# Getting Started with IBM Bluemix

## Sample Code for the FizzBuzz Workshop

# Sample code from the FizzBuzz files

## Fizzbuzz.js

```
var FizzBuzz = function (){
};

FizzBuzz.prototype.divisibleBy = function(number, divisor) {
        return number % divisor === 0;
};

FizzBuzz.prototype.convertToFizzBuzz = function(number) {
          if (this.divisibleBy(number, 15)) {
            return "FizzBuzz";
          }

          if (this.divisibleBy(number, 3)) {
            return "Fizz";
          }

          if (this.divisibleBy(number, 5)) {
            return "Buzz";
          }

          return number.toString();
};

FizzBuzz.prototype.convertRangeToFizzBuzz = function(start, end) {
          var result = [];
       var from = parseInt(start);
       var to = parseInt(end);

          for (var i = from; i <= to; i++) {
            result.push(this.convertToFizzBuzz(i));
          }

          return result;
};

module.exports = FizzBuzz;
```

## cachefizzbuzz.js

```
var sinon = require("sinon");
//var CacheFizzBuzz = require("../cachefizzbuzzpromise.js");
var CacheFizzBuzz = require("../cachefizzbuzz.js");

var testResult1 = {
    "from" : "1",
    "to" : "20",
    "result" : [ "1", "2", "Fizz", "4", "Buzz", "Fizz", "7", "8", "Fizz", "Buzz",
                 "11", "Fizz", "13", "14", "FizzBuzz", "16", "17", "Fizz", "19", "Buzz" ]
};

var testResult2 = {
    "from" : "2",
    "to" : "21",
    "result" : ["2", "Fizz", "4", "Buzz", "Fizz", "7", "8", "Fizz", "Buzz",
                "11", "Fizz", "13", "14", "FizzBuzz", "16", "17", "Fizz", "19", "Buzz", "Fizz"]
};

var DBResult1 = {
    "total_rows" : 7,
    "offset" : 6,
    "rows" : [ {
      "id" : "35523244d141fb56c2e6b8dfa58d7fed",
      "key" : [ "1", "20" ],
      "value" : null,
      "doc" : {
        "_id" : "35523244d141fb56c2e6b8dfa58d7fed",
        "_rev" : "1-a0337a71e877726cb8fda7d6ceeb2bfc",
        "from" : "1",
        "to" : "20",
        "result" : [ "1", "2", "Fizz", "4", "Buzz", "Fizz", "7", "8", "Fizz",
                     "Buzz", "11", "Fizz", "13", "14", "FizzBuzz", "16", "17", "Fizz",
                     "19", "Buzz" ]
```

```
      }
    } ]
};

var DBResult2 = {
    "total_rows" : 7,
    "offset" : 7,
    "rows" : []
};

describe("CacheFizzbuzz", function() {
      var f = new CacheFizzBuzz("http://user:password@localhost/fizzbuzz");

      describe("dbStoreCalculatedResult()", function() {
        it("calls Cloudant to store the doc", function() {

          var mock = sinon.mock(f._Cloudant);
          mock.expects("insert").withArgs(testResult1).once();

          f._dbStoreCalculatedResult(testResult1);

          mock.verify();
          mock.restore();
        });
      });

      describe("fizzBuzzRange()", function() {
            var cbFunction = function(data) {
            };

            it("calls Cloudant to get record from the fb/range view in DB",
                function() {
                  var stub = sinon.stub(f._Cloudant, "view");

                  f.fizzBuzzRange("1", "20", cbFunction);

                  expect(stub.withArgs('fb', 'range', {
                    include_docs : true, key : [ "1", "20" ]
                  }, sinon.match.any).calledOnce).to.be
                  .eql(true,
                  "Expected cloudant.view to be called only once with correct parameters");

                  f._Cloudant.view.restore();
                });
          });

      describe("processDBResult()", function() {
        var cbFunction = sinon.spy();
        var fizzBuzzSpy = null;

        beforeEach( function() {
          fizzBuzzSpy = sinon.spy(f._fizzbuzz, "convertRangeToFizzBuzz");
        });

        afterEach( function() {
          cbFunction.reset();
          f._fizzbuzz.convertRangeToFizzBuzz.restore();
        });

            it("processes the results from Cloudant with a valid result set",
                function() {
                  var storeResultsSpy = sinon.spy(f, "_dbStoreCalculatedResult");

                  f._processDBResult(false, DBResult1, "1", "20", cbFunction);

                  expect(cbFunction.withArgs(testResult1).calledOnce).to.be
                  .eql(true, "Expected processDBResult to parse DB return");
                  expect(fizzBuzzSpy).callCount(0);
                  expect(storeResultsSpy).callCount(0);

                  f._dbStoreCalculatedResult.restore();
                });

            it("calculates the results when no results found then saves to DB",
                function() {
                  storeResultsStub = sinon.stub(f, "_dbStoreCalculatedResult");

                  f._processDBResult(false, DBResult2, "2", "21", cbFunction);
```

```
                    expect(cbFunction.withArgs(testResult2).calledOnce).to.be
                    .eql(true,"Expected calculated result when no data from DB");
                    expect(fizzBuzzSpy.withArgs("2", "21").calledOnce).to.be
                    .eql(true, "convertRangeToFizzBuzz should be called to calculate results");
                    expect(storeResultsStub.withArgs(testResult2).calledOnce).to.be
                    .eql(true, "Expect calculated restuls to be stored in DB");

                    f._dbStoreCalculatedResult.restore();
                  });

              it("calculates results when DB error, but doesn't store results",
                  function() {
                    storeResultsSpy = sinon.spy(f, "_dbStoreCalculatedResult");

                    f._processDBResult(true, DBResult1, "1", "20", cbFunction);

                    expect(cbFunction.withArgs(testResult1).calledOnce).to.be
                    .eql(true, "Expected processDBResult create empty result with DB error");
                    expect(fizzBuzzSpy.withArgs("1", "20").calledOnce).to.be
                    .eql(true, "convertRangeToFizzBuzz should be called to calculate results");
                    expect(storeResultsSpy).callCount(0);

                    f._dbStoreCalculatedResult.restore();
                  });
            });

        //remaining tests go above here
});
```

## server.js

```
var express = require("express");
var app = express();
var FizzBuzz = require("./fizzbuzz");
var CacheFizzBuzz = require("./cachefizzbuzz");

var server_port = process.env.VCAP_APP_PORT || 3000;
var server_host = process.env.VCAP_APP_HOST || "localhost";
var dbURL = "";
if (process.env.VCAP_SERVICES) {
        var env = JSON.parse(process.env.VCAP_SERVICES);
        dbURL = env.cloudantNoSQLDB[0].credentials.url + "/fizzbuzz";
} else dbURL = "http://localhost:5984/fizzbuzz";

app.get("/fizzbuzz_range/:from/:to", function (req, res) {
  var fizzbuzz = new FizzBuzz();
  var from = req.params.from;
  var to = req.params.to;

  res.send({
    from: from,
    to: to,
    result: fizzbuzz.convertRangeToFizzBuzz(from, to)
  });
});

app.get("/cache_fizzbuzz_range/:from/:to", function (req, res) {
  var cachefizzbuzz = new CacheFizzBuzz(dbURL);
  var from = req.params.from;
  var to = req.params.to;

  cachefizzbuzz.fizzBuzzRange(from, to, function(data) {
          res.send(data);
  });
});

var server = app.listen(server_port, server_host, function () {

  var host = server.address().address;
  var port = server.address().port;

  console.log("Example app listening at http://%s:%s", host, port);

});
```

## test/fizzbuzz.test.js

```
var sinon = require("sinon");
var FizzBuzz = require("../fizzbuzz.js");

describe("Fizzbuzz", function() {
  var f = new FizzBuzz();

  describe("divisibleBy()", function() {
            it("when divisible", function() {
              expect(f.divisibleBy(3, 3)).to.be.eql(true);
            });

            it("when not divisible", function() {
              expect(f.divisibleBy(3, 2)).to.be.eql(false);
            });
  });

  describe("convertToFizzBuzz()", function() {
            it("when divisible by 3", function() {
              expect(f.convertToFizzBuzz(3)).to.be.equal("Fizz");
              expect(f.convertToFizzBuzz(6)).to.be.equal("Fizz");
            });

            it("when divisible by 5", function() {
              expect(f.convertToFizzBuzz(5)).to.be.equal("Buzz");
              expect(f.convertToFizzBuzz(10)).to.be.equal("Buzz");
            });

            it("when divisible by 15", function() {
              expect(f.convertToFizzBuzz(15)).to.be.equal("FizzBuzz");
              expect(f.convertToFizzBuzz(30)).to.be.equal("FizzBuzz");
            });

            it("when not divisible by 3, 5 or 15", function() {
                      expect(f.convertToFizzBuzz(4)).to.be.equal("4");
                      expect(f.convertToFizzBuzz(7)).to.be.equal("7");
            });
          });

  describe("convertRangeToFizzBuzz()", function() {
            it("returns in correct order", function() {
              expect(f.convertRangeToFizzBuzz("1", "3")).to.be.eql(["1", "2", "Fizz"]);
            });

            it("applies FizzBuzz to every number in the range", function() {
                var spy = sinon.spy(f, "convertToFizzBuzz");

                f.convertRangeToFizzBuzz("9", "50");

                for (var i = 9; i <= 50; i++) {
                  expect(spy.withArgs(i).calledOnce).to.be.eql(true, "Expected convertToFizzBuzz to
be called with " + i);
                }
                f.convertToFizzBuzz.restore();
              });
         });
});
```

