

Chall

- [Link.](#)

The screenshot shows the freeCodeCamp website interface for the 'Stock Price Checker' challenge. The page is titled 'Information Security' and 'Information Security Projects'. The challenge description states: 'Build a full stack JavaScript app that is functionally similar to this: <https://stock-price-checker.freecodecamp.rocks/>. Since all reliable stock price APIs require an API key, we've built a workaround. Use <https://stock-price-checker-proxy.freecodecamp.rocks/> to get up-to-date stock price information without needing to sign up for your own key. Working on this project will involve you writing your code using one of the following methods:

- Clone [this GitHub repo](#) and complete your project locally.
- Use our [Gitpod starter project](#) to complete your project.
- Use a site builder of your choice to complete the project. Be sure to incorporate all the files from our GitHub repo.

Instructions for completion:

1. Set the `NODE_ENV` environment variable to `test`, without quotes
2. Complete the project in `routes/api.js` or by creating a handler/controller
3. You will add any security features to `server.js`
4. You will create all of the functional tests in `tests/2_functional-tests.js`

Note Privacy Considerations: Due to the requirement that only 1 like per IP should be accepted, you will have to save IP addresses. It is important to remain compliant with data privacy laws such as the General Data Protection Regulation. One option is to get permission to save the user's data, but it is much simpler to anonymize it. For this challenge, remember to anonymize IP addresses before saving them to the database. If you need ideas on how to do this, you may choose to hash the data, truncate it, or set part of the IP address to 0.

Write the following tests in `tests/2_functional-tests.js`:

- Viewing one stock: GET request to `/api/stock-prices/`
- Viewing one stock and liking it: GET request to `/api/stock-prices/`
- Viewing the same stock and liking it again: GET request to `/api/stock-prices/`
- Viewing two stocks: GET request to `/api/stock-prices/`
- Viewing two stocks and liking them: GET request to `/api/stock-prices/`

Solution Link
ex: <https://3000-project-url.gitpod.io/>

Source Code Link
ex: <https://your-git-repo.url/files>

Remember to submit the link to your source code.

