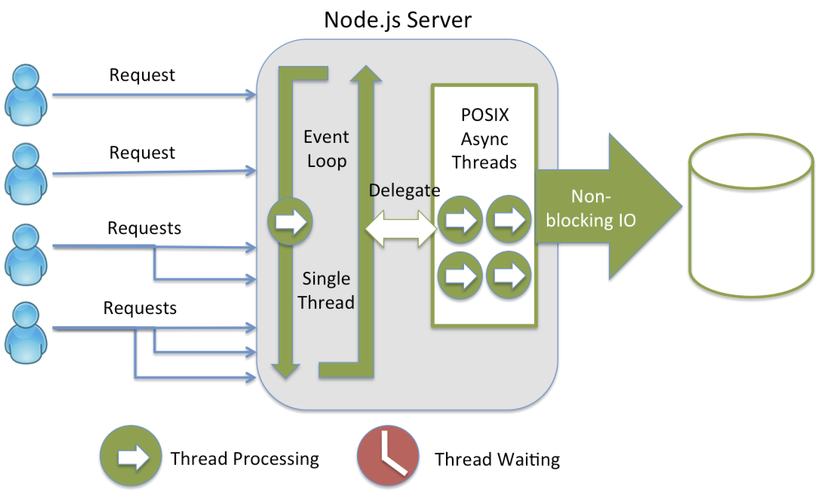
**1. NodeJS**

NodeJS là một platform cho phía server được xây dựng dựa trên Google Engine V8, với khả năng giúp lập trình viên implement phần backend của một hệ thống web application bằng Javascript.



**1.1 Những ứng dụng có thể viết bằng nodeJS:**

Websocket Servers(chat, game)

Ad servers

Cloud servers

Microservers

Restful API

Real-time applications

**1.2 Tại sao cần viết ứng dụng bằng nodeJS**

* Javascript: NodeJS viết dựa trên ngôn ngữ javascript(ngôn ngữ dễ học, sử dụng phổ biến)
* Nhanh, nhẹ: NodeJS chạy đa nền tảng phía server, sử dụng kiến trúc Event-Driven, cơ chế non-blocking IO nhẹ, hiệu quả.
* Single language: Có thể sử dụng nodeJS cho tất cả các tầng của 1 ứng dụng;
* High performance.
* Hosted anywhere.
* Dễ modify và maintain

Ngoài ra:

* Cộng đồng lớn và hoàn toàn miễn phí
* Các gói npm hỗ trợ lên tới 200.000+
* Chạy các ứng dụng real-time đa nền tảng, đa thiết bị.
* Những công ty công nghệ lớn đang sử dụng nodeJS:



**1.3 Một máy chủ javascript cơ bản cần làm được những gì?**

* Tổ chức mã nguồn thành các thành phần có thể tái sử dụng(clean code)
* Khả năng xử lý file
* Kết nối và truy cập database
* Giao tiếp và truy cập internet
* Tiếp nhận các request, trả về response (ví dụ: HTTP, FTP, SSH, AMQP)

**1.4 Cài đặt Node**

Sử dụng Homebrew

*brew install node*

*Ref:* [*https://changelog.com/posts/install-node-js-with-homebrew-on-os-x*](https://changelog.com/posts/install-node-js-with-homebrew-on-os-x)

Tool để lập trình: <https://code.visualstudio.com/>

Mã nguồn: <https://github.com/Nodejs>

**2. Các khái niệm nền tảng cơ bản của NodeJS**

**2.1 Function**

function name() {}

**2.1 Object**

* Là tập hợp các cặp Name/Value
* Là cách định nghĩa đơn giản trong javascript
* Các kiểu giá trị value có thể mang:
* Primitive: String, Number
* Object
* Function

**2.3 Object Literal**

* Là cách khai báo nhanh đối tượng trong javascript
* Gồm các cặp name/value được khai báo cách nhau dấu phẩy

var Person = {

firstName: "phuong",

lastname: "tran",

printFullName: function () {

console.log(this.firstName + this.lastname)

}

}

**2.4 Init**

* Là hàm sử dụng khởi tạo đối tượng
* Sử dụng this để truy cập các biến của đối tượng hoặc khởi tạo mới

function Persion (name, password) {

this.name = name

this.password = password

}

**2.5 Prototype**

* Prototype là khái niệm cốt lõi cơ bản trong JavaScript khá quan trọng để thực thi OOP trong JavaScript. Vì JavaScript là một prototype-based language, không có khái niệm class như các ngôn ngữ hướng đối tượng khác hướng đối tượng khác. Trong bài viết này, ta sẽ tìm hiểu cách làm việc của prototype có thể hiểu nôm na là kế thừa (inheritance) ở javascript.
* Ref: <https://topdev.vn/blog/javascript-prototype-la-gi/>

function Persion (name, password) {

this.name = name

this.password = password

}

Persion.prototype.getName = function() {

return this.name

}

Persion.prototype.getPassword = function() {

return this.password

}

Persion.prototype.getLevel = function() {

return this.level

}

Persion.prototype.level = "admin"

function User(name) {

this.name = name

}

User.prototype = new Persion()

const persion = new Persion("ti", "12233")

const user = new User("nam", "46474")

console.log(user.getName())

