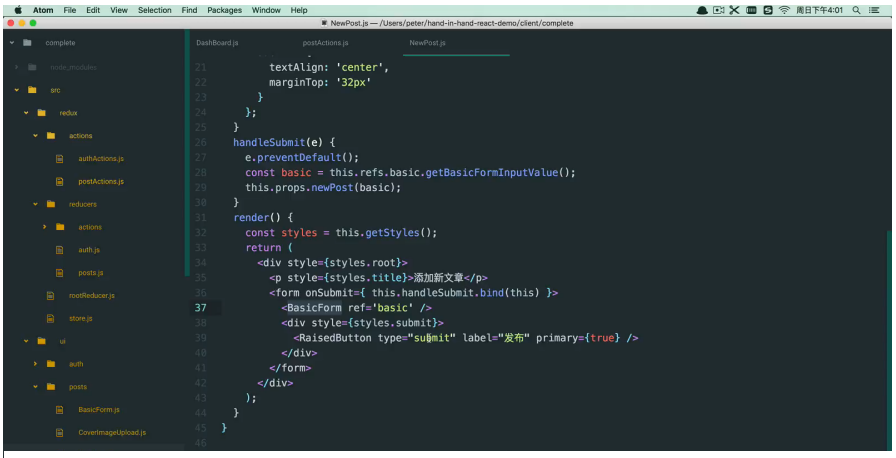

前端导航栏体现用户登录状态

前端实现用户退出登录功能

第六章：用户注册功能实现

图片保存到服务器并在前端展示出来

图片从 CoverImageUpload 组件传递给 BasicForm 组件

先修改 CoverImageUpload 组件的 `handleChange` 方法，当图片加载完毕之后，调用由父组件 BasicForm 传递进来的属性 `handleImage` 函数，把上传的图片传递给 BasicForm 组件

```
handleChange(event) {
    ...
    reader.onload = (event) => {
      ...
      this.props.handleImage(file);
```

```
    }
  }
```

图片从 BasicForm 组件传递给 NewPost 组件

在 BasicForm 组件中，把 `handleImage` 属性传递给 CoverImageUpload 组件，如下所示：

```
<CoverImageUpload handleImage={this.getImage
```

然后在 BasicForm 组件中，定义 `getImage` 方法，设置 state 变量 `file`，其值为上传的图片文件对象：

```
getImage(file) {
  this.setState({file: file});
}
```

然后，在需要被 NewPost 组件调用的 `getBasicFormInputValue` 方法中读取 state 变量 `file`，并设置为返回值的一个属性，从而 NewPost 组件得到上传的图片文件：

```
getBasicFormInputValue() {
  const name = this.refs.name.getValue();
  const content = this.refs.content.getValue(
  const file = this.state.file;
  return { name, content, file }
```

```
}
```

修改异步 Action 创建函数 newPost

修改 `redux/actions/postActions.js` 文件，修改 `newPost` 函数，重新构建传送给服务器的数据结构：

```
export function newPost(data) {
  let formData = new FormData();
  formData.append('name', data.name);
  formData.append('content', data.content);
  formData.append('post', data.file);
  return function(dispatch) {
    axios.post(`${Settings.host}/posts`, form
    ...
  }
}
```

利用 FormData 对象模拟添加新文章表单。上述代码中的 `name`、`content` 和 `post` 则对应表单中各个输入框的名字。

后端安装 multer 包

```
npm install --save multer
```

利用 multer 中间件处理 HTTP 请求头标识 `Content-Type` 为 `multipart/form-data` 类型的数据

使用 multer 中间件

修改 `server/routes.js` 文件，导入 `multer`：

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

初始化一个新的 multer 对象 `upload`，设置上传图片的保存位置，文件目录 `./public/uploads/posts` 必须手动创建：

```
var upload = multer({dest: './public/uploads/
```

然后修改添加新文章接口，在完成用户身份认证之后，再通过 multer 获取上传的文件信息：

```
app.post('/posts', requireAuth, upload.single
  var post = new Post();
  if(req.file && req.file.filename) {
    post.cover = req.file.filename;
  }
  ...
})
```

因为只是上传一张图片，所以用到了 multer 的 single 接口, 其参数 `post` 字符串对应客户端表单对象 `formData` 中包含的 `post` 字段，这两个名字必须一样。当客户端请求 /posts 接口的时候，上传的文件会保存到 `req.file` 中，其包含的文件信息如下：

```
{ fieldname: 'post',
  originalname: 'Fruit-6.png',
  encoding: '7bit',
  mimetype: 'image/png',
  destination: './public/uploads/posts',
  filename: '3ef1a6c352bff3029b78b56e78d649ba
  path: 'public/uploads/posts/3ef1a6c352bff30
  size: 16937 }
```

通过 `req.file.filename` 就能得到上传文件在磁盘上保存后的文件名，并把这个文件名保存到数据库。同时，更改返回给客户端的 JSON 数据。

posts 集合添加 cover 字段

修改 `server/models/post.js` 文件，给数据库中 `posts` 集合新添加一个 `cover` 字段：

```
var PostSchema = new Schema(
  {
    ...
    cover: { type: String }
  },
);
```

到此为止，到浏览器中新添加一篇文章，上传图片就能保存到服务器的磁盘上了，图片新的文件名也保存到数据库。

后端存储的图片供外部访问

虽然图片已经上传成功了，也知道了图片存储位置，但是图片还不能在浏览器中通过 HTML `<img src='' />` 标签显示出来。我们还需要通过 express 提供的 static，让储存在 public 目录下的静态文件供外部使用。修改 `server/index.js` 文件，添加一行代码：

```
var path = require('path');
app.use(express.static(path.join(__dirname, '
```

前端 PostItem 组件中显示上传的图片

修改 PostItem 组件，导入配置信息：

```
import { Settings } from '../../settings';
```

显示每篇文章包含的图片：

```
return (
  <div style={styles.root}>
    <div style={styles.cover}>
      <img src={`${Settings.host}/uploads/pos
    </div>
    ...
  </div>
);
```

再添加一些样式进来：

```
getStyles() {
```

```
  return {
    ...
    cover: {
      borderBottom: 'solid 1px rgba(200, 215,
      maxHeight: '300px',
      overflowY: 'hidden',
    },
    image: {
      display: 'block',
      width: '100%'
    }
  }
}
```

修改获得所有文章列表的接口

之前，我们只是把文章的标题返回给客户端，现在需要把文章包含的图片文件名返回给客户端，所有修改 GET `/posts` 接口：

```
app.get('/posts', function(req, res) {
  Post.find({}, 'name cover', function(err, p
    ...
  })
})
```