I've completed this challenge

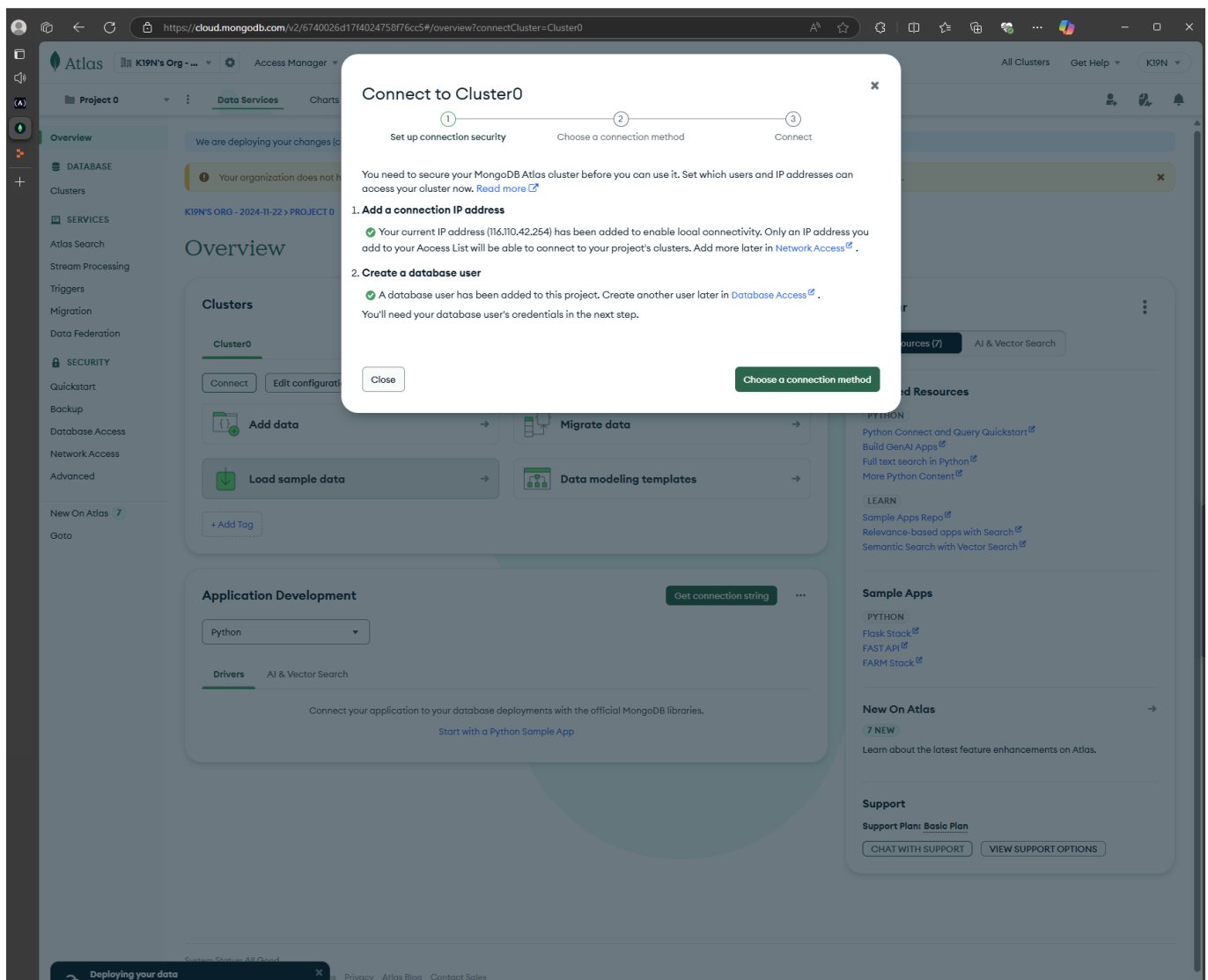
Get a Hint

- Yêu cầu bài kiểm thử cần triển khai:
 - Trong file `tests/2_functional-tests.js`, bạn cần viết các bài kiểm thử sau:
 - Xem thông tin một cổ phiếu
 - Thực hiện GET request tới endpoint `/api/stock-prices/`.
 - Xem thông tin một cổ phiếu và thích (like) nó
 - Thực hiện GET request tới endpoint `/api/stock-prices/`.
 - Xem cùng một cổ phiếu và thích (like) nó lần nữa
 - Thực hiện GET request tới endpoint `/api/stock-prices/`.
 - Xem thông tin hai cổ phiếu
 - Thực hiện GET request tới endpoint `/api/stock-prices/`.
 - Xem thông tin hai cổ phiếu và thích (like) chúng
 - Thực hiện GET request tới endpoint `/api/stock-prices/`.
- Một vài lưu ý:

- Đảm bảo ứng dụng của bạn tương thích với proxy API.
- Tuân thủ yêu cầu bảo mật IP để tránh vi phạm các quy định bảo mật dữ liệu (GDPR).
- Tích hợp đầy đủ các bài kiểm thử để kiểm tra chức năng của ứng dụng.
- Hoàn thiện đầy đủ cả phần frontend và backend để ứng dụng hoạt động trơn tru.

Solution

- Tạo cluster với **Mongoodb**.



- Tạo xong nhớ connect nha.

×

Connect to Cluster0

✓

✓

3

Set up connection security

Choose a connection method

Connect

Connecting with MongoDB Driver

1. Select your driver and version

We recommend installing and using the latest driver version.

Driver

Version

Node.js

6.7 or later

2. Install your driver

Run the following on the command line

npm install mongodb

📋

[View MongoDB Node.js Driver installation instructions.](#)

3. Add your connection string into your application code

Use this connection string in your application

☐ View full code sample

mongodb+srv://CayCon:<db_password>@cluster0.na6dd.mongodb.net/?retryWrites=true&w=majority&appName=Cluster0

📋

Replace **<db_password>** with the password for the **CayCon** database user. Ensure any option params are [URL encoded](#).

RESOURCES

[Get started with the Node.js Driver](#)

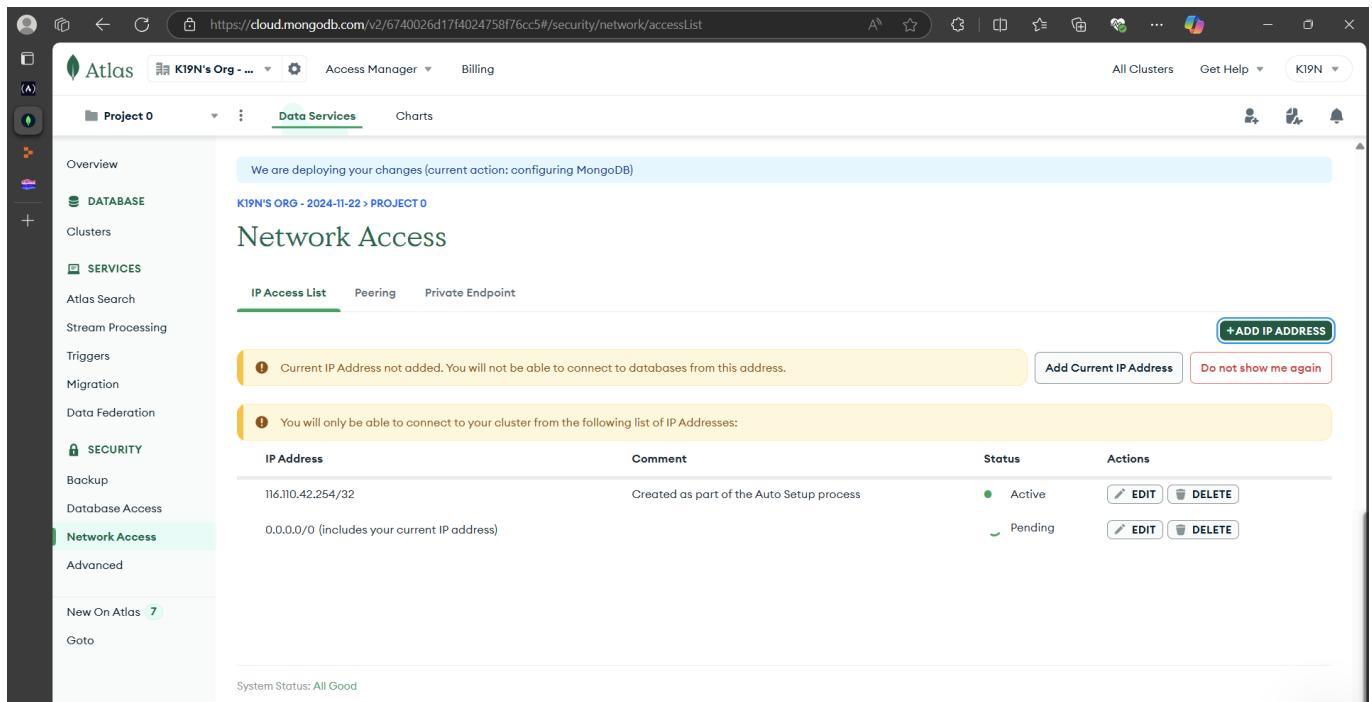
[Access your Database Users](#)

[Node.js Starter Sample App](#)

[Troubleshoot Connections](#)

