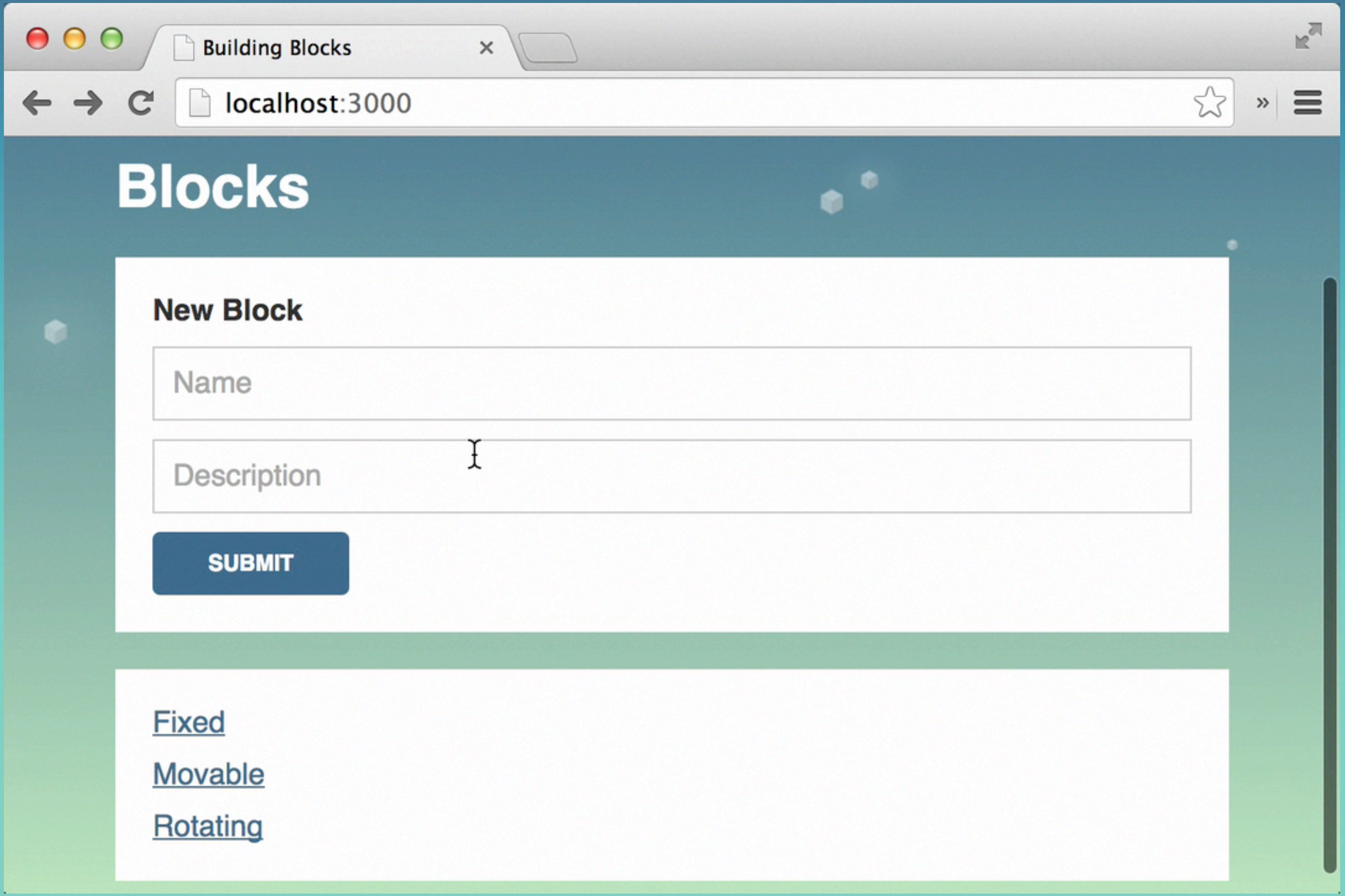
# POST Requests

Level 4 - Part I

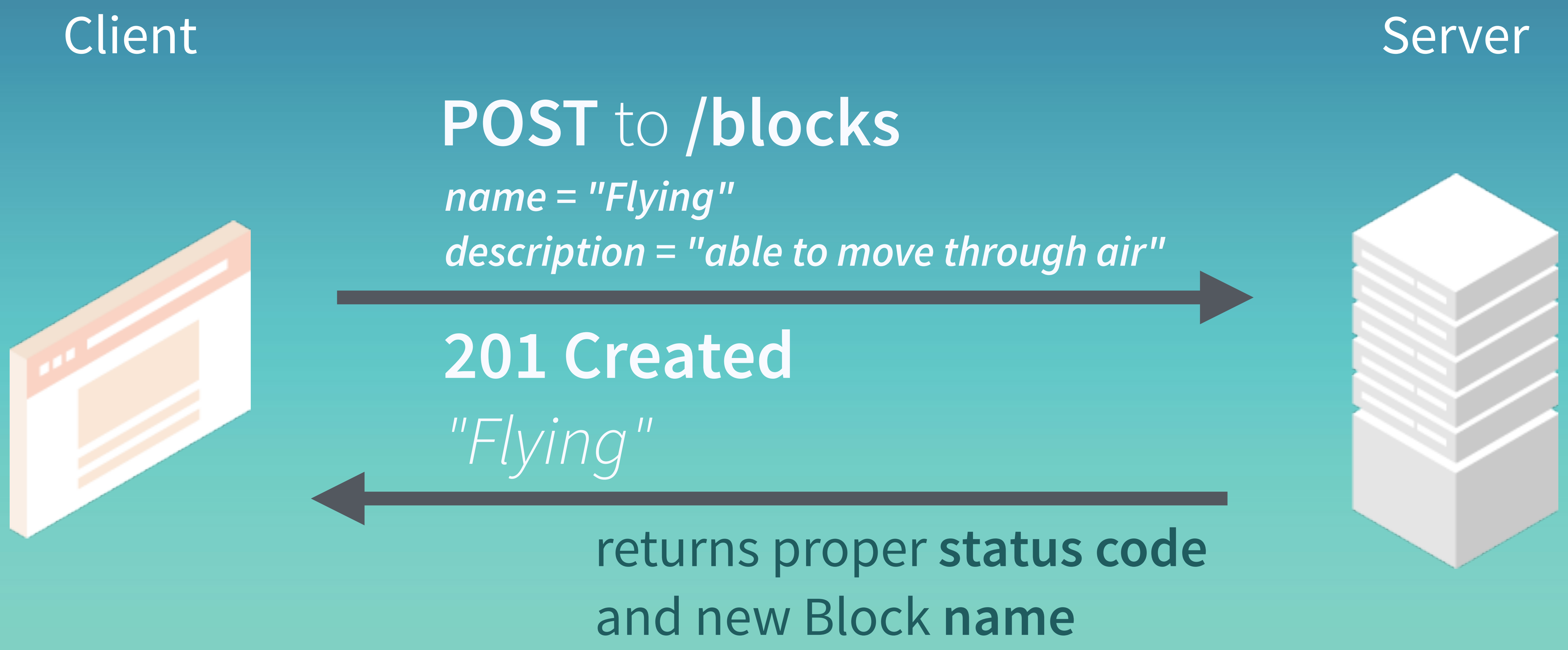
## Creating new Blocks

This is what we are going to do:

1. Add a new form
2. Create **POST** route



# Creating new Blocks



**POST** to **/blocks**

*name = "Flying"*
*description = "able to move through air"*

**201 Created**

*"Flying"*

returns proper **status code**
and new Block **name**

# Adding a form to index.html

Text field inputs will be needed for **name** and **description**

- app.js
- **public/**
  - index.html
  - jquery.js
  - client.js
  - style.css
  - bg-stars.png

index.html

```html
...
<body>

  <h1>Blocks</h1>

  <form>
    <legend>New Block</legend>
    <input name="name" placeholder="Name"><br/>
    <input name="description" placeholder="Description">
    <input type="Submit">
  </form>

  <ul class='block-list'></ul>
...
```

we'll define form attributes in **JavaScript**

# Submitting the form with JavaScript

Data is sent in a **POST** request to the **/blocks** endpoint

app.js
**public/**
index.html
jquery.js
client.js
style.css
bg-stars.png

client.js

```
$(function(){
  $.get('/blocks', appendToList);
  ...
  $('form').on('submit', function(event) {
    event.preventDefault();
    var form = $(this);
    var blockData = form.serialize();

    $.ajax({
      type: 'POST', url: '/blocks', data: blockData
    }).done(function(blockName){

    });
  });
  ...
});
```

transforms form data to **URL-encoded** notation

recently created block

# Updating the list with the new Block

We'll reuse the **appendToList** function from earlier to add new blocks to the list

- app.js
- **public/**
  - index.html
  - jquery.js
  - client.js
  - style.css
  - bg-stars.png

client.js

```
$(function(){
  $.get('/blocks', appendToList);
  ...
  $('form').on('submit', function(event) {
    event.preventDefault();
    var form = $(this);
    var blockData = form.serialize();

    $.ajax({
      type: 'POST', url: '/blocks', data: blockData
    }).done(function(blockName){
      appendToList(           );
    });
  });
  ...
});
```

same function being called

## Updating the list with the new Block

The **appendToList** function expects an **array** of Blocks

app.js
**public/**
index.html
jquery.js
**client.js**
style.css
bg-stars.png

client.js

```
$(function(){
  $.get('/blocks', appendToList);
  ...
  $('form').on('submit', function(event) {
    event.preventDefault();
    var form = $(this);
    var blockData = form.serialize();

    $.ajax({
      type: 'POST', url: '/blocks', data: blockData
    }).done(function(blockName){
      appendToList([blockName]);
    });
  });
  ...
});
```

array with the new block
as its single argument

## Clearing input fields after submission

We must clear the input text fields after posting the form

- app.js
- **public/**
  - index.html
  - jquery.js
  - client.js
  - style.css
  - bg-stars.png

client.js

```
$(function(){
  ...
  $('form').on('submit', function(event) {
    event.preventDefault();
    var form = $(this);
    var blockData = form.serialize();

    $.ajax({
      type: 'POST', url: '/blocks', data: blockData
    }).done(function(blockName){
      appendToList([blockName]);
      form.trigger('reset');
    });
  });
  ...
});
```

cleans up form text input fields

## Adding links to Blocks

client.js

- app.js
- **public/**
  - index.html
  - jquery.js
  - client.js
  - style.css
  - bg-stars.png

```
$(function(){
  ...
  $('form').on('submit', function(event) {
    ...
  });

  function appendToList(blocks) {
    var list = [];
    var content, block;
    for(var i in blocks){
      block = blocks[i];
      content = '<a href="/blocks/'+block+'">'+block+'</a>';
      list.push($('<li>', { html: content }));
    }

    $('.block-list').append(list)
  }
});
```

link to each Block's description

## Posting

Parsing depends on a middleware **not** shipped with Express

- app.js
- **public/**
  - index.html
  - jquery.js
  - client.js
  - style.css
  - bg-stars.png

```
$ npm install body-parser
```

app.js

```
var express = require('express');
var app = express();

var bodyParser = require('body-parser');
var parseUrlencoded = bodyParser.urlencoded({ extended: false });

var blocks = { ... };


...
```

forces the use of the native **querystring** Node library

# Creating a POST route

Routes can take **multiple handlers** as arguments and will call them **sequentially**

app.js

```
var express = require('express');
var app = express();

var bodyParser = require('body-parser');
var parseUrlencoded = bodyParser.urlencoded({ extended: false });

var blocks = { ... };

app.post('/blocks', parseUrlencoded, function(request, response) {

});
...
```

will run first

will run second

Using multiple route handlers is useful for **re-using middleware** that load resources, perform validations, authentication, etc.

## Reading request data

Form submitted data can be accessed through **request.body**

app.js

```
var express = require('express');
var app = express();

var bodyParser = require('body-parser');
var parseUrlencoded = bodyParser.urlencoded({ extended: false });

var blocks = { ... };

 app.post('/blocks', parseUrlencoded, function(request, response) {
   var newBlock = request.body;



 });
 ...
```

returns form data

## Creating a new Block

The form elements are parsed to object properties, **name** and **description**

app.js

```
var express = require('express');
var app = express();

var bodyParser = require('body-parser');
var parseUrlencoded = bodyParser.urlencoded({ extended: false });

var blocks = { ... };

 app.post('/blocks', parseUrlencoded, function(request, response) {
   var newBlock = request.body;
   blocks[newBlock.name] = newBlock.description;


 });
 ...
```

adds new block
to the blocks object

## Responding from a POST request

The response includes proper **status code** and the block **name**

app.js

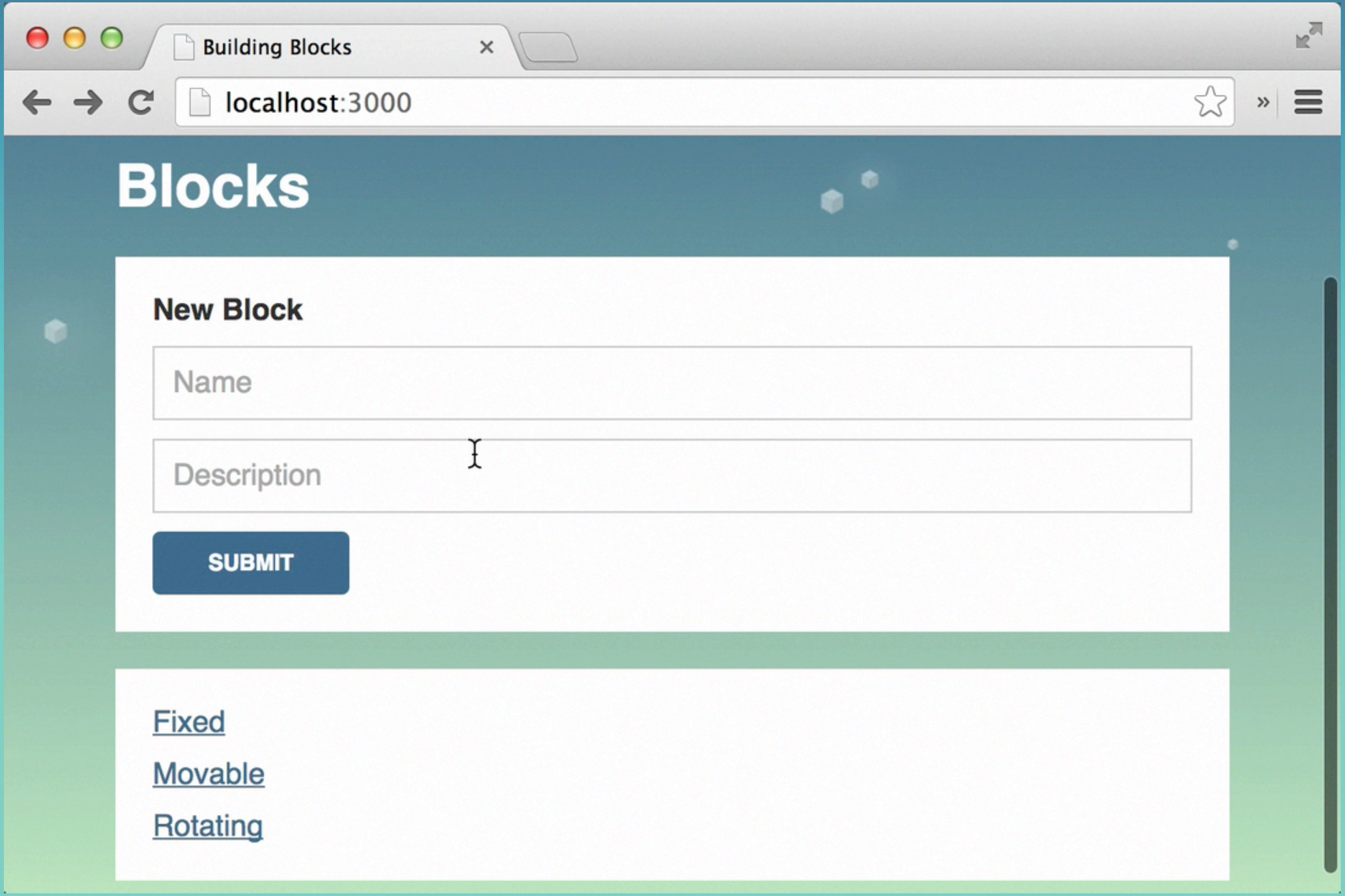
```
var express = require('express');
var app = express();

var bodyParser = require('body-parser');
var parseUrlencoded = bodyParser.urlencoded({ extended: false });

var blocks = { ... };

 app.post('/blocks', parseUrlencoded, function(request, response) {
   var newBlock = request.body;
   blocks[newBlock.name] = newBlock.description;

   response.status(201).json(newBlock.name);
 });
 ...
```

responds with new block **name**

sets the **201 Created** status code

## Testing our new feature

And it works!



Building Blocks

localhost:3000

Blocks

New Block

Name

Description

SUBMIT

Fixed

Movable

Rotating