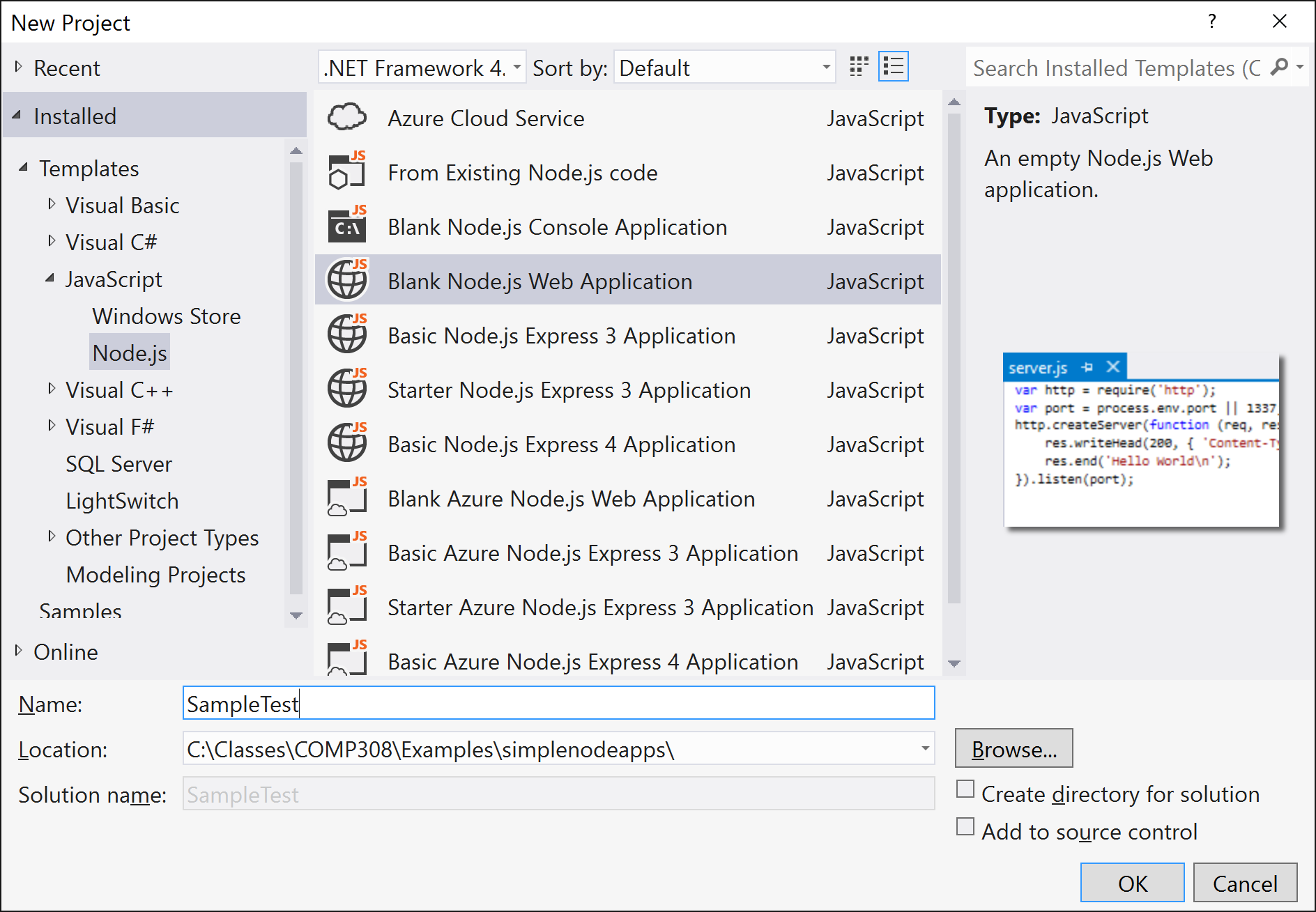
**COMP308 Sample Test**

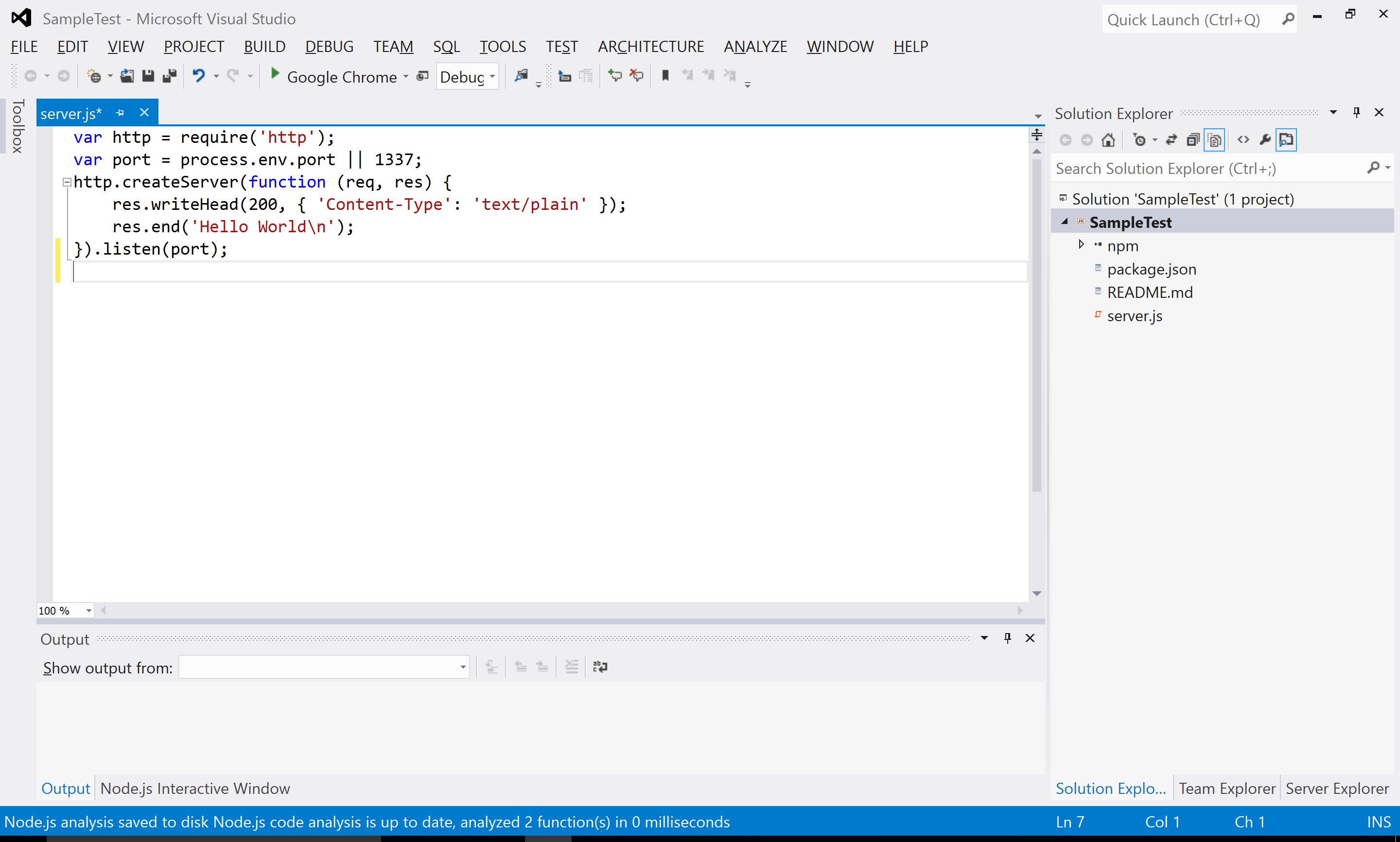
This exercise creates an Express app which allows users