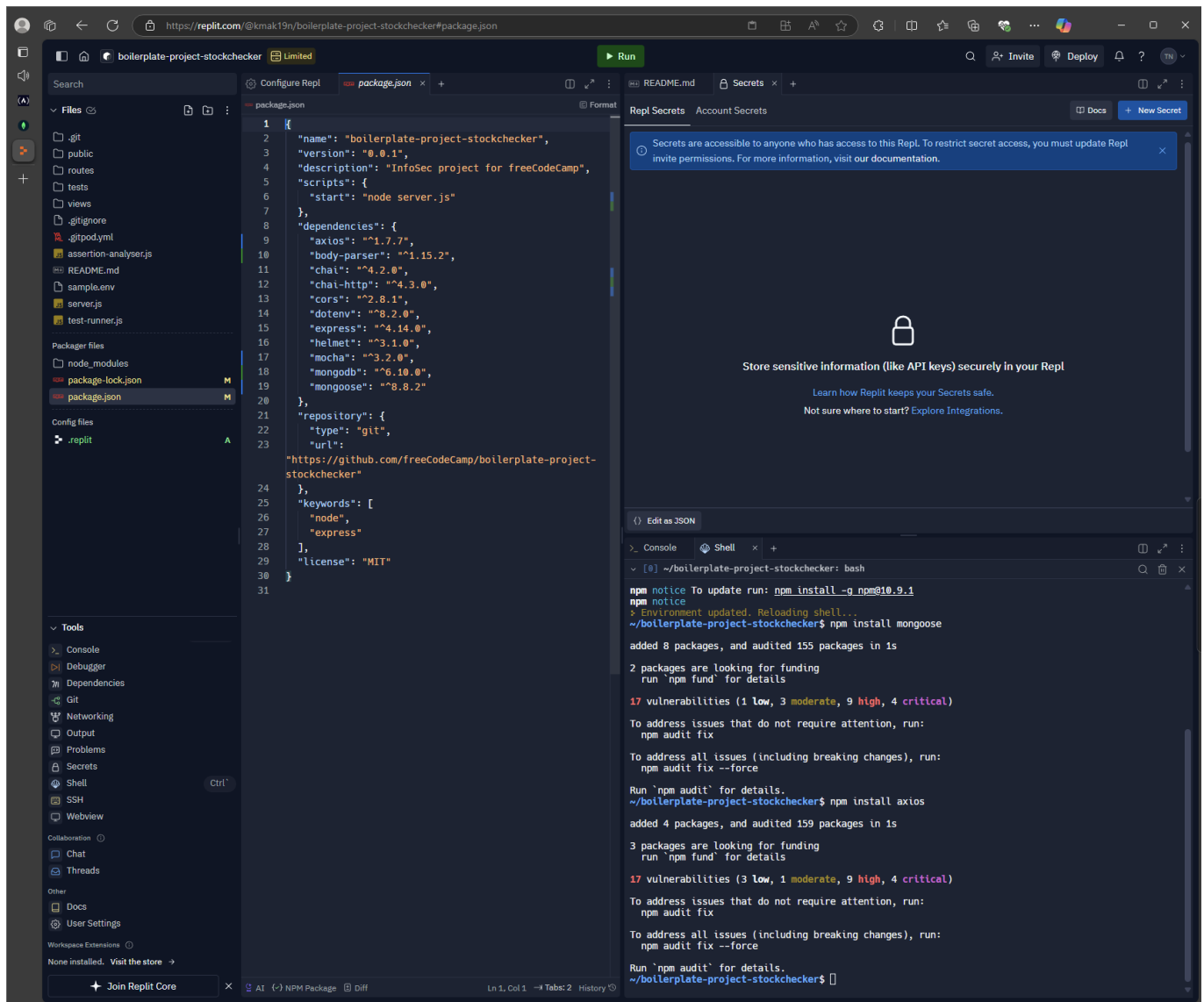
Go Back

Done



- Sau đó ta cài đặt các thứ cần thiết như **mongodb**; **mongoose**; **axios** với lệnh sau:

```
npm install mongodb
npm install mongoose
npm install axios
```



- Check ok r thì ta tạo **Secret** với phần **pass** là link từ **mongoose** ta tạo lúc nãy.
- Nhớ thay **<db_password>** thành pass mà mình tạo.
- Lưu ý là hãy encode pass bằng **encode url** nếu lỗi:)))
- Thêm các thư viện cần thiết vào **server.js**:

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

- Tiếp theo Mongoose sẽ sử dụng **global.Promise**, là **Promise** gốc được tích hợp trong JavaScript (ES6 trở lên).

```
mongoose.Promise = global.Promise;
```

