**MVC结构**

**1.基础设置:**

* 设置好view engine, views和layout路径(作为前端), 以及static文件夹
* 监听3000端口
* 设置路由
* 连接mongoose，有错报错，没错就console.log一次连接成功
* 配置env, 并且设置当目前模式不为production的时候，才会require dotenv并把它config()
* 建立author路由，包含全部author, 新的author，以及创建author(post)三个路由
* 前端views中建立partials文件夹(局部渲染)， author文件夹(author, new author)

**2.Author搜索页面:**

* 用mongoose建立author的模型, 在路由中拿到，传入建立新author的路由中
* 在前端构建表单，并把输入框绑定后端路由从模型中拿到的author的名字
* 安装并使用body-parser来接收请求体
* 在创建新author的路由中，设个对象赋值为请求体。 如果路由进入成功，在前端显示新加的名字。不成功就显示errorMessage
* **显示全部author/显示搜索的author：**
  + 用author模型的find({})获取并传到前端显示全部authors
  + 设个searchAuthor变量，并在req.query.name不为null和不为空的情况下把searchAuthor赋值为req.query， 让author模型find这个searchAuthor(如果搜索栏为空就还是显示全部)，并把searchAuthor传到前端并显示

**3.书本信息上传页面**

* 建立book模型，里面包含(title, description, publishDate, pageCount, createdAt, coverImageName和author)， 这个author实际上是author模型的id
* 建立book路由，和author构造差不多
* 去前端建立books文件夹，建立和author构造一样的new.ejs，同时在这文件夹下建立相应的\_form\_fields.ejs

\_form\_fields.ejs

* + Title 输入框
  + Author下拉选项：遍历authors, 从中拿到的author名字就是它的label，拿到的id就是它的value。 默认选中为当前author的id

Book路由：

* + 添加新书的路由: 分别拿到Author.find({})的所有对象和Book模型，并把它们以一个对象形式render到前端
* 到前端把其它输入框渲染好
* 完成表单整个添加:
  + New book的路由封装到另一个renderNewPage函数中，传入res, book, hasError(默认为false)， 如果hasError是true, 那就显示错误信息
  + 在post book的路由中，把全部需要在前端显示的传到前端，并把它们save到一个新对象中
  + Book model中配置存放图片的路径并export
  + 回到book路由中，安装并使用multer, 使用path, 拿到uploadpath，定义image type， 并用multer把uploadpath和一个filefilter函数
* 如果表单有地方没填，还是会把图片保存，防止这种情况:
  + Require fs, 在catch中如果存在新加入的coverImage, 就建立并调用removeBookCover函数
  + removeBookCover: unlink这个file

**4.书本搜索页面**

* 前端渲染搜索页面(index.ejs), 表单下面渲染书的coverImage( src拿到book模型的coverImagePath);
* 在book模型中用bookSchema建立一个virtual方法， 用get()来返回一个函数，这个函数返回path.join(‘/’, 图片路径，图片名)
* 到book路由的获取所有书的路由里，把所有书和搜索结果传到前端，用query获取搜索的值
* 最后把目前所有的书显示到前端的index.ejs
  + 在index.ejs的后端路由中把book传到前端，排序为最新的在前限制10本

**5. 图片上传到数据库(转成base64格式)**

* 去到file pond上cdn导入主css和script文件，再倒入image preview的js和css, file encode和image resize的js（这些都要放在主js前面）,再建个名字为fileUpload的js文件，用来引入这些功能
* 在fileUpload中，FilePond.registerPlugin方法分别传入preview,resize和encode方法。去到上传书图片的前端页面中，加入class=’filepond’
* 再回到fileUpload中，.setOptions（）里面定义图片比例，和长宽尺寸
* **功能:**
  + 去到book模型，删掉coverImageName, 定义coverImage(type:buffer), coverImagePath。
  + 然后去到book路由, 定义saveCover(book, coverEncoded)函数, 主要功能是如果coverEncoded(类型为string)不为空，就把它转成一个json对象。 如果这个json对象不为空，以及imageMimeTypes包含了对象的类型，那么就把book.coverImage转成buffer类型.from(cover.data, ‘base64’)， 再把coverimageType设为这个json对象的type
  + 在添加书的路由函数中调用(传入book, 请求体中的cover)。
  + 回到book模型中，如果coverImage 和 coverImageType都不为空，那么返回
* `data:${this.coverImageType};charset=utf-8;base64,${this.coverImage.toString('base64')}`;
  + (以上代码意义为把他们转换成base64格式)
  + 之后删除之前把图片存到本地的功能代码

**6. Author页面对作者的显示，更改，更新和删除**

**配置:**

* 在后端author路由中，分别定义view, edit, update和delete路由
* 去到前端authors中的index, 渲染这几个选析， 把delete单独封装成一个partial文件，这是一个post表单(传入动态url), 这个动态url就是要删除的author的id

**删改功能:**

* Edit:
  + 通过findbyid拿到当前作者的id， 然后render把这作者传到前端， 有错的话redirect到authors页面。 去到前端建立edit文件，把method改成put， cancel后回到的url改为authors/author.id
* Put:
  + 设一个author，同样通过id拿到，把它的名字赋值为请求体的名字，之后save就改完了，改完后跳转路径到authos/author.id
  + 如果有错：如果找不到这author，那就跳转回首页。找得到author但其他错误，还是render在edit页面，author不变，输出错误信息
* Delete:
  + 和put写法几乎一样，只是把save变成remove, 然后如果能找到author还有错，那就跳转回当前这个author的页面

**如果此author有作品，那就不能删除这个author:**

* 去到author模型中(用pre函数，一般这个函数在执行一个任务之前调用):
  + 用pre函数(传入remove任务，和回调函数(next))， 用当前这本书来查找包不包含这个作者的id（包含就代表这本书是属于这个作者的），再回调， 如果有错，直接next这个错，如果这个author有的书大于0, next一个新的错(不能删除这个作者), 否则就什么都不做直接继续

**Show页面:**

* 写好前端渲染
* 后端拿到用findbyid拿到符合这个id的author， 再拿到属于这个author的书（最大限制显示六本）
* Render, 传入这个author和书
* 有错的话跳转回首页

**7. Book页面对单本书的显示，更改，更新和删除**

* Show:
  + 前端写好要渲染的东西，把主页显示的图片添加链接
  + 在后端，用findById拿到这本书， .populate(‘author’)拿到这本书对应的author的所有属性(这样前端才可以显示author的名字)
* Edit:
  + 前端写好edit.ejs
  + 后端用findbyid拿到当前这本书，调用renderEditPage方法，传入res和刚拿到的这本书为参数

renderEditPage:

* + 几乎和之前的renderNewPage一样除了res.render里面的值不一样，所以定义一个renderform函数，把rendernewpage和rendereditpage功能都封装进去，多传一个form参数来改变res.render的路径

renderForm: 如果form是eidt输出error editing错误信息，不是的话就是error creating

* Update:
  + 定义book, 并用findbyid拿到当前这本书。 把这本书所有的属性赋予请求体的对应属性的值
  + 如果请求体中有cover，调用并执行saveCover(传入这本书，请求体的cover)
  + 保存这本书，并跳转到这本书的页面
  + 如果有错: 如果能找到这本书，调用并执行renderEditPage()， 找不到这本书，直接跳转回首页
* Delete: 和之前author的delete差不多， 只不过如果有error并且这本书不为空，那就输出错误信息
* 最后去到其它有图片的页面，给这些图片加上链接