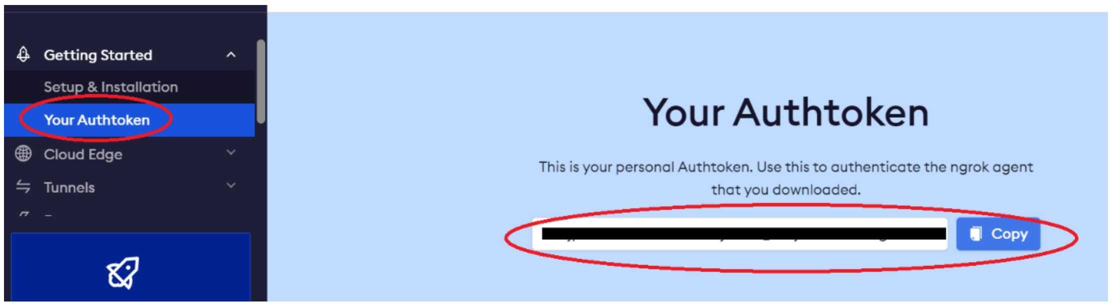
Omilia Mini-Apps: Accessing Data

Option 1: SoapUI (Easiest, most direct method)

This is the easier, more time-effective solution to test against **predefined data sets**. It will serve as a means to mock the responses from a given API. An example of this is if you have a project with at least one fully defined response provided by the client, and the expected variations of said response that would make an impact on how your application responds.

1. Download and install **SoapUI** (*SoapUI Open Source*): https://www.soapui.org/downloads/soapui/
2. Download **ngrok**: https://ngrok.com/download
3. Sign up for a free ngrok account: https://dashboard.ngrok.com/signup
   a. Get your token

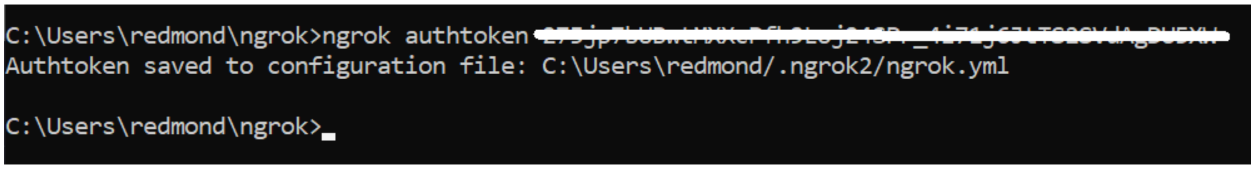


Your Authtoken

This is your personal Authtoken. Use this to authenticate the ngrok agent that you downloaded.

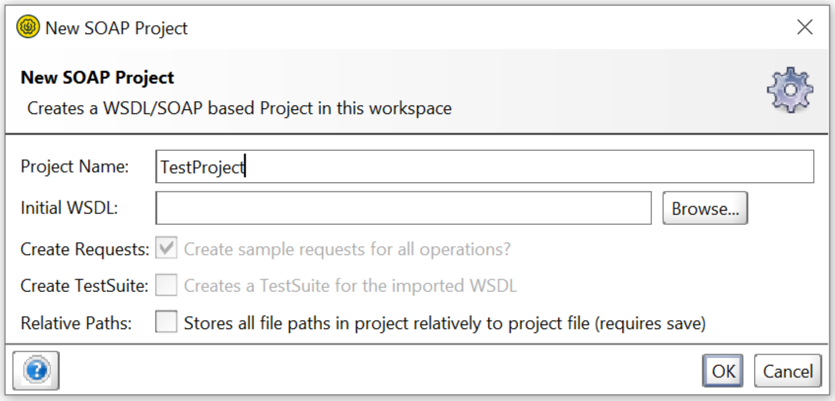
[redacted]

   b. Open CMD, navigate to the folder where ngrok.exe is found
   c. Authenticate your ngrok using the command*: ngrok authtoken* ***insertauthtokenhere***



```
C:\Users\redmond\ngrok>ngrok authtoken [redacted]
Authtoken saved to configuration file: C:\Users\redmond/.ngrok2/ngrok.yml

C:\Users\redmond\ngrok>_
```

4. Open SoapUI. Create New SOAP Project

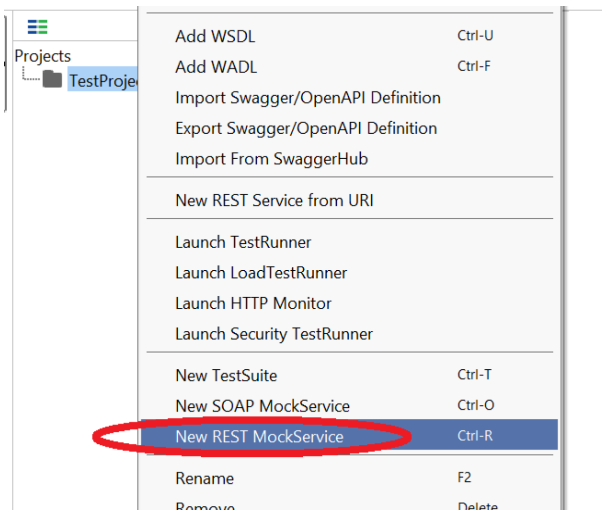


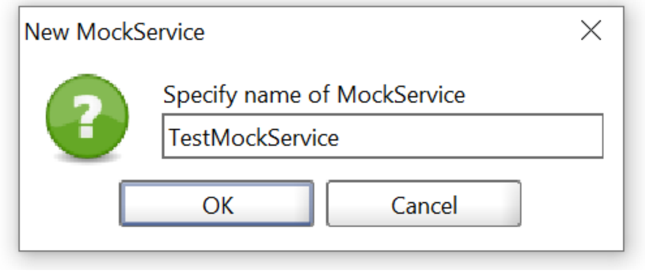
New SOAP Project

New SOAP Project
Creates a WSDL/SOAP based Project in this workspace

Project Name:	TestProject
Initial WSDL:	Browse...
Create Requests:	✓ Create sample requests for all operations?
Create TestSuite:	Creates a TestSuite for the imported WSDL
Relative Paths:	Stores all file paths in project relatively to project file (requires save)

5. Right-click project, create new REST MockService

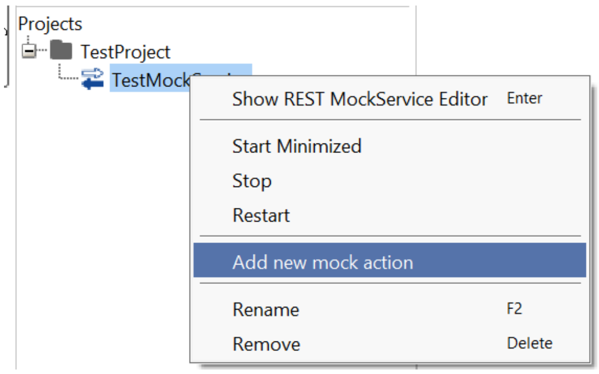

Projects
TestProje
Add WSDL Ctrl-U
Add WADL Ctrl-F
Import Swagger/OpenAPI Definition
Export Swagger/OpenAPI Definition
Import From SwaggerHub
New REST Service from URI
Launch TestRunner
Launch LoadTestRunner
Launch HTTP Monitor
Launch Security TestRunner
New TestSuite Ctrl-T
New SOAP MockService Ctrl-O
New REST MockService Ctrl-R
Rename F2
Remove Delete

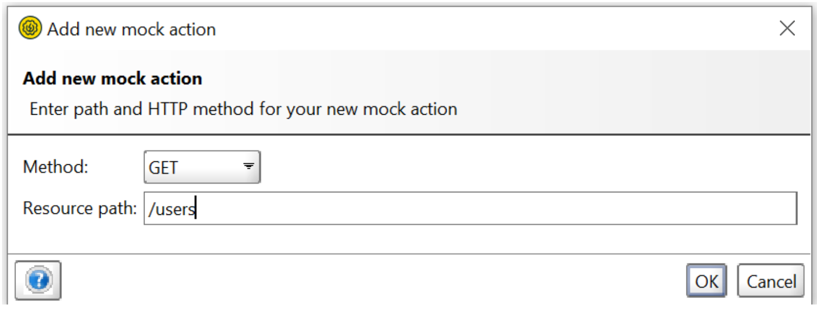


Specify name of MockService

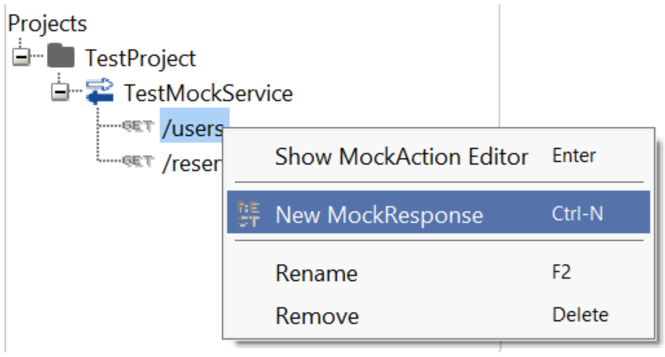
TestMockService

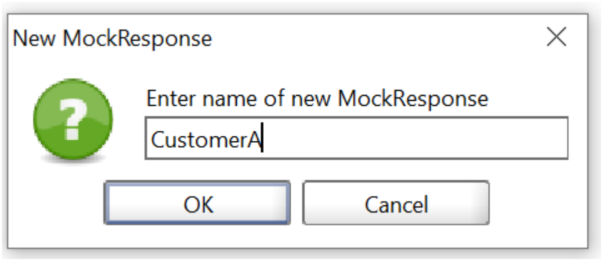
6. Right-click the new MockService, add a new mock action. For this example, we will add mock actions to retrieve users, and reservations.

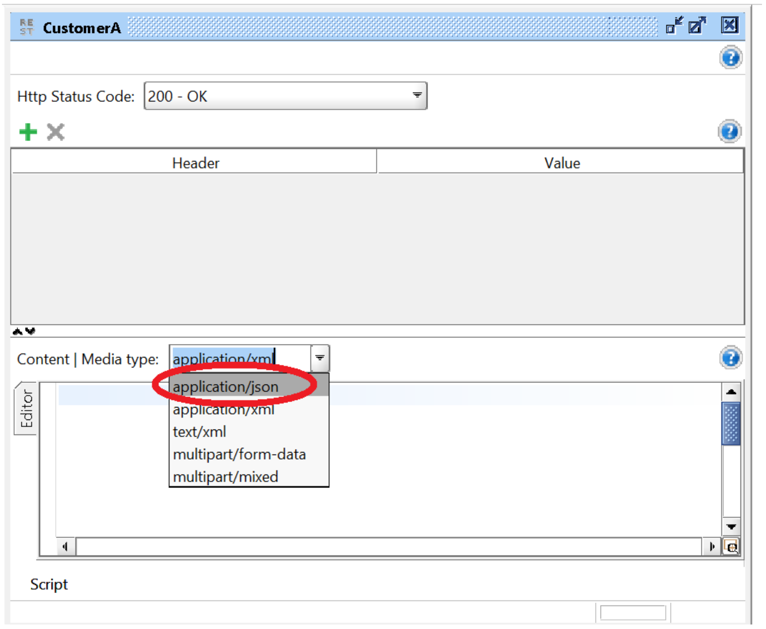


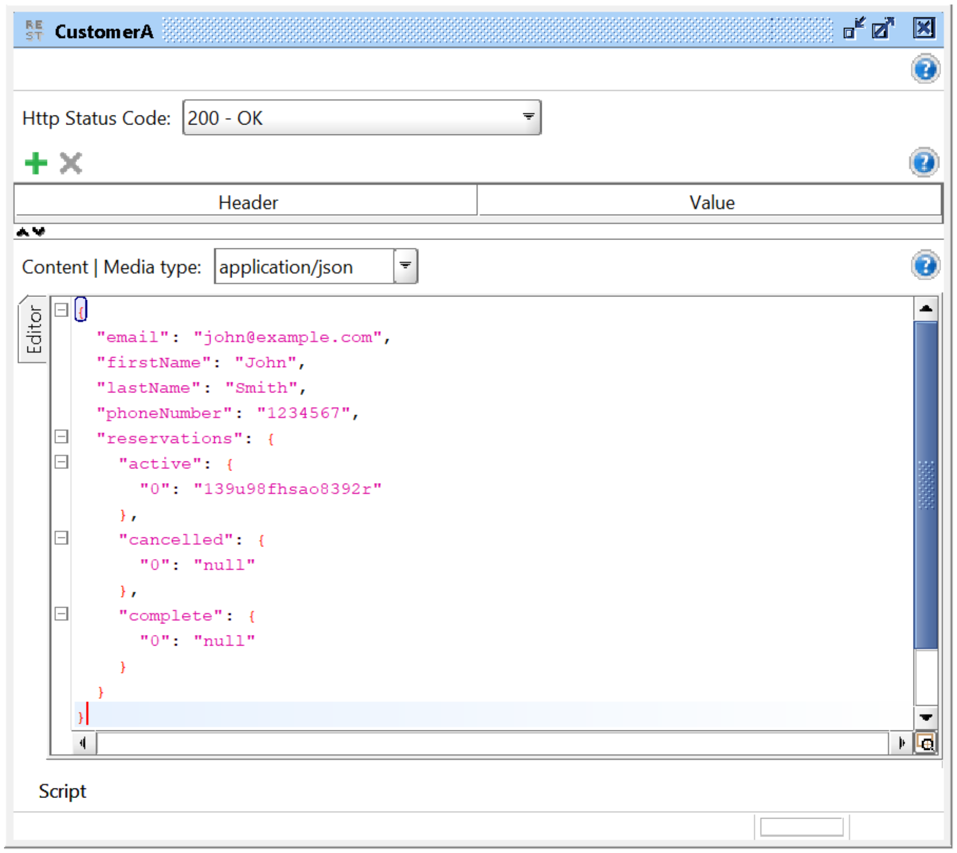


7. Right-click your new mock action, and create a mock response. In this example, the mock response will be in JSON format. Add the response data to the new mock response.







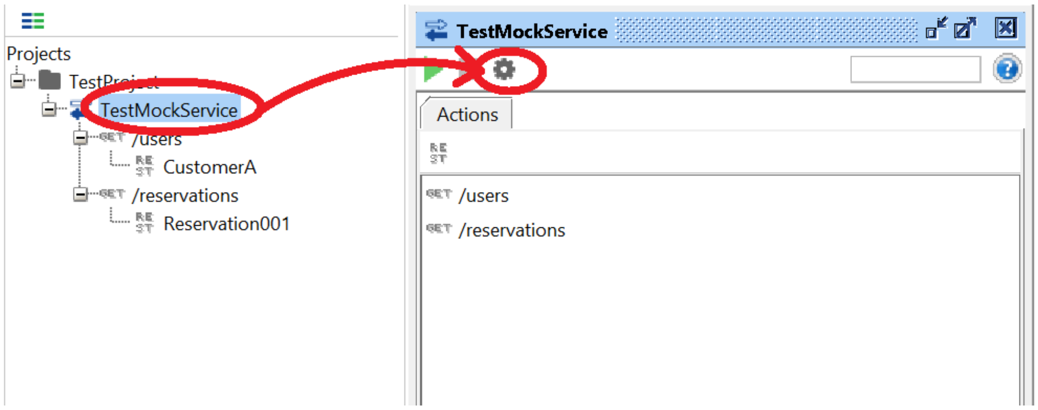


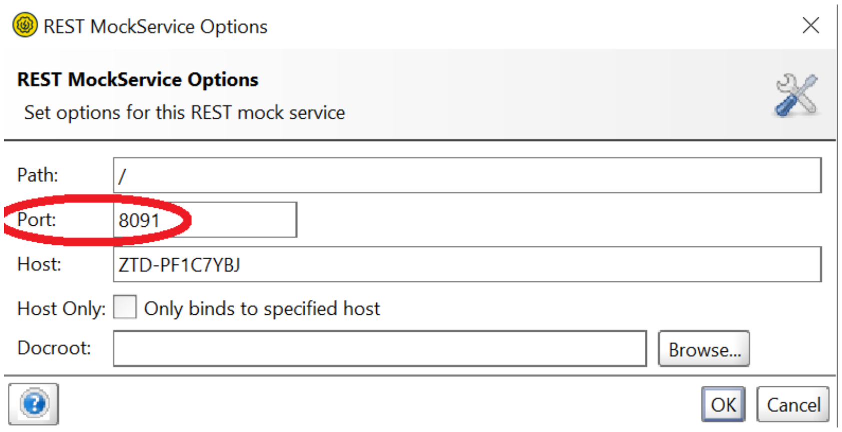
Http Status Code: 200 - OK

Content | Media type: application/json

```
{
  "email": "john@example.com",
  "firstName": "John",
  "lastName": "Smith",
  "phoneNumber": "1234567",
  "reservations": {
    "active": {
      "0": "139u98fhsao8392r"
    },
    "cancelled": {
      "0": "null"
    },
    "complete": {
      "0": "null"
    }
  }
}
```

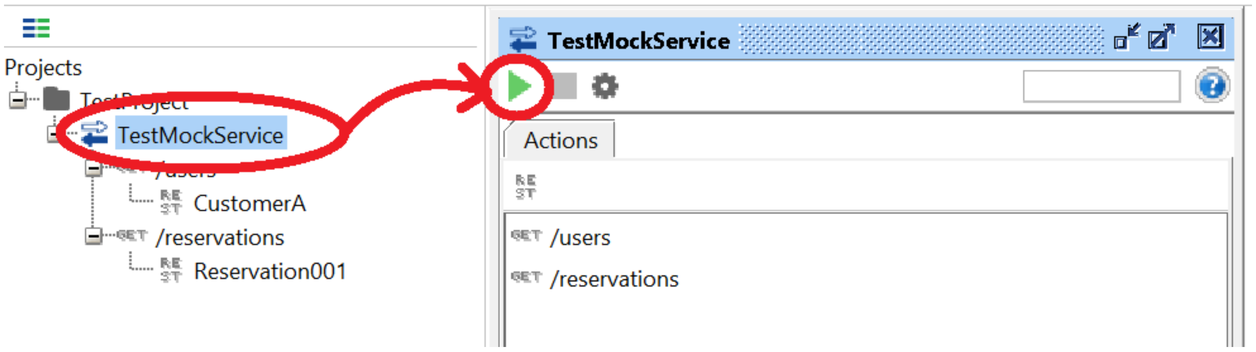
8. After adding the mock responses, go back and finish configuring your MockService. Double-click the MockService, and click the gear icon to view Options. Change the port number to an unused port. In this example, port 8091 will be used. After changing the port number, click ‘Ok’.



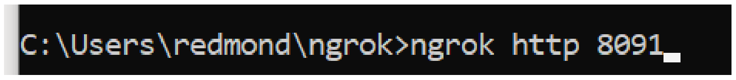


Path: /
Port: 8091
Host: ZTD-PF1C7YBJ
Host Only: [ ] Only binds to specified host
Docroot:

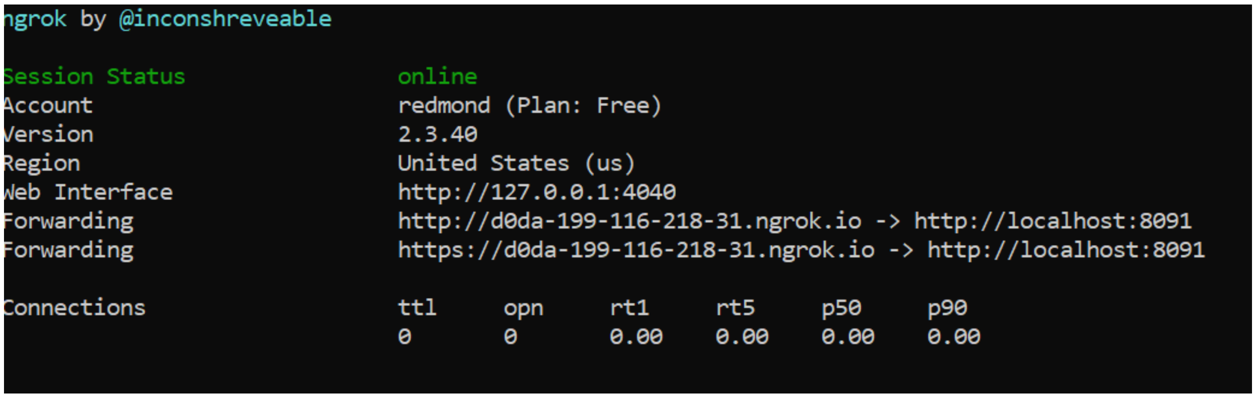
9. Double-click your MockService again. This time, click the 'Play' icon to start running your MockService. Your MockService will now be running on the port you specified.



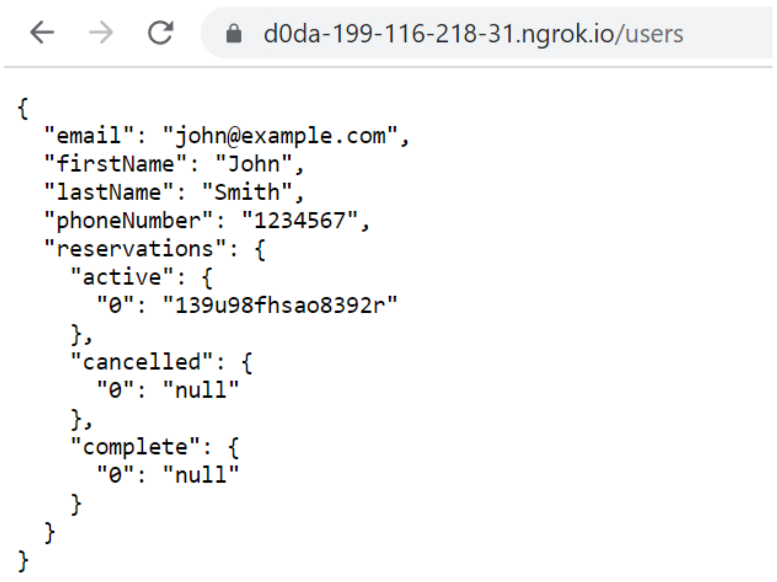
10. Open your CMD terminal. Navigate to the folder where *ngrok.exe* is located. Run the command: *ngrok http **portNumber.*** The ngrok service data will then be presented to you, you can use these URLs when accessing your mocked API.



```
C:\Users\redmond\ngrok>ngrok http 8091
```

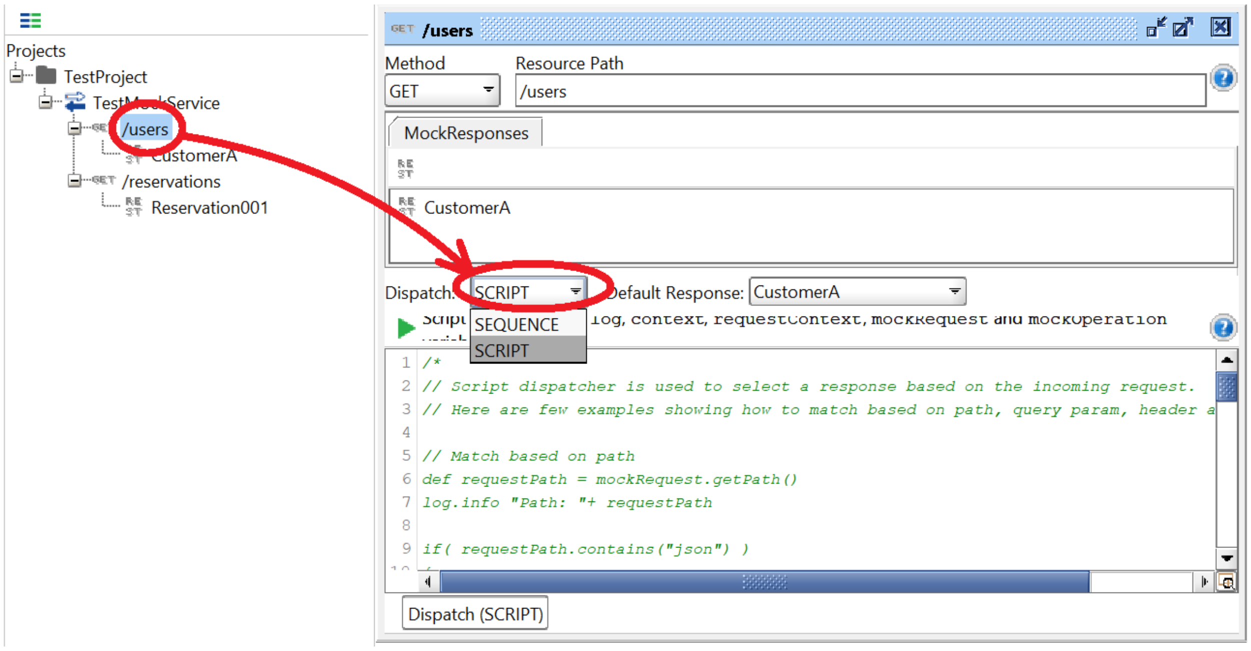


```
ngrok by @inconshreveable

Session Status                online
Account                       redmond (Plan: Free)
Version                       2.3.40
Region                        United States (us)
Web Interface                 http://127.0.0.1:4040
Forwarding                    http://d0da-199-116-218-31.ngrok.io -> http://localhost:8091
Forwarding                    https://d0da-199-116-218-31.ngrok.io -> http://localhost:8091

Connections                   ttl     opn     rt1     rt5     p50     p90
                              0       0       0.00    0.00    0.00    0.00
```



```
{
  "email": "john@example.com",
  "firstName": "John",
  "lastName": "Smith",
  "phoneNumber": "1234567",
  "reservations": {
    "active": {
      "0": "139u98fhsao8392r"
    },
    "cancelled": {
      "0": "null"
    },
    "complete": {
      "0": "null"
    }
  }
}
```

11. (Optional) Currently, your mock responses will be returned sequentially, through the order in which your mock responses are added to the relevant mock action. To change this, double-click on your mock action, and change your Dispatch from ‘SEQUENCE’ to ‘SCRIPT’. Sample code will be given in the code block to help guide you in this. Be sure to include mock responses for error responses, to handle unexpected calls.