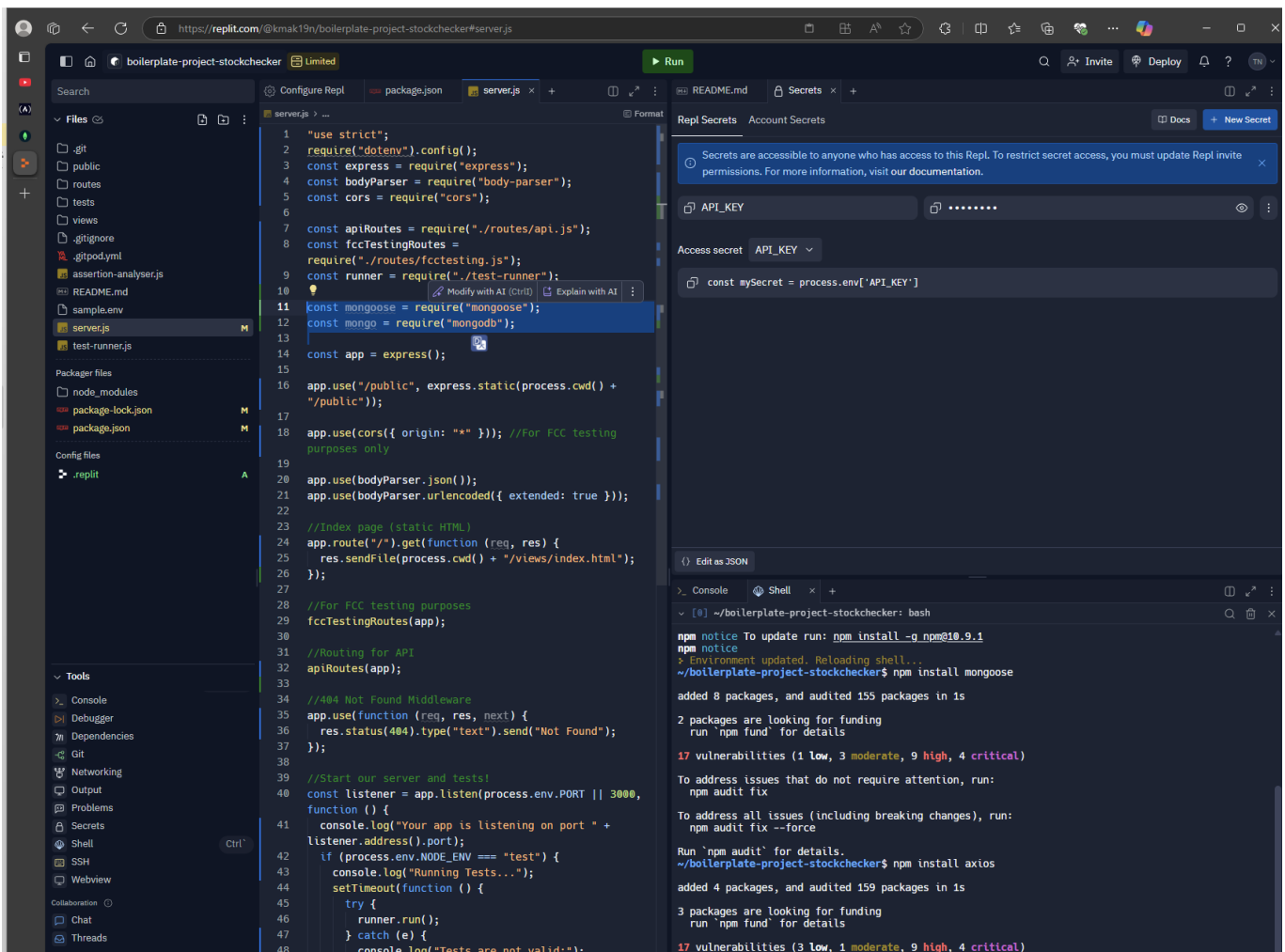
- **MONGO_URI** là chuỗi kết nối (connection string) để kết nối đến cơ sở dữ liệu MongoDB.
- Sử dụng `process.env` để lấy giá trị biến môi trường này, giúp bảo mật thông tin nhạy cảm (username, password).

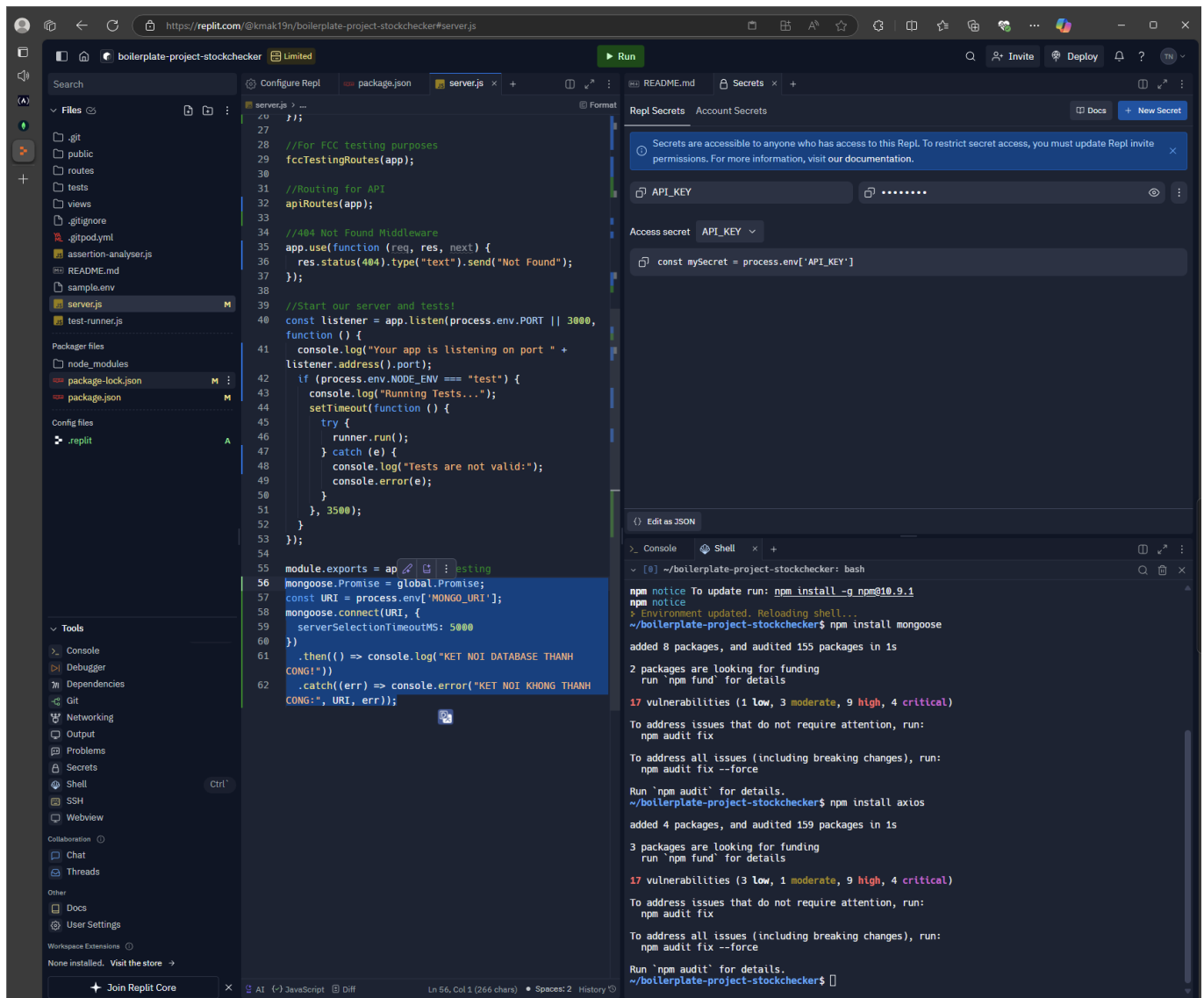
```
const URI = process.env['MONGO_URI'];
```

