INFORMATION SECURITY

Department of Information Systems - Hanyang University
Mid Term Exam – Bryan Yehuda (9024520223)

ASSIGNMENT

Your program should show response messages of the following two requests:
API doc -> REST API -> Wallet -> Get balances (https://docs.ftx.com/#get-balances)
Show the return message of this API call

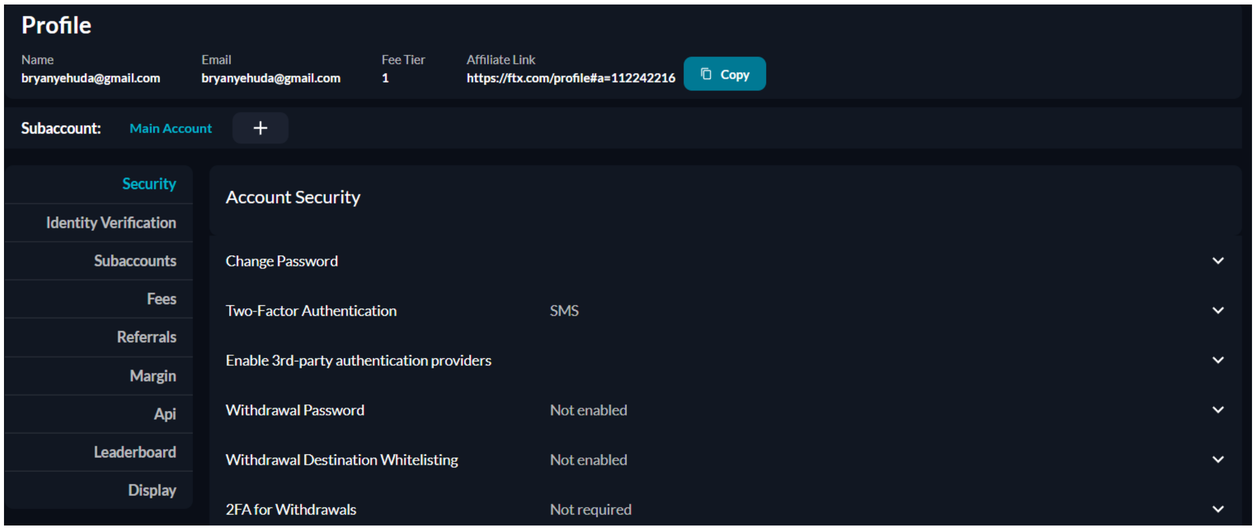
API doc -> REST API -> Open orders (https://docs.ftx.com/#get-open-orders)
Show the return message of this API call

- Provide your source codes for A and B in nicely formatted text (no screenshots).
- Screenshots of returning messages of A and B.
- Write one paragraph explaining your program and any difficulties you had. Even if you can't do the entire exam, submit as much as you can with explanation

RESULT

Registering to FTX

Before we can do any code or programming with the API Processing inside FTX's Server, we need to register for an account inside of FTX. I used my email to register for this account and it only takes me about 5 minutes for my account to be made and I am ready to browse all kinds of services that are offered by FTX. The image below is my account to be used in this project.



Profile

Name	Email	Fee Tier	Affiliate Link
bryanyehuda@gmail.com	bryanyehuda@gmail.com	1	https://ftx.com/profile#a=112242216

Subaccount: Main Account

Account Security

Change Password		
Two-Factor Authentication	SMS	
Enable 3rd-party authentication providers		
Withdrawal Password	Not enabled	
Withdrawal Destination Whitelisting	Not enabled	
2FA for Withdrawals	Not required	

