# TheoryLab Starting Guide – 21 Nov 2016

## Framework:

Mean.js - <https://meanjs.org/>

Generating new modules look at:

<https://meanjs.org/generator.html>

NPM as for server-side modules

Bower for client-side modules. When installing new modules, it gets copied to */public/lib*.

Be sure to add the scripts to the */config/assets/default.js* script.

No need to add application scripts or style files to this file as they’re automatically picked up.

[MongoDB](https://www.mongodb.com/) is the database and [Mongoose.js](http://mongoosejs.com/) is the ORM for the database.

We are using Gulp for continuous integration (CI):

**gulp serve** - to run the application

**gulp test** – to do code linting and run tests. **NB –** make sure all tests pass before committing to the repo.

**IMPORTANT:** Make sure that the configuration for testing is pointing to its own database as the tests clears the users table.

We use LESS as our CSS pre-processor:

<http://lesscss.org/>

**gulp less –** to compile LESS files.

For **in-application notifications** we use [angular-toastr](https://github.com/Foxandxss/angular-toastr).

**Form validation** should us Angular Form validation and not HTML 5 form validation.

## Source Control:

Bitbucket: <https://bitbucket.org/shemaimme/rudolph-nd> - David to provide access

The staging branch is the current branch that deploys to the staging server. Continuous Deployment (CD) will deploy the latest pushed code to the staging server.

When working on new features or bug fixes, rather create a new or feature branch.

## Continuous Deployment (CD)

Codeship - <https://app.codeship.com>

## Task + Project Management

Trello - <https://trello.com/b/szGm631k/rudolph-nd>

## Hosting

The application is currently hosted on AWS on a Linux VM.

Staging URL: <http://23.20.9.114/>

Any file uploads should go to AWS S3, only application files should sit on the server.

## Payment Gateway Integration

Stripe - <https://stripe.com/>

## Communication

Skype / Slack

## Performance Monitoring

<https://keymetrics.io/> - there is also a npm module that posts all the events to Keymetrics, so we can track issues on the AWS servers.

## Integration with DeployR

DeployR server (AWS VM server): <http://52.44.200.161:8000/deployr>

DeployR API reference docs:

<https://microsoft.github.io/deployr-api-docs/>

<https://github.com/Microsoft/js-client-library>

## Application specific

Following the Mean.js structure the functionality is broken into logical **modules**. Each module’s code is separated into a **client** and **server** folder.

**TheoryLab outputs**

At the time of writing this documentation, TheoryLab is made of 3 outputs. Posts, Datasets and Models.

**Posts** is articles created by the user, that could be linked to datasets and models. Posts is always publicly available.

**Datasets** is the data source for any further data operations. There will be readily available public datasets available, but users will be able to upload their own datasets. It will be stored in CSV format.

**Models** is provided by the DeployR server, based on the workflow and data source. The user can save their models and share it.

The user will be able to change access levels to their datasets and models, but only if they have a premium subscription. The general access level is public, but if they do have a premium subscription they can choose to make their datasets and models either **private** or **for sale**. If they choose “for sale”, they must set a price and they can add a preview note.

A model will always reference a dataset. It is important that when changing the referenced dataset or model’s access level or price, that it will be reflected in both cases. For example, the model can’t be public, if the dataset is private vice versa. Or the dataset’s price can’t be more than the model’s price. But a dataset can be sold separate from the related model, but a model can’t be sold without its dataset.

**Access Levels**

The user can choose to save public datasets/models to their LAB, where they can apply transformations/workflows against it which in return can be outputted as models and datasets. These assets could be shared publicly or sold. When the user purchases a dataset or model, the dataset/model will also be saved to their LAB.

When copying these assets to the users LAB, we will make a copy of the dataset on S3 and create a new record in the MongoDB collection with the **updated S3 reference**.

On the server-side each module has a ACL policy (ending with a **server.policy.js** extension) file which define the access control for each route. When adding a new route on the server side, you need to add the route for each role that you would like to allow access to. Currently there is 3 roles in the system:

* admin
* user
* guest

You also need to define, the HTTP method that you have rights to use this route on ie. GET, PUT, POST, DELETE.

The guest role is when not authenticated and any routes specified in for this role, won’t have access to the req.user object and it should only be routes for API’s of public accessible data.

On the client side route access is declare in to ways, either you can pass in the role on a route level as a role array under the data attribute or otherwise all routes is blocked by default, except for a list specified in \modules\core\client\app\init.js – allowedstates – which contains a list of routes that is available without authentication.

**Payment Models**

There are 2 payment business models used in the application:

One is a subscription model, where based on your subscription plan it unlocks different features for your environment.

The other is where you have an account, where you can buy and sell datasets and models.

You can’t buy and sell, if you’re on the free plan subscription.

All payments are going through Stripe, which acts as the payment gateway, but also used to setup the products for the subscription plans.

**MongoDB collections**

Below is a brief description of the MongoDB collections:

* **Articles** – this was part of the Mean.js framework, but can be removed.
* **Companies**
* **Datasets** – this is the list of datasets.
* **Models** – this is the list of models and it has a dataset field that reference a dataset above.
* **Posts** – the list of user generated posts, it links to datasets and models.
* **Postviews** – each time a post get viewed we log an entry to this collection and we increment the number of views on the posts collection. But only log it for unique users.
* **Processes** – this collection is for the lists of saved workflows. Everything was initially called processing/processes, but the language for this functionality changed to workflow.
* **Sessions** – the sessions collection is used by the passport authentication; you shouldn’t need to touch this in your code otherwise.
* **Stripeevents** – this collection is the log of Stripe events that gets generated by the stripe webhook.
* **Users** – the user’s collection has all the user credentials, subscription info etc.

**Workflows and DeployR integration**

When the user goes to his LAB, he will be presented with options to build his own workflow to transform datasets or generate models.

The workflow commands are executed against DeployR’s API using their [client API](https://github.com/Microsoft/js-client-library) tools.

There is a list of predefined workflow tasks in TheoryLab and the list will continue to grow.

A workflow task consists of the return type of either a model/dataset and you can also add options for a workflow task which will be an Angular view and its own controller.

Each workflow task generates a DeployR code snippet based on the functionality required. All this code gets generated on the server side, with the task options passed through as variables to the snippet.

All dataset read/write events happen on the DeployR side, by reading and writing to/from datasets to S3.