**2.6 Biến tham trị và biến tham chiếu**

# Truyền tham trị: Hàm được gọi bằng cách truyền trực tiếp giá trị của biến làm đối số. Thay đổi đối số bên trong hàm không ảnh hưởng đến biến được truyền từ bên ngoài hàm. Javascript luôn luôn truyền tham trị và không bao giờ thay đổi giá trị với kiểu dữ liệu nguyên thủy (String, number hoặc boolen)

* Truyền tham chiếu: Hàm được gọi bằng cách chuyển trực tiếp tham chiếu / địa chỉ của biến làm đối số. Thay đổi đối số bên trong hàm ảnh hưởng đến biến được truyền từ bên ngoài hàm. Trong các đối tượng Javascript và mảng theo sau thông qua tham chiếu.
* Ref: <https://viblo.asia/p/javascript-truyen-tham-tri-va-truyen-tham-chieu-gAm5yoAqZdb>

**2.7 Module**

* Khi một script lớn hoặc một script được tái sử dụng thì có thể tập hợp thành 1 module.
* Tồn tại độc lập, không làm ảnh hưởng hay xung đột các mã nguồn khác
* Tồn tại trong nodejs thông qua common.js

**2.8 Common.JS**

* Là nguyên tắc thỏa thuận để cấu trúc các module
* Khi tạo ra module thì dùng module.exports
* Khi gọi module thì dùng require

Ex:

var express = require('express');

var router = express.Router();

/\* GET home page. \*/

router.get('/', function(req, res, next) {

res.render('home', { todoNumber: "5" });

});

module.exports = router;

**2.9 Một số module sẵn có trong NodeJS**

Tham khảo tại <https://nodejs.org/dist/latest-v14.x/docs/api/>

Ví dụ sử dụng module util:

const util = require('util');

const debuglog = util.debuglog('foo');

debuglog('hello from foo [%d]', 123);

**2.10 Event**

* Là việc xảy ra trong ứng dụng mà khi đó chúng ta có thể lắng nghe, đáp ứng lại
* Có 2 loại chính
* System events (C++ core)
* Custom events (Javascript core)

**2.11 Event Emitter**

* Được hiểu như notification trong swift, khởi tạo 1 lắng nghe và 1 thông báo.
* Khi đối tượng EventEmitter phát ra một sự kiện, tất cả các hàm được gắn vào sự kiện cụ thể đó được gọi một cách đồng bộ.

var events = require('events');

var eventEmitter = new events.EventEmitter();

eventEmitter.on('clicked', function() {

console.log('Something is clicked!');

})

eventEmitter.addListener('clicked', function() {

console.log('Something is clicked!');

})

Ref: <https://viblo.asia/p/tim-hieu-event-va-eventemitter-trong-nodejs-bWrZnx2p5xw>

**2.12 Call và Apply**

Tham khảo tại <https://codelearn.io/sharing/bind-call-va-apply-trong-javascript>

**2.13 Class**

* Tương tự như các ngôn ngữ khác nhưng khi sử dụng cần khai báo `use strict`

`use strict`

class test {

taskCompleted(task) {

console.log("ok");

}

ex() {

var http = new chilkat.Http();

var success;

success = http.UnlockComponent("Anything for 30-day trial");

var task = http.QuickGetAsync("http://www.wikipedia.org/");

task.Run(this.taskCompleted);

}

}

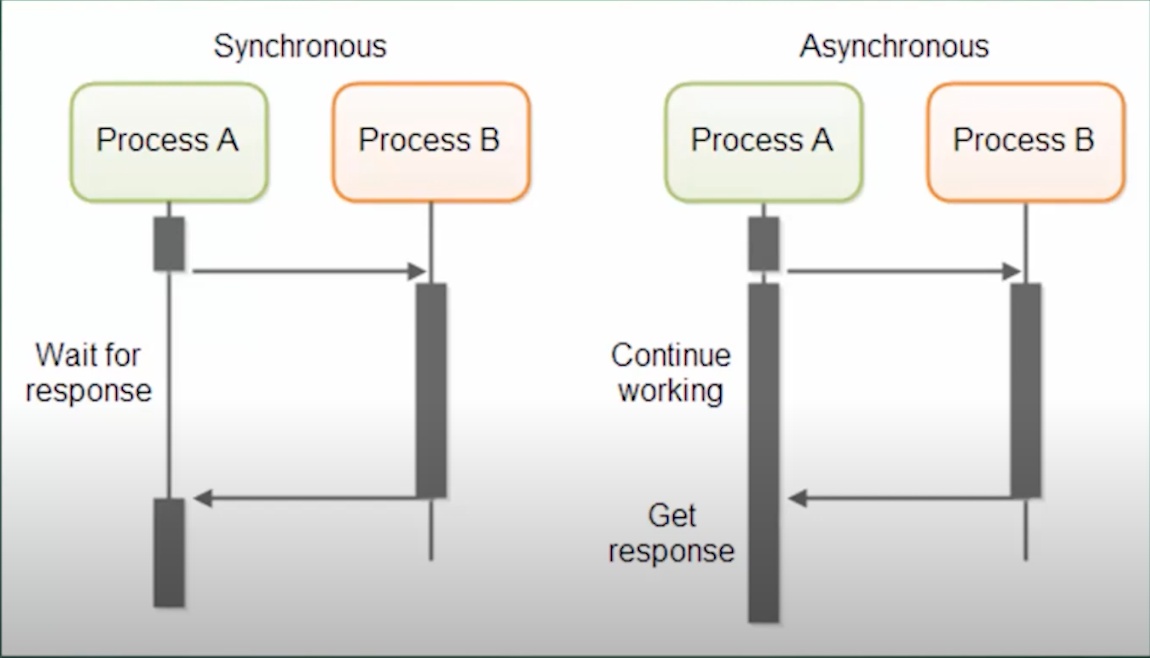
var ctest = new test();

