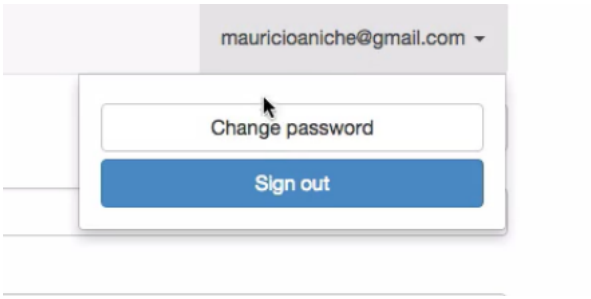
02

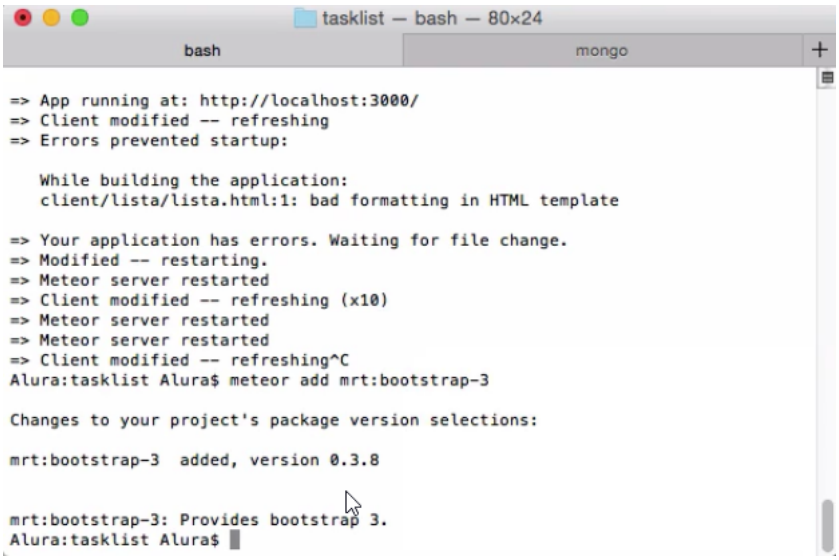
Melhorando a interface com o plugin Bootstrap

O próximo passo é cuidar um pouco da aparência da nossa página, isto é, embelezar ela. Para fazer isso utilizaremos o *Bootstrap*, pois é simples, mas deixa a aplicação com uma cara bonita.

O *Meteor* já faz isso por nós. Então, digitamos no nosso código do terminal o seguinte: `meteor add mrt:bootstrap-3` . Ele vai baixar o pacote e vai configurar isso no *Meteor*. Assim não teremos que ficar inserindo o `csc` . O *Mongo* vai fazer isso por nós. Com o pacote instalado basta usar as classes do `bootstrap` . A imagem de baixo é um *plugin* do `bootstrap` já pronto.



Na nossa tela após a finalização da instalação do *pluggin* teremos:



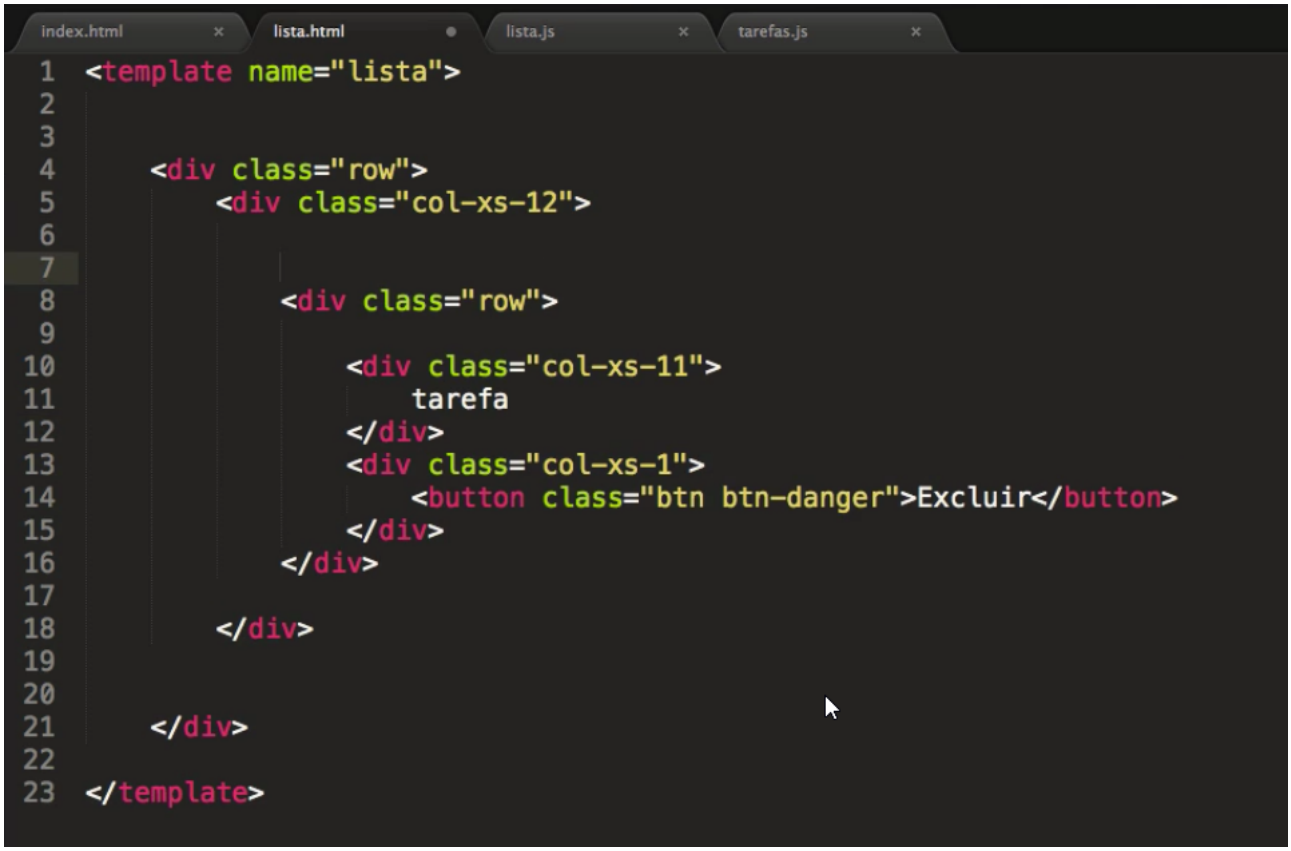
No "index.html" apagaremos o `h1` . Digitaremos dentro do `body` , `header class="container"` para criar um `container` . Na linha de baixo escreveremos `nav class="navbar navbar-default"` para adicionar um menu de navegação e ficaremos com um template *default*. Colocaremos um `div class="navbar-header"` e um `href="#" class="navbar-brand"` , esse `href` por enquanto não vai apontar para lugar nenhum . Chamaremos a aplicação de `Taskie` . Deixaremos também um espaço para o login `div class="navbar-collapse collapse"` . Acrescentaremos um `ul class="nav navbar-nav navbar-right` para ele ir para a direita. Colocaremos, ainda, dentro da lista `main class="container"` .

Teremos o seguinte:

```
<body>
    <header class="container">

        <nav class="navbar navbar-default">
            <div class="navbar-header">
                <a href="#" class="navbar-brand">
                    Taskie
                </a>
            </div>

            <div class="navbar-collapse collapse">
                <ul class="nav navbar-nav navbar-right">

                </ul>
            </div>
        </nav>
    </header>

    <main class="container">

    {{> lista}}

    </main>

</body>
```