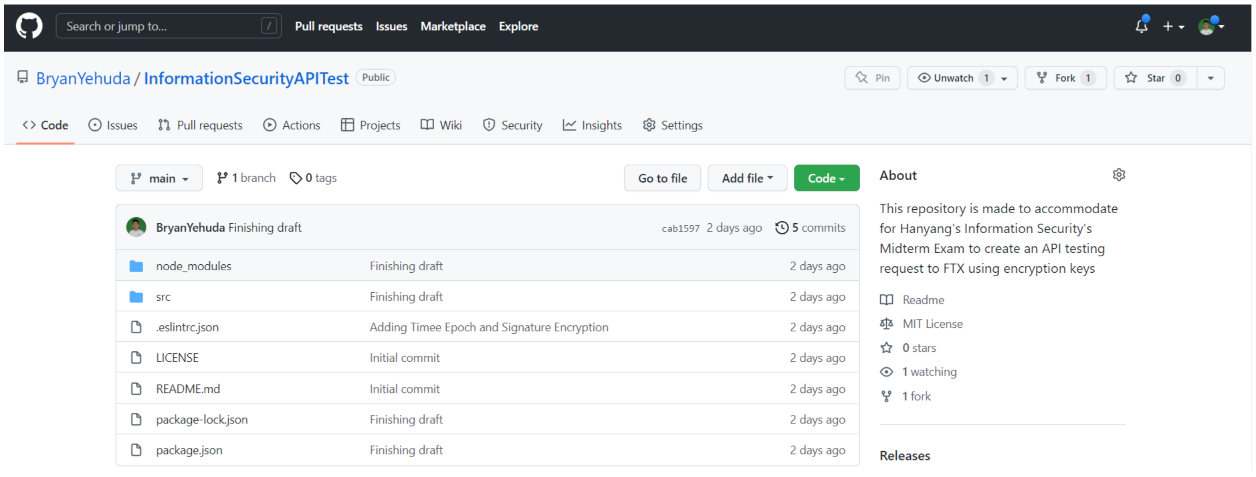
Creating an API Key

After successfully registering for an account in FTX, we can continue to create our own Public API Key and Secret API Key pairs to be used in our project. To create this API Key pair, we need to go to https://ftx.com/settings/api and scroll down to the bottom of the page. Simply press on the "Create API Key" button and FTX will automatically give you your API Key pair. Do not forget to copy this API Key pair so that it can be used in our project.

```
API KEY     : _njFKoPZrOV_9Tme2-m5vQGP8eCL3nKoJ2GTuuFU
API SECRET  : 9nHenFZftNXI5vPVcEW62lZtMpqGbhyIExQOaMTU
```

Setting Up the Environment for Javascript

I will be using Javascript for this project. Since this project will be run on the Node (Terminal) we will be installing NPM or Node Package Manager which helps us to manage all the packages needed to run Javascript on the Node. The first environment needed for this project is creating a Repository to help us contain and backtrack all of our patches into the code. The repository will be hosted in Github at this link https://github.com/BryanYehuda/InformationSecurityAPITest. This will help us to do version control on our Code.



BryanYehuda Finishing draft cab1597 2 days ago 5 commits

node_modules	Finishing draft	2 days ago
src	Finishing draft	2 days ago
.eslintrc.json	Adding Timee Epoch and Signature Encryption	2 days ago
LICENSE	Initial commit	2 days ago
README.md	Initial commit	2 days ago
package-lock.json	Finishing draft	2 days ago
package.json	Finishing draft	2 days ago