ctest.ex();

**2.14 Thực thi code không đồng bộ**

Javascript: Thực thi code đồng bộ tức là chỉ có một dòng lênh được thực thi vào một thời điểm

Nodejs: làm nhiều việc khác nữa, bao gồm thực thi javascript trong V8 Engine



**2.15 Callback**

* Một function được truyền vào lời gọi của hàm khác
* Thực thi vào một thời điểm trong tương lai sau khi hoàn tất công việc nào đó.

var firstFunction = function() {

console.log("FirstFunction")

}

var secondFunction = function() {

setTimeout(firstFunction, 5000)

console.log("SecondFunction")

}

secondFunction();

function demoCallback(callback) {

callback("data");

}

demoCallback( function(data){

console.log(data)

})

**Demo Hello word!**

var http = require("http");

http.createServer(function (req, res) {

res.writeHead("200", {"Content-Type": "text/plain"});

res.end("Hello word!")

}).listen(1023, "172.0.0.1")

**3. NPM và NPMJS**

NPM là gì? NMP là viết tắt của Node package manager là một công cụ tạo và quản lý các thư viện lập trình Javascript cho [Node.js](https://nodejs.org/). Trong cộng đồng Javascript, các lập trình viên chia sẻ hàng trăm nghìn các thư viện với các đoạn code đã thực hiện sẵn một chức năng nào đó. Nó giúp cho các dự án mới tránh phải viết lại các thành phần cơ bản, các thư viện lập trình hay thậm chí cả các [framework](https://topdev.vn/blog/framework-la-gi/).

**NPM** là một phần mềm cài đặt trên máy tính của bạn nên bạn có thể sử dụng nó để cài đặt các thư viện **Javascript** từ trên **Internet**. Để cài đặt một thư viện nào đó, chỉ cần mở cửa sổ **Terminal (hoặc CMD)** và thực thi lệnh giống dưới đây:

npm install package-name

Ref: <https://www.npmjs.com>

**4. Cassandra**

Cài đặt cassandra qua Homebrew

Cài đặt Homebrew

ruby -e "$(curl -fsSL https://raw.github.com/Homebrew/homebrew/go/install)"

Cài đặt python

brew install python

Cài đặt sql

pip install cql

Cài đặt cassandra

brew install cassandra

Ref: <https://gist.github.com/hkhamm/a9a2b45dd749e5d3b3ae>

Tiến hành truy cập cassandra

Cài đặt cassandra-driver

npm install cassandra-driver

Ref: <https://www.npmjs.com/package/cassandra-driver>

Kết nối database

const cassandra = require('cassandra-driver');

const client = new cassandra.Client({

contactPoints:['127.0.0.1:9042'],

localDataCenter: 'datacenter1',

credentials: { username: 'tphuong', password: 'tphuong' }

});

client.execute('SELECT \* FROM tphuongdb.tbl\_sings', function(err, data) {

if (err) {

// handle error

console.log('handle error.', err)

} else {

console.log('handle success.', data)

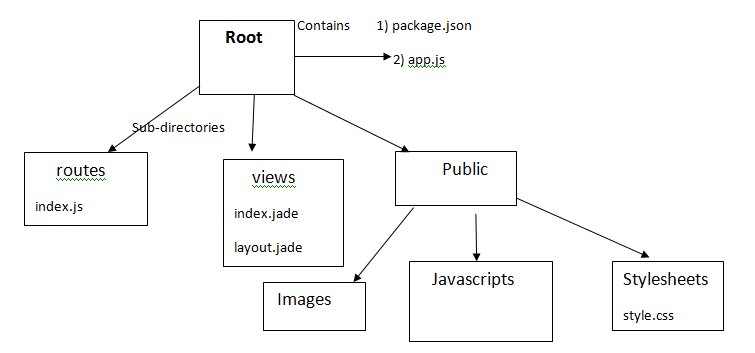
// handle success.

}

});

**5. ExpressJS**

**Expressjs** là một framework được xây dựng trên nền tảng của **Nodejs**. Nó cung cấp các tính năng mạnh mẽ để phát triển web hoặc mobile. **Expressjs** hỗ trợ các method HTTP và midleware tạo ra API vô cùng mạnh mẽ và dễ sử dụng.