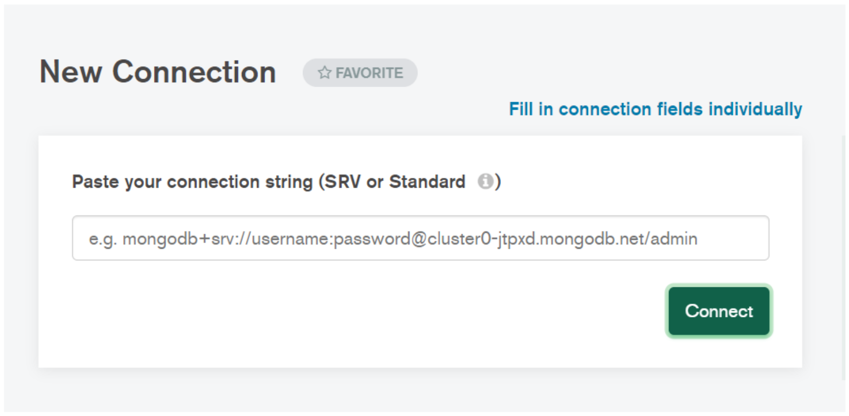


Script ... log, context, requestContext, mockRequest and mockOperation

```
/*
// Script dispatcher is used to select a response based on the incoming request.
// Here are few examples showing how to match based on path, query param, header a

// Match based on path
def requestPath = mockRequest.getPath()
log.info "Path: "+ requestPath

if( requestPath.contains("json") )
```

Option 2: NoSQL

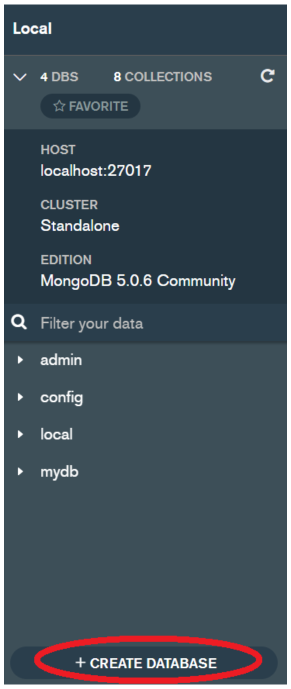
1. Download and install NoSQL database. In this tutorial, we will be using MongoDB, but the instructions can be applied to any NoSQL database.
    a. As part of the installation, be sure to include MongoDB Compass. Though MongoDB can be managed through CMD, the GUI definitely helps.
2. Open MongoDB and initialize your connection.
    a. If you are using CMD, navigate to the folder containing *mongo.exe*. By default, this will usually *be C:\Program Files\MongoDB\Server\5.0\bin*. Enter the command *mongo.exe* will start running MongoDB.
    b. If you are using MongoDB Compass, you can simply click "Connect".



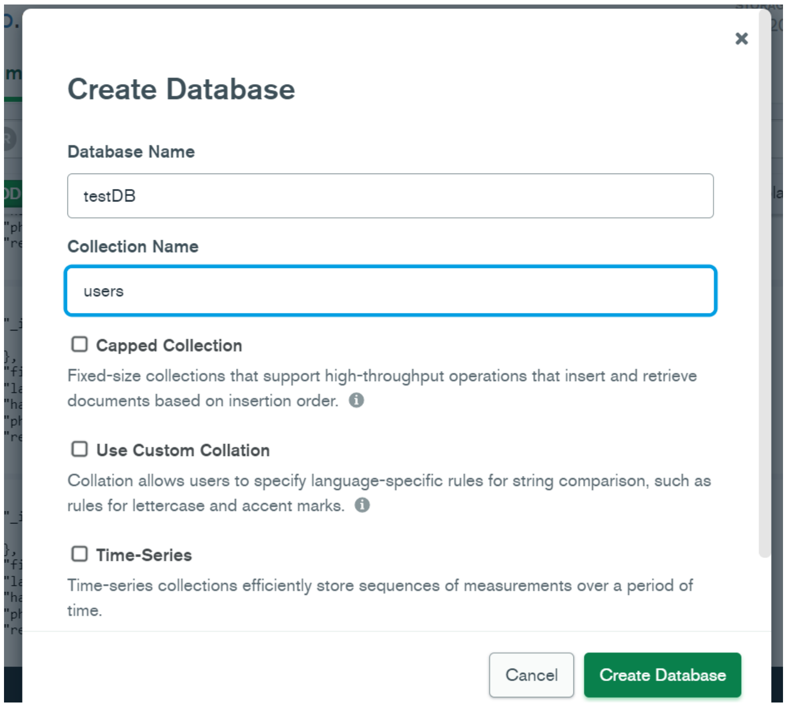
Paste your connection string (SRV or Standard)

e.g. mongodb+srv://username:password@cluster0-jtpxd.mongodb.net/admin

3. Create a new MongoDB database and collection. This can again be done either through CMD or MongoDB Compass, screenshots will be from Compass. Data entries will be stored in your *collections*.



HOST
localhost:27017
CLUSTER
Standalone
EDITION
MongoDB 5.0.6 Community



Database Name: testDB

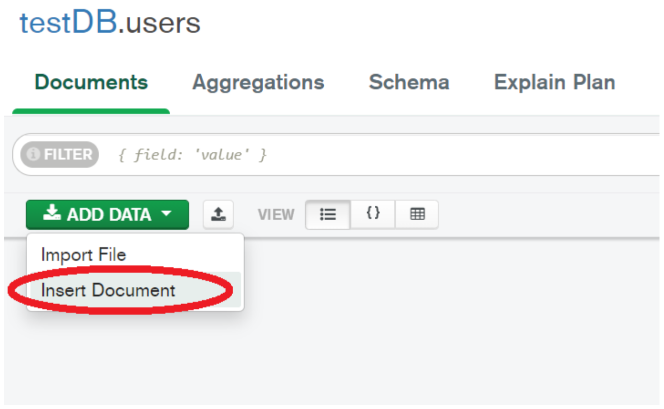
Collection Name: users

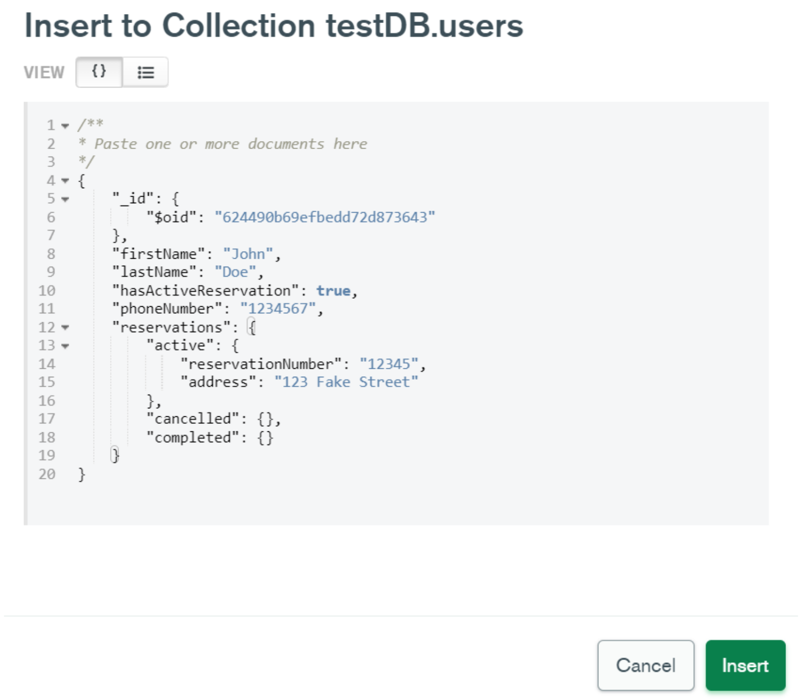
☐ Capped Collection
Fixed-size collections that support high-throughput operations that insert and retrieve documents based on insertion order.

☐ Use Custom Collation
Collation allows users to specify language-specific rules for string comparison, such as rules for lettercase and accent marks.

☐ Time-Series
Time-series collections efficiently store sequences of measurements over a period of time.

4. You can either choose to populate your collection with some initial sample data, or continue to create your API. *To continue to the API, move to Step 5*.
   To populate your collection, left-click your collection. Under the "Add Data" drop-down, click "Insert Document". You may also import a JSON or CSV file instead.





```
/**
 * Paste one or more documents here
 */
{
    "_id": {
        "$oid": "624490b69efbedd72d873643"
    },
    "firstName": "John",
    "lastName": "Doe",
    "hasActiveReservation": true,
    "phoneNumber": "1234567",
    "reservations": {
        "active": {
            "reservationNumber": "12345",
            "address": "123 Fake Street"
        },
        "cancelled": {},
        "completed": {}
    }
}
```

a. When using CMD, the command

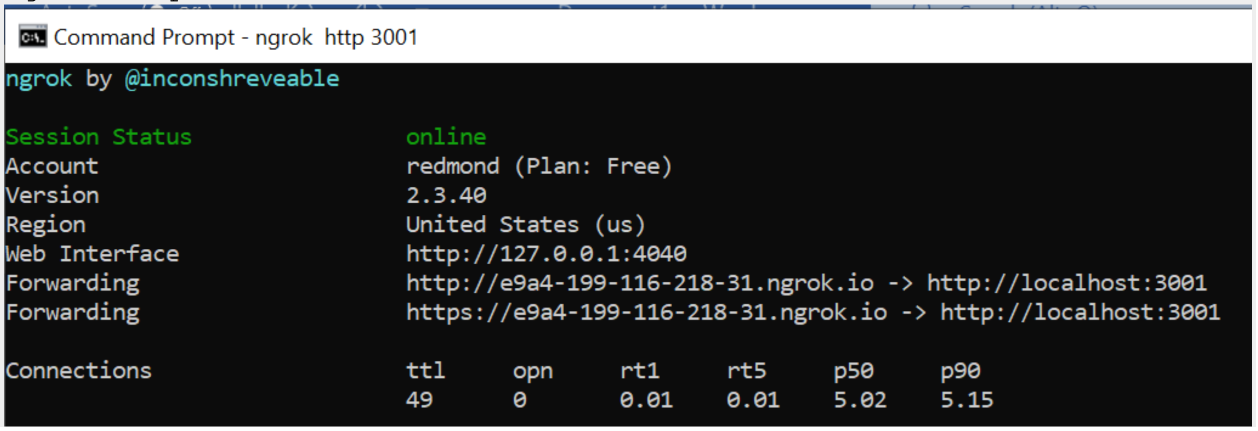
```
use collectionname
```

will allow you to populate your new collection, or create the collection if it does not already exist. From here to insert the data, run the command:

```
db.collectionname.insert({
  field: "value",
  field2: "value2"
})
```

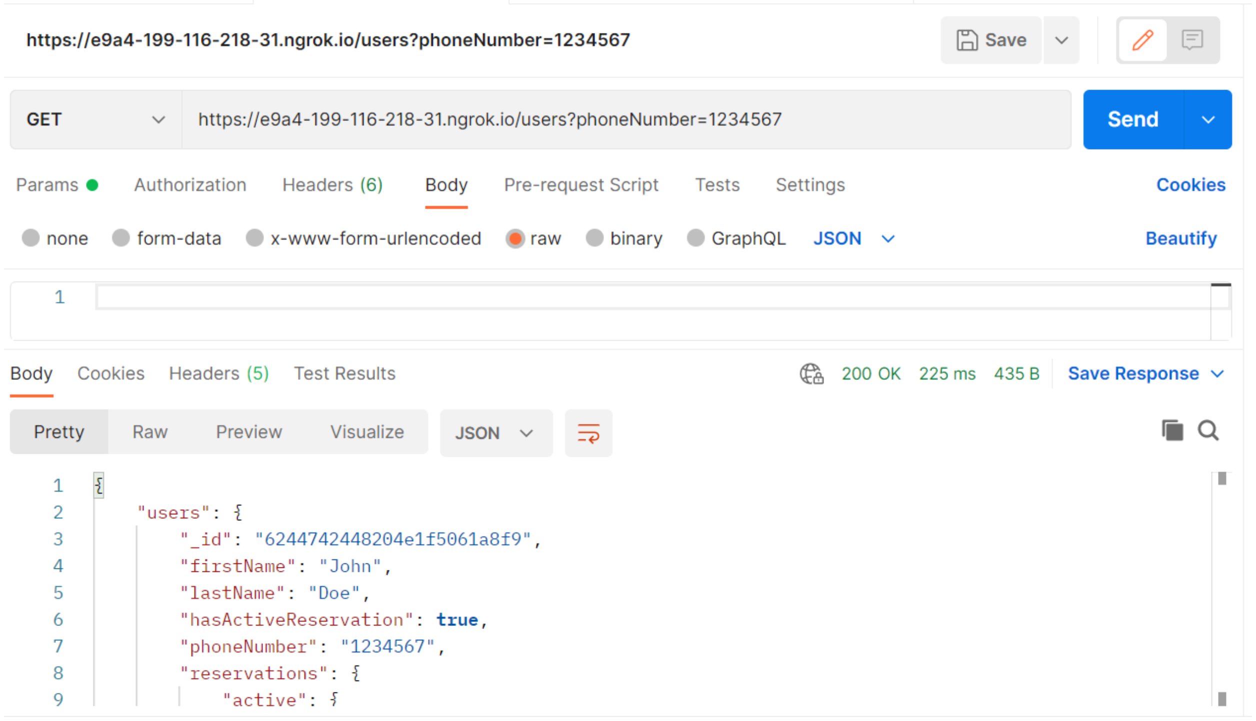
5. Create your API. In this tutorial the API will be created through NodeJS, it will be a barebones API running on localhost to serve basic queries.
    a. *The in's and out's of creating APIs is outside the scope of this tutorial, as such the code will be provided in the Appendix for your understanding. Before adding the code, the commands* ***npm init -y****,* ***npm install MongoDB****, and* ***npm install Express*** *should be ran, followed by creating a .js file for your code. Once the code is added, start your API through the terminal by running* ***node filename.js*** *in the Terminal.*
6. Follow **Steps 2-3 in Option A** to set up ngrok. From here, run ngrok on the port where your API is running. For example, to run ngrok on port 3001, run the following command:

```
ngrok http 3001
```



```
Command Prompt - ngrok  http 3001
ngrok by @inconshreveable

Session Status                online
Account                       redmond (Plan: Free)
Version                       2.3.40
Region                        United States (us)
Web Interface                 http://127.0.0.1:4040
Forwarding                    http://e9a4-199-116-218-31.ngrok.io -> http://localhost:3001
Forwarding                    https://e9a4-199-116-218-31.ngrok.io -> http://localhost:3001

Connections                   ttl     opn     rt1     rt5     p50     p90
                              49      0       0.01    0.01    5.02    5.15
```

7. Your API service should now be available through the link above. You can test this through Postman calls. Once functionality is confirmed, the link can be used for MiniApp API calls



https://e9a4-199-116-218-31.ngrok.io/users?phoneNumber=1234567

GET https://e9a4-199-116-218-31.ngrok.io/users?phoneNumber=1234567

200 OK 225 ms 435 B

```
{
    "users": {
        "_id": "6244742448204e1f5061a8f9",
        "firstName": "John",
        "lastName": "Doe",
        "hasActiveReservation": true,
        "phoneNumber": "1234567",
        "reservations": {
            "active": {
```

Appendix

NoSQL DB Setup

```
//Mongo Setup
const MongoClient = require('mongodb').MongoClient;
const url = 'mongodb://127.0.0.1:27017';

//API Setup
const express = require("express")
const app = express()
const PORT=3001;
app.use(express.json())


//Connect to MongoDB
MongoClient.connect(url, {
    useNewUrlParser: true,
    useUnifiedTopology: true
}, (err, client) => {
    if (err) {
        return console.log(err);
    }

    // Specify database you want to access
    const db = client.db('testDB');
    console.log(`MongoDB Connected: ${url}`);

    const users = db.collection('users');

    app.listen(PORT, () => console.log(`API Connected on localhost:${PORT}`))

    //API Endpoints

    //Get Users
    app.get("/users", (req, res) => {
        console.log(req.body)
        const phoneNumber = {"phoneNumber": req.query.phoneNumber}

        users.findOne(phoneNumber, (err, result) => {
            if (err) {
                console.error(err)
                res.status(500).json({ err: err })
                return
            }
            console.log(result)
            res.status(200).json({users: result})
        })
    })

    //Get Users (BACKUP via POST method)
    app.post("/users", (req, res) => {
        console.log(req.body)
        const phoneNumber = {"phoneNumber": req.body.phoneNumber}

        users.findOne(phoneNumber, (err, result) => {
            if (err) {
                console.error(err)
                res.status(500).json({ err: err })
                return
            }
            console.log(result)
            res.status(200).json({users: result})
        })
    })

    //Get All Users
    app.get("/usersList", (req, res) => {
        console.log(req.body)
        users.find().toArray((err, result) => {
            if (err) {
                console.error(err)
                res.status(500).json({ err: err })
                return
            }

            console.log(result)
            res.status(200).json({ users: result })
         })
    })
});
```