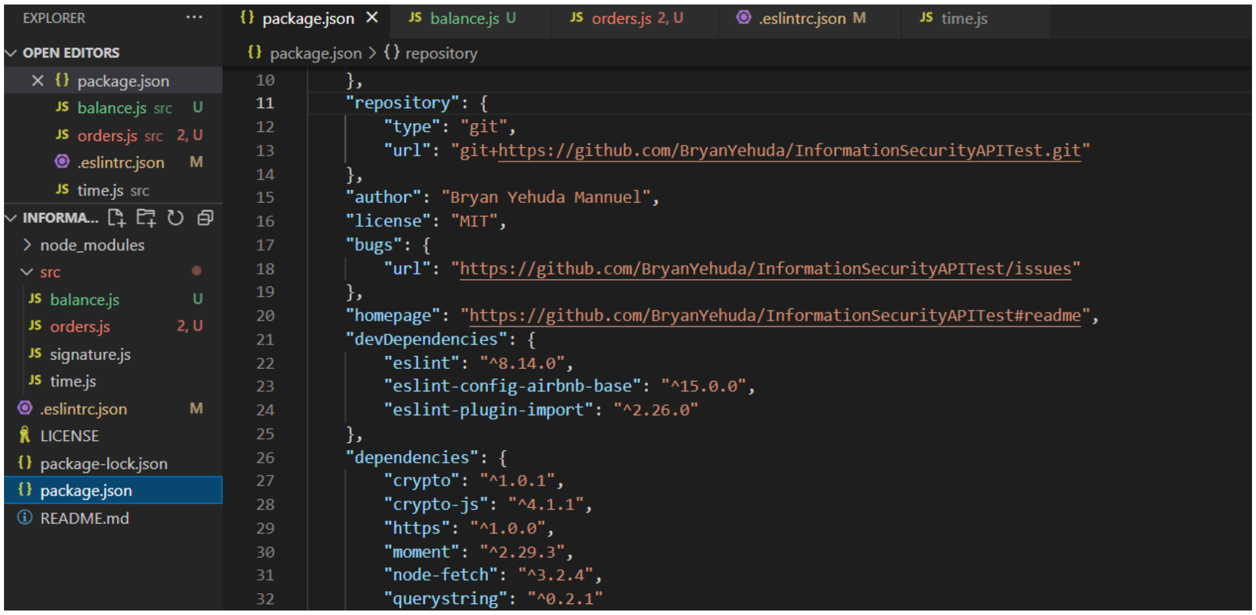
About

This repository is made to accommodate for Hanyang's Information Security's Midterm Exam to create an API testing request to FTX using encryption keys

Readme
MIT License
0 stars
1 watching
1 fork

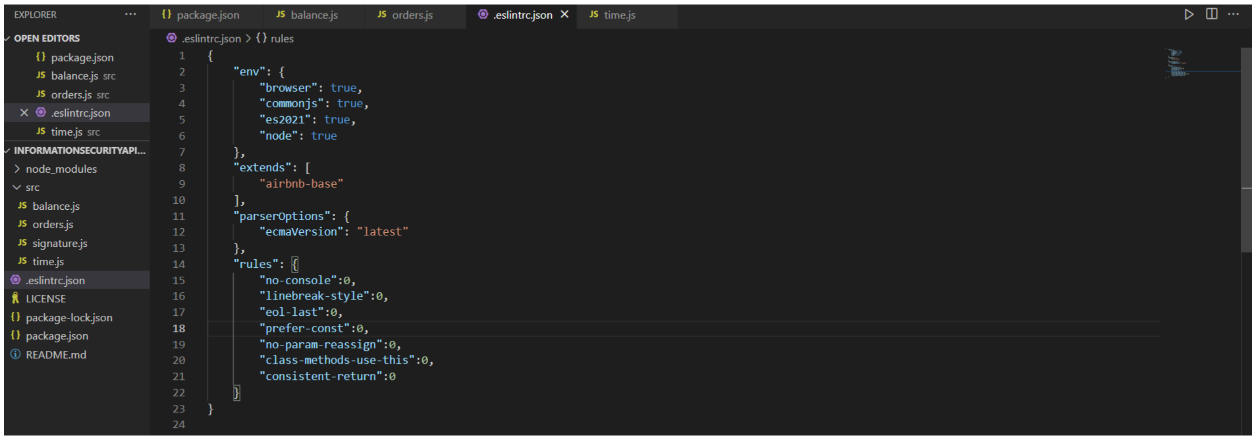
Releases

Second, we will be using Node Package Manager to create our project. Using the command "npm init" on our terminal will initialize the start and installation of Node Package Manager into our project. After that we put all of our information about the project and do not forget to state that this project will be run on the node. After providing this information, Node Package Manager will create a "package.json" that signifies that our project has been successfully integrated with Node Package Manager.



```json
    },
    "repository": {
        "type": "git",
        "url": "git+https://github.com/BryanYehuda/InformationSecurityAPITest.git"
    },
    "author": "Bryan Yehuda Mannuel",
    "license": "MIT",
    "bugs": {
        "url": "https://github.com/BryanYehuda/InformationSecurityAPITest/issues"
    },
    "homepage": "https://github.com/BryanYehuda/InformationSecurityAPITest#readme",
    "devDependencies": {
        "eslint": "^8.14.0",
        "eslint-config-airbnb-base": "^15.0.0",
        "eslint-plugin-import": "^2.26.0"
    },
    "dependencies": {
        "crypto": "^1.0.1",
        "crypto-js": "^4.1.1",
        "https": "^1.0.0",
        "moment": "^2.29.3",
        "node-fetch": "^3.2.4",
        "querystring": "^0.2.1"
```