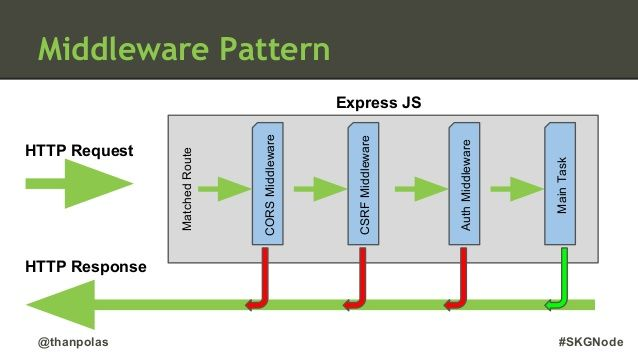
Install

npm install express --save

Ref: <https://topdev.vn/blog/express-js-la-gi/>

**6. Middleware**

**Middleware** là những đoạn mã trung gian nằm giữa các request và response. Nó nhận các request, thi hành các mệnh lệnh tương ứng trên request đó. Sau khi hoàn thành nó response (trả về) hoặc chuyển kết quả ủy thác cho một **Middleware** khác trong hàng đợi.



Hình trên mô tả 3 middleware có trong ExpressJS. Một request khi gửi đến Express sẽ được xử lý qua 5 bước như sau :

1. Tìm Route tương ứng với request
2. Dùng CORS Middleware để kiểm tra cross-origin Resource sharing của request
3. Dùng CRSF Middleware để xác thực CSRF của request, chống fake request
4. Dùng Auth Middleware để xác thực request có được truy cập hay không
5. Xử lý công việc được yêu cầu bởi request (Main Task)

Trong Express, có 5 kiểu middleware có thể sử dụng :

* Application-level middleware (middleware cấp ứng dụng)
* Router-level middleware (middlware cấp điều hướng - router)
* Error-handling middleware (middleware xử lý lỗi)
* Built-in middleware (middleware sẵn có)
* Third-party middleware (middleware của bên thứ ba)

