

# NodeJS tutorial

---

*Dr. Balázs Simon (sbalazs@iit.bme.hu), BME IIT, 2022*

## 1 Introduction

This document describes how to create a Hello World application built in NodeJS using MongoDB. Make sure you have installed Visual Studio Code, NPM and MongoDB according to the installation guide.

## 2 Preparing the NodeJS environment

At first we need some global NodeJS packages so that we can generate and run our project.

Issue the following commands from a command prompt:

```
npm install --global yo typescript
npm install --global generator-express-ts
```

The first line will install Yeoman and TypeScript. Yeoman is a tool for generating the skeleton of a NodeJS web application. TypeScript is an open-source, strongly typed superset language of JavaScript. TypeScript is created by Microsoft to make JavaScript development scalable for large production applications. TypeScript code is translated to JavaScript by the TypeScript compiler.

The second line installs the express generator for TypeScript. Express is the base of almost all NodeJS applications.

## 3 Creating the NodeJS project

We will use the **yo** command to create our project. The project will use the **express** and **mongoose** modules at runtime.

Open a console window and go into a folder in your file system that should contain the project folder of your node project (e.g. **c:\Users\cloud\Documents**). **Important: the complete path of the folder must not contain spaces or any special characters. In case your user name contains spaces or special characters, create the project folder outside the c:\Users folder.**

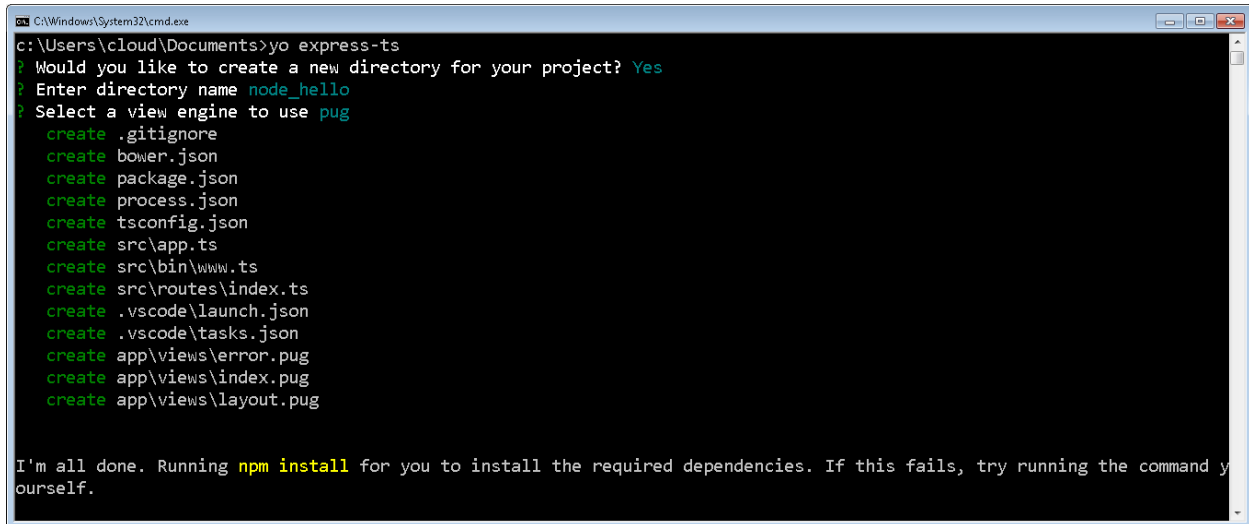
To generate the skeleton of your NodeJS project issue the following command:

```
yo express-ts
```

Follow the wizard and specify the following options:

- **Yes** for creating the project directory
- **node\_hello** as the project directory name
- **pug** as the view engine

The execution should look like this:



```
C:\Windows\System32\cmd.exe
c:\Users\cloud\Documents>yo express-ts
? Would you like to create a new directory for your project? Yes
? Enter directory name node_hello
? Select a view engine to use pug
create .gitignore
create bower.json
create package.json
create process.json
create tsconfig.json
create src/app.ts
create src/bin/www.ts
create src/routes/index.ts
create .vscode/launch.json
create .vscode/tasks.json
create app\views\error.pug
create app\views\index.pug
create app\views\layout.pug

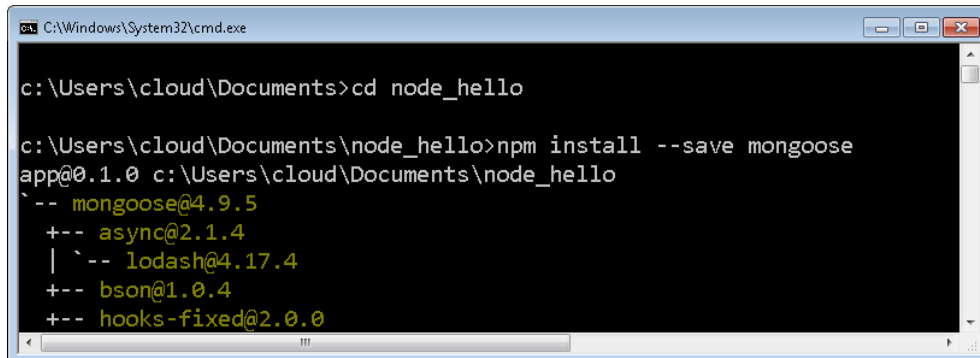
I'm all done. Running npm install for you to install the required dependencies. If this fails, try running the command yourself.
```

The project will contain the **package.json** file which contains the description of the dependencies. You already downloaded and installed all these dependencies by running the **npm install** command.

We still need to install the **mongoose** module in our application. Enter the project directory and install mongoose:

```
cd node_hello
npm install --save mongoose
```

The module should install without errors:



```
C:\Windows\System32\cmd.exe
c:\Users\cloud\Documents>cd node_hello

c:\Users\cloud\Documents\node_hello>npm install --save mongoose
app@0.1.0 c:\Users\cloud\Documents\node_hello
`-- mongoose@4.9.5
   +-- async@2.1.4
   |   `-- lodash@4.17.4
   +-- bson@1.0.4
   +-- hooks-fixed@2.0.0
```

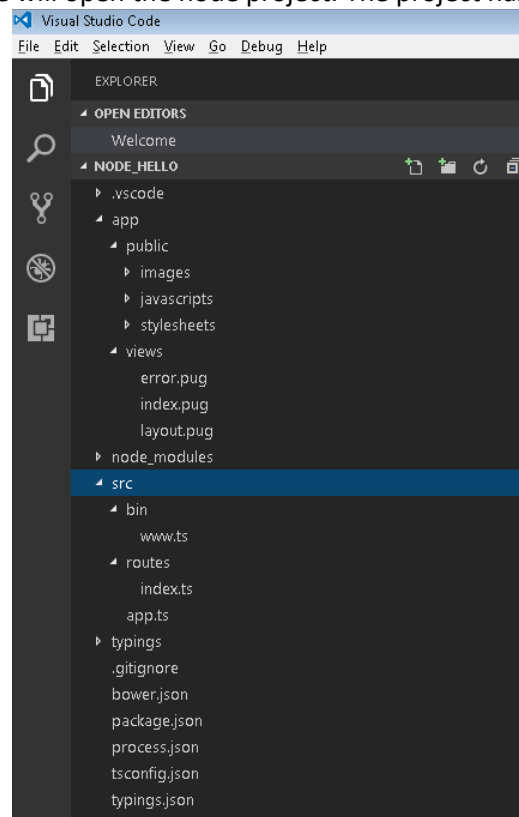
We also need the TypeScript descriptors for **express** and **mongoose**. You can install these using the following commands:

```
npm install --save @types/express
npm install --save @types/serve-static @types/express-serve-static-core @types/mime
npm install --global --save @types/mongoose
```

Now the project is ready for development.

## 4 Opening the project in VSCode

Start **Visual Studio Code** and select **File > Open Folder...** from the menu, and open the project folder you have just created. VSCode will open the node project. The project has the following structure:

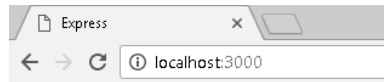


The folders and files serve the following tasks:

- **.vscode**: configuration files for VSCode
- **app**: the application code in JavaScript
  - **public**: the client-side of the application contains HTML, images, css, client-side JavaScript etc.
  - **views**: contains templates which generate HTML
- **node\_modules**: the NodeJS modules installed by **npm**
  - **@types**: the typings descriptors for NodeJS modules
- **src**: the application code in TypeScript
  - **bin/www.ts**: the main entry point for the application, this will start the HTTP server
  - **routes**: this contains **express** routes, which can be used to define what the application will do for different URL patterns
  - **app.ts**: configures the **express** application
- **bower.json**: configuration for client-side dependencies using **bower**
- **package.json**: configuration for **npm**, defines the module dependencies
- **process.json**: configuration for **pm2**, a process manager for NodeJS
- **tsconfig.json**: configuration for the TypeScript compiler **tsc**

We need the TypeScript compiler to continuously run in the background and automatically compile TypeScript files to JavaScript. To start the TypeScript compiler, press **Ctrl+Shift+B**. Select the **tsc: watch** task. This will immediately compile the existing **.ts** files in the project and will continue to do so whenever you change a **.ts** file. The generated JavaScript code will appear in the **app** folder. VSCode will also report any errors found by the TypeScript compiler.

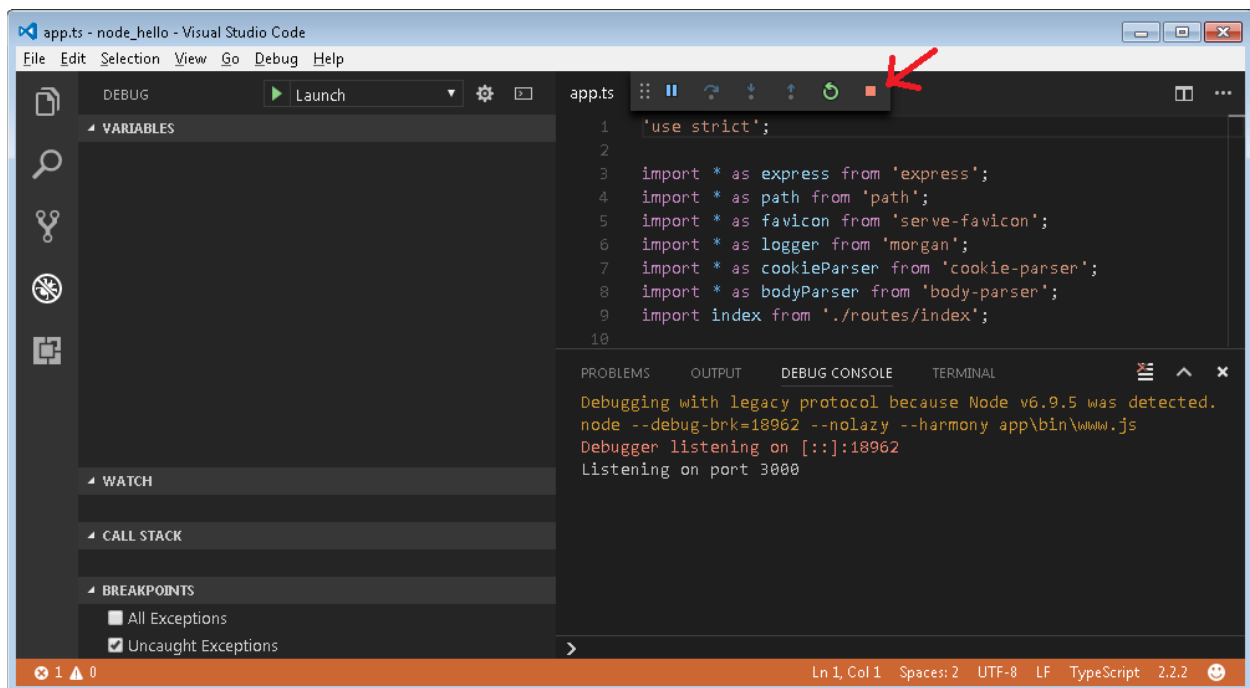
Now we can start our application by selecting **Debug / Start Debugging** from the menu or pressing **F5**. The server will start on port 3000. You can test it in the browser:



## Express

Welcome to Express

You can stop the server in VSCode by clicking the red square at the top of the window (or by pressing **Shift+F5**):



You can set breakpoints in the TypeScript file by selecting **Debug / Toggle breakpoint** from the menu or pressing **F9**. The execution will stop at that line when the server is started in debug mode (by pressing **F5**). The keyboard shortcuts for debugging are similar to the ones in Visual Studio.

## 5 Publishing a new route

Create a new folder called **services** under the **src** folder. Create a file called **hello\_service.ts** inside the **src/services** folder with the following content:

```
'use strict';

// Import express:
import * as express from 'express';
// Create a new express router:
const router = express.Router();

/* GET request: */
router.get('/', (req, res) => {
  // Get the 'name' query param:
  let name: String = req.query.name.toString();
  // Send response:
  res.send("Hello: "+name);
});

// Export the router:
export default router;
```

Next, we need to wire this router into the application's router. In order to do this, open the **src/routes/index.ts** and add the following highlighted lines:

```
'use strict';

import * as express from 'express';
import helloService from '../services/hello_service';

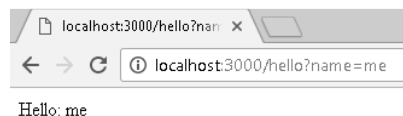
const router: express.Router = express.Router();

// Register the hello service:
router.use('/hello', helloService);

/* GET home page. */
router.get('/', (req, res, next) => {
  res.render('index', {title: 'Express'});
});

export default router;
```

Now every request to a URL starting with **/hello** will go to our **hello\_service**. Start the application and test it in a browser:



## 6 Accessing MongoDB

We need an ORM mapping from JavaScript to MongoDB to be able to access objects stored in the database conveniently.

At first, we have to define the interfaces of the objects. Create a new folder called **interfaces** under the **src** folder. Create a file called **hello.ts** inside the **src/interfaces** folder with the following content:

```
'use_strict';

export interface IHello
{
  name: string;
  message: string;
}
```

Next, we have to define the Mongoose mapping of this interface to MongoDB. Create a new folder called **schemas** under the **src** folder. Create a file called **hello.ts** inside the **src/schemas** folder with the following content (make sure to use the String type with capital 'S' here):

```
'use_strict';

import * as mongoose from 'mongoose';
import { IHello } from '../interfaces/hello';

export interface HelloEntity extends IHello, mongoose.Document { }

let HelloSchema = new mongoose.Schema({
  name: String,
  message: String
});

export var Hello = mongoose.model<HelloEntity>('Hello', HelloSchema, 'Hello');
```

Now we need to call this mapping from the service. Update the **src/services/hello\_service.ts** with the highlighted lines:

```
'use strict';

// Import express:
import * as express from 'express';
import { Hello } from '../schemas/hello';
// Create a new express router:
const router = express.Router();

/* GET request: */
```

```

router.get('/',(req,res) => {
  // Create a Mongoose query with the 'name' query param:
  let name: string = req.query.name.toString();
  let query = { name: name };
  // Execute query in the Hello schema:
  Hello.findOne(query, function(err, hello) {
    if (err) {
      res.json({info: 'Error executing query.', error: err});
    }
    if (hello) {
      res.json({info: 'Message found', data: hello.message});
    } else {
      res.json({info: 'Message not found by name: '+ name});
    }
  });
});

// Export the router:
export default router;

```

Finally, we have to configure MongoDB in the application. Add the following highlighted lines into the **src/app.ts** file:

```

'use strict';

import * as express from 'express';
import * as path from 'path';
import * as favicon from 'serve-favicon';
import * as logger from 'morgan';
import * as cookieParser from 'cookie-parser';
import * as bodyParser from 'body-parser';
import * as mongoose from 'mongoose';
import index from './routes/index';

const app: express.Express = express();

let mongoUri = 'mongodb://localhost/hello';
mongoose.connect(mongoUri, (err) => {
  if (err) {
    console.error(err.message);
    console.error(err);
  }
  else {
    console.log('Connected to MongoDB');
  }
});

```

```

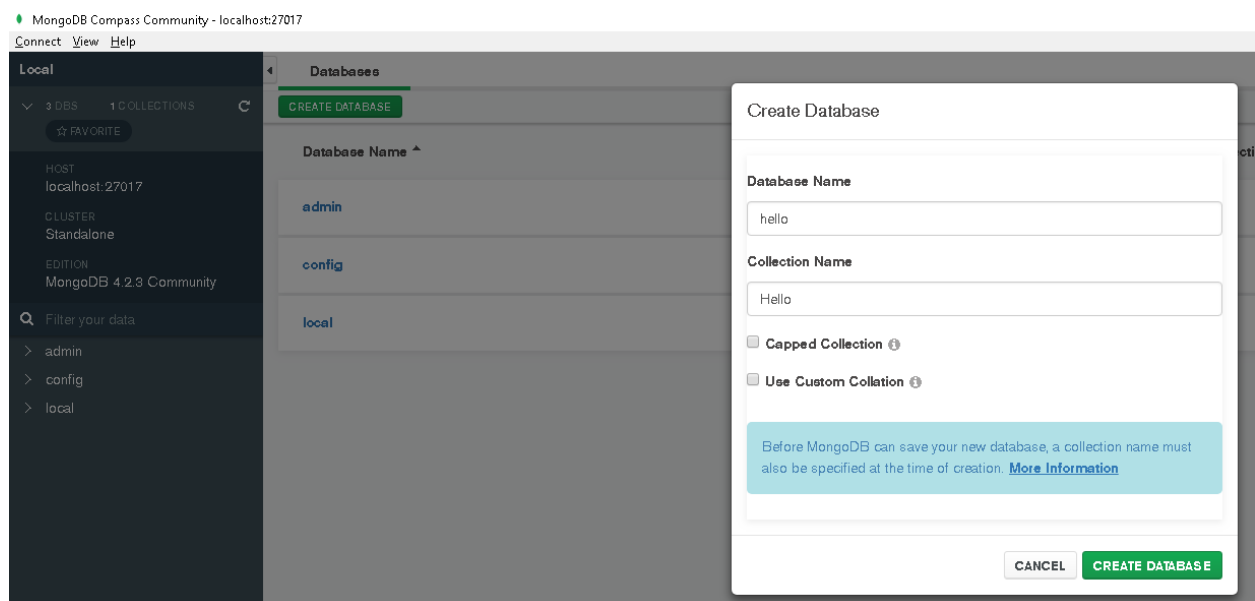
}
});

//view engine setup
app.set('views',path.join(__dirname,'views'));
app.set('view engine','pug');

//...

```

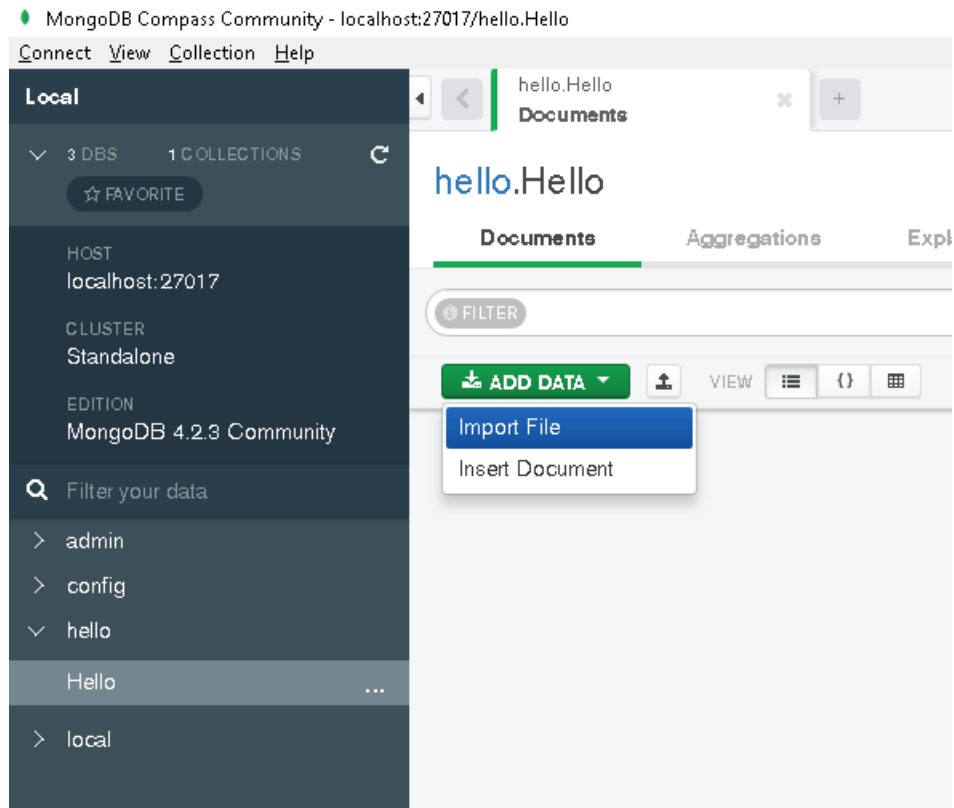
Let's add some data into the database. Open the MongoDB Compass client and create a new database called **hello** with a collection name **Hello**:



Click **Create** to create the database and the collection.



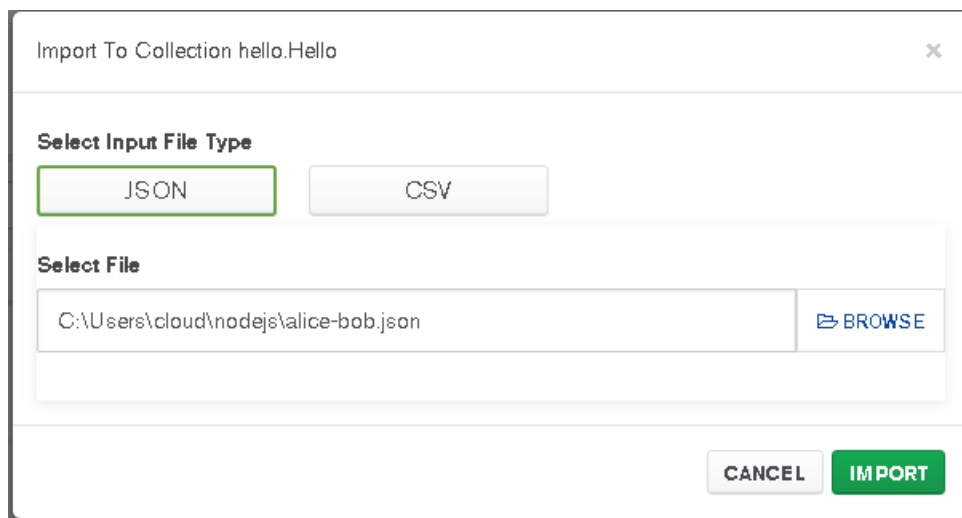
Select the **Hello** collection inside the **hello** database and click on **Add data > Import file**:



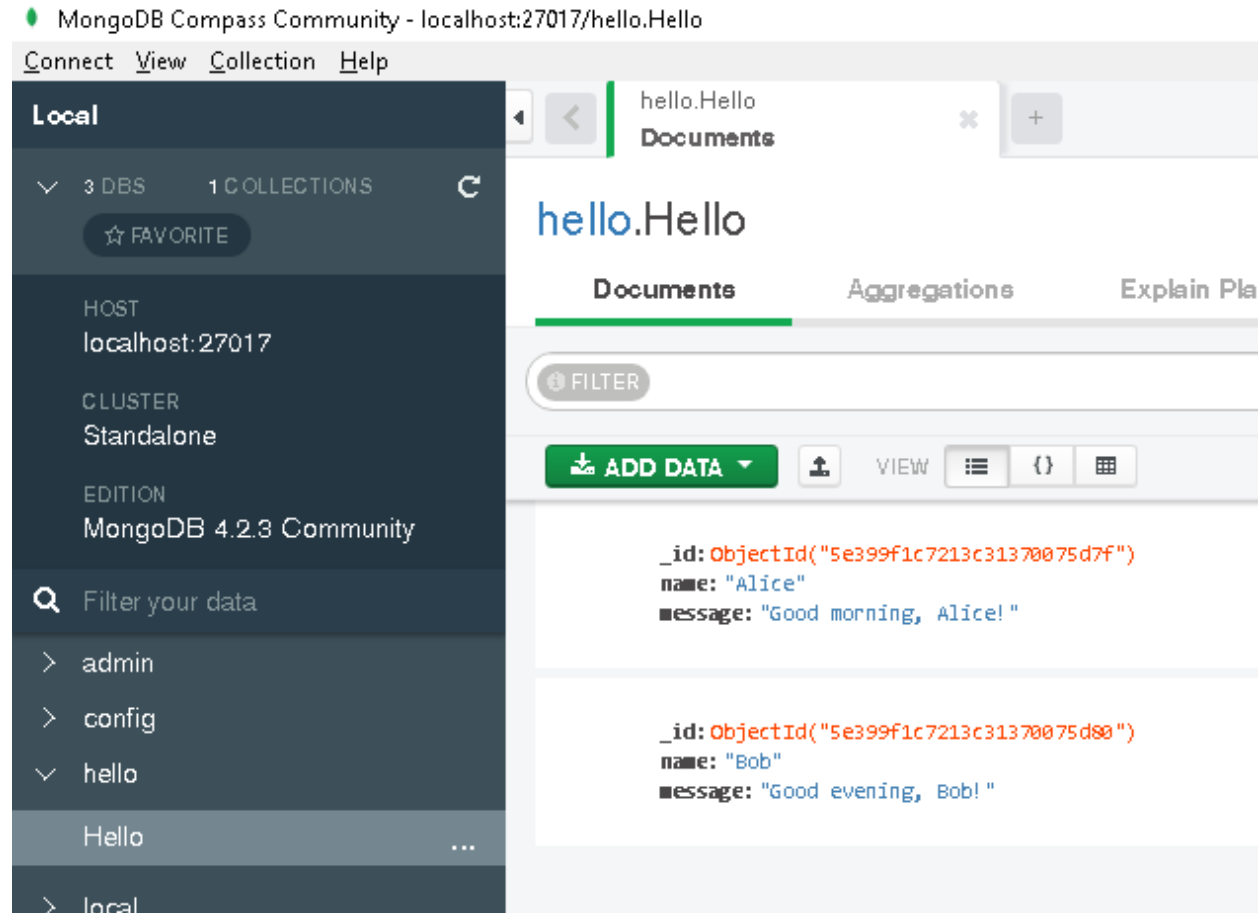
Create a JSON file with the following content:

```
{"name": "Alice", "message": "Good morning, Alice!"}  
{"name": "Bob", "message": "Good evening, Bob!"}
```

Then select this file to be imported:



This will insert two records into the database:



Now we can test our application in a browser:

