Getting started with JHipster for team project: masterclass

Team Project 2022-23

JHipster is a tech-stack generator that creates full applications stacks in using modern components. In this session, we will develop some fetures in a JHipster application. You are encouraged to follow along and try the instructions.

This session will use the terminal, gitlab and IntelliJ Ultimate however the principles are the same as if using an different IDE. n.b. use the codepad links e.g. *try it on codepad!* to copy and paste code from as PDF can be a pain to get code listings from.

This session is not exhaustive. Similar to git and CI, many of the commands are rarely used and it is completely fine to consult the documentation for JHipster when needed. This tutorial is based on the official `https://github.com/mraible/jhipster7-demo`.

Terminology

JHipster - the application generation framework we use to create the template app in team repositories.

SpringBoot - the back-end framework in the the generated app - a Springboot backend is created for us by JHipster, in Java.

Angular - the front-end framework in the the generated app - an Angular front-end is created for us by JHipster, in TypeScript.

Maven - a Java build and dependency automation tool for Java project, like our generated app. A project object model (pom.xml) is used to manage the project build and dependencies.

NPM - a package manager JavaScript and the runtime environment Node.js. NPM is used to gather dependencies for the Angular front end.

Postgres - the backend database used when the application is deployed into production.

H2DB - the backend database used when the application is developed locally.

IntelliJ Ultimate - an IDE that can be used to load the project for development. Ultimate is recommended because the community edition does not support SpringBoot projects natively.

Install pre-requisites- `git`, `java`, `npm`, `jhipster`, IntelliJ Ultimate

Install the pre-requisites for the masterclass:

- git
- java - recommended version for JHipster JDK 11
- npm - recommended: LTS v8.x.x
- jhipster - using `npm install -g jhipster/generator-jhipster#v7.x_maintenance`
- IntelliJ Ultimate https://www.jetbrains.com/community/education/#students

Get a JHipster tech stack

Clone and existing repository

To get the template tech stack simply clone the team's repository to your local machine (using SSH):

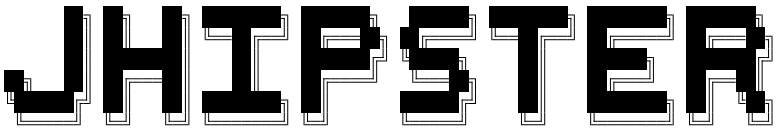
```
git clone git@git.cs.bham.ac.uk:shahs/template-teamproj-app.git
```

n.b. your repository URL will be specific to your project

Create a new app

If you wish to create a new app, in an empty folder do:

```
shah@mars example-jhip % jhipster
INFO! Using bundled JHipster
```



```
                    https://www.jhipster.tech
Welcome to JHipster v7.9.3

Application files will be generated in folder: /Users/shah/git/example-jhip
 ------------------------------------------------------------------------------------------------------------------

  Documentation for creating an application is at https://www.jhipster.tech/creating-an-app/
  If you find JHipster useful, consider sponsoring the project at https://opencollective.com/generator-jhipster
 ------------------------------------------------------------------------------------------------------------------

? Which *type* of application would you like to create? (Use arrow keys)
> Monolithic application (recommended for simple projects)
  Gateway application
  Microservice application
```

and follow the prompts. the options we used in the template app are found in `.yo-rc.json`:

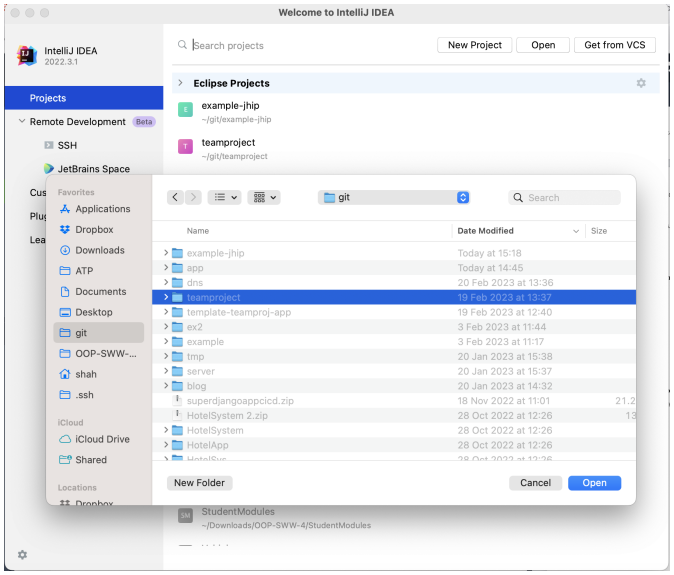
```
{
  "generator-jhipster": {
    "applicationType": "monolith",
    "authenticationType": "jwt",
    "baseName": "teamproject",
    "blueprints": [],
    "buildTool": "maven",
    "cacheProvider": "ehcache",
    "clientFramework": "angularX",
    "clientPackageManager": "npm",
    "clientTheme": "none",
..
  }
}
```