Ref: <https://viblo.asia/p/tim-hieu-ve-middleware-trong-expressjs-gVQelwaaGZJ>

**7. Demo**

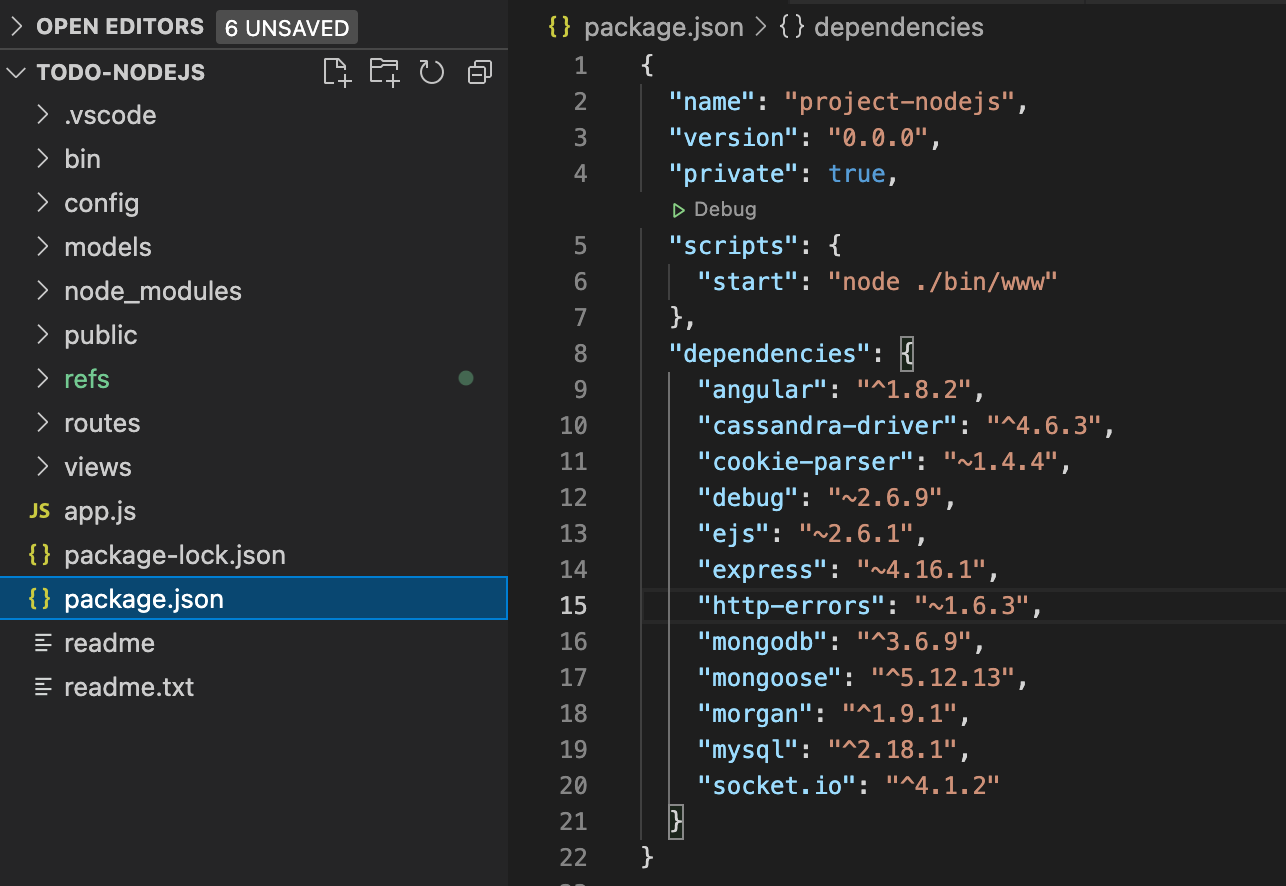
Cài đặt express và express ejs

npx express-generator

express --view=ejs myapp

Ref: <https://expressjs.com/en/starter/generator.html>

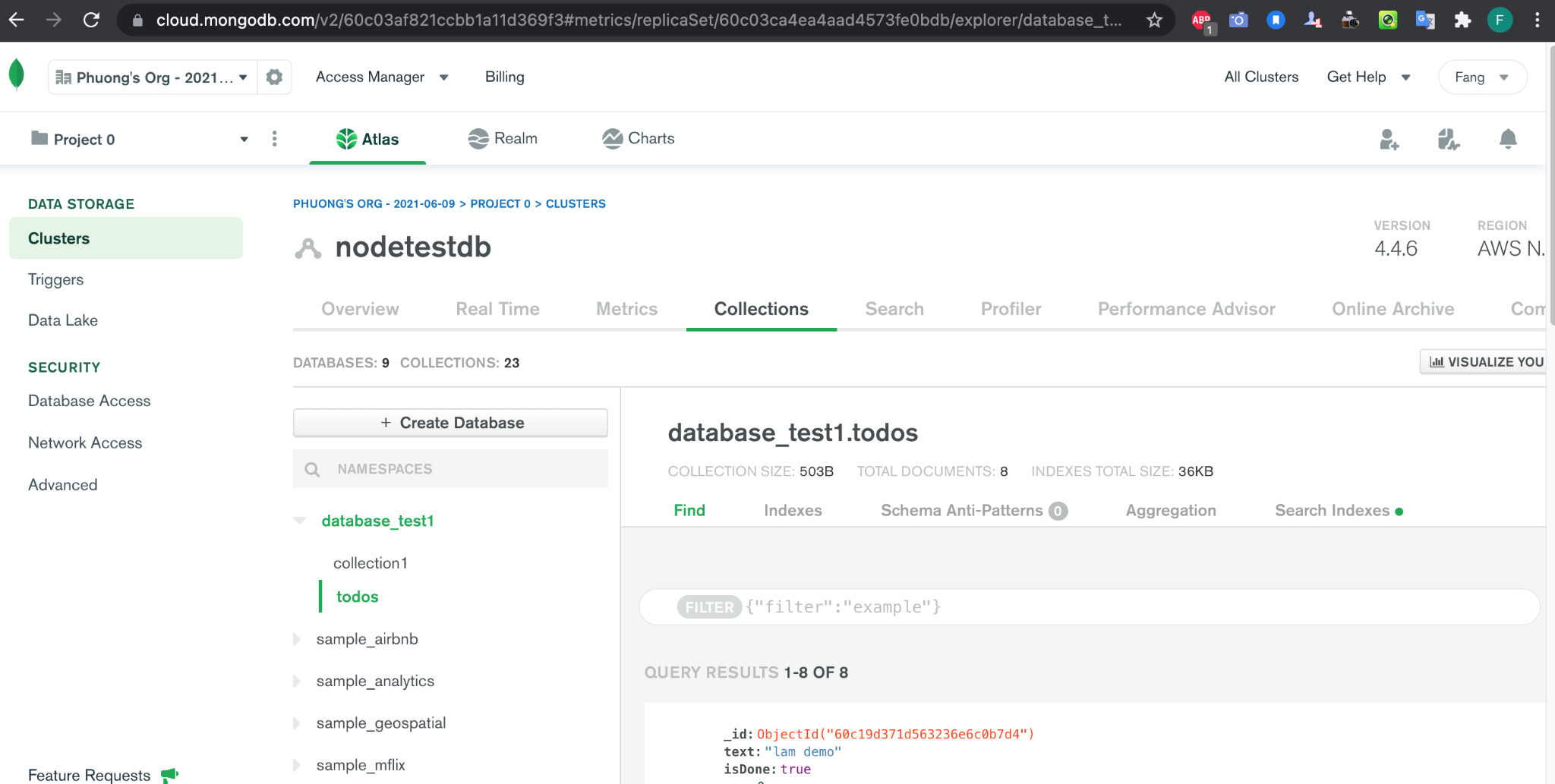
Cài đặt thêm các thư viện như: *mongoose, http-error...*



Tiến hành tạo cloud database

Đăng kí tại <https://cloud.mongodb.com/>

Tiến hành đăng kí vài tạo database mẫu.