- Thông tin MONGO_URI thường được lưu trong file .env hoặc thiết lập trên hệ thống server thay vì mã nguồn.
- Thiết lập kết nối với MongoDB
- Sử dụng lệnh mongoose.connect() để kết nối ứng dụng Node.js với MongoDB qua Mongoose.

```
mongoose
```

```
.connect(URI, {  
  serverSelectionTimeoutMS: 5000,  
})  
.then(() => console.log("KẾT NỐI DATABASE THÀNH CÔNG!"))  
.catch((err) => console.error("KẾT NỐI KHÔNG THÀNH CÔNG:", URI, err));
```





- Tiếp theo, ta sẽ tạo Mô hình và Sơ đồ.
- Mỗi cổ phiếu cần lưu thông tin về số lượt "like" và danh sách địa chỉ IP đã "like" cổ phiếu đó.
- Khởi tạo stockSchema để định nghĩa cấu trúc tài liệu trong MongoDB:
 - code: Mã cổ phiếu (kiểu String).
 - likes: Danh sách địa chỉ IP (kiểu mảng String), mặc định là mảng rỗng.
 - Tạo Model từ schema với tên là 'stock':

```
const stockSchema = new mongoose.Schema({
  code: String,
  likes: { type: [String], default: [] }
});
const Stock = mongoose.model('stock', stockSchema);
```

- Giờ ta sẽ tiến hành từng bài test 1.

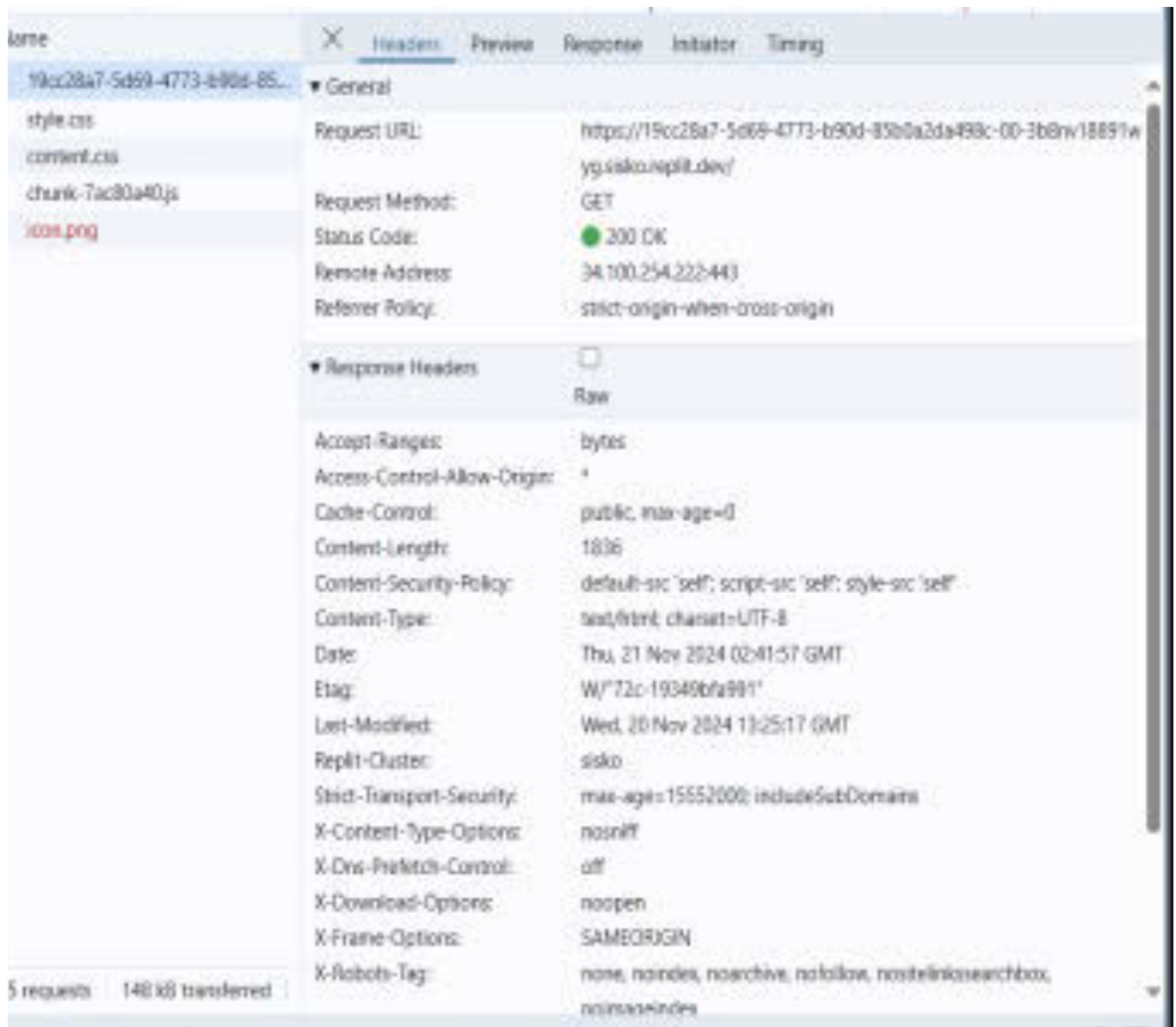
Test 1

- Tiến hành cấu hình trong file `server.js`

```
app.use(express.json());
app.use(express.json());
app.use(express.json());
app.use(express.json());

app.use(helmet.contentSecurityPolicy({ directives: {
  defaultSrc: ["'self'"],
  scriptSrc: ["'self'"],
  styleSrc: ["'self'"] }}}) )
```