n.b. you must randomise your `jwtSecretKey` in `.yo-rc.json` for production.

IntelliJ development environment

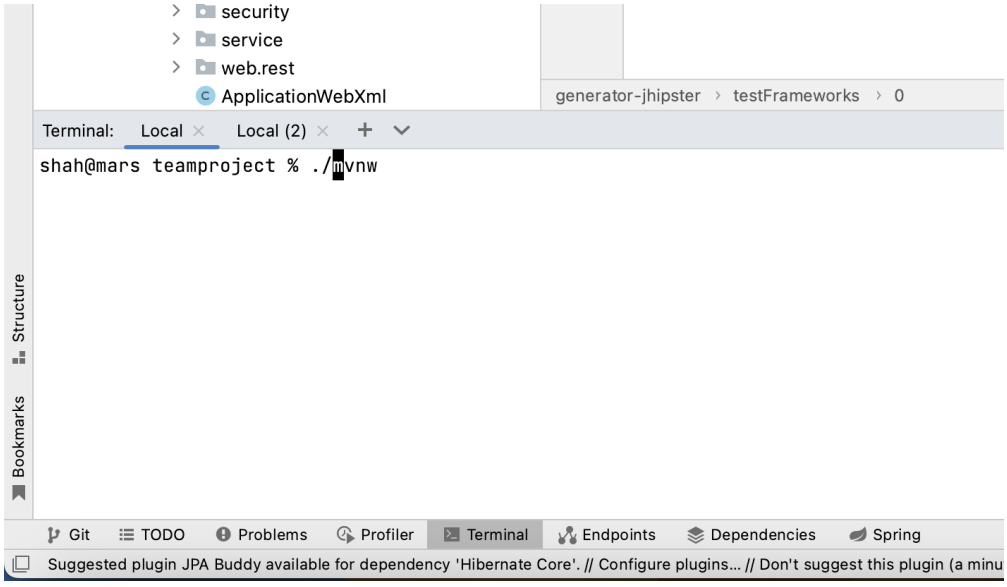
Open the project in IntelliJ

“File” ... “Open” the project folder in IntelliJ Ultimate - allow time for it to build