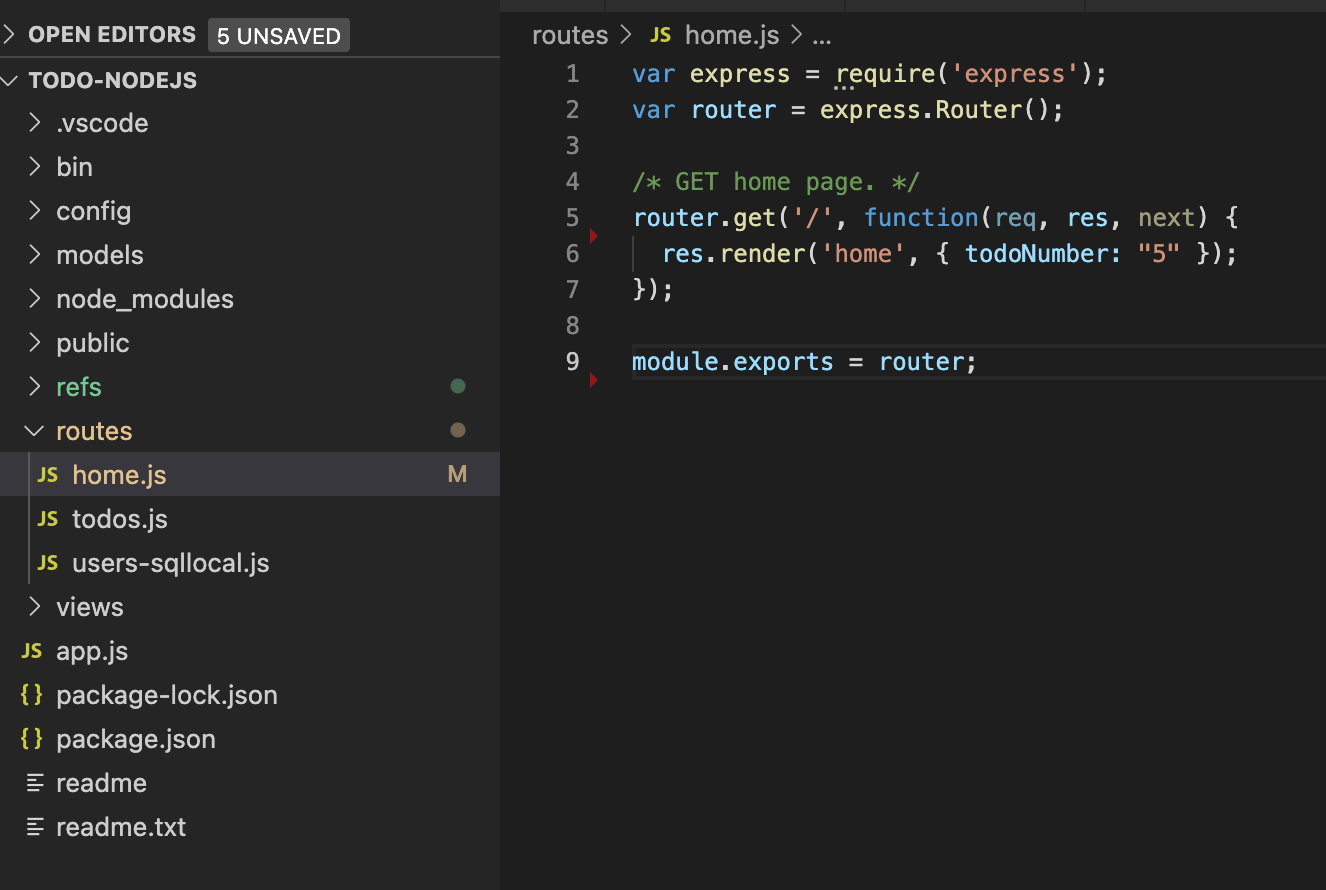


Sử dụng nmp mongoose để kết nối.

Mô hình EJS sẽ gồm những phần chính.