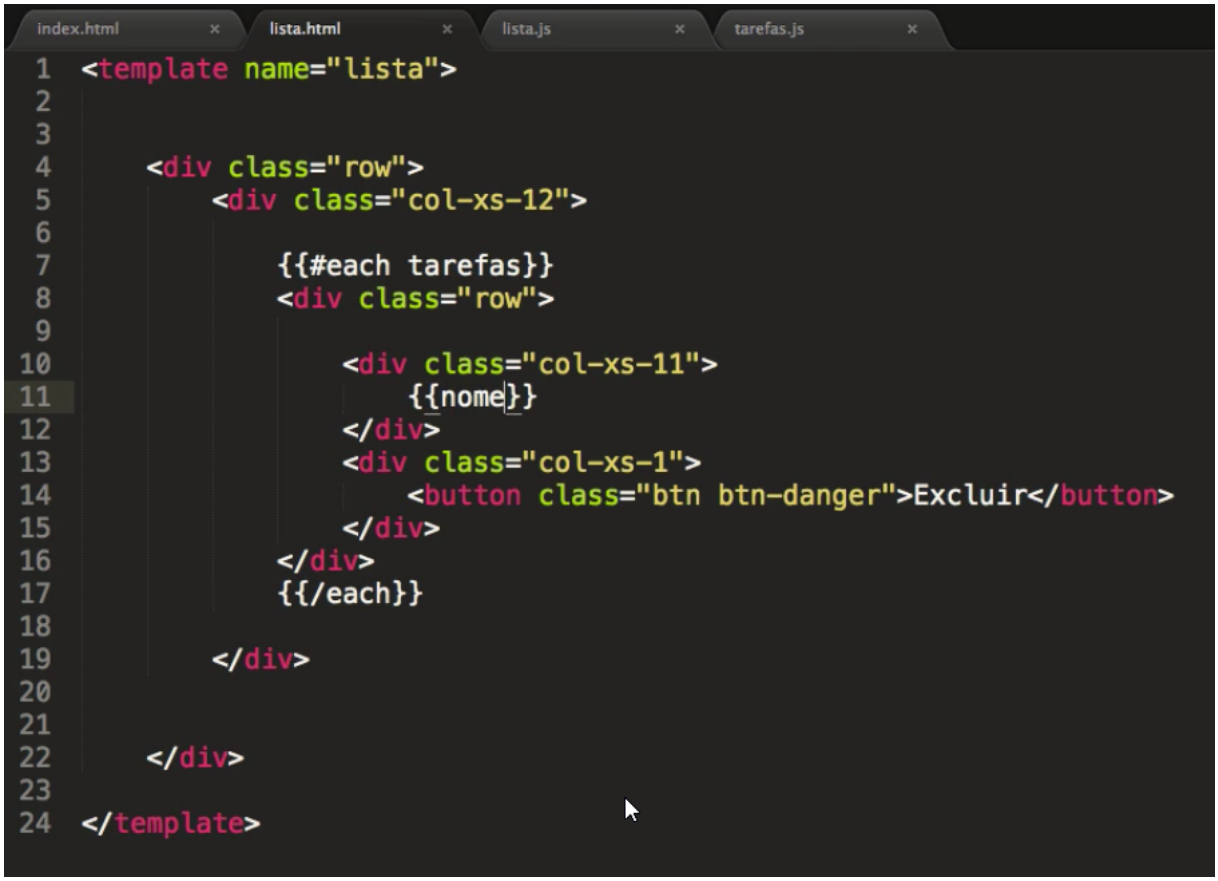
Vamos rodar a aplicação digitando "enter" e teremos o seguinte no nosso browser:

Vamos modificar a aparência da nossa lista. Através de uma `div class="row"` , na linha de baixo adicionamos `div class="col xs-12"` , para pegar tudo, e dentro dele colocamos um outro `div class="row"` e um `div class="col-xs-11"` . Na linha de baixo acrescentamos `tarefa` . Adicionaremos `div class="col-xs-1"` e o `button class="btn btn-danger>Excluir"` .

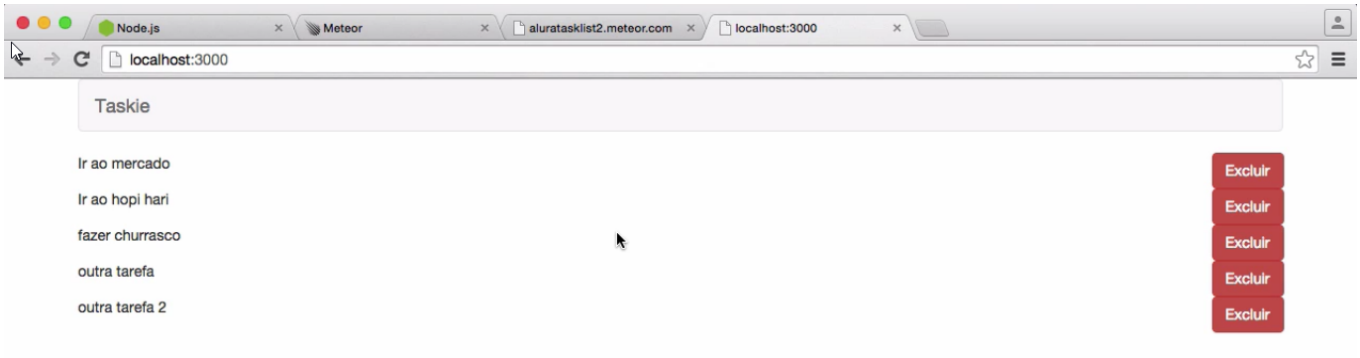


Esse código todo é o que temos que repetir com cada componente. Para indicar isso acrescentamos `{{#each tarefas}}` e colocaremos no lugar de "tarefa", o `{{nome}}` para trazer nossa lista de tarefas.