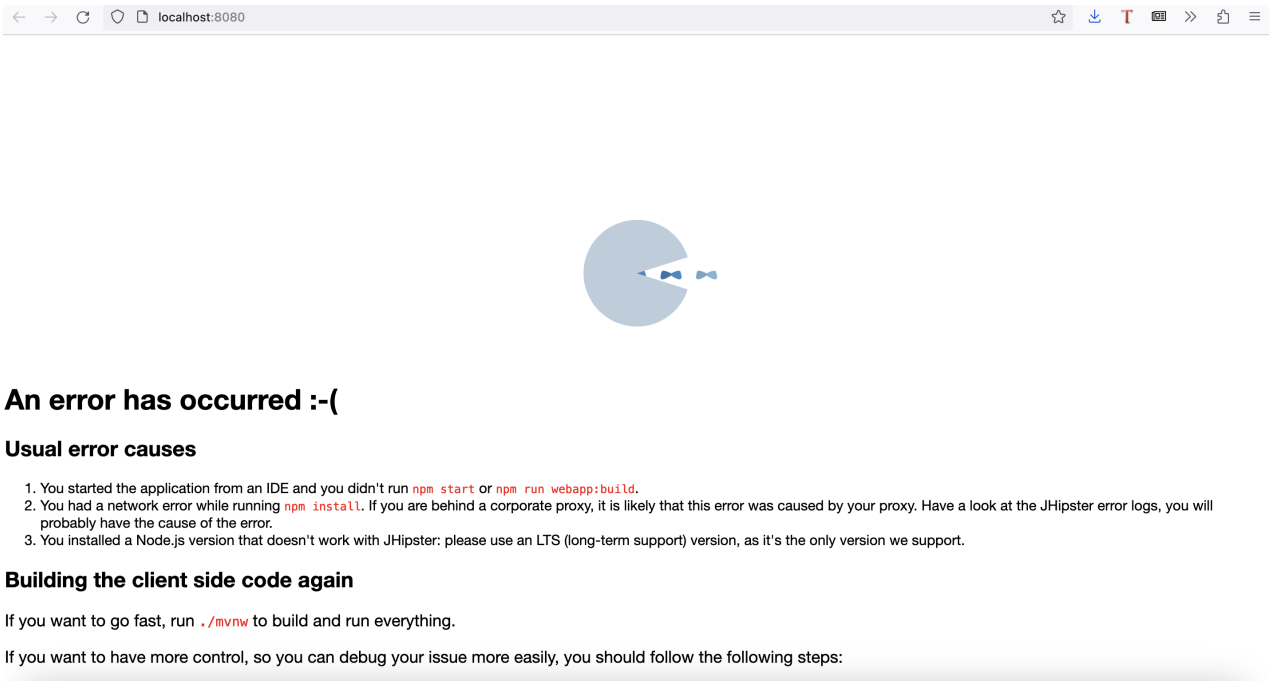


Setup the run configurations & run app in IntelliJ

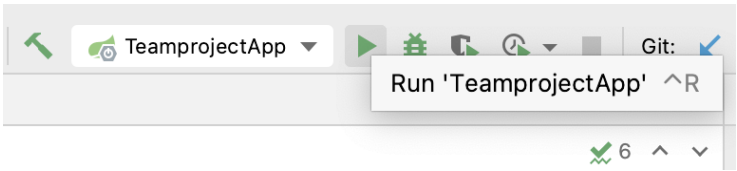
In order to run the app, you will need to run `./mvnw` once in a terminal to build the app:



Otherwise you will get the following error:

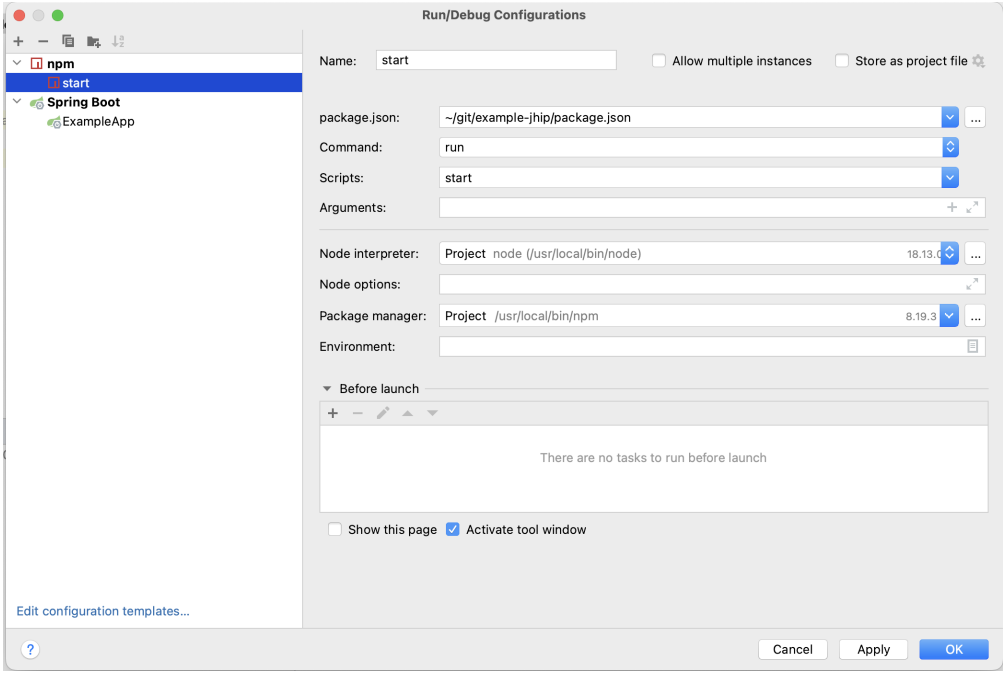


After `./mvnw` you can run the app using the detected run configuration:



Optional: create frontend-only run config

You may want to add a run config to start only the front end:



This will start the front end on http://localhost:9000. n.b. you will run into errors if the backend is not running.

Create a JDL file of core entities

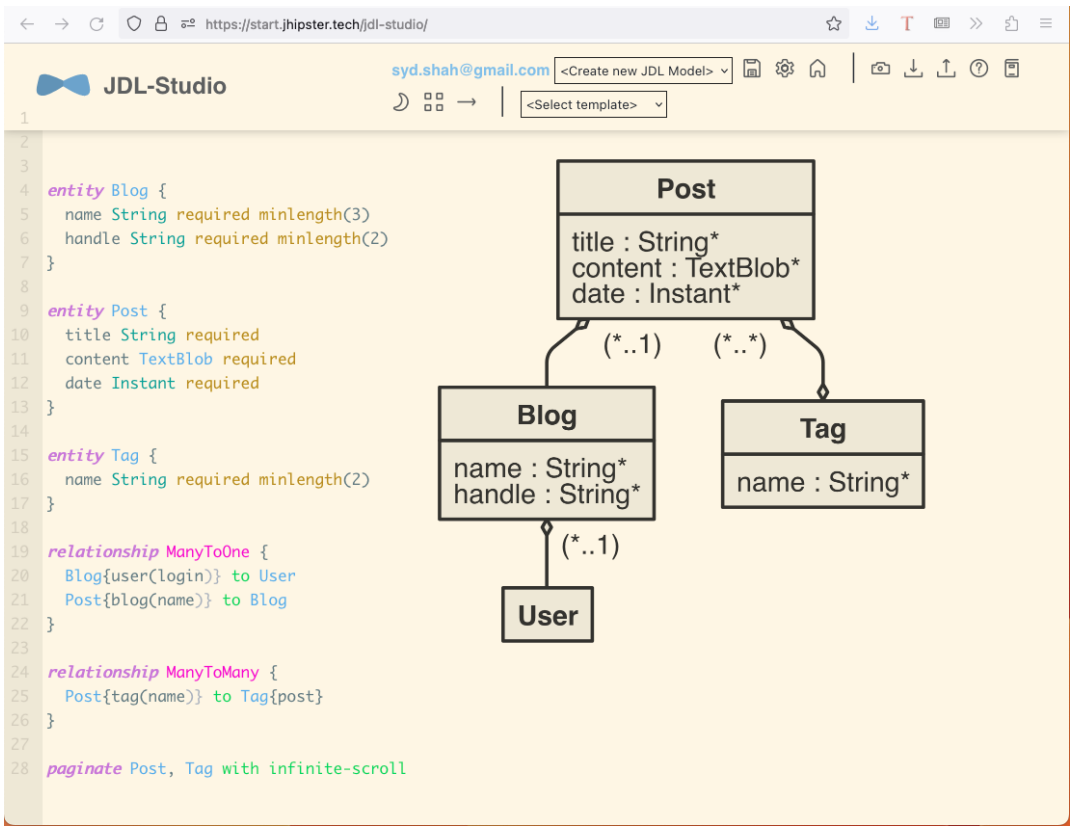
The next thing you will need to do is create a JDL file that represents the core entities in your app. For this masterclass we will use `blog.jdl`:

```
entity Blog {
  name String required minlength(3)
  handle String required minlength(2)
}

entity Post {
  title String required
  content TextBlob required
  date Instant required
}

entity Tag {
  name String required minlength(2)
}

relationship ManyToOne {
  Blog{user(login)} to User
  Post{blog(name)} to Blog
}

relationship ManyToMany {
  Post{tag(name)} to Tag{post}
}

paginate Post, Tag with infinite-scroll
```

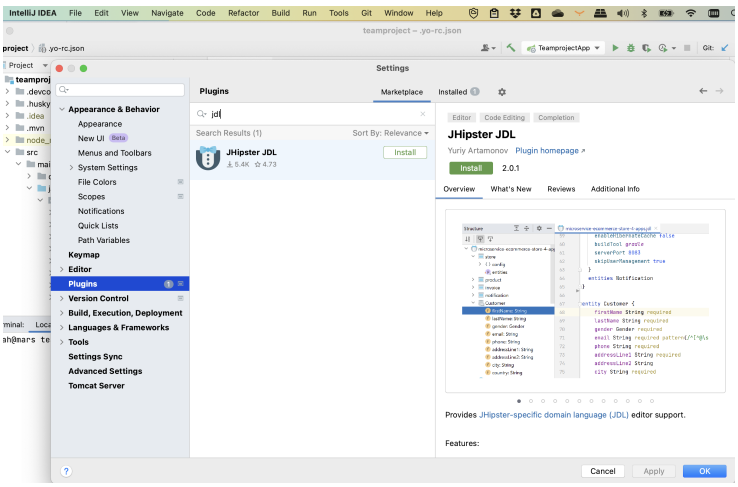
try it on codepad!

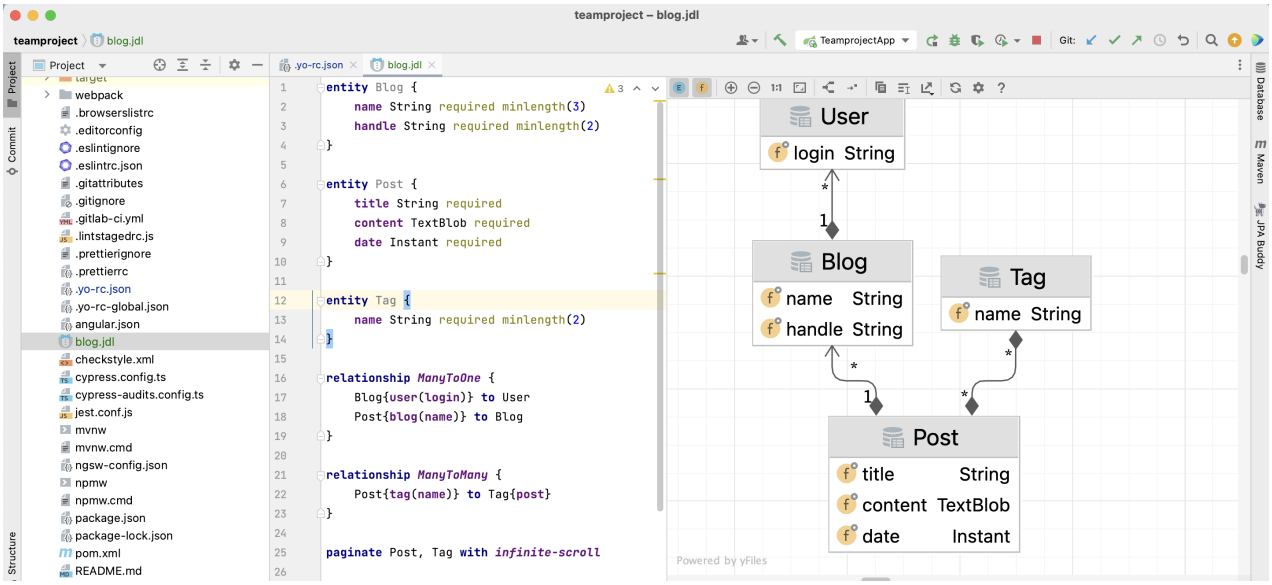
Copy the above into to a file `blog.jdl` in the root of your repository.