to manage tasks in a MongoDB database.

Use Visual Studio 2017 to complete this exercise. Run VS.

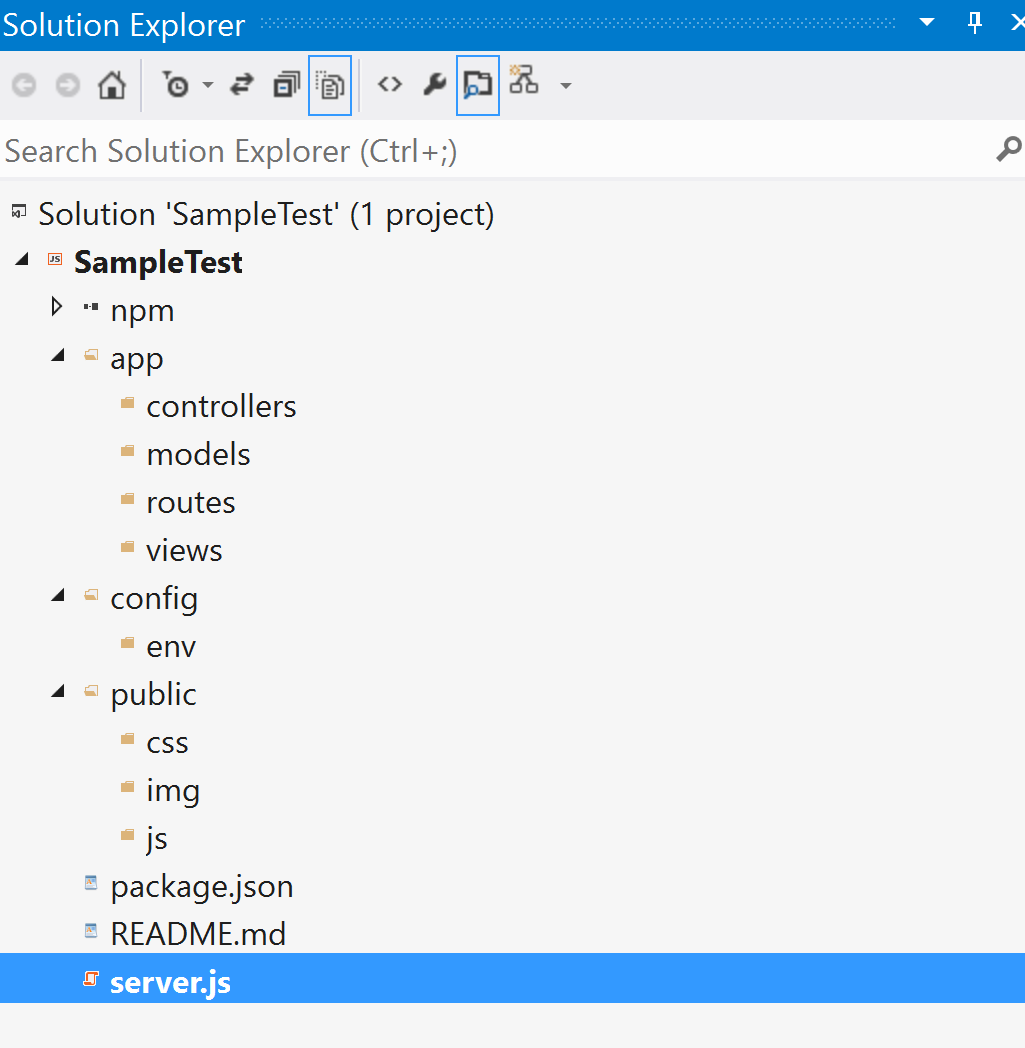
Select New Project.



