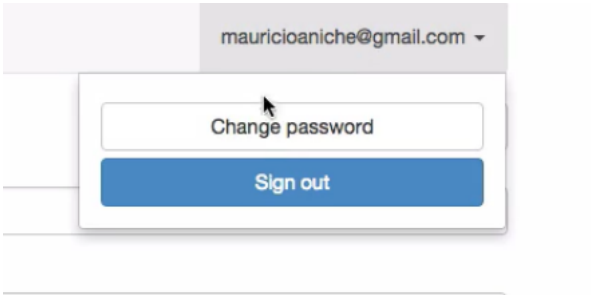
02

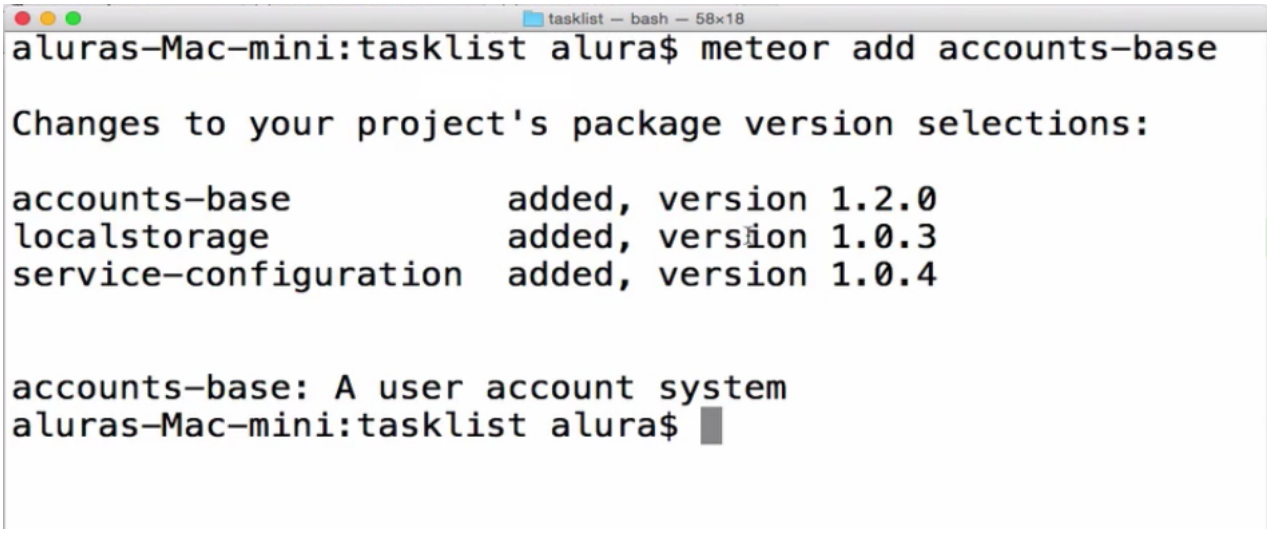
Sistema de autenticação e login

O que faremos agora é inserir novos usuários e fazer com que eles recuperem seus respectivos dados quando fizerem o *login*. Isto é, que acessem apenas os dados de suas próprias contas.

Na nossa aplicação final temos um menu que serve para entrar e sair da nossa aplicação. Este aqui:



Esse modelo de *Menu* nós já encontramos pronto em um *plugin* do *Meteor*. Para poder utilizá-lo basta instalar o *plugin* necessário. Para fazer isso vamos no terminal e escrevemos, `meteor add accounts-base` , que é um plugin básico para contas. Ele vai baixar esse *plugin*:



Agora, vamos baixar outro *plugin*, que serve para as senhas. Acrescentamos `meteor add accounts-password` . Fazemos a mesma coisa e aguardamos ele baixar:

```
accounts-base: A user account system
aluras-Mac-mini:tasklist alura$ meteor add accounts-passwo
rd

Changes to your project's package version selections:

accounts-password  added, version 1.1.1
email              added, version 1.0.6
npm-bcrypt         added, version 0.7.8_2
sha                added, version 1.0.3
srp                added, version 1.0.3


accounts-password: Password support for accounts
aluras-Mac-mini:tasklist alura$
```