This can be edited online using JDL studio:



Or using the Plugin for IntelliJ:





Then in the project root folder terminal run the command:

```
jhipster jdl blog.jdl
```

which will generate for each entity:

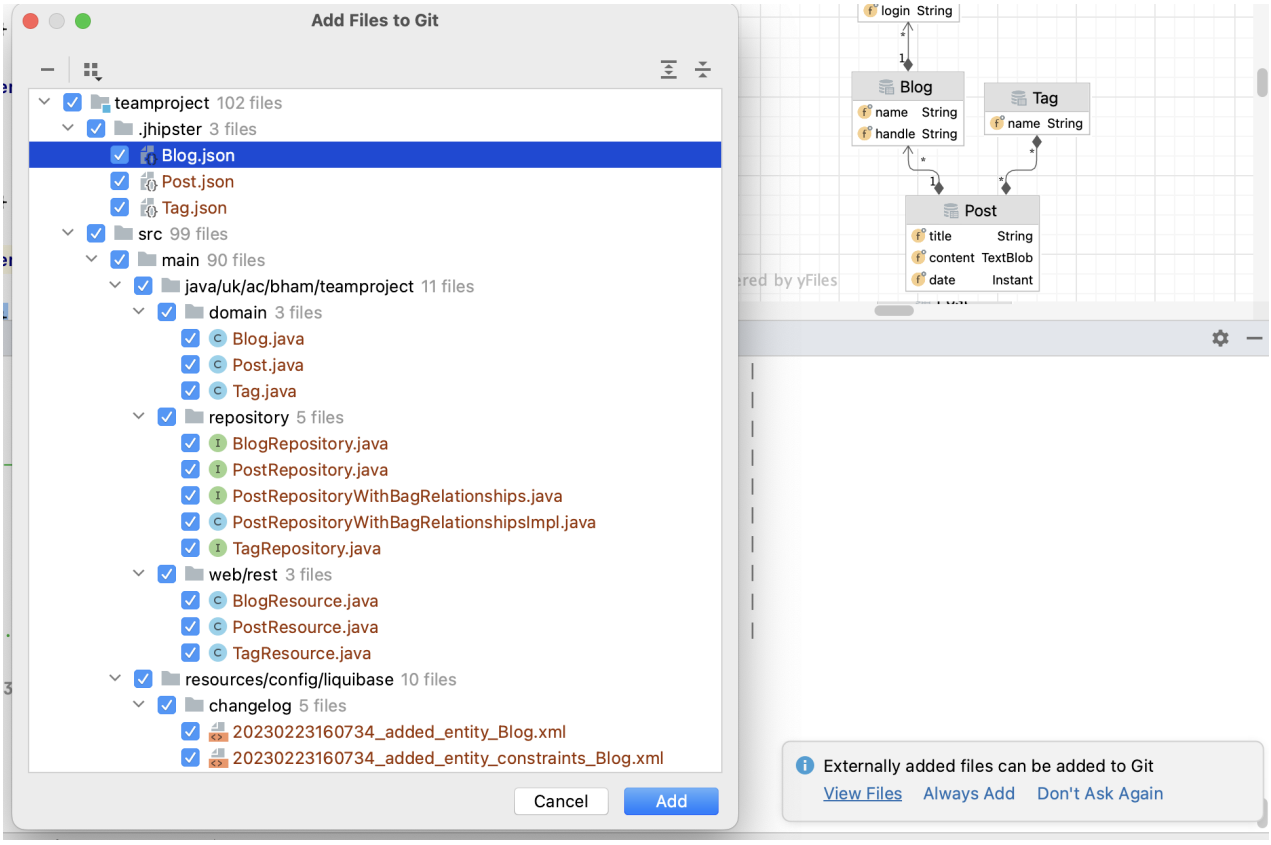
- a database table;
- a Liquibase changeset;
- a JPA entity class;
- a Spring Data JpaRepository interface;
- a Spring MVC RestController class;
- an Angular list component, edit component, service;
- several HTML pages for each component.

when asked:

```
conflict src/main/webapp/i18n/en/global.json
? Overwrite src/main/webapp/i18n/en/global.json? (ynarxdeiH)
```

enter `a` to overwrite any new files.

As you will see, JHipster has generated quite a few files. IntelliJ will prompt you add these to your git repo:



(or in a terminal `git add . /commit/push` the changes)

Optional: remove faker generated entities in dev mode

JHipster automatically generates example entities using faker.js. This is useful avoid to manually creating dummy data. However the data created is incomplete: for example, the entities do not have users.