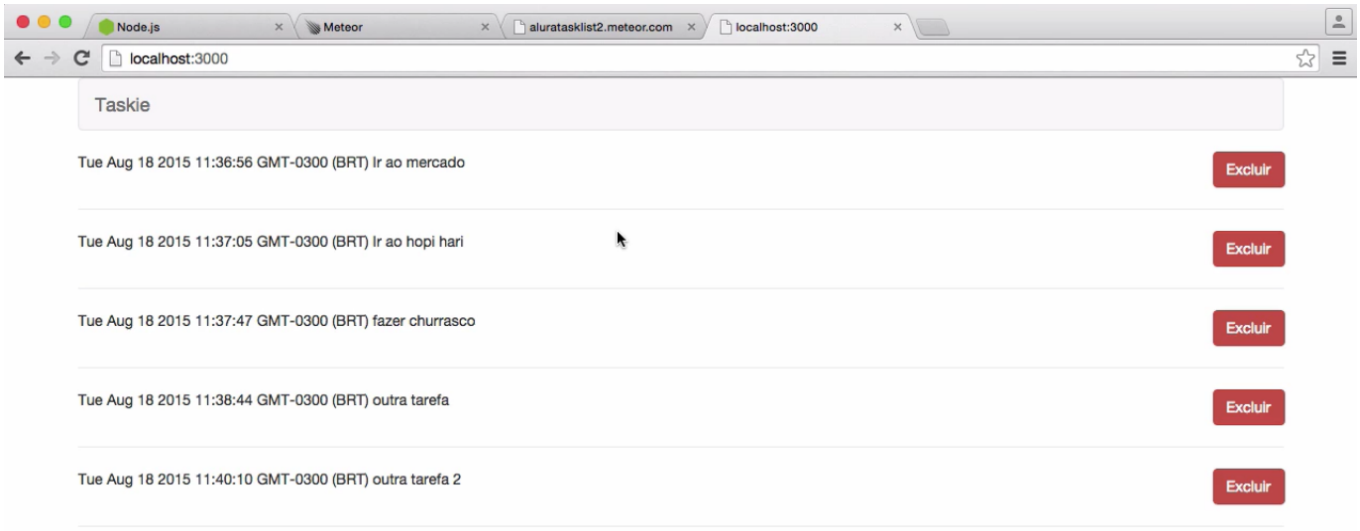
Ficaremos com:



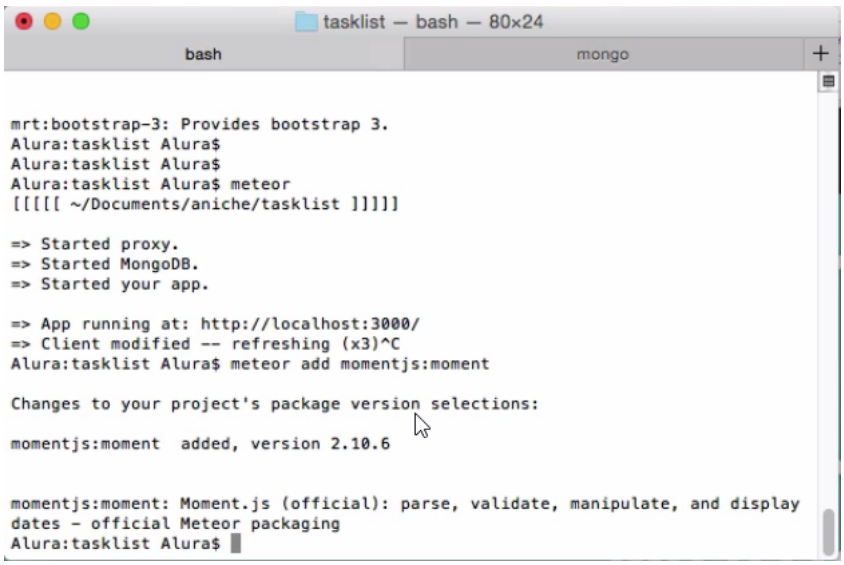
Nossa browser ficará da seguinte maneira:



Podemos acrescentar um `hr` para dar um espaçamento e podemos colocar ao lado de "nome", a "data" para ser exibida, `{{data}}` . Mas, se colocarmos apenas isso a data vai parecer muito grande em relação ao resto das informações.