Select Blank Node.js Webb Application. Name the project. Make sure both checkboxes below browser button are deselected. Click OK. You will get the following:



Create the following application structure:



Open the package.json file. Replace it with the following:

{

"name": "SampleTest",

"version": "0.0.0",

"description": "SampleTest",

"main": "server.js",

"author": {

"name": "your name",

"email": ""

},

"dependencies": {

"express": "^4.8.8",

"morgan": "^1.3.0",

"compression": "^1.0.11",

"body-parser": "^1.8.0",

"method-override": "^2.2.0",

"express-session": "^1.7.6",

"ejs": "^1.0.0",

"mongoose": "~4.3.7"

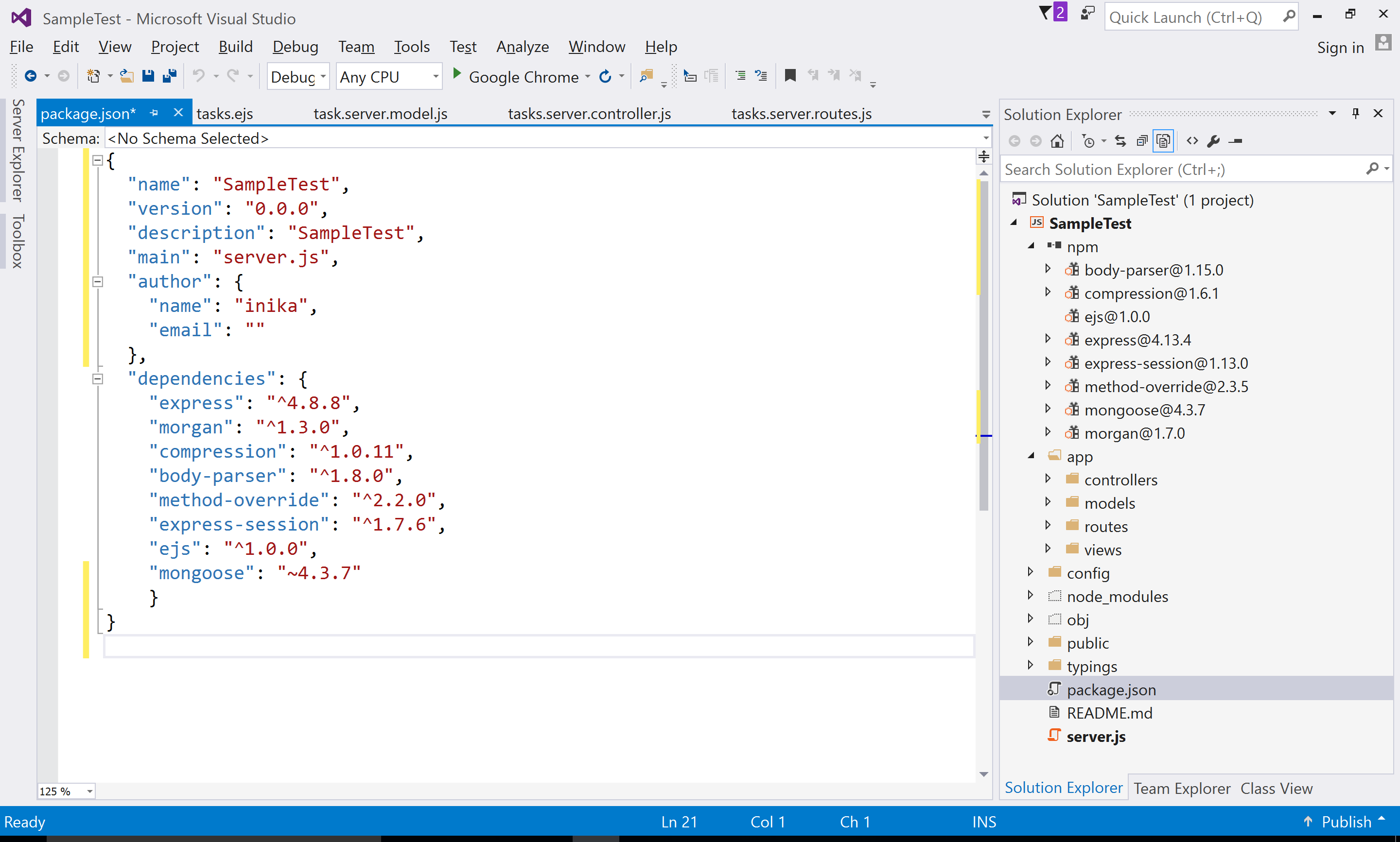
}

}

Install your application dependencies using Node.js interactive window:

**.npm install**

The dependencies should be installed and you will see the change in solution explorer:



Create a development.js file in config/env folder. Add the following code to development.js:

//Development configuration options

// 1 - database name

// 2 - a secret string to sign the session identifier

module.exports = {

db: 'mongodb://localhost/sample-db',

sessionSecret: 'myDevelopmentSecret'

};

Create a config.js file in config folder:

//This file simply loads the correct configuration file

//according to the process.env.NODE\_ENV environment variable

//which is set in server.js (it's value is 'development')

// this code will in fact return ./env/development.js

module.exports = require('./env/' + process.env.NODE\_ENV + '.js');

Create a mongoose.js file in config folder:

// Load the module dependencies:

// config.js module and mongoose module

var config = require('./config'),

mongoose = require('mongoose');

// Define the Mongoose configuration method

module.exports = function () {

// Use Mongoose to connect to MongoDB

var db = mongoose.connect(config.db);

// Return the Mongoose connection instance

return db;

};

In config folder, create an express.js file to configure your express application:

//express.js file is where we configure our Express application

//

// Load the module dependencies

var config = require('./config'),

express = require('express'),

morgan = require('morgan'),

compress = require('compression'),

bodyParser = require('body-parser'),

methodOverride = require('method-override'),

session = require('express-session');

// Create a new Express application instance

module.exports = function () {

//Create the Express application object

var app = express();

//the process.env property allows you to access predefined environment variables

//such as NODE\_ENV

// Use the 'NDOE\_ENV' variable to activate the 'morgan' logger or 'compress' middleware

if (process.env.NODE\_ENV === 'development') {

app.use(morgan('dev'));

} else if (process.env.NODE\_ENV === 'production') {

app.use(compress());

}

// Use the 'body-parser' and 'method-override' middleware functions

app.use(bodyParser.urlencoded({

extended: true

}));

app.use(bodyParser.json()); //use middleware that only parses json

app.use(methodOverride()); // use HTTP verbs such as PUT or DELETE in places where the client doesn't support it.

//saveUninitialized - forces a session that is "uninitialized" to be saved to the store

//resave - forces the session to be saved back to the session store

// Configure the 'session' middleware

app.use(session({

saveUninitialized: true,

resave: true,

secret: config.sessionSecret

}));

//Configure Express to use EJS module as the default template engine

// Set the application view engine and 'views' folder

app.set('views', './app/views');

app.set('view engine', 'ejs');

app.engine('html', require('ejs').renderFile);

//bootstrap the app using the controller and routing modules

//The express.static() middleware takes one argument

//to determine the location of the static folder

//Configure static file serving

app.use(express.static('./public'));

return app;

};

To initialize your Mongoose configuration, go back to your server.js file, and change it to look like the following code snippet:

// The server.js file is the main file of your Node.js application

// It will load the express.js file as a module to bootstrap your Express application

//

//The process.env.NODE\_ENV variable is set to the default 'development‘

//value if itdoesn 't exist.

// Set the 'NODE\_ENV' variable

process.env.NODE\_ENV = process.env.NODE\_ENV || 'development';

// Load the module dependencies

var mongoose = require('./config/mongoose'),

express = require('./config/express');

// Create a new Mongoose connection instance

var db = mongoose();

// Create a new Express application instance

var app = express();

// Use the Express application instance to listen to the '3000' port

app.listen(3000);

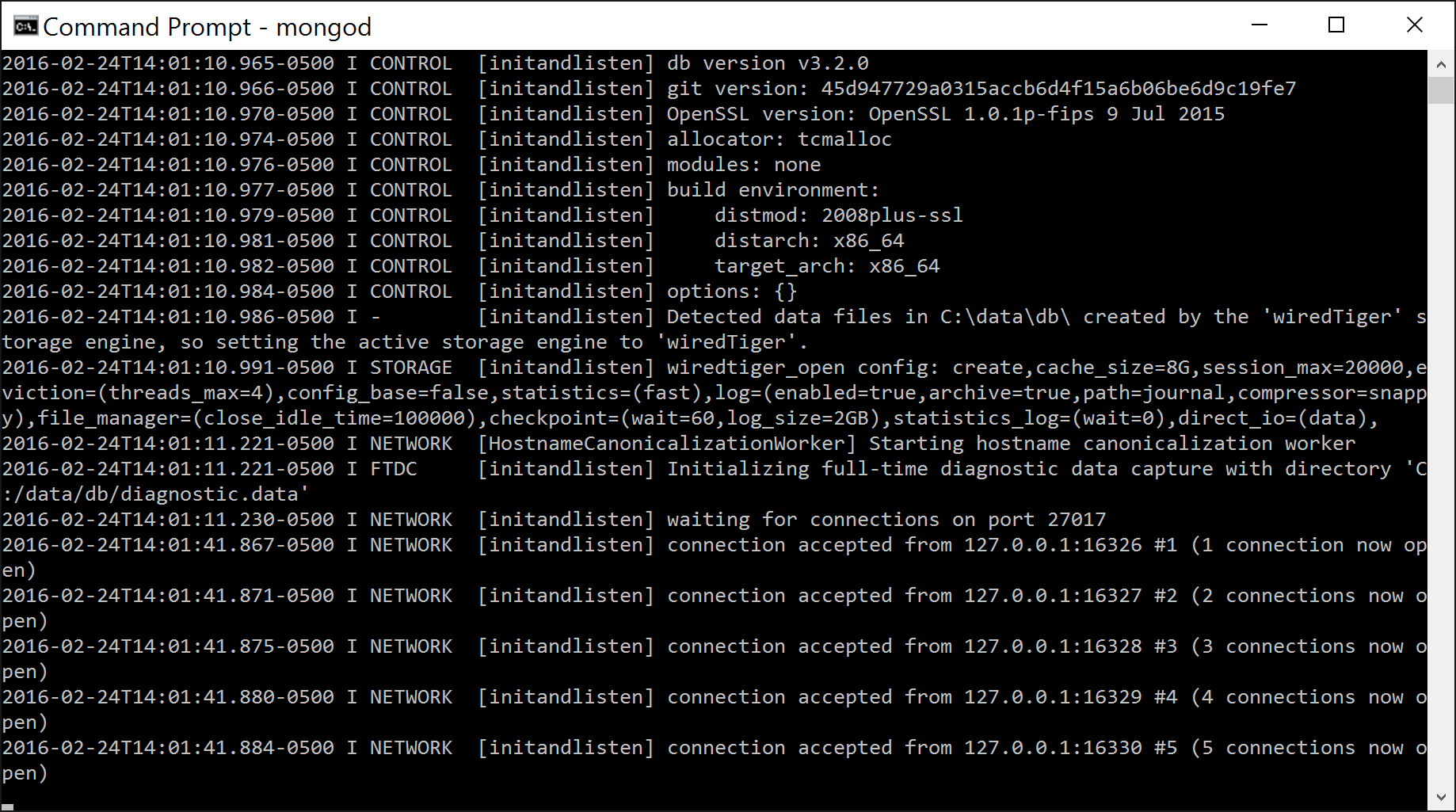
// Use the module.exports property to expose our Express application instance for external usage

module.exports = app; //returns the application object

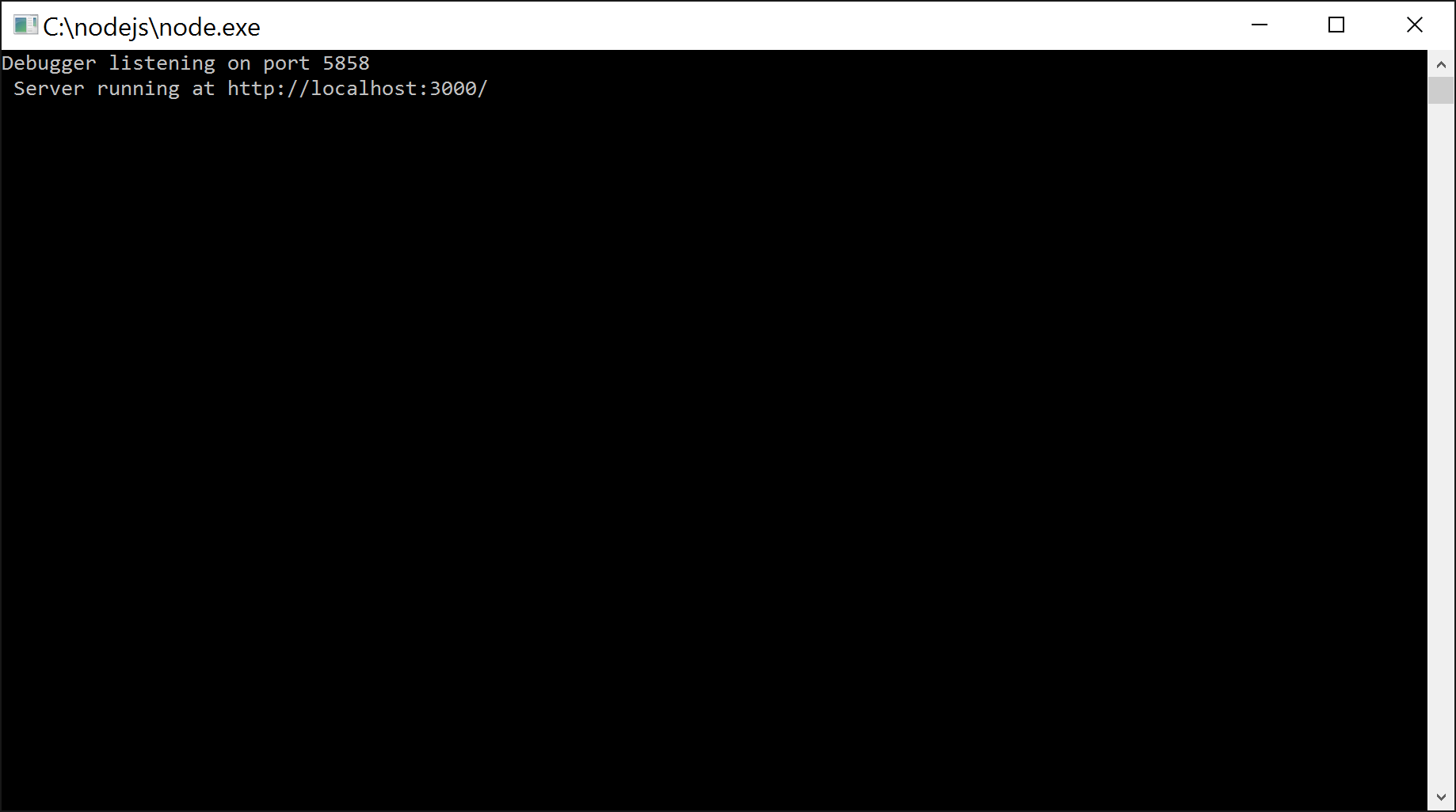
// Log the server status to the console

console.log('Server running at http://localhost:3000/');

Run mongod.exe to start the MongoDb server:



You can now run the application to see if the connection is accepted.



**Creating the task model**

To create your first schema, go to the app/models folder and create a new file named task.server.model.js. In this file, paste the following lines of code:

// Load the Mongoose module and Schema object

var mongoose = require('mongoose'),

Schema = mongoose.Schema;

// Define a new 'TaskSchema'

var TaskSchema = new Schema({

taskId: { type: String, unique: true, required: true },

taskName: String,

taskDescription: String,

startDate: {

type: Date,

// Create a default 'created' value

default: Date.now

},

endDate: {

type: Date,

// Create a default 'created' value

default: Date.now

},

owner: String

});

// Create the 'Task' model out of the 'TaskSchema'

mongoose.model('Task', TaskSchema);

**Registering the Task model**

Before you can start using the Task model, you will need to include the task.server.model.js file in your Mongoose configuration file in order to register the Task model. To do so, change your config/mongoose.js file to look like the following code snippet:

// Load the module dependencies:

// config.js module and mongoose module

var config = require('./config'),

mongoose = require('mongoose');

// Define the Mongoose configuration method

module.exports = function () {

// Use Mongoose to connect to MongoDB

var db = mongoose.connect(config.db);

// Load the 'Task' model

require('../app/models/task.server.model');

// Return the Mongoose connection instance

return db;

};

**Creating new tasks using save()**

To keep things organized, it is better that you create a Tasks controller that will handle all task-related operations. Under the app/controllers folder, create a new file named tasks.server.controller.js and paste the following lines of code:

// Load the 'Task' Mongoose model

var Task = require('mongoose').model('Task');

// Create a new 'createTask' controller method

exports.createTask = function (req, res, next) {

// Create a new instance of the 'Task' Mongoose model

var task = new Task(req.body);

// Use the 'Task' instance's 'save' method to save a new task document

task.save(function (err) {

if (err) {

// Call the next middleware with an error message

return next(err);

} else {

// Use the 'response' object to send a JSON response

res.json(task);

}

});

};

**Testing the controller**

To test your new controller, let's add a set of user-related routes that call the controller's methods. Begin by creating a file named tasks.server.routes.js inside the app/routes folder. In this newly created file, paste the following lines of code:

// Load the 'tasks' controller

var tasks = require('../../app/controllers/tasks.server.controller');

// Define the routes module' method

module.exports = function (app) {

// Set up the 'users' base routes

// a post request to /tasks will execute createTask method in tasks.server.controller

app.route('/tasks').post(tasks.createTask);

};

Load the routing file in express.js. The following code is your new express.js file:

//express.js file is where we configure our Express application

//

// Load the module dependencies

var config = require('./config'),

express = require('express'),

morgan = require('morgan'),

compress = require('compression'),

bodyParser = require('body-parser'),

methodOverride = require('method-override'),

session = require('express-session');

// Create a new Express application instance

module.exports = function () {

//Create the Express application object

var app = express();

//the process.env property allows you to access predefined environment variables

//such as NODE\_ENV

// Use the 'NDOE\_ENV' variable to activate the 'morgan' logger or 'compress' middleware

if (process.env.NODE\_ENV === 'development') {

app.use(morgan('dev'));

} else if (process.env.NODE\_ENV === 'production') {

app.use(compress());

}

// Use the 'body-parser' and 'method-override' middleware functions

app.use(bodyParser.urlencoded({

extended: true

}));

app.use(bodyParser.json()); //use middleware that only parses json

app.use(methodOverride()); // use HTTP verbs such as PUT or DELETE in places where the client doesn't support it.

//saveUninitialized - forces a session that is "uninitialized" to be saved to the store

//resave - forces the session to be saved back to the session store

// Configure the 'session' middleware

app.use(session({

saveUninitialized: true,

resave: true,

secret: config.sessionSecret

}));

//Configure Express to use EJS module as the default template engine

// Set the application view engine and 'views' folder

app.set('views', './app/views');

app.set('view engine', 'ejs');

app.engine('html', require('ejs').renderFile);

//bootstrap the app using the controller and routing modules

// Load the routing files

require('../app/routes/tasks.server.routes.js')(app);

//The express.static() middleware takes one argument

//to determine the location of the static folder

//Configure static file serving

app.use(express.static('./public'));

return app;

};

**Creating views.**

In app/views create an index.ejs file and paste the following code:

<!DOCTYPE html>

<html>

<head>

<title>Simple Task App</title>

</head>

<body>

<a href="/tasks">Add Task</a>

<br>

<a href="/list\_tasks">List Tasks</a>

</body>

</html>

This view allows the user to select the proper link for adding or reading tasks.

**Creating a view to send a POST request**

In app/views create an add\_task.ejs file and paste the following code:

<!DOCTYPE html>

<html>

<head>

<title>

<%=title %>

</title>

</head>

<body>

<form action="/tasks" method="post">

<div>

<label>Task ID:</label>

<input type="text" name="taskId" />

</div>

<div>

<label>Task Name:</label>

<input type="text" name="taskName" />

</div>

<div>

<label>Task Description:</label>

<input type="text" name="taskDescription" />

</div>

<div>

<label>Start Date:</label>

<input type="text" name="startDate" />

</div>

<div>

<label>End Date:</label>

<input type="text" name="endDate" />

</div>

<div>

<label>Owner:</label>

<input type="text" name="owner" />

</div>

<div>

<input type="submit" value="Create Task" />

</div>

</form>

</body>

</html>

In app/controllers add another controller to display the view. Name it index.server.controller.js.

Paste the following code:

// Create a new 'render' controller method

exports.render = function (req, res) {

// Use the 'response' object to render the 'index' view with a 'title' property

res.render('index', {title: 'Sample Test'} );

};

// Create a new 'renderAdd' controller method

exports.renderAdd = function (req, res) {

// Use the 'response' object to render the 'add\_task' view with a 'title' property

res.render('add\_task', { title: 'Add New task' });

};

Modify the routes to handle a GET request for displaying the above view.

Your tasks.server.routes.js file should look like the following:

//Load the index controller

var index = require('../../app/controllers/index.server.controller');

// Load the 'tasks' controller

var tasks = require('../../app/controllers/tasks.server.controller');

// Define the routes module' method

module.exports = function (app) {

// Set up the 'users' base routes

//

//show the 'index' page if a GET request is made to root

app.route('/').get(index.render);

//show the 'add\_task' page if a GET request is made to /tasks

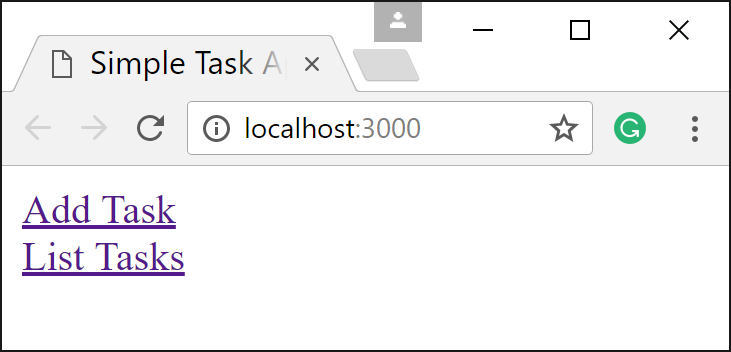
app.route('/tasks').get(index.renderAdd);

// a post request to /tasks will execute createTask method in tasks.server.controller

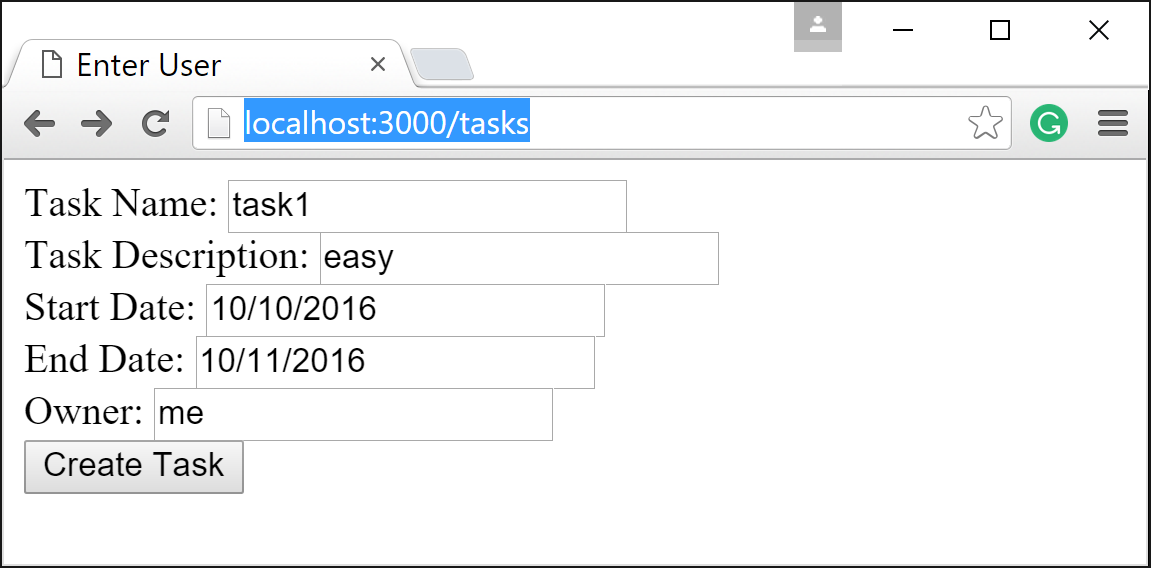
app.route('/tasks').post(tasks.createTask);

};

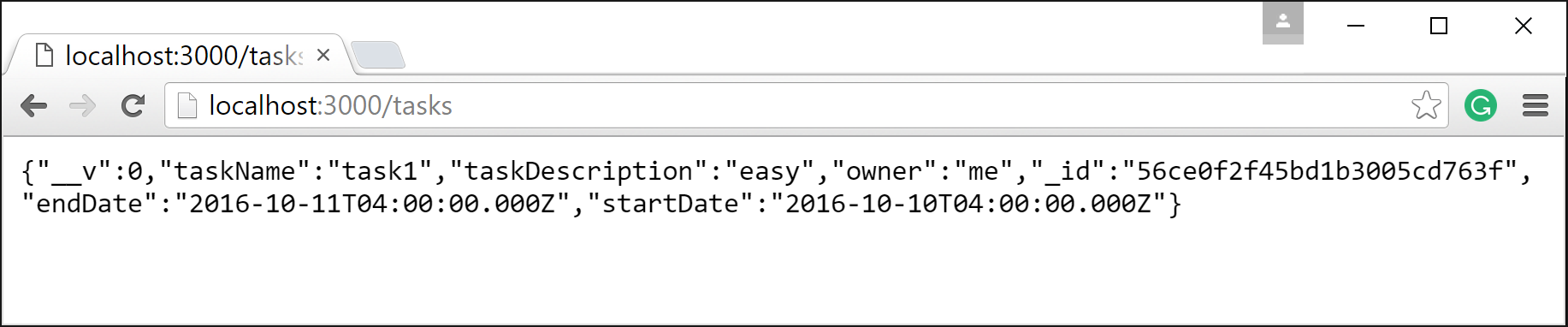
You are now ready to test your app. Run the application and navigate to <http://localhost:3000>.



Enter the information as shown below:



Click Create Task button. The task should be entered in your MongoDB database and you will get the following view:



Let’s now add functionalities to list the tasks and update existing tasks.

Go to tasks.server.controller.js and add the following method to read all tasks:

// Create a new 'readTasks' controller method

exports.readTasks = function (req, res, next) {

console.log('in readTasks')

// Use the 'Task' static 'find' method to retrieve the list of items

Task.find({}, function (err, tasks) {

console.log(tasks)

if (err) {

// Call the next middleware with an error message

console.log('some error in readTask method')

return next(err);

} else {

//

res.render('tasks', {

title: 'Tasks',

tasks: tasks

});

}

});

};

Create another view, tasks.ejs to display the tasks:

<!DOCTYPE html>

<html xmlns="http://www.w3.org/1999/xhtml">

<head>

<title><%= title %></title>

</head>

<body>

<form method = "post" action="/edit\_task">

<table align="center" >

<tr>

<td>

<div>

<label>Task ID:</label>

</div>

</td>

<td>

<div>

<label>Task Name:</label>

</div>

</td>

<td>

<div>

<label>Task Description:</label>

</div>

</td>

<td>

<div>

<label>Start Date:</label>

</div>

</td>

<td>

<div>

<label>End date </label>

</div>

</td>

<td>

<div>

<label>Owner </label>

</div>

</td>

</tr>

<% tasks.forEach(function(task){ %>

<tr>

<td>

<div>

<input type="text" name="taskId" value="<%= task.taskId %>"/>

</div>

</td>

<td>

<div>

<input type="text" name="taskName" value="<%= task.taskName %>"/>

</div>

</td>

<td>

<div>

<input type="text" name="taskDescription" value="<%= task.taskDescription %>"/>

</div>

</td>

<td>

<div>

<input type="text" name="startDate" value="<%= task.startDate %>" />

</div>

</td>

<td>

<div>

<input type="text" name="endDate" value="<%= task.endDate %>" />

</div>

</td>

<td>

<div>

<input type="text" name="owner" value="<%= task.owner %>" />

</div>

</td>

<td>

<div>

<input type="submit" value="Update Task" style="width: 200px"/>

</div>

</td>

</tr>

<% })%>

</table>

</form>

</body>

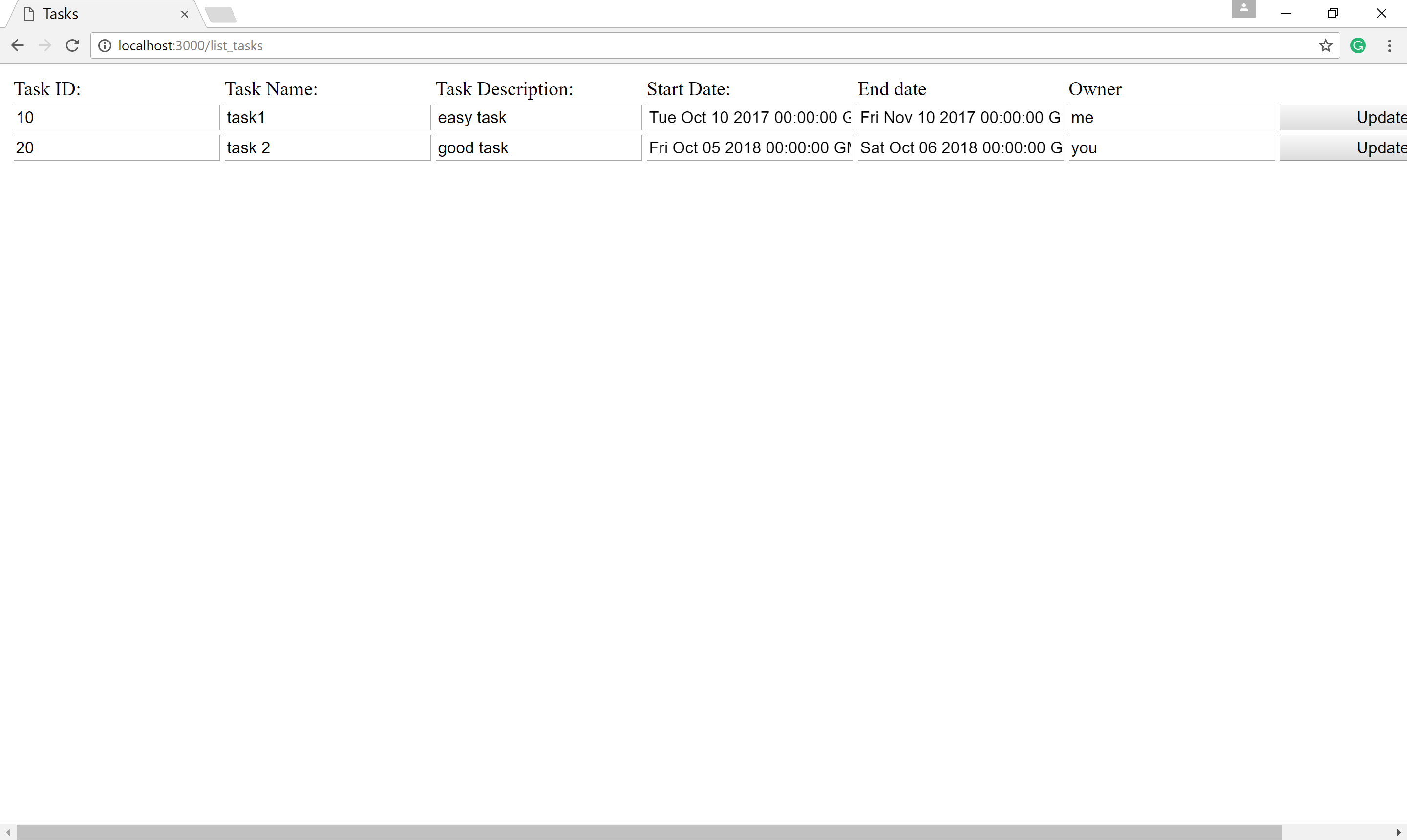
</html>

Note the use of ejs **forEach** function to loop over tasks and dynamically create and populate the rows of the table.

Add this routing code to routing file:

app.route('/list\_tasks').get(tasks.readTasks);

That’s it! Run the app again and choose list tasks.



Now you can add the update functionality as in MongoDBCRUDTest example.