This is in the dev profile only, disable in:

`src/main/resources/config/application-dev.yml` (line 48 remove faker)

Delete the H2 database in `target/h2db`

Business logic: change who can view items

One issue with the application is that the core data models does not follow reasonable security principles. Blogs and posts should only be viewable by the user who created them. Currently posts and blogs can be viewed by all users and edited by any user. To make it so that only the current user can see their own blogs, edit the file:

`src/main/java/uk/ac/bham/teamproject/blog/web/rest/BlogResource.java`

the method to change is:

```
1 public List<Blog> getAllBlogs(@RequestParam(required = false, defaultValue =
    "false") boolean eagerload) {
2     log.debug("REST request to get all Blogs");
3     if (eagerload) {
```

```
        return blogRepository.findAllWithEagerRelationships();
    else {
         return blogRepository.findAll();
    }
}
```
try it on codepad!

change it so that it is:

```
@GetMapping("/blogs")
public List<Blog> getAllBlogs(@RequestParam(required = false, defaultValue =
    "false") boolean eagerload) {
    log.debug("REST request to get all Blogs");
    return blogRepository.findByUserIsCurrentUser();
}
```
try it on codepad!

Similarly, to make it so that only the current user can see their own posts, edit the file:

`src/main/java/uk/ac/bham/teamproject/blog/web/rest/PostResource.java.getAllPosts()`

replace:

```
Page<Entry> page;
if (eagerload) {
    page = postRepository.findAllWithEagerRelationships(pageable);
} else {
    page = postRepository.findAll(pageable);
}
```
try it on codepad!

with:

```
page = postRepository.findByBlogUserLoginOrderByDateDesc(
    SecurityUtils.getCurrentUserLogin().orElse(null), pageable);

```
try it on codepad!

and in the file:

`src/main/java/uk/ac/bham/teamproject/repository/PostRepository.java`

create the method:

```
Page<Post> findByBlogUserLoginOrderByDateDesc(String currentUserLogin,
    Pageable pageable);

```
try it on codepad!

Front-end changes

HTML in Posts

We want the front end to allow HTML in the content field of a blog post. To achieve this, edit the file:

`src/main/webapp/app/entities/post/list/post.component.html`

replace:

```
<td>{{ post.content }}</td>}
```

with:

```
<td [innerHTML]="post.content"></td>
```

Blog layout

We want the front end to look more like a blog, to do this, edit:

`src/main/webapp/app/entities/post/list/post.component.html`

change:

```
<div class="table-responsive">
```

to:

```
<div class="table-responsive" *ngIf="posts && posts.length > 0">
  <div infinite-scroll (scrolled)="loadPage(page + 1)"
      [infiniteScrollDisabled]="page >= links['last']"
      [infiniteScrollDistance]="0">
    <div *ngFor="let entry of posts; trackBy: trackId">
      <a [routerLink]="['/post', entry.id, 'view' ]">
        <h2>{{entry.title}}</h2>
      </a>
      <small>Posted on {{entry.date}} by {{entry.blog}}</small>
      <div [innerHTML]="entry.content"></div>
      <div class="btn-group mb-2 mt-1">
        <button type="submit"
                [routerLink]="['/entry', entry.id, 'edit']"
                class="btn btn-primary btn-sm">
          <fa-icon [icon]="'pencil-alt'"></fa-icon>
          <span class="d-none d-md-inline"
              jhiTranslate="entity.action.edit">Edit</span>
        </button>
        <button type="submit"
                [routerLink]="['/', 'entry', { outlets: { popup: entry.id +
                    '/delete'} }]"
                replaceUrl="true"
                queryParamsHandling="merge"
                class="btn btn-danger btn-sm">
          <fa-icon [icon]="'times'"></fa-icon>
          <span class="d-none d-md-inline"
              jhiTranslate="entity.action.delete">Delete</span>
        </button>
      </div>
    </div>
  </div>
</div>                                                    try it on codepad!
```

Deployment

The deployed app from the masterclass can be found here:

http://mc.bham.team:8080

login using username: user password: user to see the changes in action.