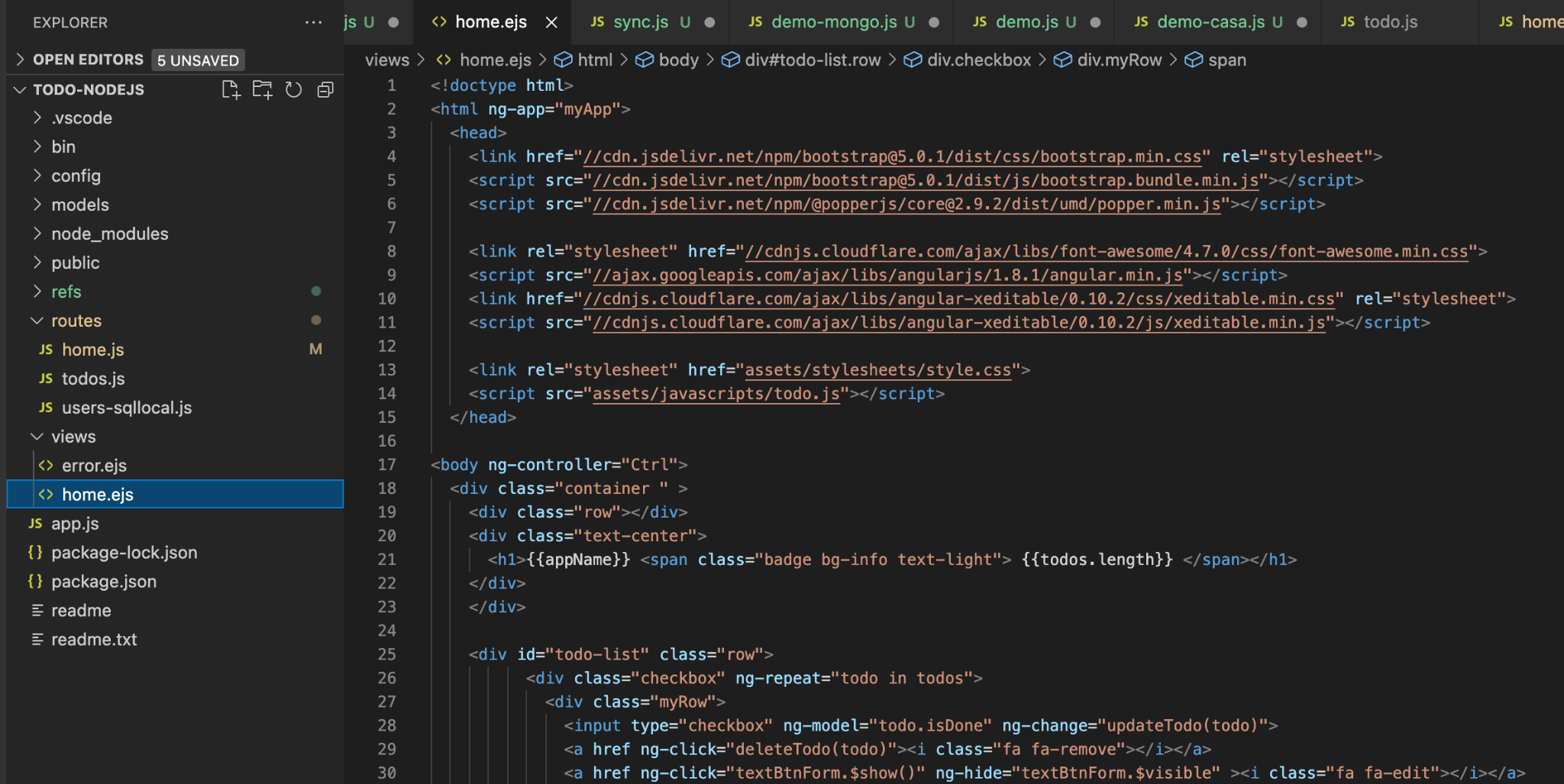
**Router**

Kết nối vào tạo ra nhưng APIs



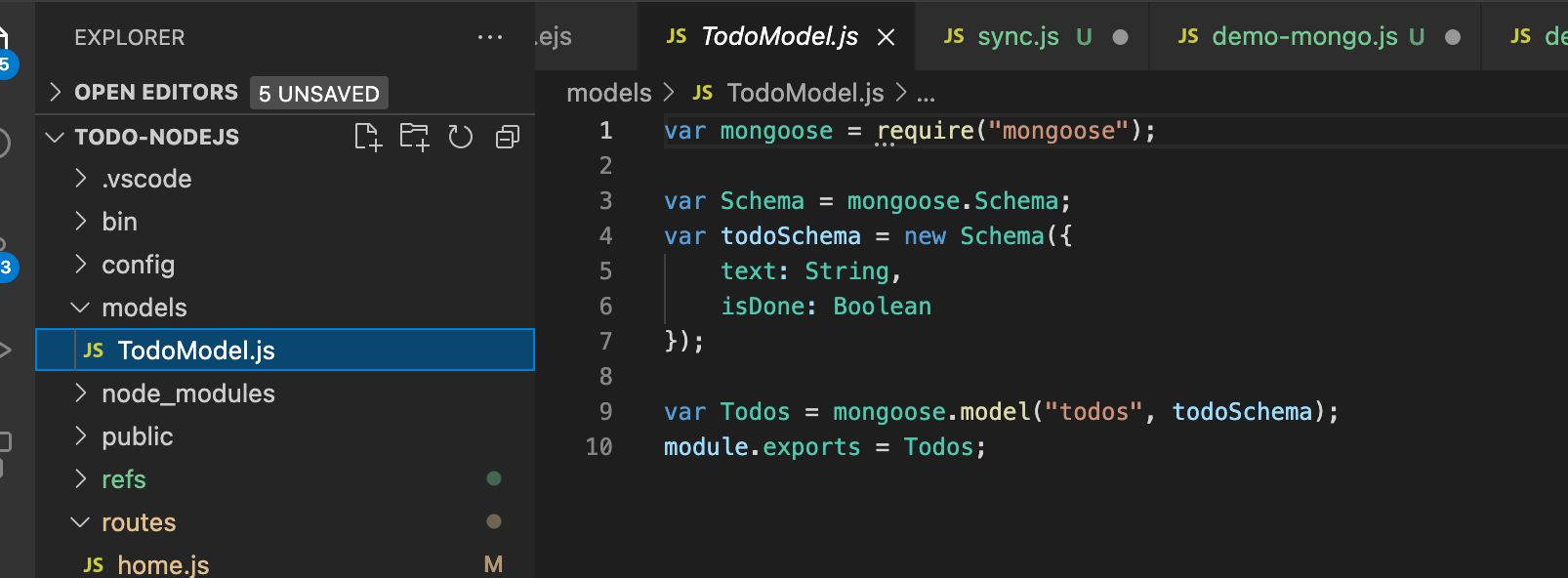
**Views**

Tạo và hiển thị UI front end



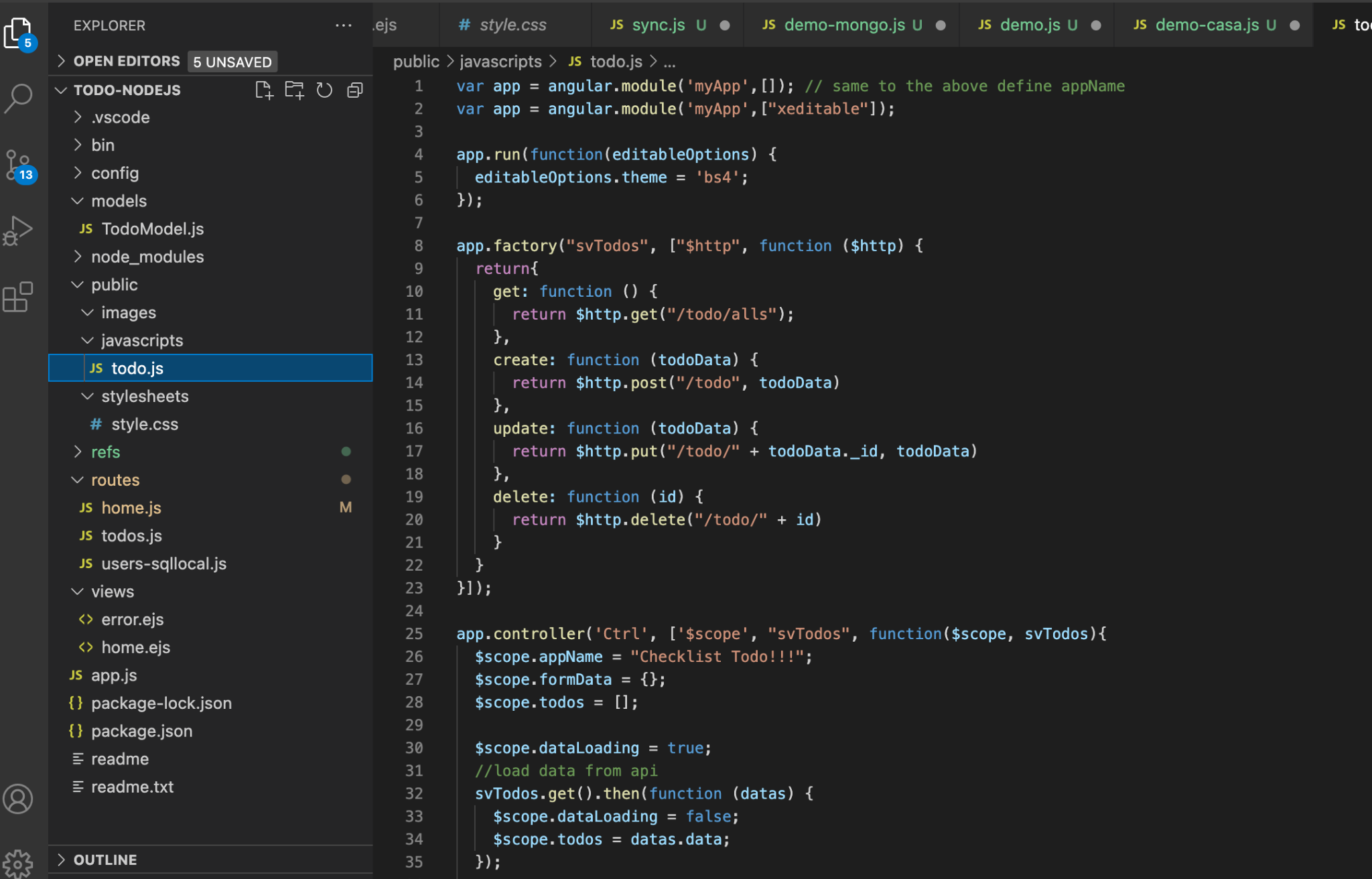
**Models**

Khởi tạo các model kết nối database



**Javascripts**

Binding data từ model sang views



**Source demo**

<https://github.com/trantuanphuong/todo-nodejs>

**Deploy dự án:**

Đăng kí một tài khoản tại <https://dashboard.heroku.com/> và liên kết với github sau đó deploy dự án.

Results:

<https://demochecklistjs.herokuapp.com/>