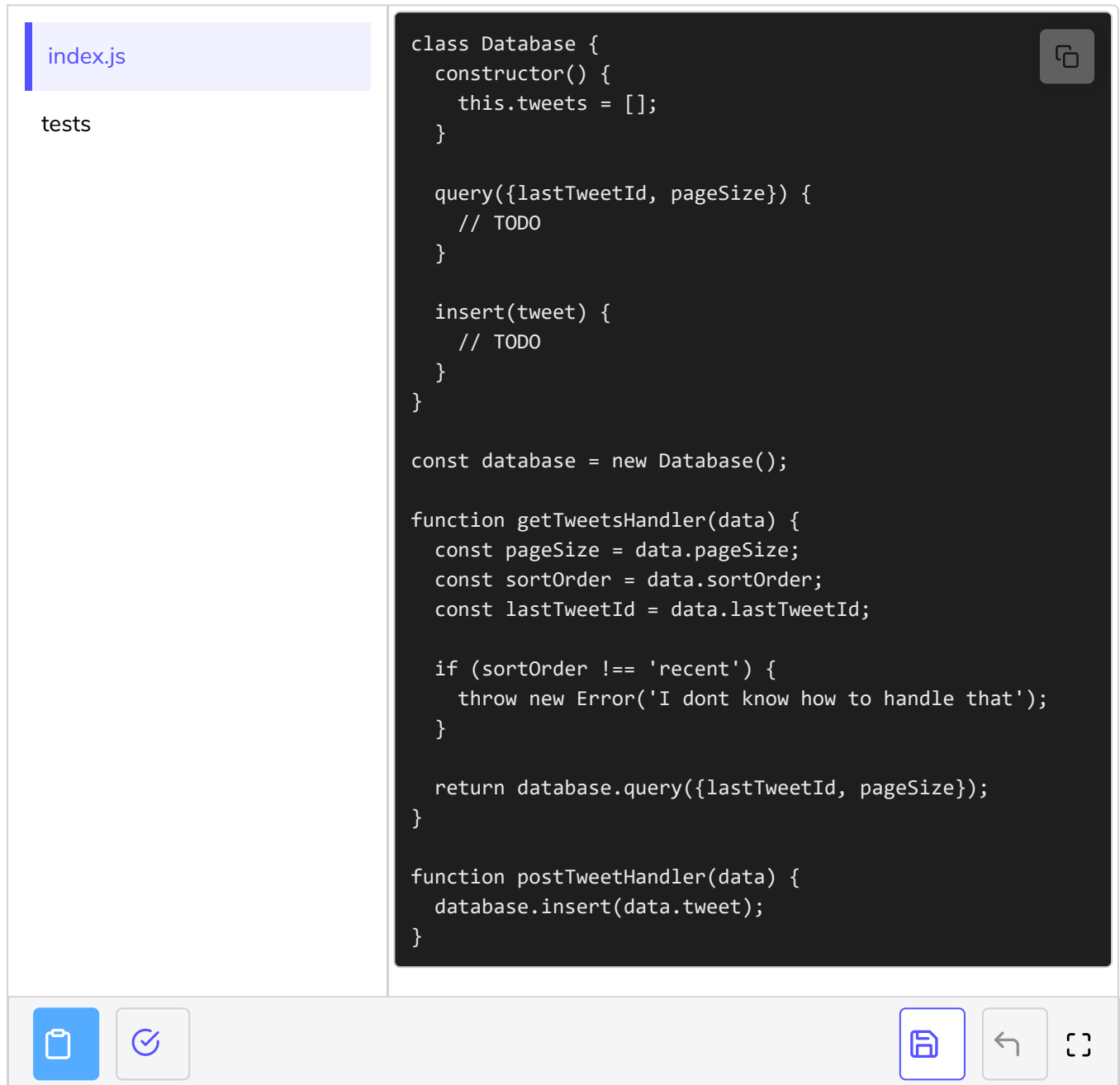
Unit tests

Let's actually write some unit tests first. Our initial get should return an empty list. We'll post one tweet, and check that the following `get` includes that tweet. We'll post another, and check that the `get` returns the tweets in order of most recent (based on when the server received it)

recent (based on when the server received it).

The implementation should be very short. We're not taking into consideration `lastTweetId` or `pageSize` yet.

You can view the actual tests on the “test” tab.



The screenshot shows a code editor with a file explorer on the left and a code editor on the right. The file explorer shows a file named `index.js` and a folder named `tests`. The code editor displays the following JavaScript code:

```
class Database {
  constructor() {
    this.tweets = [];
  }

  query({lastTweetId, pageSize}) {
    // TODO
  }

  insert(tweet) {
    // TODO
  }
}

const database = new Database();

function getTweetsHandler(data) {
  const pageSize = data.pageSize;
  const sortOrder = data.sortOrder;
  const lastTweetId = data.lastTweetId;

  if (sortOrder !== 'recent') {
    throw new Error('I dont know how to handle that');
  }

  return database.query({lastTweetId, pageSize});
}

function postTweetHandler(data) {
  database.insert(data.tweet);
}
```

It's not important for us to go over the full implementation of the server. Let's assume we now have a working server that'll handle requests correctly.