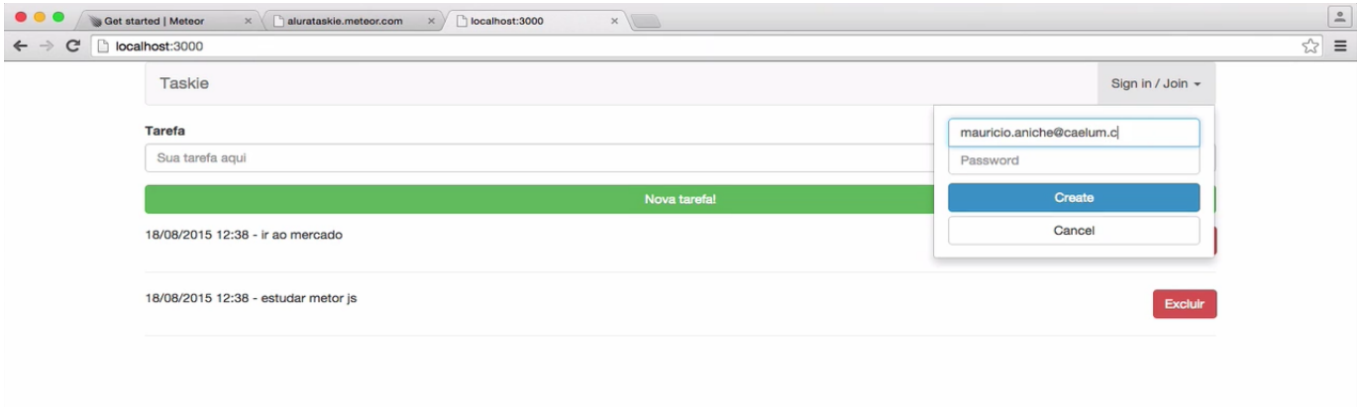
Adicionaremos um último *pluging*, que servirá para deixar nosso site mais bonito. Escreveremos `meteor add ian:accounts-ui-bootstrap-3` .

```
accounts-password: Password support for accounts
aluras-Mac-mini:tasklist alura$ meteor add ian:accounts-ui
-bootstrap-3

Changes to your project's package version selections:

anti:i18n                    added, version 0.4.3
ian:accounts-ui-bootstrap-3  added, version 1.2.79
stylus                       added, version 1.0.7


ian:accounts-ui-bootstrap-3: Bootstrap-styled accounts-ui
with multi-language support.
aluras-Mac-mini:tasklist alura$
```

Vamos na sublime, na aba "index.html" e colocamos nosso componente `{{> loginButtons}}` .

```
 6    <header class="container">
 7
 8        <nav class="navbar navbar-default">
 9            <div class="navbar-header">
10                <a href="#" class="navbar-brand">
11                    Taskie
12                </a>
13            </div>
14
15            <div class="navbar-collapse collapse">
16                <ul class="nav navbar-nav navbar-right">
17                {{> loginButtons}}
18                </ul>
19            </div>
20        </nav>
21    </header>
22
```

Vamos dar um refrash e ver o que acontece:

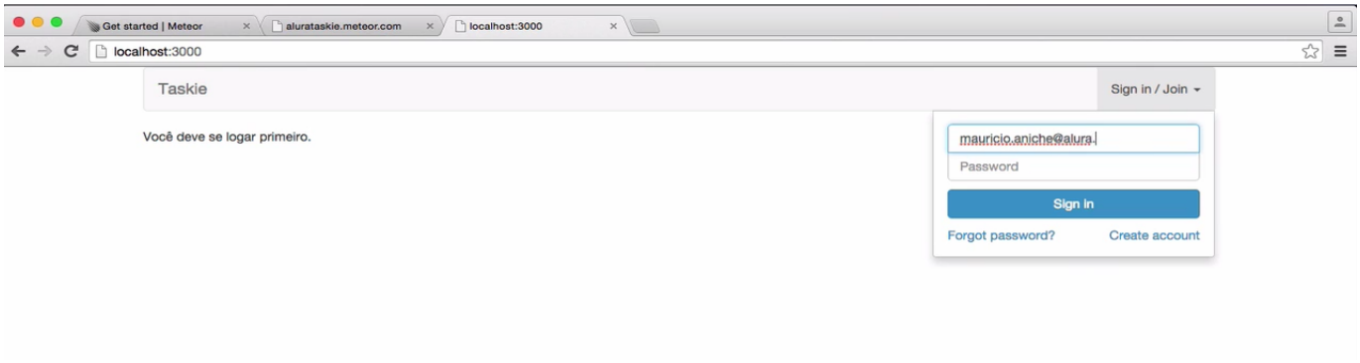


Essa parte está toda funcionando e resta, apenas, que nossa aplicação se comporte de acordo. Por exemplo, nossa aplicação não pode mostrar a lista de um usuário "deslogado" para outro usuário que está "logado".

Para isso vamos voltar na aba `index.html` e acrescentamos, acima de `{{> novo}}` , o seguinte: `{{#if currentUser}}` . Caso o usuário não esteja logado, podemos acrescentar uma mensagem, através do `{{else}}` , para que ele realize o "login" da aplicação. Digitamos `{{else}}` na linha de baixo de `{{> lista}}` e escreveremos a seguinte mensagem: "Você deve se logar primeiro". Ficaremos com:

```
24
25      {{#if currentUser}}
26          {{> novo}}
27          {{> lista}}
28      {{else}}
29          Você deve se logar primeiro.
30      {{/if}}
```

Se formos na nossa página da web veremos que a mensagem está aparente. Também, conseguimos logar e deslogar da nossa *single page*.



