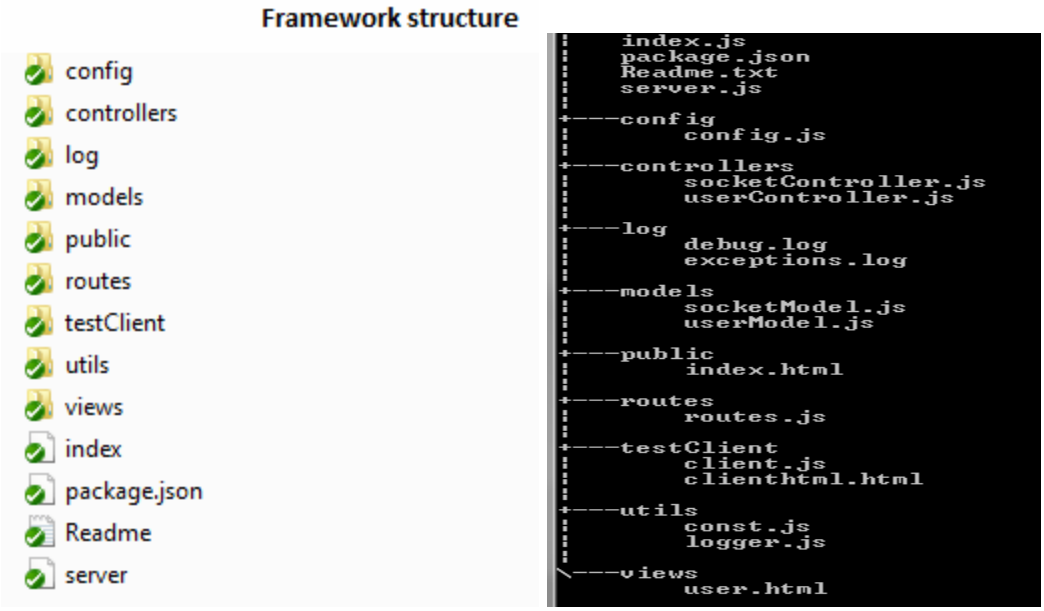
Node js Structured Framework

This framework is REST API based one and it can be used for fast api development. User can easily develop api functionality without doing much work.

Framework structure is shown in fig.



Framework structure

The structure of the files in an application is very important.It will help us to expand the current application structure by adding more modules and make an good app. Before doing that one , we should clearly understand the basic structure and how to make simple api function using current app.

We will separate core functions of our app into sensible and efficient file structure.

This framework contains some important modules,

- Configuration
- Routing
- Logging
- Socket io programming.
- Data management
- Client testing

Configuration

This is the main configuration file (config/config.js) that contains all configurations needed for the entire application to work. DB config, server config, app config, socket config etc.

Routing

Restify server based routing configuration implemented.The file routes/routes.js file exports the Router object that handles all the route to application framework.

Server

Node Restify server is used for framework development. server.js file is used to configure server, socket, and handlers.Once everything is configured correctly, you need to call initApp() function that will start application.

There is one index file in the root folder and it is entry point of application and db connection, server initialisation start from here.This file will invoke the framework server and start application.

Just need to run node index.js (Development or production mode can set by using NODE_ENV)

and you can see that server is started.
Open the browser and type url :http://localhost:8888/
Gives : "Welcome to REST API service --- \n"

Data management

This framework has separate directory structure for each file model, controller etc. Router routes the url to one function inside the controller. This framework structure contains controllers, models etc.

Controller navigate through the application and handlers (functions) inside controller call the appropriate model and returns the data. Data is taken from models, models retrieves the data from database, for different database we have to use different node database modules. Here we have used the sql server node database module.

Socket io programming

Socket io connection established through the same port used for api development. We have modularise the code to controller and model.There is no view present in this structured Framework.Current version only covers the basics of the socket io implementation and in future we are planning to implement multiple room Real Time Chat servert.

.

Data handling in CRUD operations.

Here we have already implemented a CRUD operation api functionalities using the current api framework.It includes user controller, model, routes etc.

Logging

In the current framework we are using winston log module.We can easily log the the info, error, warn and debug using this logger module.

```
// app logger file
var winston = require('winston');

var logger = new (winston.Logger)({
  transports: [
    new (winston.transports.Console)({ json: false, timestamp: true }),
    new winston.transports.File({ filename: './log/' + 'debug.log', json: false })
  ],
  exceptionHandlers: [
    new (winston.transports.Console)({ json: false, timestamp: true }),
    new winston.transports.File({ filename: './log/' + '/exceptions.log', json: false })
  ],
  exitOnError: false
});
// Set log levels
logger.setLevels({debug:0, info: 1, silly:2, warn: 3, error:4});

module.exports = logger;
```

this file is stored in utils/logger.js file.

In the application we can use like,

// Log information
logger.info('Application server started. ');
Two log files are created inside log folder for debug and exception log.

Client testing

This module is very important one and it can be used for REST api functional testing. Here is one html and js file, html file is used for socket io connection testing and js file one is used for testing

```
// Client.js file for testing the service

// Create Json client to test the server
var client = restify.createJsonClient({
  url: 'http://localhost:8888',
  version: '*'
});
// Checking server side connection
// Connect to server
var checkConnection = function(){
  client.get('/', function(err, req, res, obj) {
      console.log('%j', obj);
  });
}
// get userdetails
var getUserData = function(){
  client.get('/getAllusers', function(err, req, res, obj) {
      console.log('%j', obj);
  });
}
```

.

How to use this one ?

```
var client = require('./testClient/client');
// call client
client.clientConnect();
client.getUsers();
```

Will output the results in console.

Create a REST api simple function using our framework

We present here a demo of single api function development using our own framework. Let's begin.

1) **Create your own custom controller**

```
// New controller

// Custom model initialisation
var custom = require('../models/CustomModel')
var customModel = new custom();

function CustomController(){}

CustomController.prototype.index = function(req, res, next){

    customModel.index(req, res);
    return next();
}
module.exports = CustomController;
```

create /controllers/customController.js file and update the file with above code and save it.

2) **Create custom model**

```
// Custom Model
function CustomModel(){}

CustomModel.prototype.index = function(req, res) {
    var data = 'Custom model data';

    res.send(data);
}
module.exports = CustomModel;
```

Drop this bits of code in models/customModel.js.

3) **Add custom controller to server.js**

```
// Service handlers for each module
var userservice = require("./controllers/userController");
var socketHandler = require("./controllers/socketController");
var customHandler = require("./controllers/customController");
```

```
/*
  Create all service handler objects here .
*/
function createHandlerObjects(){
    servicehandlers = {
        user:  new userservice(),
        custom: new customHandler()
    };
}
```

Update the server.js with code shown in above. Initialise service handlers with new custom controller object.

Currently we are storing our action objects in service handlers object.

4) **Add Url Routing**

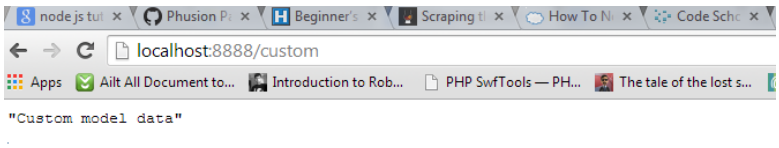
Edit routes\routes.js and update it.

```
Router.prototype.configUrls = function() {
    this.server.get('/', this.controllers.user.init);
    this.server.get('/getAllusers', this.controllers.user.getAllUserdetails);
    this.server.post('/update', this.controllers.user.updateUser);
    this.server.get('/delete/:id', this.controllers.user.deleteUser);
    this.server.post('/adduser', this.controllers.user.addUser);
    // Custom controller routes
    this.server.get('/custom', this.controllers.custom.index);
```

Now everything is completed and the new REST api function developed using our own new structured framework. It is very simple.

Run the server steps

Go to application folder -> cd appfolder
Start node server by typing - > node index.js
Open the browser and type -> http://localhost:8888/custom



It is working fine. The above mentioned example is simple one.

For database related method, you can check userModel.js file

```
UserModel.prototype.getUserdetails = function(req, res) {

    if (! isConnected) return(res.send("No database connection."));

    request.query("SELECT * FROM userTable", function(err, results){
        console.dir(results);
        if (err)
            res.end('Err'+err);
        else {
            res.writeHead(200,{'Content-Type':'application/json'});
            res.end(JSON.stringify(results));
        }
    })

}
```

Here we are using the sql server database and mssql node module included here.

Moving forward

In the future we want our framework to include cool features like
- Have multiple database(mysql, sqlite etc) configs
- Views development
- Secure api development
- Real time multiple room chat server implementation.