- Restart lại là ok rồi



- Nếu không cấu hình, ứng dụng sẽ gặp các rủi ro bảo mật nghiêm trọng:
 - Dễ bị tấn công XSS (Cross-Site Scripting):

- Kẻ tấn công có thể chèn mã JavaScript độc hại qua:
 - Trường nhập liệu của người dùng.
 - Các tập lệnh từ bên thứ ba bị xâm phạm.
- Tải tài nguyên từ nguồn không đáng tin cậy:
 - Ứng dụng có thể vô tình tải script hoặc style từ domain bên ngoài không an toàn, tạo cơ hội cho kẻ tấn công chèn mã độc.
- Nguy cơ bị Clickjacking:
 - Nếu thiếu header bảo mật, ứng dụng có thể bị nhúng trong iframe độc hại, dẫn đến việc người dùng thực hiện hành động không mong muốn mà không nhận ra.
- Không tuân thủ tiêu chuẩn bảo mật:
 - CSP (Content Security Policy) là tiêu chuẩn bảo mật quan trọng. Nếu không thiết lập, ứng dụng của bạn có thể bị trình duyệt hoặc các công cụ kiểm tra đánh dấu là không an toàn.

Test 2

- Đầu tiên là lập **saveStock**
- Hàm này quản lý việc lưu hoặc cập nhật thông tin cổ phiếu trong cơ sở dữ liệu:
 - Trường hợp cổ phiếu chưa tồn tại:
 - Tạo mới một tài liệu cổ phiếu với mã code.
 - Nếu người dùng thích cổ phiếu (like = true), thêm địa chỉ IP vào mảng likes.
 - Lưu tài liệu vừa tạo vào cơ sở dữ liệu.
 - Trường hợp cổ phiếu đã tồn tại:
 - Nếu người dùng thích cổ phiếu (like = true) và IP của họ chưa có trong mảng likes, thêm IP vào danh sách.
 - Lưu các thay đổi.
 - Kết quả cuối cùng của hàm là trả về thông tin đã lưu hoặc tạo mới cổ phiếu.

```
function saveStock(code, like, ip) {  
  return Stock.findOne({ code: code }).then((foundStock) => {  
    if (!foundStock) {  
      let newStock = new Stock({ code: code, likes: like ? [ip] : [] });  
      return newStock.save();  
    } else {  
      if (like && foundStock.likes.indexOf(ip) === -1) {  
        foundStock.likes.push(ip);  
      }  
      return foundStock.save();  
    }  
  });  
}
```

- Tiếp theo là hàm **parseData**
- Hàm này xử lý dữ liệu kết hợp từ cơ sở dữ liệu và API để định dạng thông tin trả về cho client:
 - Kiểm tra thông tin giá cổ phiếu:
 - Nếu thông tin hợp lệ, trả về mã cổ phiếu, giá và số lượt "like".
 - Nếu không, báo lỗi và trả về giá N/A.
 - Tính chênh lệch lượt "like" (nếu có nhiều hơn một cổ phiếu):
 - Tính toán sự chênh lệch lượt "like" giữa các cổ phiếu và thêm trường rel_likes.
 - Kết quả cuối cùng:
 - Trả về dữ liệu đã định dạng hoặc ghi log nếu có lỗi.

```
function parseData(results) {
  let stockData = results.map(([saveResult, requestResult]) => {
    let stockInfo = requestResult.data;
    if (stockInfo && stockInfo.latestPrice) {
      return {
        stock: saveResult.code,
        price: stockInfo.latestPrice,
        likes: saveResult.likes ? saveResult.likes.length : 0
      };
    } else {
      console.error(`Error: No valid price data for stock code ${saveResult.code}`);
      return {
        stock: saveResult.code,
        price: 'N/A',
        likes: saveResult.likes ? saveResult.likes.length : 0
      };
    }
  });

  if (stockData.length > 1) {
    stockData[0].rel_likes = stockData[0].likes - stockData[1].likes;
    stockData[1].rel_likes = stockData[1].likes - stockData[0].likes;
  }

  console.log(stockData);
  return stockData;
}
```

Test 3

- Ta cần xử lý yêu cầu GET cho một mã cổ phiếu, không tính lượt "like" và luôn trả về likes: 0. Nếu không có giá hợp lệ từ API, trả về giá trị mặc định là 0. Bảo vệ bằng cách xử lý lỗi và trả về thông báo lỗi nếu

có sự cố trong quá trình gửi yêu cầu hoặc nhận phản hồi từ API.

- Mã cổ phiếu (ví dụ: "aapl", "tsla") được lấy từ tham số stock trong query, và được chuyển thành chữ hoa để đảm bảo tính nhất quán khi gửi tới API.
- Đây là yêu cầu HTTP GET sử dụng thư viện axios để lấy thông tin giá cổ phiếu từ một API bên ngoài (<https://stock-price-checker-proxy.freecodecamp.rocks/v1/stock/{stockName}/quote>). - Địa chỉ này trả về dữ liệu về mã cổ phiếu mà người dùng yêu cầu.
- **stockName** là mã cổ phiếu được chuyển thành chữ hoa.
- **axios.get()** gửi yêu cầu GET và nhận phản hồi từ API.
- Sau khi nhận phản hồi, nếu dữ liệu giá không có, giá trị mặc định là 0 sẽ được sử dụng.
- Dữ liệu trả về cho người dùng dưới dạng JSON, bao gồm: mã cổ phiếu (stock), giá cổ phiếu (price), và số lượt "like" (likes).

```
if (typeof req.query.stock === 'string') {
  let stockName = req.query.stock.toUpperCase();
  axios.get(`https://stock-price-checker-
proxy.freecodecamp.rocks/v1/stock/${stockName}/quote`)
    .then((response) => {
      const stockInfo = response.data || {};
      res.json({
        stockData: {
          stock: stockName,
          price: stockInfo.latestPrice || 0,
          likes: saveStock.likes.length,
        },
      });
    })
    .catch((error) => {
      console.error('Error processing stock:', error);
      res.status(500).send('Error processing stock');
    });
}
```

Test 4

- Để lấy địa chỉ IP thực của người dùng trong ứng dụng Express, bạn cần kích hoạt thuộc tính 'trust proxy' trong tệp server.js:

```
app.set('trust proxy', true);
```

- Sau đó, bạn có thể lấy địa chỉ IP thực từ yêu cầu bằng cách sử dụng req.ip:

```
const userIP = req.ip;
```

- Tiếp theo, ta tạo biến để nhận tham số stock và like từ query string, đồng thời lấy địa chỉ IP của người dùng từ req.ip.

```
app.route('/api/stock-prices')
  .get((req, res) => {
    const stockQuery = req.query.stock; // Mã cổ phiếu từ query
    const like = req.query.like === 'true'; // Kiểm tra xem có "like" không
    const userIP = req.ip; // Lấy địa chỉ IP của người dùng
```

- Gọi hàm saveStock để lưu hoặc cập nhật mã cổ phiếu và xử lý lượt "like":

```
    const stockName = stockQuery.toUpperCase(); // Viết hoa mã cổ phiếu
    saveStock(stockName, like, userIP)
      .then((savedStock) => {
        // Lấy thông tin giá cổ phiếu từ API bên ngoài
        return axios.get(`https://stock-price-checker-
proxy.freecodecamp.rocks/v1/stock/${stockName}/quote`)
          .then((response) => {
            const stockInfo = response.data || {}; // Nếu không có dữ liệu, dùng
đối tượng rỗng
            res.json({
              stockData: {
                stock: savedStock.code,
                price: stockInfo.latestPrice || 'N/A', // Giá cổ phiếu hoặc 'N/A'
nếu không có
                likes: savedStock.likes.length, // Số lượt like
              },
            });
          });
      });
    });
  });
```

- Như vậy, ứng dụng sẽ lưu thông tin cổ phiếu, xử lý lượt "like", lấy giá cổ phiếu từ API và trả về dữ liệu dưới dạng JSON.

Test 5

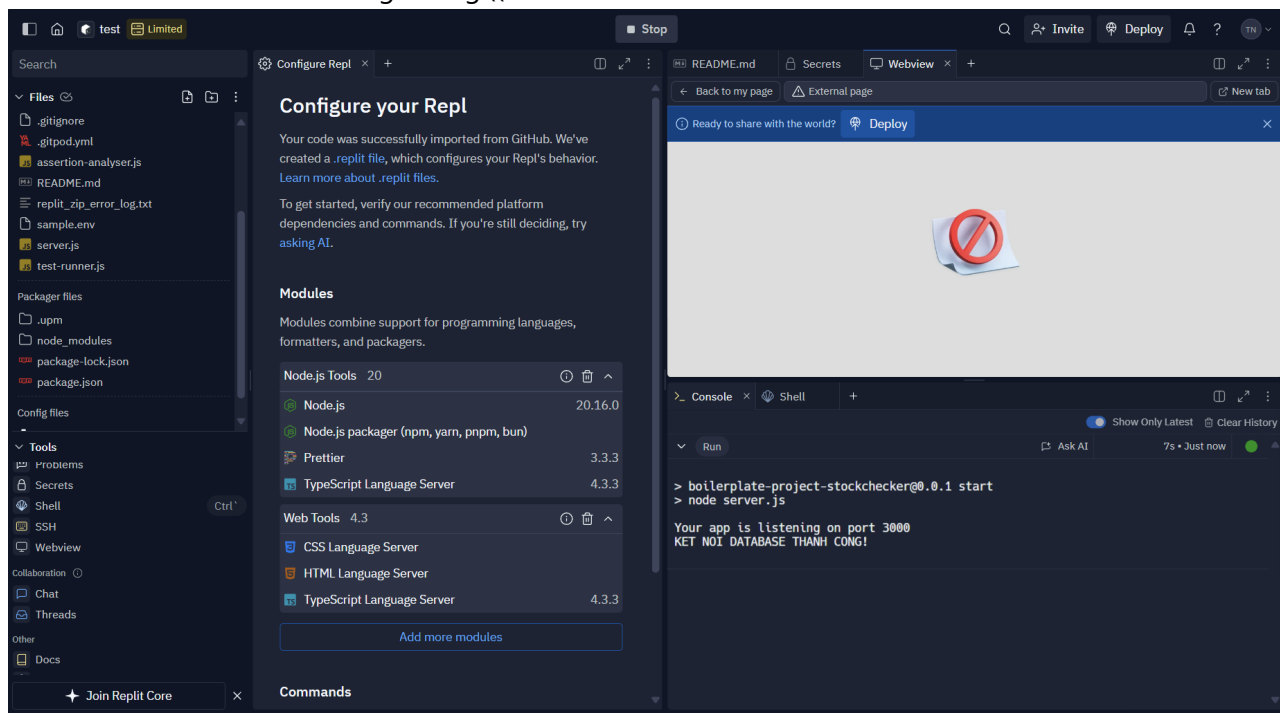
- Đoạn mã **if (stockResults.length === 2)** đảm bảo chỉ thực hiện khi mảng stockResults chứa đúng hai cổ phiếu.
- **rel_likes** là một thuộc tính bổ sung cho mỗi cổ phiếu, dùng để biểu thị sự chênh lệch về lượt thích giữa hai cổ phiếu.
- Tính toán sự chênh lệch lượt thích giữa cổ phiếu đầu tiên (stockResults[0]) và cổ phiếu thứ hai (stockResults[1]):

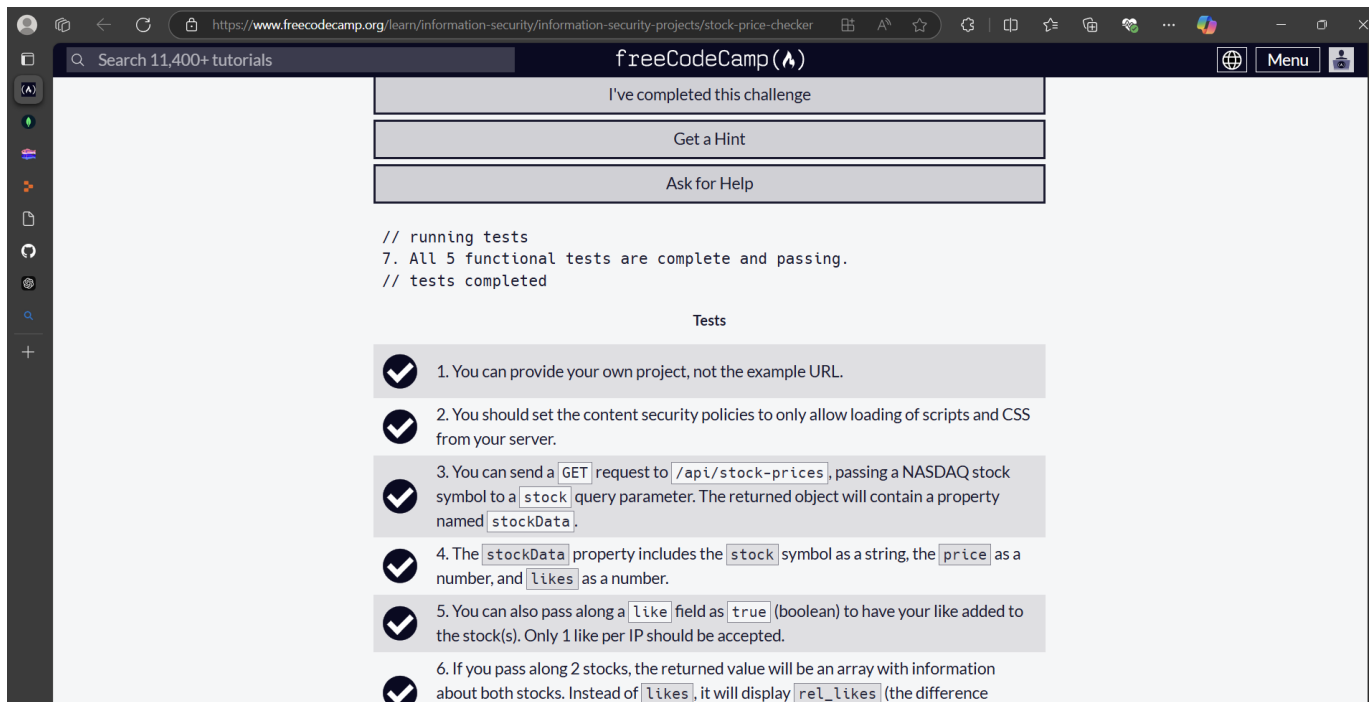
```
stockResults[0].rel_likes = stockResults[0].likes - stockResults[1].likes;
```

- Nếu cổ phiếu đầu tiên có nhiều lượt thích hơn, rel_likes sẽ là số dương.
- Nếu cổ phiếu đầu tiên có ít lượt thích hơn, rel_likes sẽ là số âm.
- Nếu lượt thích bằng nhau, rel_likes sẽ bằng 0.
- Tính toán cho cổ phiếu thứ hai:

```
stockResults[1].rel_likes = stockResults[1].likes - stockResults[0].likes;
```

- Giá trị này luôn là nghịch đảo của giá trị rel_likes của cổ phiếu đầu tiên. Ví dụ, nếu stockResults[0].rel_likes = 3, thì stockResults[1].rel_likes = -3.
- Nếu không trả về dữ liệu đầy đủ qua res.json(), hệ quả sẽ là:
 - Client sẽ không nhận được kết quả rõ ràng và đầy đủ về cổ phiếu. Cụ thể:
 - Nếu không trả về giá (price), người dùng sẽ không biết giá hiện tại của cổ phiếu.
 - Nếu không trả về số lượt thích (likes), người dùng sẽ không biết mức độ yêu thích của cổ phiếu.
- Hành vi của API: Nếu không trả về dữ liệu đầy đủ, API có thể trả về lỗi hoặc dữ liệu thiếu, không đáp ứng được yêu cầu của các test case.
- Submit link để check xem đúng không:((





The screenshot shows the freeCodeCamp interface for a challenge titled 'stock-price-checker'. The browser address bar shows the URL: `https://www.freecodecamp.org/learn/information-security/information-security-projects/stock-price-checker`. The page has a dark header with the freeCodeCamp logo and a search bar. Below the header, there are three buttons: 'I've completed this challenge', 'Get a Hint', and 'Ask for Help'. The main content area displays the following text:

```
// running tests
7. All 5 functional tests are complete and passing.
// tests completed
```

Below this text, there is a section titled 'Tests' with a list of six items, each preceded by a checkmark icon:

1. You can provide your own project, not the example URL.
2. You should set the content security policies to only allow loading of scripts and CSS from your server.
3. You can send a `GET` request to `/api/stock-prices`, passing a NASDAQ stock symbol to a `stock` query parameter. The returned object will contain a property named `stockData`.
4. The `stockData` property includes the `stock` symbol as a string, the `price` as a number, and `likes` as a number.
5. You can also pass along a `like` field as `true` (boolean) to have your like added to the stock(s). Only 1 like per IP should be accepted.
6. If you pass along 2 stocks, the returned value will be an array with information about both stocks. Instead of `likes`, it will display `rel_likes` (the difference

- xong r nhaaaa