Lastly, since we are using Javascript that is notoriously known for not having any regular formatting standard, we will be using and installing ESlint as a linter to format all of our codes and warn us if there are any errors or formatting issues. To install it just type “npm install eslint” and NPM will install it for you. Then we will initialize ESlint using “npm init @eslint/config” and provide all the required information. After finishing it ESlint will create a “.eslintrc.json” as a configuration settings for this project. And after this, we are all set up to continue with our project.



```json
{
    "env": {
        "browser": true,
        "commonjs": true,
        "es2021": true,
        "node": true
    },
    "extends": [
        "airbnb-base"
    ],
    "parserOptions": {
        "ecmaVersion": "latest"
    },
    "rules": {
        "no-console":0,
        "linebreak-style":0,
        "eol-last":0,
        "prefer-const":0,
        "no-param-reassign":0,
        "class-methods-use-this":0,
        "consistent-return":0
    }
}
```

The Code

There are 2 main parts in the code section of this project. The first one is called “balance.js” which will be used to get our wallet balance in part A and the second one is called “orders.js” which will be used to get our open orders in the market.

```javascript
// balance.js
import { createHmac } from 'crypto';
import { Agent, request as _request } from 'https';
import { stringify } from 'querystring';

const USER_AGENT = 'BRYAN';
const DEFAULT_ENDPOINT = 'ftx.com';
const DEFAULT_HEADER_PREFIX = 'FTX';

// We are creating a Balance class to accommodate our Request to the FTX's Server
class Balance {
  constructor(setting) {
    this.ua = USER_AGENT;
    this.timeout = 90 * 1000;
    this.agent = new Agent({
      keepAlive: true,
      timeout: 90 * 1000,
      keepAliveMsecs: 1000 * 60,
    });

    if (!setting) {
      return;
    }

    if (setting.key && setting.secret) {
      this.key = setting.key;
      this.secret = setting.secret;
    }

    if (setting.timeout) {
      this.timeout = setting.timeout;
    }

    if (setting.userAgent) {
      this.ua += ` | ${setting.userAgent}`;
    }

    this.endpoint = setting.endpoint || DEFAULT_ENDPOINT;
    this.headerPrefix = setting.headerPrefix || DEFAULT_HEADER_PREFIX;
```

```
}

// We are creating a draft so that we save time when we are sending our
// request because we already have a template of what to be sent
createDraft({
  path,
  method,
  data,
  timeout,
}) {
  if (!timeout) {
    timeout = this.timeout;
  }

  path = `/api${path}`;
  let paydata = '';
  if (method === 'GET' && data) {
    path += `?${stringify(data)}`;
  } else if (data) {
    paydata = JSON.stringify(data);
  }

  const start = +new Date();

  const signature = createHmac('sha256', this.secret)
    .update(start + method + path + paydata).digest('hex');

  const options = {
    host: this.endpoint,
    path,
    method,
    agent: this.agent,
    headers: {
      'User-Agent': this.ua,
      'content-type': 'application/json',
      Accept: 'application/json',
      'X-Requested-With': 'XMLHttpRequest',
      [`${this.headerPrefix}-TS`]: start,
      [`${this.headerPrefix}-KEY`]: this.key,
      [`${this.headerPrefix}-SIGN`]: signature,
    },
    timeout,
    paydata,
  };

  return options;
}

// We are sending the request to FTX's server using the specified
// API Key, Method, and path
requestDraft(draft) {
  return new Promise((resolve, reject) => {
    const req = _request(draft, (res) => {
      res.setEncoding('utf8');
      let buffer = '';
      res.on('data', (data) => {
        buffer += data;
      });
      res.on('end', () => {
        if (res.statusCode >= 300) {
          let message;
          let data;

          try {
            data = JSON.parse(buffer);
            message = data;
          } catch (e) {
            message = buffer;
          }

          console.error('ERROR!', res.statusCode, message);
          const error = new Error(message.error);
          error.statusCode = res.statusCode;
          return reject(error);
        }

        let data;
        try {
```

```javascript
            data = JSON.parse(buffer);
          } catch (err) {
            console.error('JSON ERROR!', buffer);
            return reject(new Error('Json error'));
          }

          resolve(data);
        });
      });

      req.on('error', (err) => {
        reject(err);
      });

      req.on('socket', (socket) => {
        if (socket.connecting) {
          socket.setNoDelay(true);
          socket.setTimeout(draft.timeout);
          socket.on('timeout', () => {
            req.abort();
          });
        }
      });

      req.end(draft.paydata);
    });
  }

  request(props) {
    return this.requestDraft(this.createDraft(props));
  }
}

// The Api Key needed to make a Request to FTX's Server
const ftx = new Balance({
  key: '_njFKoPZrOV_9Tme2-m5vQGP8eCL3nKoJ2GTuuFU',
  secret: '9nHenFZftNXI5vPVcEW62lZtMpqGbhyIExQOaMTU',
});
```

```
// The path and method needed to make a Request to FTX's Server
ftx.request({
  method: 'GET',
  path: '/wallet/balances',
}).then(console.log); // Then we output the results
```

```
// orders.js
import { createHmac } from 'crypto';
import { Agent, request as _request } from 'https';
import { stringify } from 'querystring';

const USER_AGENT = 'BRYAN';
const DEFAULT_ENDPOINT = 'ftx.com';
const DEFAULT_HEADER_PREFIX = 'FTX';

// We are creating a Balance class to accommodate our Request to the FTX's Server
class Orders {
  constructor(setting) {
    this.ua = USER_AGENT;
    this.timeout = 90 * 1000;

    this.agent = new Agent({
      keepAlive: true,
      timeout: 90 * 1000,
      keepAliveMsecs: 1000 * 60,
    });

    if (!setting) {
      return;
    }

    if (setting.key && setting.secret) {
      this.key = setting.key;
      this.secret = setting.secret;
    }

    if (setting.timeout) {
      this.timeout = setting.timeout;
    }
    if (setting.userAgent) {
      this.ua += ` | ${setting.userAgent}`;
    }

    this.endpoint = setting.endpoint || DEFAULT_ENDPOINT;
    this.headerPrefix = setting.headerPrefix || DEFAULT_HEADER_PREFIX;
  }

  // We are creating a draft so that we save time when we are sending our
  // request because we already have a template of what to be sent
  createDraft({
    path,
    method,
    data,
    timeout,
  }) {
    if (!timeout) {
      timeout = this.timeout;
    }

    path = `/api${path}`;
    let paydata = '';
    if (method === 'GET' && data) {
      path += `?${stringify(data)}`;
    } else if (data) {
      paydata = JSON.stringify(data);
    }

    const start = +new Date();

    const signature = createHmac('sha256', this.secret)
      .update(start + method + path + paydata).digest('hex');

    const options = {
      host: this.endpoint,
      path,
      method,
      agent: this.agent,
      headers: {
        'User-Agent': this.ua,
        'content-type': 'application/json',
        Accept: 'application/json',
        'X-Requested-With': 'XMLHttpRequest',
        [`${this.headerPrefix}-TS`]: start,
        [`${this.headerPrefix}-KEY`]: this.key,
        [`${this.headerPrefix}-SIGN`]: signature,
      },
      timeout,
      paydata,
    };

    return options;
  }

  // We are sending the request to FTX's server using the specified
  // API Key, Method, and path
  requestDraft(draft) {
    return new Promise((resolve, reject) => {
      const req = _request(draft, (res) => {
        res.setEncoding('utf8');
        let buffer = '';
        res.on('data', (data) => {
          buffer += data;
        });
        res.on('end', () => {
          if (res.statusCode >= 300) {
            let message;
            let data;

            try {
              data = JSON.parse(buffer);
              message = data;
            } catch (e) {
              message = buffer;
            }

            console.error('ERROR!', res.statusCode, message);
```

```
            const error = new Error(message.error);
            error.statusCode = res.statusCode;
            return reject(error);
          }

          let data;
          try {
            data = JSON.parse(buffer);
          } catch (err) {
            console.error('JSON ERROR!', buffer);
            return reject(new Error('Json error'));
          }

          resolve(data);
        });
      });

      req.on('error', (err) => {
        reject(err);
      });

      req.on('socket', (socket) => {
        if (socket.connecting) {
          socket.setNoDelay(true);
          socket.setTimeout(draft.timeout);
          socket.on('timeout', () => {
            req.abort();
          });
        }
      });

      req.end(draft.paydata);
    });
  }

  request(props) {
    return this.requestDraft(this.createDraft(props));
  }
}
```

```
// The Api Key needed to make a Request to FTX's Server
const ftx = new Orders({
  key: '_njFKoPZrOV_9Tme2-m5vQGP8eCL3nKoJ2GTuuFU',
  secret: '9nHenFZftNXI5vPVcEW62lZtMpqGbhyIExQOaMTU',
});

// The path and method needed to make a Request to FTX's Server
ftx.request({
  method: 'GET',
  path: '/orders?market=BTC-PERP',
}).then(console.log); // Then we output the results
```

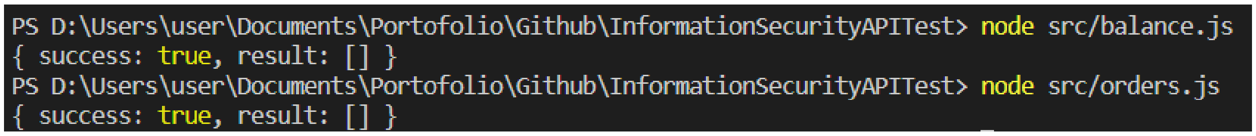
There are 5 sections in both code that we need to look at. The first one is the Balance class for balance.js and the Orders class for orders.js. This class is needed so that we can accommodate our requests to FTX's server inside a class. This class contains the user agent which is me that makes this request to the server, the timeout which calculates how much time is needed for a request to be flagged as a failed one, the secret key and public key, the endpoint, and the header to be sent to FTX.

The second one is the createDraft method. This method is needed to save our time when we are making a request to FTX's server. This draft acts as a template for our request so that the next time we make requests we do not need to iterate in this step anymore. We can just use this draft and fill it with the information needed and send it right away to FTX's server.

The third one is the requestDraft method. This method fills all the required information on the createDraft method's draft with our information. When all of the required information is completed, it will send the draft as a request to FTX's server. Then it will fetch the responses given by FTX and resolve those responses to be shown on the terminal.

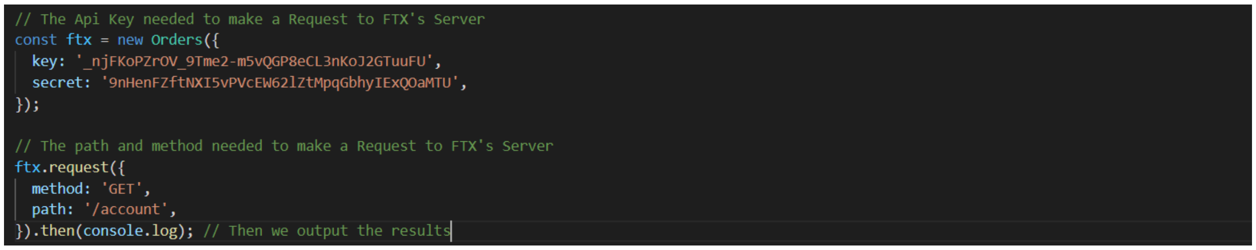
The fourth one, is the new Balance or new Orders constant, where we put our Public API Key and Secret API Key as a constant to be used when filling required information. The last one is the request method, which triggers the requestDraft method by telling them where to send the request and what method will be used to send it. Then it will show the resolved responses sent by FTX into the terminal.

The Results



```
PS D:\Users\user\Documents\Portofolio\Github\InformationSecurityAPITest> node src/balance.js
{ success: true, result: [] }
PS D:\Users\user\Documents\Portofolio\Github\InformationSecurityAPITest> node src/orders.js
{ success: true, result: [] }
```

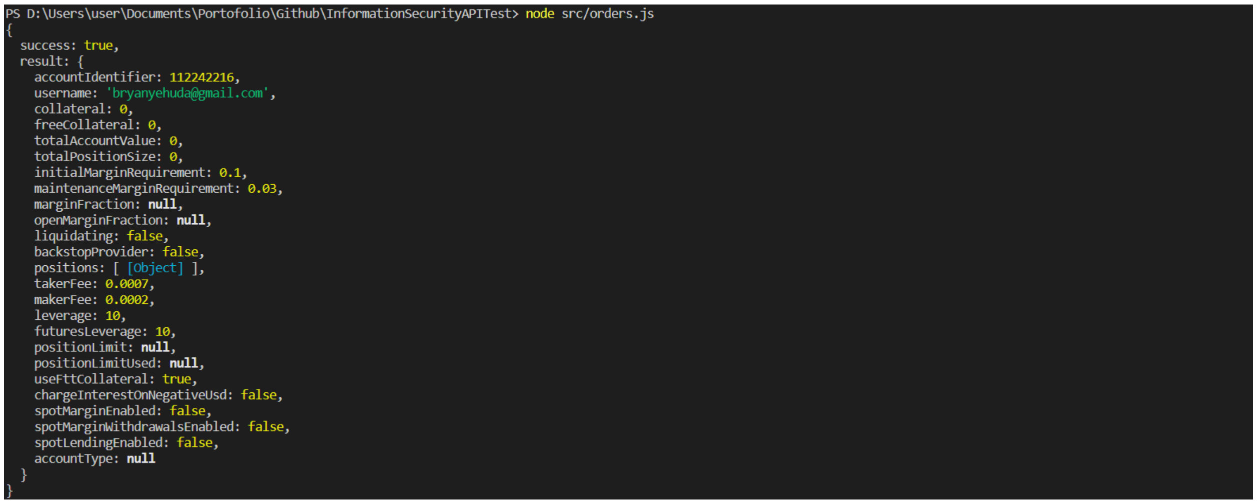
These are the results when both of the code are run on the terminal. When we run balance.js it shows that the request is successful but it does not show any balance since our balance is still zero. The same results are true for orders.js since we also did not have any open orders. Still curious, I wanted to try to get results on the terminal by requesting something that we already have, which is our own account. To get this we can sent a request to /account according to https://docs.ftx.com/#account.



```javascript
// The Api Key needed to make a Request to FTX's Server
const ftx = new Orders({
  key: '_njFKoPZrOV_9Tme2-m5vQGP8eCL3nKoJ2GTuuFU',
  secret: '9nHenFZftNXI5vPVcEW62lZtMpqGbhyIExQOaMTU',
});

// The path and method needed to make a Request to FTX's Server
ftx.request({
  method: 'GET',
  path: '/account',
}).then(console.log); // Then we output the results
```

And by sending a request to that endpoint, we get a response back since our own account is not empty. Below is the response that we got by sending those requests. This means our code is running successfully and that indicates that we can get any kind of responses from FTX by using this code. That is the end of this report and thank you for your time and consideration.



```
PS D:\Users\user\Documents\Portofolio\Github\InformationSecurityAPITest> node src/orders.js
{
  success: true,
  result: {
    accountIdentifier: 112242216,
    username: 'bryanyehuda@gmail.com',
    collateral: 0,
    freeCollateral: 0,
    totalAccountValue: 0,
    totalPositionSize: 0,
    initialMarginRequirement: 0.1,
    maintenanceMarginRequirement: 0.03,
    marginFraction: null,
    openMarginFraction: null,
    liquidating: false,
    backstopProvider: false,
    positions: [ [Object] ],
    takerFee: 0.0007,
    makerFee: 0.0002,
    leverage: 10,
    futuresLeverage: 10,
    positionLimit: null,
    positionLimitUsed: null,
    useFttCollateral: true,
    chargeInterestOnNegativeUsd: false,
    spotMarginEnabled: false,
    spotMarginWithdrawalsEnabled: false,
    spotLendingEnabled: false,
    accountType: null
  }
}
```