## test/cachefizzbuzz.test.js

```
var sinon = require("sinon");
var CacheFizzBuzz = require("../cachefizzbuzz.js");

var testResult1 = {
    "from" : "1",
    "to" : "20",
    "result" : [ "1", "2", "Fizz", "4", "Buzz", "Fizz", "7", "8", "Fizz", "Buzz",
                 "11", "Fizz", "13", "14", "FizzBuzz", "16", "17", "Fizz", "19", "Buzz" ]
};

var testResult2 = {
    "from" : "2",
    "to" : "21",
    "result" : ["2", "Fizz", "4", "Buzz", "Fizz", "7", "8", "Fizz", "Buzz",
                "11", "Fizz", "13", "14", "FizzBuzz", "16", "17", "Fizz", "19", "Buzz", "Fizz"]
};
```

```
var DBResult1 = {
    "total_rows" : 7,
    "offset" : 6,
    "rows" : [ {
      "id" : "35523244d141fb56c2e6b8dfa58d7fed",
      "key" : [ "1", "20" ],
      "value" : null,
      "doc" : {
        "_id" : "35523244d141fb56c2e6b8dfa58d7fed",
        "_rev" : "1-a0337a71e877726cb8fda7d6ceeb2bfc",
        "from" : "1",
        "to" : "20",
        "result" : [ "1", "2", "Fizz", "4", "Buzz", "Fizz", "7", "8", "Fizz",
                     "Buzz", "11", "Fizz", "13", "14", "FizzBuzz", "16", "17", "Fizz",
                     "19", "Buzz" ]
      }
    } ]
};

var DBResult2 = {
    "total_rows" : 7,
    "offset" : 7,
    "rows" : []
};

describe("CacheFizzbuzz", function() {
      var f = new CacheFizzBuzz("http://user:password@localhost/fizzbuzz");

      describe("dbStoreCalculatedResult()", function() {
        it("calls Cloudant to store the doc", function() {

          var mock = sinon.mock(f._Cloudant);
          mock.expects("insert").withArgs(testResult1).once();

          f._dbStoreCalculatedResult(testResult1);

          mock.verify();
          mock.restore();
        });
      });

      describe("fizzBuzzRange()", function() {
            var cbFunction = function(data) {
            };

            it("calls Cloudant to get record from the fb/range view in DB",
                function() {
                  var stub = sinon.stub(f._Cloudant, "view");

                  f.fizzBuzzRange("1", "20", cbFunction);

                  expect(stub.withArgs('fb', 'range', {
                    include_docs : true, key : [ "1", "20" ]
                  }, sinon.match.any).calledOnce).to.be
                  .eql(true,
                  "Expected cloudant.view to be called only once with correct parameters");

                  f._Cloudant.view.restore();
                });
          });

      describe("processDBResult()", function() {
        var cbFunction = sinon.spy();
        var fizzBuzzSpy = null;

        beforeEach( function() {
          fizzBuzzSpy = sinon.spy(f._fizzbuzz, "convertRangeToFizzBuzz");
        });

        afterEach( function() {
          cbFunction.reset();
          f._fizzbuzz.convertRangeToFizzBuzz.restore();
        });

            it("processes the results from Cloudant with a valid result set",
                function() {
                  var storeResultsSpy = sinon.spy(f, "_dbStoreCalculatedResult");
```

```
                f._processDBResult(false, DBResult1, "1", "20", cbFunction);

                expect(cbFunction.withArgs(testResult1).calledOnce).to.be
                .eql(true, "Expected processDBResult to parse DB return");
                expect(fizzBuzzSpy).callCount(0);
                expect(storeResultsSpy).callCount(0);

                f._dbStoreCalculatedResult.restore();
              });

            it("calculates the results when no results found then saves to DB",
                function() {
                  storeResultsStub = sinon.stub(f, "_dbStoreCalculatedResult");

                  f._processDBResult(false, DBResult2, "2", "21", cbFunction);

                  expect(cbFunction.withArgs(testResult2).calledOnce).to.be
                  .eql(true,"Expected calculated result when no data from DB");
                  expect(fizzBuzzSpy.withArgs("2", "21").calledOnce).to.be
                  .eql(true, "convertRangeToFizzBuzz should be called to calculate results");
                  expect(storeResultsStub.withArgs(testResult2).calledOnce).to.be
                  .eql(true, "Expect calculated restuls to be stored in DB");

                  f._dbStoreCalculatedResult.restore();
                });

            it("calculates results when DB error, but doesn't store results",
                function() {
                  storeResultsSpy = sinon.spy(f, "_dbStoreCalculatedResult");

                  f._processDBResult(true, DBResult1, "1", "20", cbFunction);

                  expect(cbFunction.withArgs(testResult1).calledOnce).to.be
                  .eql(true, "Expected processDBResult create empty result with DB error");
                  expect(fizzBuzzSpy.withArgs("1", "20").calledOnce).to.be
                  .eql(true, "convertRangeToFizzBuzz should be called to calculate results");
                  expect(storeResultsSpy).callCount(0);

                  f._dbStoreCalculatedResult.restore();
                });
          });

      //remaining tests go above here
});
```