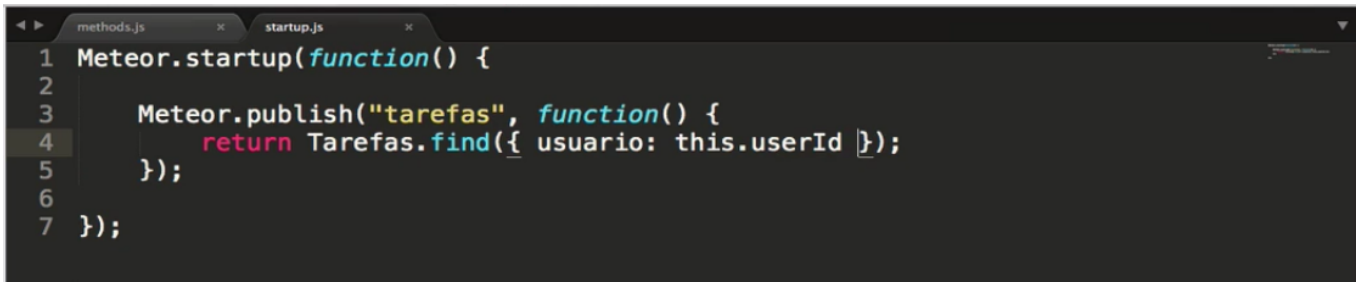
Mas, para resolver o problema de um usuário que consegue, ainda, acessar os dados de outros usuários precisamos separar algumas coisas no nosso código.

Vamos ao arquivo "novo.js" e acrescentamos no `Meteor.call` um `usuario` e a tarefa a qual pertence o usuário, `this.userId` . O `userId` , nos fornece o `Id` do usuário que está logado. Teremos:

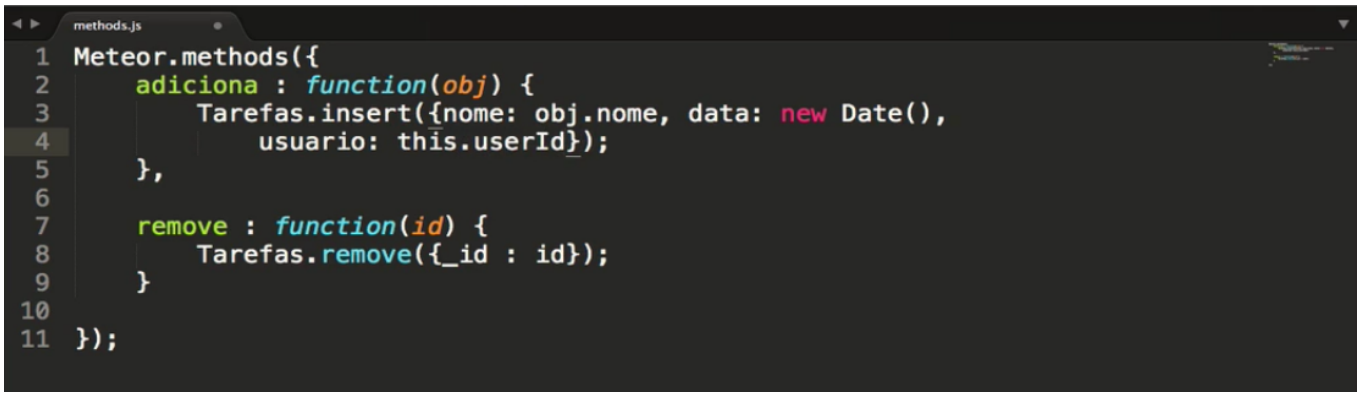
```
Meteor.call("adiciona", { nome: nome, usuario: this.userId } )
```

Agora, sempre que tivermos uma tarefa nova, salvaremos não apenas a tarefa como o usuário dela. A tarefa agora se compõe de "nome" e "usuário". Nossa tela ficará da seguinte maneira:

Nossa próxima etapa é modificar o `publish` que está na `startup.js` . Como não tínhamos descriminado o que queríamos publicar, se deixarmos como ele está, todos os dados de outros usuário serão também exibidos. Queremos publicar, apenas, os dados que fazem sentido. Por isso acrescentamos que queremos apenas os dados do usuário, para isso completaremos os parênteses do `Tarefa.find` com `usuario: this.userId` e ficaremos com: `return Tarefas.find({ usuario: this.userId })` . Isso fará um filtro para que os dados sejam apenas os do usuário logado.



Temos que inserir também no `methods.js` que a tarefa que queremos ver é aquela que pertence ao usuário que está logado. Por isso, acrescentamos, abaixo de `Tarefas.insert` um `usuario: this.userId` . Agora, toda vez que ele for abrir a aplicação vai mostrar, o nome, a data e também o usuário que está logado e a sua respectiva lista de tarefas. Teremos:



O *Mongo* tem os dados de diversos alunos, mas ele só irá mostrar aqueles alunos que estiverem logados. Vamos ver o exemplo de dois usuários:

O Guilherme: