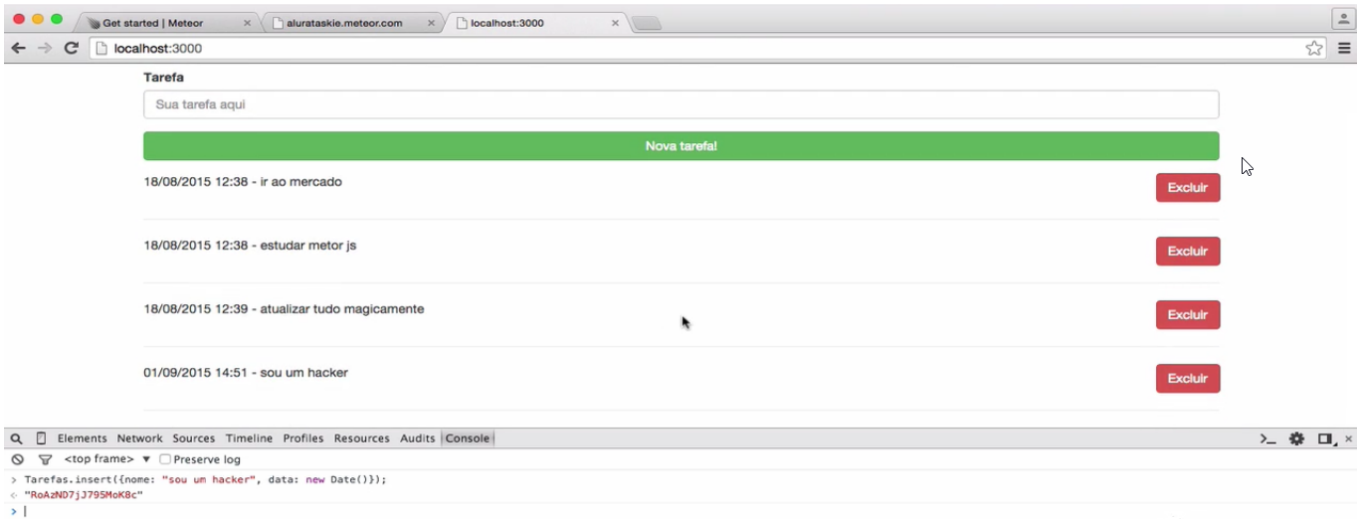
02

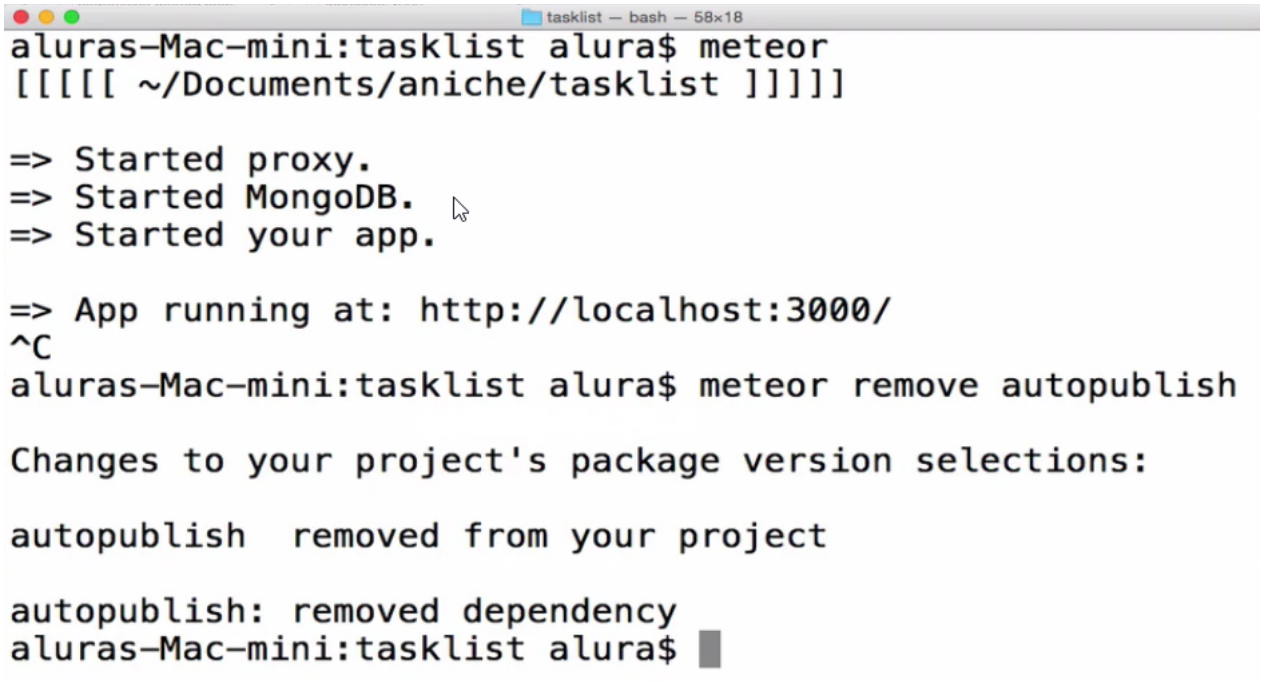
Publish e Subscribe

Nossa aplicação está quase pronta. Mas, ela ainda tem um grande problema.

Vamos ver qual é esse problema! Abrindo o *console* do *chrome* na nossa *single page aplication* podemos digitar um *javascript* e como, estamos em uma página que tem *Meteor*, então, todo o seu ambiente está carregado. Nada impede que a gente digite `Tarefa.insert{(nome: "sou um hacker", data: "new Date"()})` . Se dermos um enter veremos a falha de segurança que nossa aplicação possui, pois nossa *Taskie* será toda alterada.



O *Meteor* vêm com uma espécie de *autopublisher*. Ele pega o banco de dados de produção e cria uma cópia no cliente. Assim, as operações que fazemos no cliente, refletem no servidor. Para impedir que esse tipo de situação, a de um hacker modificando sua página, temos que retirar esse `autopublisher` . Vamos no terminal e digitamos `meteor remove autopublish` .



Só que, ao rodar a aplicação novamente, teremos problemas pois, ela já não vai mais estar inserindo as novas tarefas direito. Temos que fazer ela voltar a funcionar.

O primeiro passo é fazer a lista trazer os dados. Isto é, o servidor mandar os dados.

Vamos compreender...

O nome *autopublish* deixa claro como o *Meteor* funciona, o *publish* publica os dados para o "cliente", isto é, faz uma espécie de cópia dos dados do "servidor" e envia para o "cliente". É como se a nossa *single page aplication* tivesse uma "cópia" do *Mongo db*. O *autopublish* publica o banco de dados inteiro. Temos que evitar que isso ocorra e para ajeitar as coisas vamos realizar um procedimento manual.

Agora, vamos usar a pasta "server" e vamos criar um arquivo chamado `startup.js` . E escrevemos, `Meteor.startup(function)` .

Na próxima linha acrescentamos um `Meteor.publish` e nomearemos ele de "tarefas", também adicionaremos uma função e, assim, teremos `Meteor.publish("tarefas", function()` . Acrescentaremos, ainda, na próxima linha, um `return` , o retorno dessa função, que será copiado para o banco do cliente e como, por enquanto, queremos copiar tudo, completamos com "Tarefas.find". Ficaremos com, `return Tarefas.find({})` .

Se o servidor fez o *publish* o cliente, precisa fazer o *subscribe*. Na pasta "cliente" vou criar um arquivo "index.js" e vamos digitar a mesma coisa que fizemos no arquivo "startup". Escreveremos `Meteor.statup(function()` . Só que essa é a "startup" do cliente.

Em seguida, duas linhas abaixo do que já escrevemos, digitamos `Meteor.subscribe` e completaremos com o nome do quê queremos subescrever, no caso, tarefas. Teremos, `Meteor.subscribe("tarefas")` .

Agora, o *Mongo* do cliente tem uma cópia do banco.

Vamos reparar uma coisa, o `Tarefa.find` possui comportamentos distintos dependendo de onde ele estiver. Isto é, se o `Tarefas.find` estiver no cliente ele vai buscar pelo banco do cliente, mas se ele estivesse no servidor, ele buscaria o banco no servidor. Esse código é "isomorfo", ou seja, possui formas diferentes dependendo de onde ele está.

O próximo passo é fazer o "adiciona" funcionar. Para isso, teremos que mandar os dados para o servidor.

Vamos criar um novo arquivo no servidor chamado `methods.js` . O "Methods" é como se fosse um serviço da *web* que a aplicação vai disponibilizar para ela mesma.

Vamos escrever `Meteor.methods` e na próxima linha vamos adicionar um método `adiciona` que receberá um objeto. Teremos `adiciona : function(obj)` . Com esse objeto poderemos escrever `Tarefas.insert` e o nome que está no objeto, por exemplo, `obj.nome` e, para pegar a data do servidor, escrevemos `data: new Date` e teremos ao todo: `Tarefas.insert({nome:obj.nome, data: new Date})` .

Nossa tela ficará:

```
Meteor.methods({
    adiciona : function(obj) {
        Tarefas.insert({nome: obj.nome, data: new Date()});
    }

});
```

No código do cliente, no "novo.js" acrescentaremos `Metheor.call` abaixo do `//Tarefas.insert{(nome: nome, data: new Date()})` que devemos desativar, pois mudamos de onde vêm os dados. Como é para "adicionar" devemos completar o `Meteor.call` com `adiciona` entre os parênteses. Também temos que acrescentar um objeto, um mapa, que tem um atributo "nome" e o valor dele que é o campo "nome". Ficaremos com:

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

E ao todo teremos:

```
Template.novo.events({

    "submit form": function(e, template) {
        e.preventDefault();

        var input = $("#tarefa");
        var nome = input.val();

        //Tarefas.insert({nome: nome, data: new Date()});
        Meteor.call("adiciona", { nome: nome } );

        input.val("");
    }
});
```

O insert já não funcionará mais direto do código do cliente, pois bloqueamos o *autopublish*. Agora, teremos que chamar um método, que aqui chamamos de "adiciona" e no "cliente", esse método é consumido.

Observação: Se fizermos um teste e ele não funcionar podemos abrir o *console* para ver o que ocorreu. Por exemplo, ele poderia indicar que "Tarefa is not defined". Poderíamos voltar no "novo.js" e verificar que na verdade é "Tarefas" no plural e arrumar esse erro.

Então, já conseguimos concertar o erro do *autopublish* e, por fim, poderemos acrescentar novas tarefas sem se preocupar com a segurança.