

## webapplog: [programming weblog]

Software engineering, Node.js, JavaScript and startups.

# Express.js and Mongoose Example: Building HackHall

**Note:** This text is a part of the [Express.js Guide: The Comprehensive Book on Express.js](#).

The [HackHall](#) project was written using Backbone.js and Underscore for the front-end app, and Express.js, MongoDB via Mongoose for the back-end REST API server.

**Example:** The HackHall source code is in the public [GitHub repository](#).

The live demo is accessible at [hackhall.com](#) either with Angellist or pre-filled email (1@1.com) and password (1).

## What is HackHall

HackHall (ex-Accelerator.IO) is an open-source invite-only social network and collaboration tool for hackers, hipsters, entrepreneurs and pirates (just kidding). HackHall is akin to Reddit, plus Hacker News, plus Facebook Groups with curation.

The HackHall project is in its early stage and roughly a beta. We plan to extend the code base in the future and bring in more people to share skills, wisdom and passion for programming.

In this chapter, we'll cover the [1.0 release](#) which has:

- Email and password authentication
- Mongoose models and schemas
- Express.js structure with routes in modules
- JSON REST API
- Express.js error handling
- Front-end client Backbone.js app (for more info on Backbone.js, download/read online our [Rapid Prototyping with JS](#) tutorials)
- Environmental variables with Foreman's .env
- TDD with Mocha
- Basic Makefile setup

## Running HackHall

To get the source code, you can navigate to hackhall folder or clone from GitHub:

```
$ git clone git@github.com:azat-co/hackhall
$ git checkout 1.0
$ npm install
```

If you plan to test an AngelList (optional), HackHall is using Heroku and Foreman setup for AngelList API keys storing them in environmental variables, so we need to add .env file like this (below are fake values):

```
ANGELLIST_CLIENT_ID=254C0335-5F9A-4607-87C0
ANGELLIST_CLIENT_SECRET=99F5C1AC-C5F7-44E6-81A1-8DF4FC42B8D9
```

The keys are obtainable at [angel.co/api](https://angel.co/api) after someone creates and registers his/her AngelList app.

and third-party libraries are out of scope of this book. However, you can find enough materials [online](#) and in [Rapid Prototyping with JS](#).

To start MongoDB server, open a new terminal window and run:

```
$ mongod
```

Go back to the project folder and run:

```
$ foreman start
```

After MongoDB is running on localhost with default port 27017, it's possible to seed the database `hackhall` with default admin user by running `seed.js` mongo script:

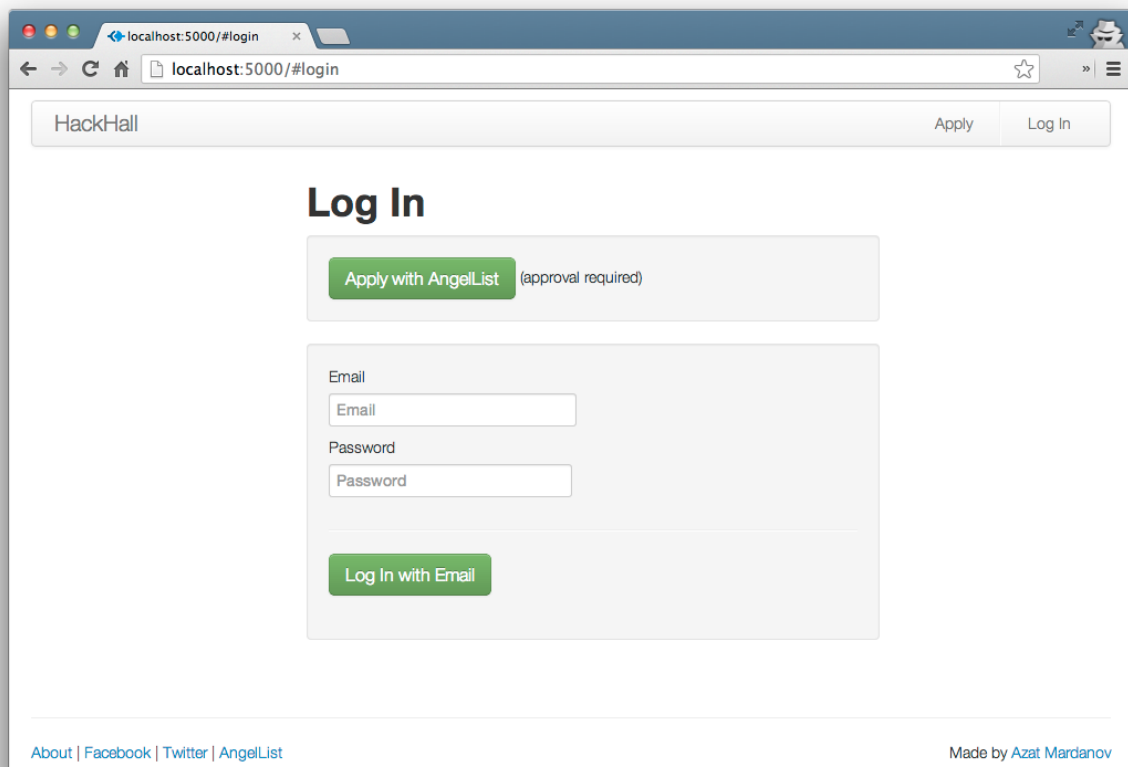
```
$ mongo localhost:27017/hackhall seed.js
```

Feel free to modify `seed.js` to your liking (beware that it erases all previous data!):

```
db.dropDatabase();
var seedUser = {
  firstName: 'Azat',
  lastName: 'Mardanov',
  displayName: 'Azat Mardanov',
  password: '1',
  email: '1@1.com',
```

```
approved: true,  
  admin: true  
};  
db.users.save(seedUser);
```

If you open your browser at <http://localhost:5000>, you should see the login screen.



Enter username and password to get in (the ones from the `seed.js` file).

HackHall Apply Log In

## Log In

[Apply with AngelList](#) (approval required)

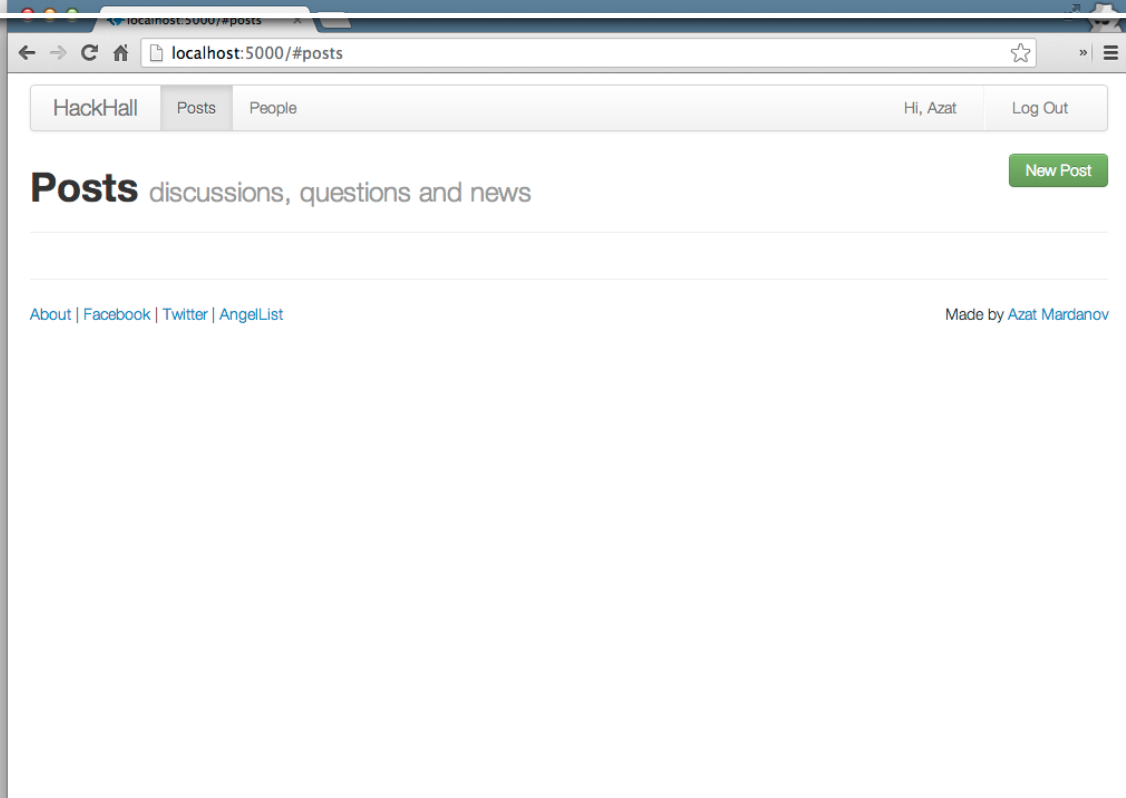
Email

Password

[Log In with Email](#)

[About](#) | [Facebook](#) | [Twitter](#) | [AngelList](#) Made by [Azat Mardanov](#)

After successful authentication, users are redirected to the Posts page:



where they can create a post (e.g., a question):

The screenshot shows a web browser window with the address bar displaying `localhost:5000/#posts/new`. The page has a header with the site name 'HackHall', navigation links 'Posts' and 'People', a user greeting 'Hi, Azat', and a 'Log Out' button. The main content area is titled 'New Post' and contains a form with three fields: 'Title' with the value 'Express.js patterns?' and a hint 'Keep it short and meaningful.', 'URL' with the value 'URL' and a hint 'To start discussion or ask a question, leave URL blank.', and 'Text' with the value 'What are the best Express.js patterns?'. A 'Submit' button is located below the form. The footer contains links 'About | Facebook | Twitter | AngelList' and the text 'Made by Azat Mardanov'.

HackHall Posts People Hi, Azat Log Out

### New Post

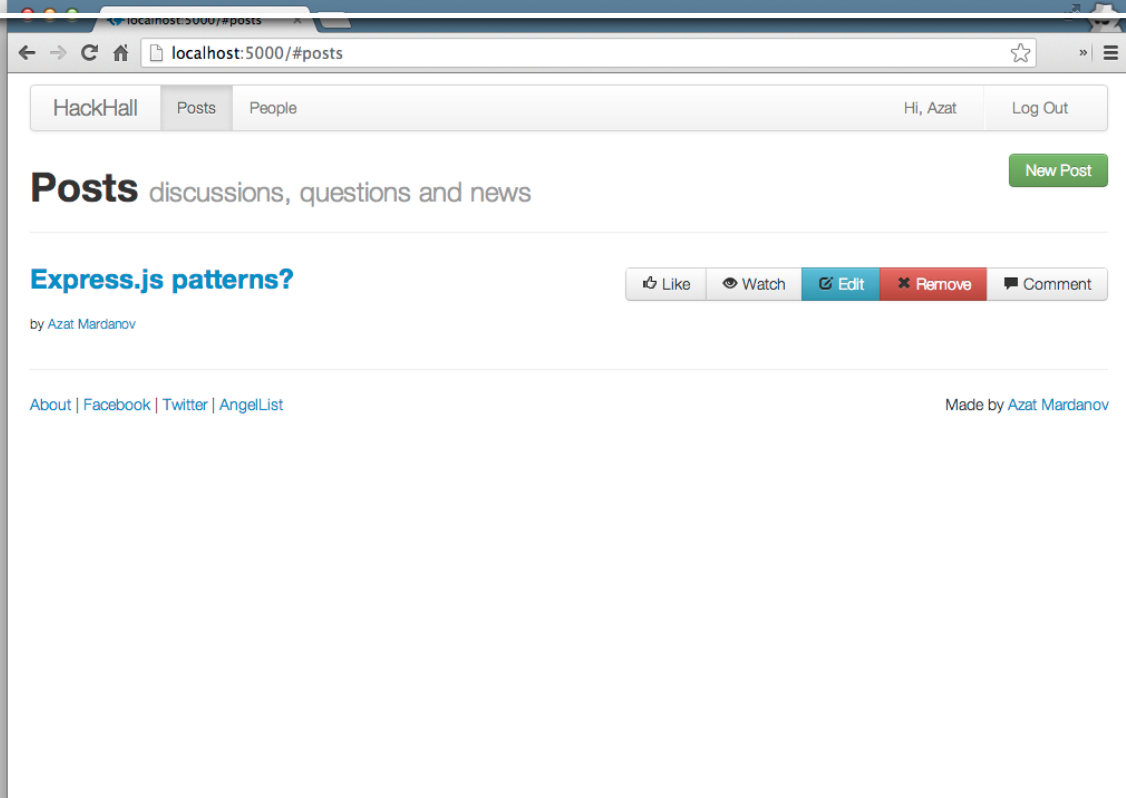
Title  Keep it short and meaningful.

URL  To start discussion or ask a question, leave URL blank.

Text  Leave the URL blank otherwise the Text will be ignored.

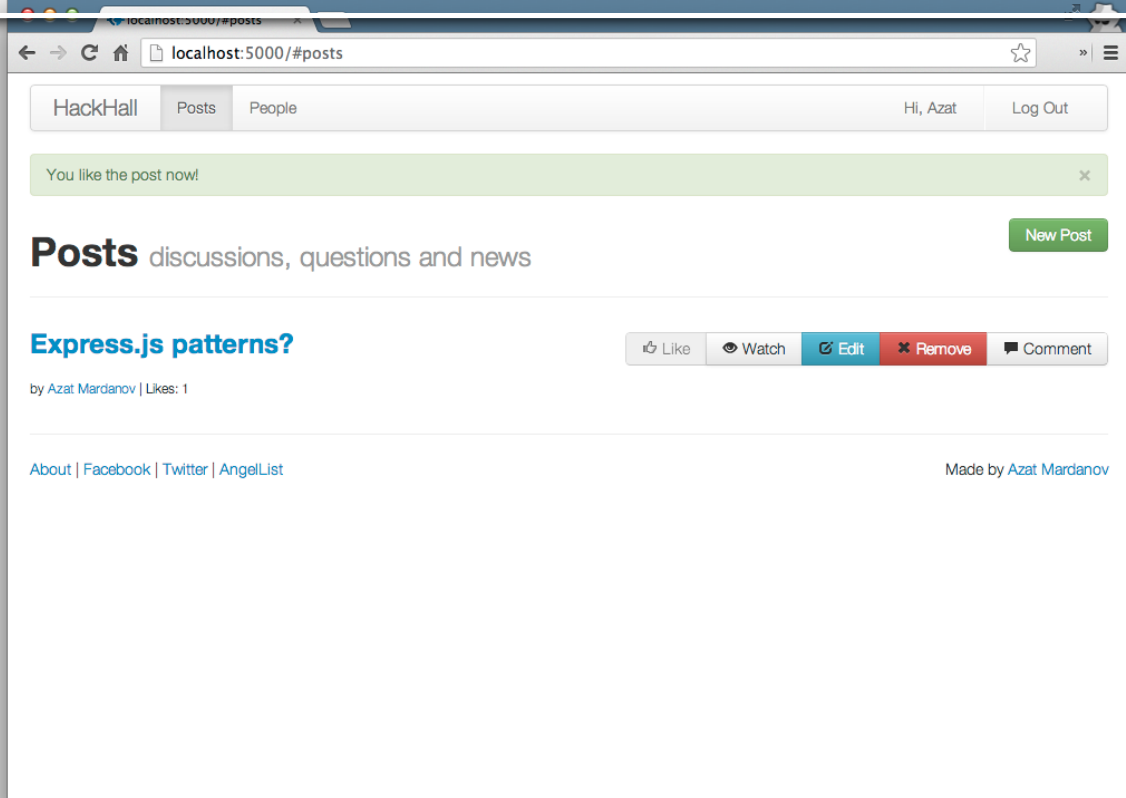
[About](#) | [Facebook](#) | [Twitter](#) | [AngelList](#) Made by Azat Mardanov

Save the post:

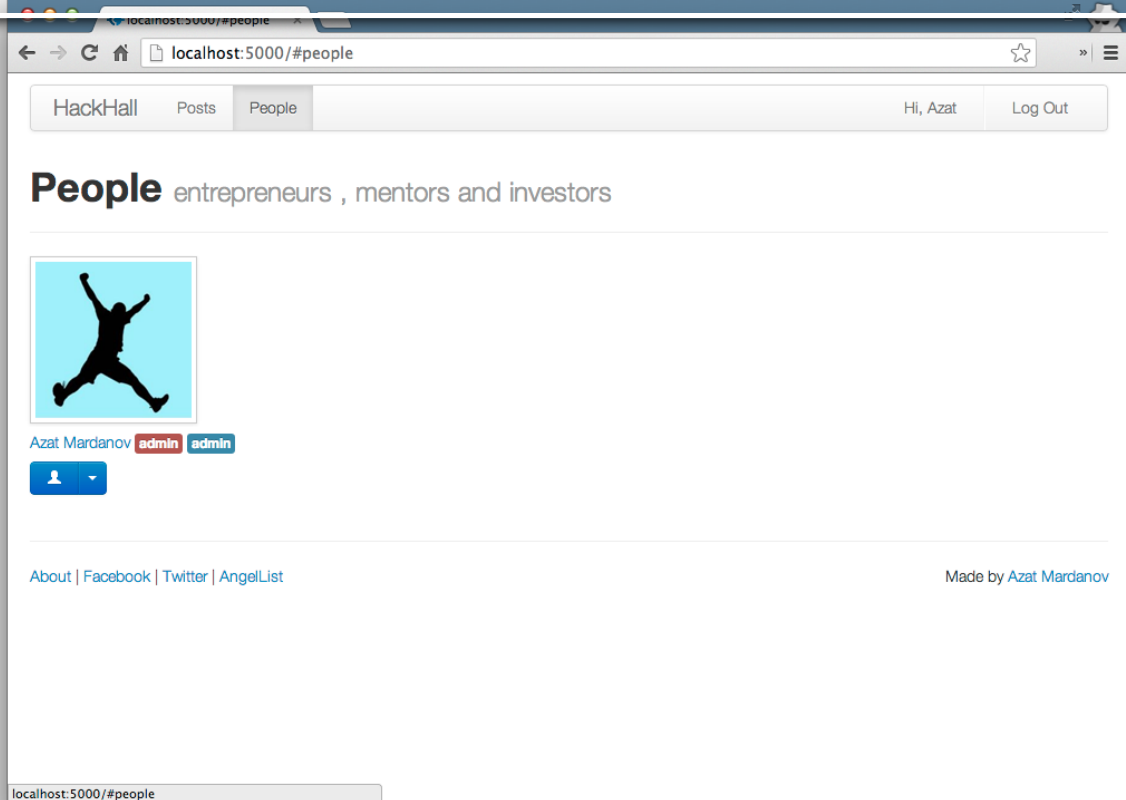


Like posts:

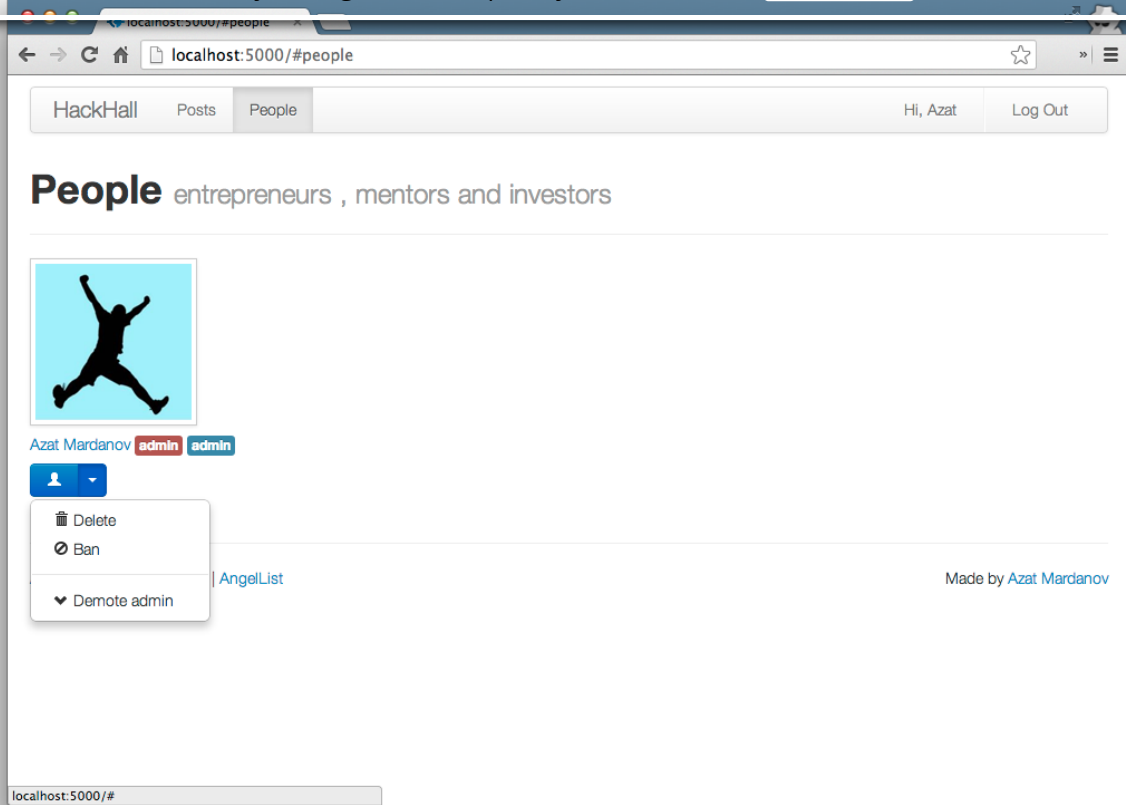




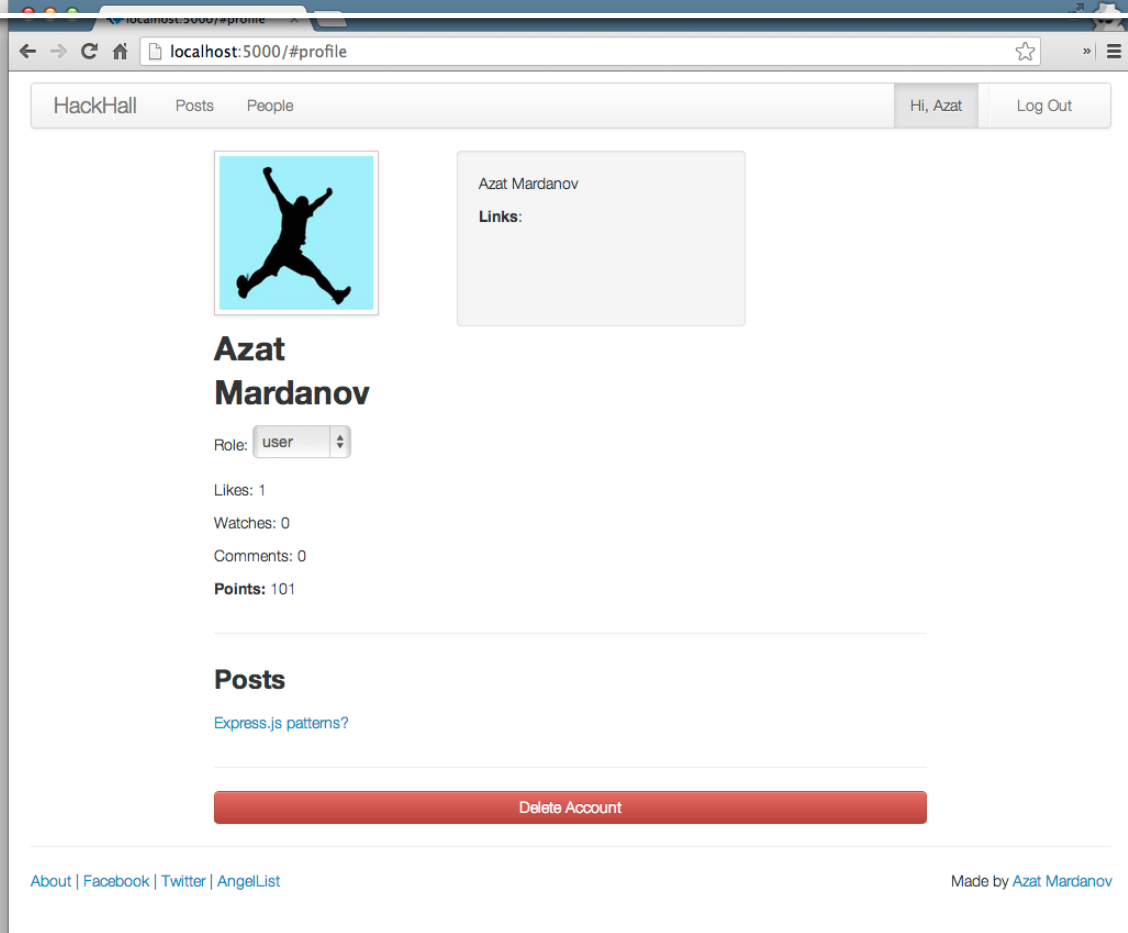
Visit other users' profiles:



If they have admin rights, users can approve applicants:



and manage their account on Profile page:

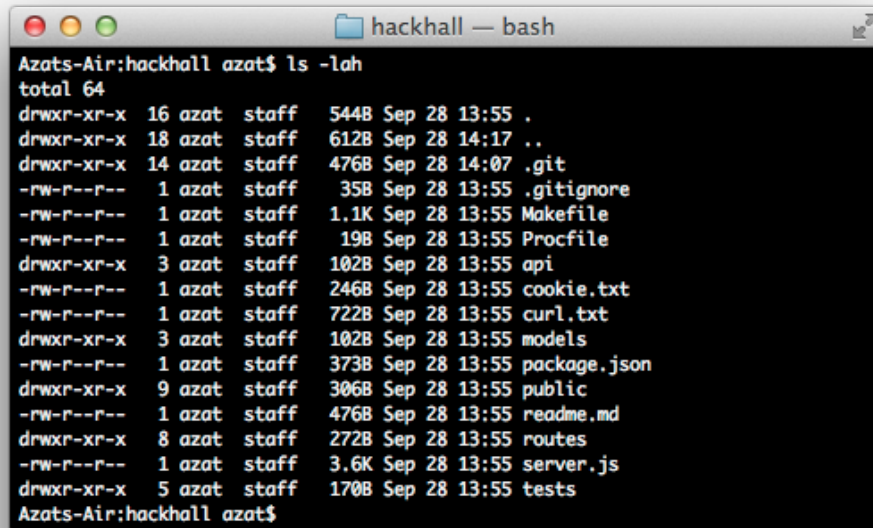


## Structure

Here are what each of the folders and files have:

- /api: app-shared routes
- /models: Mongoose models
- /public: Backbone app, static files like front-end JavaScript, CSS, HTML
- /routes: REST API routes
- /tests: Mocha tests
- .gitignore: list of files that git should ignore
- Makefile: make file to run tests

- `package.json`: NPM dependencies and HackHall meta data
- `readme.md`: description
- `server.js`: main HackHall server file



A terminal window titled 'hackhall — bash' showing the output of the command `ls -lah`. The output lists files and directories with their permissions, sizes, and timestamps. The files include `.`, `..`, `.git`, `.gitignore`, `Makefile`, `Procfile`, `api`, `cookie.txt`, `curl.txt`, `models`, `package.json`, `public`, `readme.md`, `routes`, `server.js`, and `tests`.

```
Azats-Air:hackhall azat$ ls -lah
total 64
drwxr-xr-x  16 azat  staff  544B Sep 28 13:55 .
drwxr-xr-x  18 azat  staff  612B Sep 28 14:17 ..
drwxr-xr-x  14 azat  staff  476B Sep 28 14:07 .git
-rw-r--r--   1 azat  staff   35B Sep 28 13:55 .gitignore
-rw-r--r--   1 azat  staff  1.1K Sep 28 13:55 Makefile
-rw-r--r--   1 azat  staff   19B Sep 28 13:55 Procfile
drwxr-xr-x   3 azat  staff  102B Sep 28 13:55 api
-rw-r--r--   1 azat  staff  246B Sep 28 13:55 cookie.txt
-rw-r--r--   1 azat  staff  722B Sep 28 13:55 curl.txt
drwxr-xr-x   3 azat  staff  102B Sep 28 13:55 models
-rw-r--r--   1 azat  staff  373B Sep 28 13:55 package.json
drwxr-xr-x   9 azat  staff  306B Sep 28 13:55 public
-rw-r--r--   1 azat  staff  476B Sep 28 13:55 readme.md
drwxr-xr-x   8 azat  staff  272B Sep 28 13:55 routes
-rw-r--r--   1 azat  staff  3.6K Sep 28 13:55 server.js
drwxr-xr-x   5 azat  staff  170B Sep 28 13:55 tests
Azats-Air:hackhall azat$
```

## Express.js App

Let's jump straight to the `server.js` file and learn how it's implemented. Firstly, we declare dependencies:

```
var express = require('express'),
    routes = require('./routes'),
    http = require('http'),
    util = require('util'),
    oauth = require('oauth'),
    querystring = require('querystring');
```

is populated by Heroku, and in case of a local setup fallback on 3000.

```
var app = express();
app.configure(function(){
  app.set('port', process.env.PORT || 3000 );
  app.use(express.favicon());
  app.use(express.logger('dev'));
  app.use(express.bodyParser());
  app.use(express.methodOverride());
```

The values passed to cookieParser and session middlewares are needed for authentication. Obviously, session secrets are supposed to be private:

```
app.use(express.cookieParser('asd;lfkajs;ldfkj'));
app.use(express.session({
  secret: '<h1>WHEEYEEE</h1>',
  key: 'sid',
  cookie: {
    secret: true,
    expires: false
  }
}));
```

This is how we serve our front-end client Backbone.js app and other static files like CSS:

```
app.use(express.static(__dirname + '/public'));
app.use(app.router);
});
```

handler dedicated to AJAX/XHR requests from the Backbone.js app (responds with JSON):

```
app.configure(function() {
  app.use(logErrors);
  app.use(clientErrorHandler);
  app.use(errorHandler);
});

function logErrors(err, req, res, next) {
  console.error('logErrors', err.toString());
  next(err);
}

function clientErrorHandler(err, req, res, next) {
  console.error('clientErrors ', err.toString());
  res.send(500, { error: err.toString()});
  if (req.xhr) {
    console.error(err);
    res.send(500, { error: err.toString()});
  } else {
    next(err);
  }
}

function errorHandler(err, req, res, next) {
  console.error('lastErrors ', err.toString());
  res.send(500, {error: err.toString()});
}
```

In the same way that we determine `process.env.PORT` and fallback on local setup value 3000, we do a similar thing with MongoDB connection string:

```
var dbUrl = process.env.MONGOHQ_URL
  || 'mongodb://@127.0.0.1:27017/hackhall';
var mongoose = require('mongoose');
var connection = mongoose.createConnection(dbUrl);
connection.on('error', console.error.bind(console,
  'connection error:'));;
```

Sometime it's a good idea to log on the connection open event:

```
connection.once('open', function () {
  console.info('connected to database')
});
```

The Mongoose models live in the `models` folder:

```
var models = require('./models');
```

This middleware will provide access to two collections within our routes methods:

```
function db (req, res, next) {
  req.db = {
    User: connection.model('User', models.User, 'users'),
    Post: connection.model('Post', models.Post, 'posts')
  };
  return next();
}
```



```
checkUser = routes.main.checkUser;
checkAdmin = routes.main.checkAdmin;
checkApplicant = routes.main.checkApplicant;
```

## AngelList OAuth routes:

```
app.get('/auth/angellist', routes.auth.angellist);
app.get('/auth/angellist/callback',
  routes.auth.angellistCallback,
  routes.auth.angellistLogin,
  db,
  routes.users.findOrAddUser);
```

Main application routes including api/profile return user session if user is logged in:

```
//MAIN
app.get('/api/profile', checkUser, db, routes.main.profile);
app.del('/api/profile', checkUser, db, routes.main.delProfile);
app.post('/api/login', db, routes.main.login);
app.post('/api/logout', routes.main.logout);
```

The POST requests for creating users and posts:

```
//POSTS
app.get('/api/posts', checkUser, db, routes.posts.getPosts);
app.post('/api/posts', checkUser, db, routes.posts.add);
```

```
app.put('/api/posts/:id', checkUser, db, routes.posts.updatePost);
app.del('/api/posts/:id', checkUser, db, routes.posts.del);

//USERS
app.get('/api/users', checkUser, db, routes.users.getUsers);
app.get('/api/users/:id', checkUser, db, routes.users.getUser);
app.post('/api/users', checkAdmin, db, routes.users.add);
app.put('/api/users/:id', checkAdmin, db, routes.users.update);
app.del('/api/users/:id', checkAdmin, db, routes.users.del);
```

These routes are for new members that haven't been approved yet:

```
//APPLICATION
app.post('/api/application',
  checkAdmin,
  db,
  routes.application.add);
app.put('/api/application',
  checkApplicant,
  db,
  routes.application.update);
app.get('/api/application',
  checkApplicant,
  db,
  routes.application.get);
```

The catch all else route:

```
app.get('*', function(req, res){
  res.send(404);
});
```

being executed as a stand alone or as an imported module:

```
http.createServer(app);
if (require.main === module) {
  app.listen(app.get('port'), function(){
    console.info('Express server listening on port '
      + app.get('port'));
  });
}
else {
  console.info('Running app as a module')
  exports.app = app;
}
```

The full source code for `hackhall/server.js`:

```
var express = require('express'),
    routes = require('./routes'),
    http = require('http'),
    util = require('util'),
    oauth = require('oauth'),
    querystring = require('querystring');

var app = express();
app.configure(function(){
  app.set('port', process.env.PORT || 3000 );
  app.use(express.favicon());
  app.use(express.logger('dev'));
  app.use(express.bodyParser());
  app.use(express.methodOverride());
  app.use(express.cookieParser('asd;lfkajs;ldfkj'));
  app.use(express.session({
    secret: '<h1>WHEEYEEE</h1>',
```

```
    cookie: {
      secret: true,
      expires: false
    }
  }));
// app.use(express.csrf());
// app.use(function(req, res, next) {
//   res.locals.csrf = req.session._cstf;
//   return next();
// });
app.use(express.static(__dirname + '/public'));
app.use(app.router);
});

app.configure(function() {
  app.use(logErrors);
  app.use(clientErrorHandler);
  app.use(errorHandler);
});

function logErrors(err, req, res, next) {
  console.error('logErrors', err.toString());
  next(err);
}

function clientErrorHandler(err, req, res, next) {
  console.error('clientErrors ', err.toString());
  res.send(500, { error: err.toString()});
  if (req.xhr) {
    console.error(err);
    res.send(500, { error: err.toString()});
  } else {
    next(err);
  }
}

function errorHandler(err, req, res, next) {
  console.error('lastErrors ', err.toString());
```

```
}

var dbUrl = process.env.MONGOHQ_URL || 'mongodb://@127.0.0.1:27017/hackhall';
var mongoose = require('mongoose');
var connection = mongoose.createConnection(dbUrl);
connection.on('error', console.error.bind(console, 'connection error:'));
connection.once('open', function () {
  console.info('connected to database')
});

var models = require('./models');
function db (req, res, next) {
  req.db = {
    User: connection.model('User', models.User, 'users'),
    Post: connection.model('Post', models.Post, 'posts')
  };
  return next();
}

checkUser = routes.main.checkUser;
checkAdmin = routes.main.checkAdmin;
checkApplicant = routes.main.checkApplicant;

app.get('/auth/angellist', routes.auth.angellist);
app.get('/auth/angellist/callback',
  routes.auth.angellistCallback,
  routes.auth.angellistLogin,
  db,
  routes.users.findOrAddUser);

//MAIN
app.get('/api/profile', checkUser, db, routes.main.profile);
app.del('/api/profile', checkUser, db, routes.main.delProfile);
app.post('/api/login', db, routes.main.login);
app.post('/api/logout', routes.main.logout);

//POSTS
app.get('/api/posts', checkUser, db, routes.posts.getPosts);
app.post('/api/posts', checkUser, db, routes.posts.add);
```

```
app.put('/api/posts/:id', checkUser, db, routes.posts.updatePost);
app.del('/api/posts/:id', checkUser, db, routes.posts.del);

//USERS
app.get('/api/users', checkUser, db, routes.users.getUsers);
app.get('/api/users/:id', checkUser, db, routes.users.getUser);
app.post('/api/users', checkAdmin, db, routes.users.add);
app.put('/api/users/:id', checkAdmin, db, routes.users.update);
app.del('/api/users/:id', checkAdmin, db, routes.users.del);

//APPLICATION
app.post('/api/application', checkAdmin, db, routes.application.add);
app.put('/api/application', checkApplicant, db, routes.application.update);
app.get('/api/application', checkApplicant, db, routes.application.get);

app.get('*', function(req, res){
  res.send(404);
});

http.createServer(app);
if (require.main === module) {
  app.listen(app.get('port'), function(){
    console.info('Express server listening on port ' + app.get('port'));
  });
}
else {
  console.info('Running app as a module')
  exports.app = app;
}
```

## Routes

The HackHall routes reside in `hackhall/routes` folder and are groped into

- `hackhall/routes/index.js`: bridge between `server.js` and other routes in the folder
- `hackhall/routes/auth.js`: routes that handle OAuth dance with AngelList API
- `hackhall/routes/main.js`: login, logout and other routes
- `hackhall/routes/users.js`: routes related to users REST API
- `hackhall/routes/application.js`: submission of application to become a user
- `hackhall/routes/posts.js`: routes related to posts REST API

### `index.js`

Let's peak into `hackhall/routes/index.js` where we include other modules:

```
exports.posts = require('./posts');
exports.main = require('./main');
exports.users = require('./users');
exports.application = require('./application');
exports.auth = require('./auth');
```

### `auth.js`

In this module, we'll handle OAuth *dance* with AngelList API. To do so, we'll have to rely on `https` library:

```
var https = require('https');
```

website and stored in environment variables:

```
var angelListClientId = process.env.ANGELLIST_CLIENT_ID;
var angelListClientSecret = process.env.ANGELLIST_CLIENT_SECRET;
```

The method will redirect users to angel.co website for authentication:

```
exports.angelList = function(req, res) {
  res.redirect('https://angel.co/api/oauth/authorize?client_id=' + angelListClientId + '&sc
}
```

After users allow our app to access their information, AngelList sends them back to this route for us to make a new (HTTPS) request to retrieve the token:

```
exports.angelListCallback = function(req, res, next) {
  var token;
  var buf = '';
  var data;
  // console.log('/api/oauth/token?client_id='
  //+ angelListClientId
  //+ '&client_secret='
  //+ angelListClientSecret
  //+ '&code='
  //+ req.query.code
  //+ '&grant_type=authorization_code');
  var angelReq = https.request({
    host: 'angel.co',
    path: '/api/oauth/token?client_id='
      + angelListClientId
```



```
+ angelListClientSecret
+ '&code='
+ req.query.code
+ '&grant_type=authorization_code',
port: 443,
method: 'POST',
headers: {
  'content-length': 0
},
},
function(angelRes) {
  angelRes.on('data', function(buffer) {
    buf += buffer;
  });
  angelRes.on('end', function() {
    try {
      data = JSON.parse(buf.toString('utf-8'));
    } catch (e) {
      if (e) return res.send(e);
    }
    if (!data || !data.access_token) return res.send(500);
    token = data.access_token;
    req.session.angelListAccessToken = token;
    if (token) next();
    else res.send(500);
  });
});
angelReq.end();
angelReq.on('error', function(e) {
  console.error(e);
  next(e);
});
}
```

Directly call AngleList API with the token from the previous middleware to get user information:

```
exports.angelListLogin = function(req, res, next) {
  token = req.session.angelListAccessToken;
  httpsRequest = https.request({
    host: 'api.angel.co',
    path: '/1/me?access_token=' + token,
    port: 443,
    method: 'GET'
  },
  function(httpsResponse) {
    httpsResponse.on('data', function(buffer) {
      data = JSON.parse(buffer.toString('utf-8'));
      if (data) {
        req.angelProfile = data;
        next();
      }
    });
  }
);
httpsRequest.end();
httpsRequest.on('error', function(e) {
  console.error(e);
});
};
```

The full source code for `hackhall/routes/auth.js` files:

```
var https = require('https');

var angelListClientId = process.env.ANGELLIST_CLIENT_ID;
var angelListClientSecret = process.env.ANGELLIST_CLIENT_SECRET;

exports.angelList = function(req, res) {
  res.redirect('https://angel.co/api/oauth/authorize?client_id=' + angelListClientId + '&sc
}

exports.angelListCallback = function(req, res, next) {
```

```
var buf = '';
var data;
// console.log('/api/oauth/token?client_id=' + angelListClientId + '&client_secret=' + ar
var angelReq = https.request({
  host: 'angel.co',
  path: '/api/oauth/token?client_id=' + angelListClientId + '&client_secret=' + angeli
  port: 443,
  method: 'POST',
  headers: {
    'content-length': 0
  }
},
function(angelRes) {

  angelRes.on('data', function(buffer) {
    buf += buffer;
  });
  angelRes.on('end', function() {
    try {
      data = JSON.parse(buf.toString('utf-8'));
    } catch (e) {
      if (e) return res.send(e);
    }
    if (!data || !data.access_token) return res.send(500);
    token = data.access_token;
    req.session.angelListAccessToken = token;
    if (token) next();
    else res.send(500);
  });
});
angelReq.end();
angelReq.on('error', function(e) {
  console.error(e);
  next(e);
});
}
exports.angelListLogin = function(req, res, next) {
  token = req.session.angelListAccessToken;
```

```
host: 'api.angel.co',
path: '/1/me?access_token=' + token,
port: 443,
method: 'GET'
},
function(httpsResponse) {
  httpsResponse.on('data', function(buffer) {
    data = JSON.parse(buffer.toString('utf-8'));
    if (data) {
      req.angelProfile = data;
      next();
    }
  });
}
);
httpsRequest.end();
httpsRequest.on('error', function(e) {
  console.error(e);
});
};
```

## main.js

The `hackhall/routes/main.js` file might be interesting as well.

The `checkAdmin()` function performs authentication for admin privileges. If the session object doesn't carry proper flag, we call Express.js `next()` function with an error object:

```
exports.checkAdmin = function(request, response, next) {
  if (request.session
    && request.session.auth
    && request.session.userId
    && request.session.admin) {
```

```
    return next();  
  } else {  
    next('User is not an administrator.');  }  
};
```

Similarly, we can check only for the user without checking for admin rights:

```
exports.checkUser = function(req, res, next) {  
  if (req.session && req.session.auth && req.session.userId  
    && (req.session.user.approved || req.session.admin)) {  
    console.info('Access USER: ' + req.session.userId);  
    return next();  
  } else {  
    next('User is not logged in.');  }  
};
```

Application is just an unapproved user object and we can also check for that:

```
exports.checkApplicant = function(req, res, next) {  
  if (req.session && req.session.auth && req.session.userId  
    && (!req.session.user.approved || req.session.admin)) {  
    console.info('Access USER: ' + req.session.userId);  
    return next();  
  } else {  
    next('User is not logged in.');  }  
};
```

database. On success, we store the user object in the session and proceed; otherwise the request fails:

```
exports.login = function(req, res, next) {
  req.db.User.findOne({
    email: req.body.email,
    password: req.body.password
  },
  null, {
    safe: true
  },
  function(err, user) {
    if (err) return next(err);
    if (user) {
      req.session.auth = true;
      req.session.userId = user._id.toHexString();
      req.session.user = user;
      if (user.admin) {
        req.session.admin = true;
      }
      console.info('Login USER: ' + req.session.userId);
      res.json(200, {
        msg: 'Authorized'
      });
    } else {
      next(new Error('User is not found.'));
    }
  });
};
```

The logging out process removes any session information:

```
exports.logout = function(req, res) {
```

```
req.session.destroy(function(error) {  
  if (!error) {  
    res.send({  
      msg: 'Logged out'  
    });  
  }  
});  
};
```

This route is used for both the Profile page as well as by Backbone.js for user authentication:

```
exports.profile = function(req, res, next) {  
  req.db.User.findById(req.session.userId, 'firstName lastName'  
    + 'displayName headline photoUrl admin'  
    + 'approved banned role angelUrl twitterUrl'  
    + 'facebookUrl linkedinUrl githubUrl', function(err, obj) {  
    if (err) next(err);  
    if (!obj) next(new Error('User is not found'));  
    req.db.Post.find({  
      author: {  
        id: obj._id,  
        name: obj.displayName  
      }  
    }, null, {  
      sort: {  
        'created': -1  
      }  
    }, function(err, list) {  
      if (err) next(err);  
      obj.posts.own = list || [];  
      req.db.Post.find({  
        likes: obj._id  
      }, null, {  
        sort: {
```

```
}
```

This logic finds Posts and Comments made by the user:

```
    }, function(err, list) {
      if (err) next(err);
      obj.posts.likes = list || [];
      req.db.Post.find({
        watches: obj._id
      }, null, {
        sort: {
          'created': -1
        }
      }, function(err, list) {
        if (err) next(err);
        obj.posts.watches = list || [];
        req.db.Post.find({
          'comments.author.id': obj._id
        }, null, {
          sort: {
            'created': -1
          }
        }, function(err, list) {
          if (err) next(err);
          obj.posts.comments = [];
          list.forEach(function(value, key, list) {
            obj.posts.comments.push(
              value.comments.filter(
                function(el, i, arr) {
                  return (el.author.id.toString() == obj._id.toString());
                }
              )
            );
          });
        });
      });
      res.json(200, obj);
    });
```



```
});  
});  
});  
});  
};
```

It's important to allow users to delete their profiles:

```
exports.delProfile = function(req, res, next) {  
  console.log('del profile');  
  console.log(req.session.userId);  
  req.db.User.findByIdAndRemove(req.session.user._id, {},  
    function(err, obj) {  
      if (err) next(err);  
      req.session.destroy(function(error) {  
        if (err) {  
          next(err)  
        }  
      });  
      res.json(200, obj);  
    }  
  );  
};
```

The full source code of `hackhall/routes/main.js` files:

```
exports.checkAdmin = function(request, response, next) {  
  if (request.session && request.session.auth && request.session.userId && request.session.  
    console.info('Access ADMIN: ' + request.session.userId);  
    return next();  
  } else {  
    next('User is not an administrator.');
```

```
};
```

```
exports.checkUser = function(req, res, next) {  
  if (req.session && req.session.auth && req.session.userId && (req.session.user.approved |  
    console.info('Access USER: ' + req.session.userId);  
    return next();  
  } else {  
    next('User is not logged in.');
```

```
};  
  
exports.checkApplicant = function(req, res, next) {  
  if (req.session && req.session.auth && req.session.userId && (!req.session.user.approved  
    console.info('Access USER: ' + req.session.userId);  
    return next();  
  } else {  
    next('User is not logged in.');
```

```
};  
  
exports.login = function(req, res, next) {  
  req.db.User.findOne({  
    email: req.body.email,  
    password: req.body.password  
  },  
  null, {  
    safe: true  
  },  
  function(err, user) {  
    if (err) return next(err);  
    if (user) {  
      req.session.auth = true;  
      req.session.userId = user._id.toHexString();  
      req.session.user = user;  
      if (user.admin) {  
        req.session.admin = true;  
      }  
      console.info('Login USER: ' + req.session.userId);
```

```
      msg: 'Authorized'
    });
  } else {
    next(new Error('User is not found.'));
  }
});
};

exports.logout = function(req, res) {
  console.info('Logout USER: ' + req.session.userId);
  req.session.destroy(function(error) {
    if (!error) {
      res.send({
        msg: 'Logged out'
      });
    }
  });
};

exports.profile = function(req, res, next) {
  req.db.User.findById(req.session.userId, 'firstName lastName displayName headline photoUr
  if (err) next(err);
  if (!obj) next(new Error('User is not found'));
  req.db.Post.find({
    author: {
      id: obj._id,
      name: obj.displayName
    }
  }, null, {
    sort: {
      'created': -1
    }
  }, function(err, list) {
    if (err) next(err);
    obj.posts.own = list || [];
    req.db.Post.find({
      likes: obj._id
    }, null, {
```

```
    'created': -1
  }
}, function(err, list) {
  if (err) next(err);
  obj.posts.likes = list || [];
  req.db.Post.find({
    watches: obj._id
  }, null, {
    sort: {
      'created': -1
    }
  }, function(err, list) {
    if (err) next(err);
    obj.posts.watches = list || [];
    req.db.Post.find({
      'comments.author.id': obj._id
    }, null, {
      sort: {
        'created': -1
      }
    }, function(err, list) {
      if (err) next(err);
      obj.posts.comments = [];
      list.forEach(function(value, key, list) {
        obj.posts.comments.push(value.comments.filter(function(el, i, arr) {
          return (el.author.id.toString() == obj._id.toString());
        }));
      });
      res.json(200, obj);
    });
  });
});
});
});
});

exports.delProfile = function(req, res, next) {
  console.log('del profile');
```

```
req.db.User.findByIdAndRemove(req.session.user._id, {}, function(err, obj) {
  if (err) next(err);
  req.session.destroy(function(error) {
    if (err) {
      next(err)
    }
  });
  res.json(200, obj);
});
```

## users.js

The full source code for `hackhall/routes/users.js` files:

```
objectId = require('mongodb').ObjectID;

exports.getUsers = function(req, res, next) {
  if (req.session.auth && req.session.userId) {
    req.db.User.find({}, 'firstName lastName displayName headline photoUrl admin approved t
    if (err) next(err);
    res.json(200, list);
  });
} else {
  next('User is not recognized.')
}
}

exports.getUser = function(req, res, next) {
  req.db.User.findById(req.params.id, 'firstName lastName displayName headline photoUrl adn
  if (err) next(err);
  if (!obj) next(new Error('User is not found'));
  req.db.Post.find({
    author: {
      id: obj._id,
```

```
    }, null, {
      sort: {
        'created': -1
      }
    }, function(err, list) {
      if (err) next(err);
      obj.posts.own = list || [];
      req.db.Post.find({
        likes: obj._id
      }, null, {
        sort: {
          'created': -1
        }
      }, function(err, list) {
        if (err) next(err);
        obj.posts.likes = list || [];
        req.db.Post.find({
          watches: obj._id
        }, null, {
          sort: {
            'created': -1
          }
        }, function(err, list) {
          if (err) next(err);
          obj.posts.watches = list || [];
          req.db.Post.find({
            'comments.author.id': obj._id
          }, null, {
            sort: {
              'created': -1
            }
          }, function(err, list) {
            if (err) next(err);
            obj.posts.comments = [];
            list.forEach(function(value, key, list) {
              obj.posts.comments.push(value.comments.filter(function(el, i, arr) {
                return (el.author.id.toString() == obj._id.toString());
              }));
            });
          });
        });
      });
    });
  });
}
```

```
    });
    res.json(200, obj);
  });
});
});
});
});
};

exports.add = function(req, res, next) {
  var user = new req.db.User(req.body);
  user.save(function(err) {
    if (err) next(err);
    res.json(user);
  });
};

exports.update = function(req, res, next) {
  var obj = req.body;
  obj.updated = new Date();
  delete obj._id;
  req.db.User.findByIdAndUpdate(req.params.id, {
    $set: obj
  }, {
    new: true
  }, function(err, obj) {
    if (err) next(err);
    res.json(200, obj);
  });
};

exports.del = function(req, res, next) {
  req.db.User.findByIdAndRemove(req.params.id, function(err, obj) {
    if (err) next(err);
    res.json(200, obj);
  });
};
```

```
data = req.angelProfile;
req.db.User.findOne({
  angellistId: data.id
}, function(err, obj) {
  console.log('angellistLogin4');
  if (err) next(err);
  console.warn(obj);
  if (!obj) {
    req.db.User.create({
      angellistId: data.id,
      angelToken: token,
      angellistProfile: data,
      email: data.email,
      firstName: data.name.split(' ')[0],
      lastName: data.name.split(' ')[1],
      displayName: data.name,
      headline: data.bio,
      photoUrl: data.image,
      angelUrl: data.angellist_url,
      twitterUrl: data.twitter_url,
      facebookUrl: data.facebook_url,
      linkedinUrl: data.linkedin_url,
      githubUrl: data.github_url
    }, function(err, obj) { //remember the scope of variables!
      if (err) next(err);
      console.log(obj);
      req.session.auth = true;
      req.session.userId = obj._id;
      req.session.user = obj;
      req.session.admin = false; //assing regular user role by default
      res.redirect('/#application');
      // }
    });
  } else { //user is in the database
    req.session.auth = true;
    req.session.userId = obj._id;
    req.session.user = obj;
    req.session.admin = obj.admin; //false; //assing regular user role by default
```



```
    res.redirect('/#posts');
  } else {
    res.redirect('/#application');
  }
}
}))
}
```

## applications.js

In the current version, submitting and approving an application won't trigger an email notification. Therefore, users have to come back to the website to check their status.

Merely add a user object (with approved=false by default) to the database:

```
exports.add = function(req, res, next) {
  req.db.User.create({
    firstName: req.body.firstName,
    lastName: req.body.lastName,
    displayName: req.body.displayName,
    headline: req.body.headline,
    photoUrl: req.body.photoUrl,
    password: req.body.password,
    email: req.body.email,
    angellist: {
      blah: 'blah'
    },
    angelUrl: req.body.angelUrl,
    twitterUrl: req.body.twitterUrl,
    facebookUrl: req.body.facebookUrl,
    linkedinUrl: req.body.linkedinUrl,
    githubUrl: req.body.githubUrl
  }, function(err, obj) {
    if (err) next(err);
```

```
res.json(200, obj);  
  })  
};
```

Let the users update information in their applications:

```
exports.update = function(req, res, next) {  
  var data = {};  
  Object.keys(req.body).forEach(function(k) {  
    if (req.body[k]) {  
      data[k] = req.body[k];  
    }  
  });  
  delete data._id;  
  req.db.User.findByIdAndUpdate(req.session.user._id, {  
    $set: data  
  }, function(err, obj) {  
    if (err) next(err);  
    if (!obj) next('Cannot save.')
```

```
    res.json(200, obj);  
  });  
};
```

Select a particular object with the `get()` function:

```
exports.get = function(req, res, next) {  
  req.db.User.findById(req.session.user._id,  
    'firstName lastName photoUrl headline displayName'  
    + 'angelUrl facebookUrl twitterUrl linkedinUrl'  
    + 'githubUrl', {}, function(err, obj) {  
    if (err) next(err);  
    if (!obj) next('cannot find');
```

```
}  
};
```

The full source code of `hackhall/routes/applications.js` files:

```
***Error:** File "applications.js" does not exist at this path*
```

### **posts.js**

The last routes module that we bisect is `hackhall/routes/posts.js`. It takes care of adding, editing and removing posts, as well as commenting, watching and liking.

We use object ID for conversion from HEX strings to proper objects:

```
objectId = require('mongodb').ObjectID;
```

The coloring is nice for logging but of course optional. We accomplish it with escape sequences:

```
var red, blue, reset;  
red   = '\u001b[31m';  
blue  = '\u001b[34m';  
reset = '\u001b[0m';  
console.log(red + 'This is red' + reset + ' while ' + blue + ' this is blue' + reset);
```

The default values for the pagination of posts:

```
var LIMIT = 10;
var SKIP = 0;
```

The `add()` function handles creation of new posts:

```
exports.add = function(req, res, next) {
  if (req.body) {
    req.db.Post.create({
      title: req.body.title,
      text: req.body.text || null,
      url: req.body.url || null,
      author: {
        id: req.session.user._id,
        name: req.session.user.displayName
      }
    }, function(err, docs) {
      if (err) {
        console.error(err);
        next(err);
      } else {
        res.json(200, docs);
      }
    });
  } else {
    next(new Error('No data'));
  }
};
```

To retrieve the list of posts:

```
var limit = req.query.limit || LIMIT;
var skip = req.query.skip || SKIP;
req.db.Post.find({}, null, {
  limit: limit,
  skip: skip,
  sort: {
    '_id': -1
  }
}, function(err, obj) {
  if (!obj) next('There are not posts.');
```

---

```
  obj.forEach(function(item, i, list) {
    if (req.session.user.admin) {
      item.admin = true;
    } else {
      item.admin = false;
    }
    if (item.author.id == req.session.userId) {
      item.own = true;
    } else {
      item.own = false;
    }
    if (item.likes
      && item.likes.indexOf(req.session.userId) > -1) {
      item.like = true;
    } else {
      item.like = false;
    }
    if (item.watches
      && item.watches.indexOf(req.session.userId) > -1) {
      item.watch = true;
    } else {
      item.watch = false;
    }
  });
  var body = {};
  body.limit = limit;
  body.skip = skip;
  body.posts = obj;
```

```
    if (err) next(err);  
    body.total = total;  
    res.json(200, body);  
  });  
});  
};
```

For the individual post page, we need the `getPost()` method:

```
exports.getPost = function(req, res, next) {  
  if (req.params.id) {  
    req.db.Post.findById(req.params.id, {  
      title: true,  
      text: true,  
      url: true,  
      author: true,  
      comments: true,  
      watches: true,  
      likes: true  
    }, function(err, obj) {  
      if (err) next(err);  
      if (!obj) {  
        next('Nothing is found.');      } else {  
        res.json(200, obj);  
      }  
    });  
  } else {  
    next('No post id');  }  
};
```

The `del()` function removes specific posts from the database. The `find-`

However, the same thing can be accomplished with just `remove()`.

```
exports.del = function(req, res, next) {
  req.db.Post.findById(req.params.id, function(err, obj) {
    if (err) next(err);
    if (req.session.admin || req.session.userId === obj.author.id) {
      obj.remove();
      res.json(200, obj);
    } else {
      next('User is not authorized to delete post.');
```

To like the post, we update the post item by prepending `post.likes` array with the ID of the user:

```
function likePost(req, res, next) {
  req.db.Post.findByIdAndUpdate(req.body._id, {
    $push: {
      likes: req.session.userId
    }
  }, {}, function(err, obj) {
    if (err) {
      next(err);
    } else {
      res.json(200, obj);
    }
  });
};
```

Likewise, when a user performs the watch action, the system adds a new

```
function watchPost(req, res, next) {
  req.db.Post.findByIdAndUpdate(req.body._id, {
    $push: {
      watches: req.session.userId
    }
  }, {}, function(err, obj) {
    if (err) next(err);
    else {
      res.json(200, obj);
    }
  });
};
```

The `updatePost()` is what calls like or watch functions based on the action flag sent with request. In addition, the `updatePost()` processes the changes to the post and comments:

```
exports.updatePost = function(req, res, next) {
  var anyAction = false;
  if (req.body._id && req.params.id) {
    if (req.body && req.body.action == 'like') {
      anyAction = true;
      likePost(req, res);
    }
    if (req.body && req.body.action == 'watch') {
      anyAction = true;
      watchPost(req, res);
    }
    if (req.body && req.body.action == 'comment'
      && req.body.comment && req.params.id) {
      anyAction = true;
    }
  }
};
```



```
$push: {
  comments: {
    author: {
      id: req.session.userId,
      name: req.session.user.displayName
    },
    text: req.body.comment
  }
}
}, {
  safe: true,
  new: true
}, function(err, obj) {
  if (err) throw err;
  res.json(200, obj);
});
}

if (req.session.auth && req.session.userId && req.body
  && req.body.action !== 'comment' &&
  req.body.action !== 'watch' && req.body !== 'like' &&
  req.params.id && (req.body.author.id === req.session.user._id
  || req.session.user.admin)) {
  req.db.Post.findById(req.params.id, function(err, doc) {
    if (err) next(err);
    doc.title = req.body.title;
    doc.text = req.body.text || null;
    doc.url = req.body.url || null;
    doc.save(function(e, d) {
      if (e) next(e);
      res.json(200, d);
    });
  })
} else {
  if (!anyAction) next('Something went wrong.');
```

```
}

} else {
  next('No post ID.');
```

```
};
```

The full source code for the `hackhall/routes/posts.js` file:

```
objectId = require('mongodb').ObjectID;
var red, blue, reset;
red = '\u001b[31m';
blue = '\u001b[34m';
reset = '\u001b[0m';
console.log(red + 'This is red' + reset + ' while ' + blue + ' this is blue' + reset);

var LIMIT = 10;
var SKIP = 0;

exports.add = function(req, res, next) {
  if (req.body) {
    req.db.Post.create({
      title: req.body.title,
      text: req.body.text || null,
      url: req.body.url || null,
      author: {
        id: req.session.user._id,
        name: req.session.user.displayName
      }
    }, function(err, docs) {
      if (err) {
        console.error(err);
        next(err);
      } else {
        res.json(200, docs);
      }
    });
  } else {
    next(new Error('No data'));
  }
}
```

```
};
```

```
exports.getPosts = function(req, res, next) {
  var limit = req.query.limit || LIMIT;
  var skip = req.query.skip || SKIP;
  req.db.Post.find({}, null, {
    limit: limit,
    skip: skip,
    sort: {
      '_id': -1
    }
  }, function(err, obj) {
    if (!obj) next('There are not posts.');
```

```
    obj.forEach(function(item, i, list) {
      if (req.session.user.admin) {
        item.admin = true;
      } else {
        item.admin = false;
      }
      if (item.author.id == req.session.userId) {
        item.own = true;
      } else {
        item.own = false;
      }
      if (item.likes && item.likes.indexOf(req.session.userId) > -1) {
        item.like = true;
      } else {
        item.like = false;
      }
      if (item.watches && item.watches.indexOf(req.session.userId) > -1) {
        item.watch = true;
      } else {
        item.watch = false;
      }
    });
    var body = {};
    body.limit = limit;
    body.skip = skip;
```

```
req.db.Post.count({}, function(err, total) {
  if (err) next(err);
  body.total = total;
  res.json(200, body);
});
});
};

exports.getPost = function(req, res, next) {
  if (req.params.id) {
    req.db.Post.findById(req.params.id, {
      title: true,
      text: true,
      url: true,
      author: true,
      comments: true,
      watches: true,
      likes: true
    }, function(err, obj) {
      if (err) next(err);
      if (!obj) {
        next('Nothing is found.');
```

---

```
      } else {
        res.json(200, obj);
      }
    });
  } else {
    next('No post id');
  }
};

exports.del = function(req, res, next) {
  req.db.Post.findById(req.params.id, function(err, obj) {
    if (err) next(err);
    if (req.session.admin || req.session.userId === obj.author.id) {
      obj.remove();
      res.json(200, obj);
    }
  });
};
```

```
    next('User is not authorized to delete post.');
```

---

```
  }  
  })  
};
```

```
function likePost(req, res, next) {  
  req.db.Post.findByIdAndUpdate(req.body._id, {  
    $push: {  
      likes: req.session.userId  
    }  
  }, {}, function(err, obj) {  
    if (err) {  
      next(err);  
    } else {  
      res.json(200, obj);  
    }  
  });  
};
```

```
function watchPost(req, res, next) {  
  req.db.Post.findByIdAndUpdate(req.body._id, {  
    $push: {  
      watches: req.session.userId  
    }  
  }, {}, function(err, obj) {  
    if (err) next(err);  
    else {  
      res.json(200, obj);  
    }  
  });  
};
```

```
exports.updatePost = function(req, res, next) {  
  var anyAction = false;  
  if (req.body._id && req.params.id) {  
    if (req.body && req.body.action == 'like') {  
      anyAction = true;  
      likePost(req, res);  
    }  
  }  
}
```

```
if (req.body && req.body.action == 'watch') {
  anyAction = true;
  watchPost(req, res);
}

if (req.body && req.body.action == 'comment' && req.body.comment && req.params.id) {
  anyAction = true;
  req.db.Post.findByIdAndUpdate(req.params.id, {
    $push: {
      comments: {
        author: {
          id: req.session.userId,
          name: req.session.user.displayName
        },
        text: req.body.comment
      }
    }
  }, {
    safe: true,
    new: true
  }, function(err, obj) {
    if (err) throw err;
    res.json(200, obj);
  });
}

if (req.session.auth && req.session.userId && req.body && req.body.action != 'comment'
  req.body.action != 'watch' && req.body != 'like' &&
  req.params.id && (req.body.author.id == req.session.user._id || req.session.user.admin)
  req.db.Post.findById(req.params.id, function(err, doc) {
    if (err) next(err);
    doc.title = req.body.title;
    doc.text = req.body.text || null;
    doc.url = req.body.url || null;
    doc.save(function(e, d) {
      if (e) next(e);
      res.json(200, d);
    });
  })
} else {
```

```
}  
  
    } else {  
      next('No post ID.');    }  
  };  
};
```

## Mongoose Models

Ideally, in a big application, we would break down each model into a separate file. Right now in the HackHall app, we have them all in `hackhall/models/index.js`.

As always, our dependencies look better at the top:

```
var mongoose = require('mongoose');  
var Schema = mongoose.Schema;  
var roles = 'user staff mentor investor founder'.split(' ');
```

The Post model represents post with its likes, comments and watches.

```
exports.Post = new Schema ({  
  title: {  
    required: true,  
    type: String,  
    trim: true,  
    // match: /^[[:alpha:][:space:][:punct:]]{1,100}$/  
    match: /^[\\w ,.!~]{1,100}$/  
  },  
  url: {  
    type: String,
```

```
max: 1000
},
text: {
  type: String,
  trim: true,
  max: 2000
},
comments: [{
  text: {
    type: String,
    trim: true,
    max: 2000
  },
  author: {
    id: {
      type: Schema.Types.ObjectId,
      ref: 'User'
    },
    name: String
  }
}],
watches: [{
  type: Schema.Types.ObjectId,
  ref: 'User'
}],
likes: [{
  type: Schema.Types.ObjectId,
  ref: 'User'
}],
author: {
  id: {
    type: Schema.Types.ObjectId,
    ref: 'User',
    required: true
  },
  name: {
    type: String,
    required: true
  }
}
```



```
},
created: {
  type: Date,
  default: Date.now,
  required: true
},
updated: {
  type: Date,
  default: Date.now,
  required: true
},
own: Boolean,
like: Boolean,
watch: Boolean,
admin: Boolean,
action: String
});
```

The User model can also serve as an application object (when `approved=false`):

```
exports.User = new Schema({
  angelListId: String,
  angelListProfile: Schema.Types.Mixed,
  angelToken: String,
  firstName: {
    type: String,
    required: true,
    trim: true
  },
  lastName: {
    type: String,
    required: true,
    trim: true
  },
});
```

```
    type: String,
    required: true,
    trim: true
  },
  password: String,
  email: {
    type: String,
    required: true,
    trim: true
  },
  role: {
    type: String,
    enum: roles,
    required: true,
    default: roles[0]
  },
  approved: {
    type: Boolean,
    default: false
  },
  banned: {
    type: Boolean,
    default: false
  },
  admin: {
    type: Boolean,
    default: false
  },
  headline: String,
  photoUrl: String,
  angellist: Schema.Types.Mixed,
  created: {
    type: Date,
    default: Date.now
  },
  updated: {
    type: Date, default: Date.now
  },
}
```

```
twitterUrl: String,  
facebookUrl: String,  
linkedinUrl: String,  
githubUrl: String,  
own: Boolean,  
posts: {  
  own: [Schema.Types.Mixed],  
  likes: [Schema.Types.Mixed],  
  watches: [Schema.Types.Mixed],  
  comments: [Schema.Types.Mixed]  
}  
});
```

The full source code for `hackhall/models/index.js`:

```
var mongoose = require('mongoose');  
var Schema = mongoose.Schema;  
var roles = 'user staff mentor investor founder'.split(' ');  
  
exports.Post = new Schema ({  
  title: {  
    required: true,  
    type: String,  
    trim: true,  
    // match: /^[[:alpha:][:space:][:punct:]]{1,100}$//  
    match: /^[[:w ,.!?]]{1,100}$//  
  },  
  url: {  
    type: String,  
    trim: true,  
    max: 1000  
  },  
  text: {  
    type: String,  
    trim: true,
```

```
},
comments: [{
  text: {
    type: String,
    trim: true,
    max: 2000
  },
  author: {
    id: {
      type: Schema.Types.ObjectId,
      ref: 'User'
    },
    name: String
  }
}],
watches: [{
  type: Schema.Types.ObjectId,
  ref: 'User'
}],
likes: [{
  type: Schema.Types.ObjectId,
  ref: 'User'
}],
author: {
  id: {
    type: Schema.Types.ObjectId,
    ref: 'User',
    required: true
  },
  name: {
    type: String,
    required: true
  }
},
created: {
  type: Date,
  default: Date.now,
  required: true
}
```

```
    updated: {
      type: Date,
      default: Date.now,
      required: true
    },
    own: Boolean,
    like: Boolean,
    watch: Boolean,
    admin: Boolean,
    action: String
  });

exports.User = new Schema({
  angellistId: String,
  angellistProfile: Schema.Types.Mixed,
  angelToken: String,
  firstName: {
    type: String,
    required: true,
    trim: true
  },
  lastName: {
    type: String,
    required: true,
    trim: true
  },
  displayName: {
    type: String,
    required: true,
    trim: true
  },
  password: String,
  email: {
    type: String,
    required: true,
    trim: true
  },
  role: {
```

```
enum: roles,
  required: true,
  default: roles[0]
},
approved: {
  type: Boolean,
  default: false
},
banned: {
  type: Boolean,
  default: false
},
admin: {
  type: Boolean,
  default: false
},
headline: String,
photoUrl: String,
angelList: Schema.Types.Mixed,
created: {
  type: Date,
  default: Date.now
},
updated: {
  type: Date, default: Date.now
},
angelUrl: String,
twitterUrl: String,
facebookUrl: String,
linkedinUrl: String,
githubUrl: String,
own: Boolean,
posts: {
  own: [Schema.Types.Mixed],
  likes: [Schema.Types.Mixed],
  watches: [Schema.Types.Mixed],
  comments: [Schema.Types.Mixed]
}
```

## Mocha Tests

One of the benefits of using REST API server architecture is that each route and the application as a whole become very testable. The assurance of the passed tests is a wonderful supplement during development — the so called test-driven development approach.

To run tests, we utilize Makefile:

```
REPORTER = list
MOCHA_OPTS = --ui tdd --ignore-leaks

test:
    clear
    echo Starting test *****
    ./node_modules/mocha/bin/mocha \
    --reporter $(REPORTER) \
    $(MOCHA_OPTS) \
    tests/*.js
    echo Ending test

test-w:
    ./node_modules/mocha/bin/mocha \
    --reporter $(REPORTER) \
    --growl \
    --watch \
    $(MOCHA_OPTS) \
    tests/*.js

users:
    mocha tests/users.js --ui tdd --reporter list --ignore-leaks

posts:
```

```
echo Starting test *****
./node_modules/mocha/bin/mocha \
--reporter $(REPORTER) \
$(MOCHA_OPTS) \
tests/posts.js
echo Ending test

application:
  mocha tests/application.js --ui tdd --reporter list --ignore-leaks

.PHONY: test test-w posts application
```

Therefore, we can start tests with: `$ make` command.

All the 20 tests should pass:



```
hackhall — bash
Access USER: 52478fc96eee8397666b9b9f
GET /api/users 200 7ms - 221b
. Test log in check /api/users: 11ms
teardown
  Test log in check /api/posts: setup
Access USER: 52478fc96eee8397666b9b9f
GET /api/posts 200 6ms - 59b
. Test log in check /api/posts: 9ms
teardown
  User control new user POST /api/users: Access USER: 52478fc96eee8397666b9b9f
Login USER: 52478fc96eee8397666b9b9f
POST /api/login 200 19ms - 25b
Access ADMIN: 52478fc96eee8397666b9b9f
POST /api/users 200 5ms - 394b
. User control new user POST /api/users: 29ms
GET /api/profile 200 17ms - 285b
  User control get user list and check for new user GET /api/users: Access USER
: 52478fc96eee8397666b9b9f
GET /api/users 200 5ms - 433b
. User control get user list and check for new user GET /api/users: 6ms
  User control Approve User: PUT /api/users/undefined: Access ADMIN: 52478fc96e
ee8397666b9b9f
PUT /api/users/52478fd7c9351d5e50000003 200 6ms - 393b
Access USER: 52478fc96eee8397666b9b9f
GET /api/users/52478fd7c9351d5e50000003 200 9ms - 277b
. User control Approve User: PUT /api/users/undefined: 20ms
  User control Banned User: PUT /api/users/undefined: Access ADMIN: 52478fc96e
ee8397666b9b9f
PUT /api/users/52478fd7c9351d5e50000003 200 5ms - 392b
Access USER: 52478fc96eee8397666b9b9f
GET /api/users/52478fd7c9351d5e50000003 200 12ms - 276b
. User control Banned User: PUT /api/users/undefined: 20ms
  User control Promote User: PUT /api/users/undefined: Access ADMIN: 52478fc96e
ee8397666b9b9f
PUT /api/users/52478fd7c9351d5e50000003 200 5ms - 391b
Access USER: 52478fc96eee8397666b9b9f
GET /api/users/52478fd7c9351d5e50000003 200 7ms - 275b
. User control Promote User: PUT /api/users/undefined: 16ms
  User control Delete User: DELETE /api/users/:id: Access ADMIN: 52478fc96eee83
97666b9b9f
DELETE /api/users/52478fd7c9351d5e50000003 200 5ms - 391b
Access USER: 52478fc96eee8397666b9b9f
GET /api/users 200 3ms - 221b
. User control Delete User: DELETE /api/users/:id: 12ms

20 passing (359ms)

echo Ending test
Ending test
Azats-Air:hackhall azat$
```

The HackHall tests live in tests folder and consist of:

- hackhall/tests/application.js: functional tests for unapproved users information
- hackhall/tests/posts.js: functional tests for posts

Test use a library called [superagent](#)([GitHub](#)).

The full content for `hackhall/tests/application.js`:

```
var app = require ('../server').app,
    assert = require('assert'),
    request = require('superagent');

app.listen(app.get('port'), function(){
  console.log('Express server listening on port ' + app.get('port'));
});

var user1 = request.agent();
var port = 'http://localhost:'+app.get('port');
var userId;

suite('APPLICATION API', function (){
  suiteSetup(function(done){
    done();
  });
  test('log in as admin', function(done){
    user1.post(port+'/api/login').send({email:'1@1.com',password:'1'}).end(function(res){
      assert.equal(res.status,200);
      done();
    });
  });
  test('get profile for admin',function(done){
    user1.get(port+'/api/profile').end(function(res){
      assert.equal(res.status,200);
      done();
    });
  });
  test('submit applicaton for user 3@3.com', function(done){
    user1.post(port+'/api/application').send({
      firstName: 'Dummy',
```

```
    displayName: 'Dummy Application',
    password: '3',
    email: '3@3.com',
    headline: 'Dummy Appliation',
    photoUrl: '/img/user.png',
    angelList: {blah:'blah'},
    angelUrl: 'http://angel.co.com/someuser',
    twitterUrl: 'http://twitter.com/someuser',
    facebookUrl: 'http://facebook.com/someuser',
    linkedinUrl: 'http://linkedin.com/someuser',
    githubUrl: 'http://github.com/someuser'
  }).end(function(res){
    assert.equal(res.status,200);
    userId = res.body._id;
    done();
  });

});

test('logout admin',function(done){
  user1.post(port+'/api/logout').end(function(res){
    assert.equal(res.status,200);
    done();
  });
});

test('get profile again after logging out',function(done){
  user1.get(port+'/api/profile').end(function(res){
    assert.equal(res.status,500);
    done();
  });
});

test('log in as user3 - unapproved', function(done){
  user1.post(port+'/api/login').send({email:'3@3.com',password:'3'}).end(function(res){
    assert.equal(res.status,200);
    done();
  });
});

test('get user application', function(done){
  user1.get(port+'/api/application/').end(function(res){
```

```
    assert.equal(res.status, 200);
    done();
  });
});
test('update user application', function(done){
  user1.put(port+'/api/application/').send({
    firstName: 'boo'}).end(function(res){
    // console.log(res.body)
    assert.equal(res.status, 200);
    done();
  });
});
test('get user application', function(done){
  user1.get(port+'/api/application/').end(function(res){
    // console.log(res.body)
    assert.equal(res.status, 200);
    done();
  });
});
test('check for posts - fail (unapproved?)', function(done){
  user1.get(port+'/api/posts/').end(function(res){
    // console.log(res.body)
    assert.equal(res.status, 500);

    done();
  });
});
test('logout user',function(done){
  user1.post(port+'/api/logout').end(function(res){
    assert.equal(res.status,200);
    done();
  });
});
test('log in as admin', function(done){
  user1.post(port+'/api/login').send({email:'1@1.com',password:'1'}).end(function(res){
    assert.equal(res.status,200);
    done();
  });
});
```

```
test('delete users', function(done){
  user1.del(port+'/api/users/'+userId).end(function(res){
    assert.equal(res.status, 200);
    done();
  });
});

test('logout admin', function(done){
  user1.post(port+'/api/logout').end(function(res){
    assert.equal(res.status, 200);
    done();
  });
});

test('log in as user - should fail', function(done){
  user1.post(port+'/api/login').send({email: '3@3.com', password: '3'}).end(function(res){
    // console.log(res.body)
    assert.equal(res.status, 500);

    done();
  });
});

test('check for posts - must fail', function(done){
  user1.get(port+'/api/posts/').end(function(res){
    // console.log(res.body)
    assert.equal(res.status, 500);

    done();
  });
});

suiteTeardown(function(done){
  done();
});

});
```

The full content for `hackhall/tests/posts.js`:

```
var app = require('../server').app,
    assert = require('assert'),
    request = require('superagent');

app.listen(app.get('port'), function() {
  console.log('Express server listening on port ' + app.get('port'));
});

var user1 = request.agent();
var port = 'http://localhost:' + app.get('port');
var postId;

suite('POSTS API', function() {
  suiteSetup(function(done) {
    done();
  });
  test('log in', function(done) {
    user1.post(port + '/api/login').send({
      email: '1@1.com',
      password: '1'
    }).end(function(res) {
      assert.equal(res.status, 200);
      done();
    });
  });
  test('add post', function(done) {
    user1.post(port + '/api/posts').send({
      title: 'Yo Test Title',
      text: 'Yo Test text',
      url: ''
    }).end(function(res) {
      assert.equal(res.status, 200);
      postId = res.body._id;
      done();
    });
  });

  test('get profile', function(done) {
```

```
    assert.equal(res.status, 200);
    done();
  });
});
test('post get', function(done) {
  // console.log('000'+postId);
  user1.get(port + '/api/posts/' + postId).end(function(res) {
    assert.equal(res.status, 200);
    assert.equal(res.body._id, postId);
    done();
  });
});
test('delete post', function(done) {
  user1.del(port + '/api/posts/' + postId).end(function(res) {
    assert.equal(res.status, 200);
    done();
  });
});
test('check for deleted post', function(done) {
  user1.get(port + '/api/posts/' + postId).end(function(res) {
    // console.log(res.body)
    assert.equal(res.status, 500);

    done();
  });
});
suiteTeardown(function(done) {
  done();
});
});
```

The full content for `hackhall/tests/users.js`:

```
var app = require('../server').app,
```

```
request = require('superagent');
// http = require('support/http');

var user1 = request.agent();
var port = 'http://localhost:' + app.get('port');

app.listen(app.get('port'), function() {
  console.log('Express server listening on port ' + app.get('port'));
});

suite('Test root', function() {
  setup(function(done) {
    console.log('setup');

    done();
  });

  test('check /', function(done) {
    request.get('http://localhost:3000').end(function(res) {
      assert.equal(res.status, 200);
      done();
    });
  });

  test('check /api/profile', function(done) {
    request.get('http://localhost:' + app.get('port') + '/api/profile').end(function(res) {
      assert.equal(res.status, 500);
      done();
    });
  });

  test('check /api/users', function(done) {
    user1.get('http://localhost:' + app.get('port') + '/api/users').end(function(res) {
      assert.equal(res.status, 500);
      // console.log(res.text.length);
      done();
    });
    // done();
  });
});
```



```
user1.get('http://localhost:' + app.get('port') + '/api/posts').end(function(res) {
  assert.equal(res.status, 500);
  // console.log(res.text.length);
  done();
});
// done();
});
teardown(function(done) {
  console.log('teardown');
  done();
});

});

suite('Test log in', function() {
  setup(function(done) {
    console.log('setup');

    done();
  });
  test('login', function(done) {
    user1.post('http://localhost:3000/api/login').send({
      email: '1@1.com',
      password: '1'
    }).end(function(res) {
      assert.equal(res.status, 200);
      done();
    });
  });

});
test('check /api/profile', function(done) {
  user1.get('http://localhost:' + app.get('port') + '/api/profile').end(function(res) {
    assert.equal(res.status, 200);
    // console.log(res.text.length);
    done();
  });
  // done();
});
});
```

```
user1.get('http://localhost:' + app.get('port') + '/api/users').end(function(res) {
  assert.equal(res.status, 200);
  // console.log(res.text);
  done();
});
// done();
});
test('check /api/posts', function(done) {
  user1.get('http://localhost:' + app.get('port') + '/api/posts').end(function(res) {
    assert.equal(res.status, 200);
    // console.log(res.text.length);
    done();
  });
  // done();
});

teardown(function(done) {
  console.log('teardown');
  done();
});

});
suite('User control', function() {
  var user2 = {
    firstName: 'Bob',
    lastName: 'Dilan',
    displayName: 'Bob Dilan',
    email: '2@2.com'
  };
  suiteSetup(function(done) {
    user1.post('http://localhost:3000/api/login').send({
      email: '1@1.com',
      password: '1'
    }).end(function(res) {
      assert.equal(res.status, 200);
      // done();
    });
    user1.get('http://localhost:' + app.get('port') + '/api/profile').end(function(res) {
```

```
// console.log(res.text.length);
// done();
});

done();
})

test('new user POST /api/users', function(done) {
  user1.post(port + '/api/users')
    .send(user2)
    .end(function(res) {
      assert.equal(res.status, 200);
      // console.log(res.text.length);
      user2 = res.body;
      // console.log(user2)
      done();
    })
});

test('get user list and check for new user GET /api/users', function(done) {
  user1.get('http://localhost:' + app.get('port') + '/api/users').end(function(res) {
    assert.equal(res.status, 200);
    // console.log(res.body)
    var user3 = res.body.filter(function(el, i, list) {
      return (el._id == user2._id);
    });
    assert(user3.length === 1);
    // assert(res.body.indexOf(user2)>-1);
    // console.log(res.body.length)
    done();
  })
});

test('Approve User: PUT /api/users/' + user2._id, function(done) {
  assert(user2._id != '');
  user1.put(port + '/api/users/' + user2._id)
    .send({
      approved: true
    })
    .end(function(res) {
```

```
// console.log(res.text.length);

assert(res.body.approved);

user1.get(port + '/api/users/' + user2._id).end(function(res) {
  assert(res.status, 200);
  assert(res.body.approved);
  done();
})

})

});

test('Banned User: PUT /api/users/' + user2._id, function(done) {
  assert(user2._id != '');
  user1.put(port + '/api/users/' + user2._id)
    .send({
      banned: true
    })
    .end(function(res) {
      assert.equal(res.status, 200);
      // console.log(res.text.length);
      assert(res.body.banned);
      user1.get(port + '/api/users/' + user2._id).end(function(res) {
        assert(res.status, 200);
        assert(res.body.banned);
        done();
      })
    })
  });

test('Promote User: PUT /api/users/' + user2._id, function(done) {
  assert(user2._id != '');
  user1.put(port + '/api/users/' + user2._id)
    .send({
      admin: true
    })
    .end(function(res) {
      assert.equal(res.status, 200);
      // console.log(res.text.length);
      assert(res.body.admin);
```

```
    assert(res.status, 200);
    assert(res.body.admin);
    done();
  })

  })
});

test('Delete User: DELETE /api/users/:id', function(done) {
  assert(user2._id != '');
  user1.del(port + '/api/users/' + user2._id)
    .end(function(res) {
      assert.equal(res.status, 200);
      // console.log('id:' + user2._id)
      user1.get(port + '/api/users').end(function(res) {
        assert.equal(res.status, 200);
        var user3 = res.body.filter(function(el, i, list) {
          return (el._id === user2._id);
        });
        // console.log('***');
        // console.warn(user3);
        assert(user3.length === 0);
        done();
      });
    });
});

});
});

// app.close();
// console.log(app)
```

**Warning:** Please don't store plain passwords/keys in the database. Any serious production app should at least [salt the passwords](#) before storing them.

## Conclusion

applications components, such as REST API architecture, OAuth, Mon-goose and its models, MVC structure of Express.js apps, access to environment variables, etc.



11



4



This entry was posted in Book, MongoDB, Node.js, Startups, Tutorials and tagged express.js, expressjs, hackhall, MongoDB, node.js, rest api on November 5, 2013 [<http://webapplog.com/express-js-and-mon-goose-example-building-hackhall/>] .



nottinhill

September 22, 2014 at 7:11 am

Massive tutorial, thanks!

---

Pingback: [Javascript Hacks for Hipsters | SoshiTech](#)