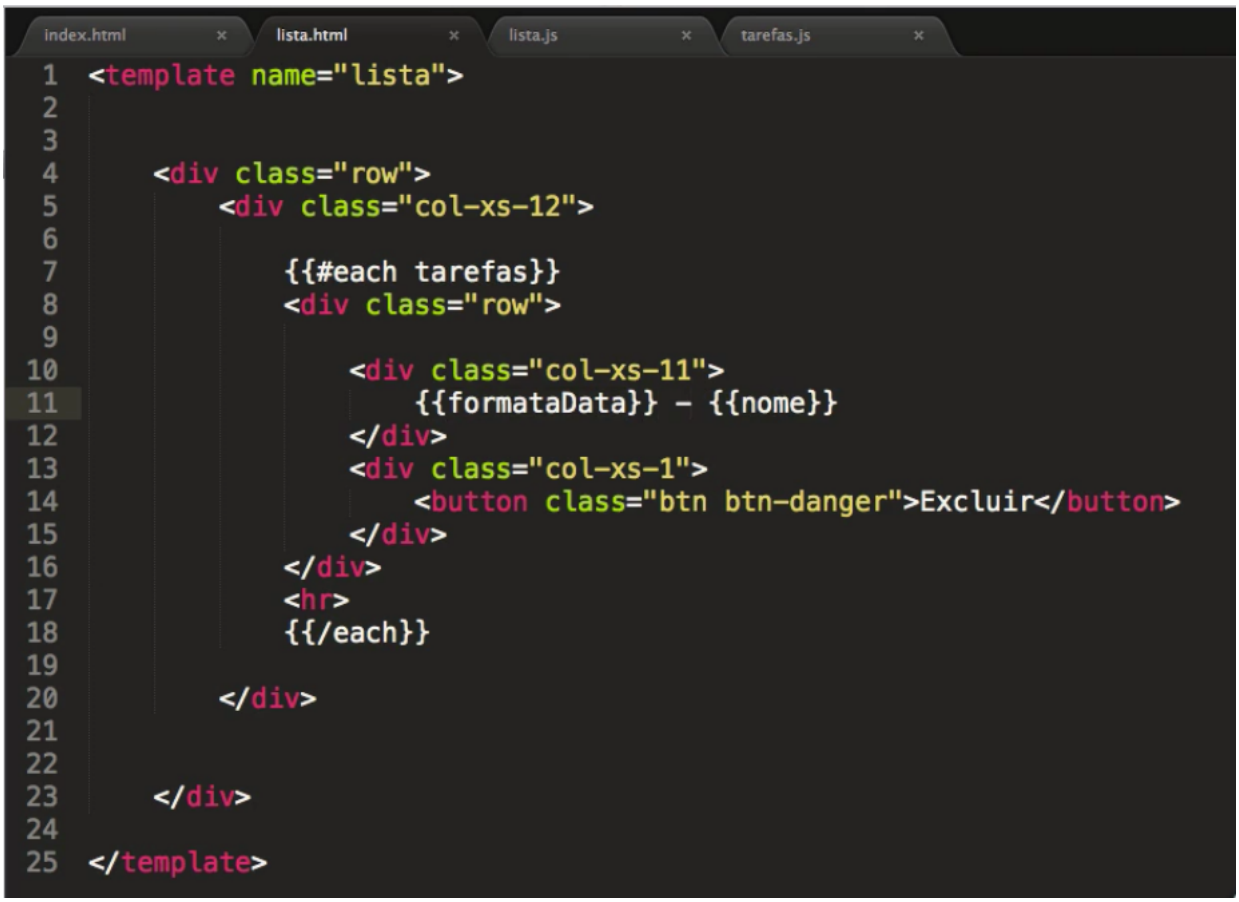


Existe uma biblioteca de *Java Script* que chama-se "Moment Js". Como é uma biblioteca muito comum no mundo do *javacript*, o *Meteor*, já tem uma maneira de fazer isso fácil. Vamos digitar no terminal, `meteor add momentjs:moment` , e ele vai instalar o pacote. Após baixar, acabamos de adquirir a biblioteca de `Moment.js` . Teremos:



Vamos usar a biblioteca. Tudo o que é `javascript` fica na `lista.js` . então, vamos na aba `lista.js` e adicionamos uma função, `formaData : function` e acrescentamos na próxima linha um `return moment(this.data)format('DD/MM/YYYY HH:mm')` . Lembrando que é necessário separar as várias funções por vírgula. Ficaremos com:

Voltamos na `lista.html` e alteramos o "data", por "formataData". Podemos acrescentar um hífen entre o "formataData" e "nome" e teremos `{{formataData}} - {{nome}}` . Assim, a data ficará separada das atividades por um hífen.



O *Meteor* sabe que o `this` do "lista.js" é cada uma das tarefas. O `this.data` é a data da tarefa e o *Meteor* a formata conforme o que inserimos no código `DD/MM/YYYY HH:mm` .

Vamos subir a aplicação de novo, digitando `meteor` no terminal. Aqui, é um costume, parar a aplicação quando um pacote novo for instalado, para não sujar o código e não bagunçar muito.

Teremos o seguinte: