# Chapter-4

Construction

## 4.1 Implementation

### 4.1.1 Implementation Details

**Schemas**

**Request Schema code:-**

var mongoose = require("mongoose");

var Schema = mongoose.Schema;

var RequestSchema = new Schema({

user: {type: Schema.ObjectId, ref: 'User', required: true},

address: {lat: {type: Number, required: true}, lng: {type: Number, required: true}},

status: {type: String, required: true, enum: ['submitted', 'engineerAlloted', 'contractorAlloted', 'processing', 'completed'], default: 'submitted'},

engineer: {type: Schema.ObjectId, ref: 'User'},

contractor: {type: Schema.ObjectId, ref: 'User'},

});

module.exports = mongoose.model("Request", RequestSchema);

**User Schema code:-**

var mongoose = require("mongoose");

var Schema = mongoose.Schema;

var UserSchema = new Schema({

first\_name: {type: String, required: true},

last\_name: {type: String, required: true},

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

mobile: {type: String, required: true},

password: {type: String, required: true},

user\_type: {type: String, required: true, enum: ['user', 'engineer', 'contractor', 'admin'], default: 'user'}

});

module.exports = mongoose.model("User", UserSchema);

**Index routes code:-**

var express = require('express');

var router = express.Router();

var request\_controller = require('../controllers/requestController');

var user\_controller = require('../controllers/userController');

//Display Login Form GET

router.get('/', user\_controller.user\_login\_get);

//Handle Login Form POST

router.post('/login', user\_controller.user\_login\_post);

//Display Signup Form GET

router.get('/signup', user\_controller.user\_create\_get);

//Handle Signup Form POST

router.post('/signup', user\_controller.user\_create\_post);

module.exports = router;

**User routes code:-**

var express = require('express');

var router = express.Router();

var request\_controller = require('../controllers/requestController');

var user\_controller = require('../controllers/userController');

//Display dashboard GET

router.get('/dashboard', user\_controller.user\_dashboard\_get);

//Handle profile update POST

router.post('/update', user\_controller.user\_update\_post);

//Handle Admin Signup POST

router.post('/signup', user\_controller.admin\_create\_user\_post);

//Handle logout GET

router.get('/logout', user\_controller.user\_logout\_get);

module.exports = router;

**Request routes code:-**

var express = require('express');

var router = express.Router();

var request\_controller = require('../controllers/requestController');

var user\_controller = require('../controllers/userController');

//Handle Generate Request POST

router.post('/generate', request\_controller.user\_generate\_request\_post);

//Update Request POST

router.post('/update', request\_controller.request\_update\_post);

//Handle Request delete

router.post('/delete', request\_controller.delete\_request\_post);

//View Map for engineer POST

router.post('/map', request\_controller.view\_map\_post);

module.exports = router;

**Authorization Controller:-**

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

if (req.session.user) {

next(); //If session exists, proceed to page

} else {

var err = new Error("Not logged in!");

next(err); //Error, trying to access unauthorized page!

}

}

**Request Controller:-**

var User = require('../models/user');

var Request = require('../models/request');

var async = require('async');

var auth = require('./authorizationPermissions');

//Handle Generate Request POST

exports.user\_generate\_request\_post = function (req, res, next) {

var requestDetail = {

user: req.session.user,

address: {

lat: Number(req.body.lat),

lng: Number(req.body.lng)

},

status: "submitted"

}

var request = new Request(requestDetail);

request.save(function(err, saved){

if(err){

return next(err);

} else {

res.redirect('/user/dashboard');

}

})

}

//Update Request POST

exports.request\_update\_post = function(req, res, next){

var request = new Request({

\_id: req.body.request\_id,

user: req.body.request\_user\_id,

address: {

lat: req.body.request\_add\_lat,

lng: req.body.request\_add\_lng

},

status: req.body.status,

        engineer: req.body.engineer,

        contractor: req.body.contractor

});

// res.send(req.body);

    Request.findByIdAndUpdate(req.body.request\_id, request, function (err, updated) {

        if (err) {

            return next(err);

        } else {

            console.log("request updated");

            res.redirect('/user/dashboard');

        }

    })

}

exports.delete\_request\_post = function(req, res, next){

requestid = req.body.request\_id;

Request.findByIdAndRemove(requestid, function(err, removed){

if(err) next(err);

console.log("request deleted");

res.redirect('/user/dashboard');

})

}

//View Map for engineer POST

exports.view\_map\_post = function(req, res, next){

async.parallel({

        lat: function (cb) {

lat = req.body.request\_lat;

cb(null, lat);

        },

        lng: function (cb) {

lng = req.body.request\_lng;

cb(null, lng);

        },

    }, function (err, data) {

console.log(data);

        res.render('map', { data: data });

    });

}

**User Controller:-**

var User = require('../models/user');

var Request = require('../models/request');

var async = require('async');

var auth = require('./authorizationPermissions');

//Display SignUp Form GET

exports.user\_create\_get = function (req, res, next) {

    res.render('signup');

}

//Handle SignUp Form POST

exports.user\_create\_post = function (req, res, next) {

    User.findOne({ 'email': req.body.email }).exec(function (err, found\_user) {

        if (err) { next(err); }

        if (found\_user) {

            err = new Error("User Exist");

            return next(err);

        } else {

            var userdetail = {

                first\_name: req.body.fname,

                last\_name: req.body.lname,

                password: req.body.password,

                email: req.body.email,

                mobile: req.body.mobile,

                user\_type: req.body.userType

            }

            var user = new User(userdetail);

            user.save(function (err, user) {

                if (err) { return next(err); }

                req.session.user = user;

                res.redirect('/user/dashboard');

            });

        }

    });

}

//Handle SignUp Form from Admin POST

exports.admin\_create\_user\_post = function (req, res, next) {

    User.findOne({ 'email': req.body.email }).exec(function (err, found\_user) {

        if (err) { next(err); }

        if (found\_user) {

            err = new Error("User Exist");

            return next(err);

        } else {

            var userdetail = {

                first\_name: req.body.fname,

                last\_name: req.body.lname,

                password: req.body.password,

                email: req.body.email,

                mobile: req.body.mobile,

                user\_type: req.body.userType

            }

            var user = new User(userdetail);

            user.save(function (err, user) {

                if (err) { return next(err); }

                res.redirect('/user/dashboard');

            });

        }

    });

}

//Display login form GET

exports.user\_login\_get = function (req, res, next) {

    res.render('index');

}

//Handle Login form POST

exports.user\_login\_post = function (req, res, next) {

    User.findOne({ email: req.body.email, password: req.body.password }, function (err, user) {

        if (err) { return next(err); }

        if (!user) {

            var err = new Error("User not found");

            err.status = 403;

            return next(err);

        }

        req.session.user = user;

        res.redirect('/user/dashboard');

    });

}

//Functions to display various Dashboards

function userDashboard(req, res, next) {

    async.parallel({

        requests: function (cb) {

            Request.find({user: req.session.user.\_id}).populate({

                path: 'user',

                model: 'User',

                select: 'first\_name last\_name mobile'

            }).exec(function (err, requests) {

                if (err) return next(err);

                cb(null, requests);

            });

        },

        user: function (cb) {

            var user = req.session.user;

            cb(null, user);

        },

        engineers: function (cb) {

            User.find({ user\_type: 'engineer' }, 'first\_name last\_name', function (err, engineers) {

                if (err) return next(err);

                cb(null, engineers);

            });

        },

        contractors: function (cb) {

            User.find({ user\_type: 'contractor' }, 'first\_name last\_name', function (err, contractors) {

                if (err) return next(err);

                cb(null, contractors);

            });

        }

    }, function (err, data) {

        res.render('userdash', { data: data });

    });

}

function adminDashboard(req, res, next) {

    async.parallel({

        requests: function (cb) {

            Request.find().populate({

                path: 'user',

                model: 'User',

                select: 'first\_name last\_name mobile'

            }).populate({

                path: 'engineer',

                model: 'User',

                select: 'first\_name last\_name'

            }).populate({

                path: 'contractor',

                model: 'User',

                select: 'first\_name last\_name'

            }).exec(function (err, requests) {

                if (err) return next(err);

                console.log(requests.engineer);

                cb(null, requests);

            });

        },

        user: function (cb) {

            var user = req.session.user;

            cb(null, user);

        },

        engineers: function (cb) {

            User.find({ user\_type: 'engineer' }, 'first\_name last\_name', function (err, engineers) {

                if (err) return next(err);

                cb(null, engineers);

            });

        },

        contractors: function (cb) {

            User.find({ user\_type: 'contractor' }, 'first\_name last\_name', function (err, contractors) {

                if (err) return next(err);

                cb(null, contractors);

            });

        }

    }, function (err, data) {

        res.render('admindash', { data: data });

    });

}

function engineerDashboard(req, res, next) {

    async.parallel({

        requests: function (cb) {

            Request.find({ engineer: req.session.user.\_id }).populate({

                path: 'user',

                model: 'User',

                select: 'first\_name last\_name mobile'

            }).exec(function (err, requests) {

                if (err) return next(err);

                cb(null, requests);

            });

        },

        user: function (cb) {

            var user = req.session.user;

            cb(null, user);

        },

    }, function (err, data) {

        res.render('engineerdash', { data: data });

    });

}

function contractorDashboard(req, res, next) {

    async.parallel({

        requests: function (cb) {

            Request.find({contractor: req.session.user.\_id}).populate({

                path: 'user',

                model: 'User',

                select: 'first\_name last\_name mobile'

            }).exec(function (err, requests) {

                if (err) return next(err);

                cb(null, requests);

            });

        },

        tender: function (cb) {

            Request.find({ status: 'engineerAlloted' }).populate({

                path: 'user',

                model: 'User',

                select: 'first\_name last\_name mobile'

            }).exec(function(err, tender){

                if (err) return next(err);

                cb(null, tender)

            });

        },

        user: function (cb) {

            var user = req.session.user;

            cb(null, user);

        },

        engineers: function (cb) {

            User.find({ user\_type: 'engineer' }, 'first\_name last\_name mobile', function (err, engineers) {

                if (err) return next(err);

                cb(null, engineers);

            });

        },

    }, function (err, data) {

        res.render('contractordash', { data: data });

    });

}

//Display dashboard GET

exports.user\_dashboard\_get = function (req, res, next) {

    var userType = req.session.user.user\_type;

    if (userType == 'admin') {

        adminDashboard(req, res, next);

    } else if (userType == 'user') {

        userDashboard(req, res, next);

    } else if (userType == 'contractor') {

        contractorDashboard(req, res, next);

    } else if(userType == 'engineer') {

        engineerDashboard(req, res, next);

    } else {

        res.next(err);

    }

}

//Handle update profile POST

exports.user\_update\_post = function (req, res, next) {

    userid = req.session.user.\_id;

    var user = new User({

        first\_name: req.body.fname,

        last\_name: req.body.lname,

        password: req.body.password,

        email: req.body.email,

        mobile: req.body.mobile,

        \_id: req.session.user.\_id,

        user\_type: req.session.user.user\_type

    });

    User.findByIdAndUpdate(userid, user, function (err, updated) {

        if (err) {

            return next(err);

        } else {

            req.session.user = user;

            console.log("user updated");

            res.redirect('/user/dashboard');

        }

    })

};

//HAndle user logout POST

exports.user\_logout\_get = function (req, res, next) {

    req.session.destroy(function () {

        res.redirect('/');

    });

};

#### 4.1.1.1 Software Details

* NodeJS
* HTML5
* CSS3
* Node Modules
* StarUML
* Bootstrap
* MongoDB

#### 4.1.1.2 Hardware Details

* Server: -
  + 4GB RAM
  + Quad Core Processor(Preferred)
  + 1GB Storage
* Client: -
  + 1GB RAM
  + Dual Core Processor
  + 1GB Storage

## 4.2 Testing

The reason behind testing is to find errors. Every program or software has errors in it, against the common view that there are no errors in it if the program or software is working. Executing the programs with the intention of finding the errors in it is therefore testing; hence a successful test is one which finds errors.

Testing is an activity; however, it is restricted to being performed after the development phase is complete but is carried parallel with all stages of system development, starting with requirement specification. A test case is a set of the data that a system will process as normal input. The software units developed in the system are modules and routines that are assembled and integrated to perform the required function of the system. Test results once gathered and evaluated, provide a qualitative indication of the software quality and reliability and serve as basis for design modification if required. The testing phase of the implementations works accurately and efficiently before live operation commences.

### 4.2.1 White Box Testing

This method of software testing tests internal structures or workings of an application, as opposed to its functionality (i.e. black-box testing). In white-box testing an internal perspective of the system, as well as programming skills, are required and used to design test cases. The tester chooses inputs to exercise paths through the code and determine the appropriate outputs.

### 4.2.2 Black Box Testing

This method of software testing tests the functionality of an application as opposed to its internal structures or working. Specific knowledge of the internal structure and programming knowledge in general is not required. It uses external descriptions of the software, including specifications, requirements, and designs to derive test cases. The test designer selects valid and invalid inputs and determines the correct output.

**4.2.3 Unit Testing**

The unit testing was done after the coding phase was done. The purpose of the unit testing was to locate errors on the module, independent of the other modules. Some changes in the coding were done during the testing. Finally, all the modules were individually tested from bottom up starting with smallest and lowest modules and proceeding one at a time.

**4.2.4 Integration Testing**

Once the unit was over, all the modules were integrated for integration testing. External and internal interfaces are implemented and work as per design, the performance of the module is not degraded.

**4.2.5** **Validation Testing**

At the culmination of integration testing, software is said to be completely

assembled as a package; interfacing errors have been uncovered and corrected. Then as a final series of software test, validation tests were carried out.

**4.2.6 Acceptance Testing**

This is the final stage in the testing process before the system is accepted for operational use. Any requirement problem or requirement definition problem revealed from acceptance testing are considered and made error free.