|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | | | **Our Black Gems API** | | |  | | |
|  | |  | | | | |  | |
|  | | | | **Taighlor Moultrie and Roderick Bishop** |  | | | |
|  | | | | **12-07-2021**  **—**  **CS460 Senior Project  South Carolina State University**  **—**  **Dr. Biswajit Biswal** |  | | | |
|  | Table of Contents  [Abstract 3](#_Toc89866691)  [Inspiration 3](#_Toc89866692)  [Introduction and Project Overview 5](#_Toc89866693)  [Project Measures of Success & Expectations: 5](#_Toc89866694)  [Requirements 6](#_Toc89866695)  [Data Requirements: 6](#_Toc89866696)  [Functional Requirements 6](#_Toc89866697)  [Project Design 7](#_Toc89866698)  [UMLs: 7](#_Toc89866699)  [API implementation 8](#_Toc89866700)  [Controllers 8](#_Toc89866701)  [Repositories 8](#_Toc89866702)  [Data Models 8](#_Toc89866703)  [Service 8](#_Toc89866704)  [Tools used 9](#_Toc89866705)  [Database implementation 9](#_Toc89866706)  [Database Overview and Design: 10](#_Toc89866707)  [Data definition: 12](#_Toc89866708)  [Operations & violation resolution: 13](#_Toc89866709)  [Final Project Outcomes 14](#_Toc89866710)  [Project Measures of Success Revisited: 14](#_Toc89866711)  [mVP: 14](#_Toc89866712)  [Contribution: 15](#_Toc89866713)  [Learning Statements: 16](#_Toc89866714)  [Future for Project 16](#_Toc89866715)  [References & Resources 17](#_Toc89866716) | | | | | | |  |

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | Abstract | | | | | | |  |
|  |  | | |  |  | | |  |
|  |  | |  | | |  | |  |
|  |  |  | In current times, African Americans are woefully underserved when it comes to having our history presented in a way that is modern, truthful, and impactful. So many of our “firsts” and “mosts”, as well as our creations and contributions have been buried underneath other history deemed more important. It’s time to display that greatness in a way that works for the future. Our Black Gems API is both a celebration of Black or African American contributions to society. This API intends to serve clients with what we call “Gems”. Gems are the enlightening facts, figures, inventions, and songs generated by Black people but not celebrated enough. | | |  |  |  |
|  | | | | |
|  |  |  |  |
|  |  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Inspiration | | |  |
|  |  |  |  |  |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | “A black man who lays claim  to his unadulterated knowledge  of self comes second only to God.  A black woman who knows  her worth comes second to none.”  ― Anan$i | | | | |  |
|  | |  |  |  |  | |
|  |  | | | | | ‘ |

# Introduction and Project Overview

African Americans are woefully underserved when it comes to having our history presented in a way that is modern, truthful, and impactful. So much of our history has been buried, lied about, and thus forgotten. It’s time to display that greatness in a way that works for the future.

Our Black Gems API(OBGAPI) is both a celebration of Black and African American contributions to society and a new realization of an older idea. OBGAPI is an open-source application programming interface built on a .NET stack with four endpoints that users will be able to retrieve data relating to inventions of, facts about, songs by, and information on figures in Black history. Conceptually, OBGAPI is a technical solution to a social issue, the erasure of the Black figure.

# Project Measures of Success & Expectations:

Our minimally viable product will be a stable version of the app that can:

* Four distinct endpoints, with one that is demo-ready (facts)
* Demo can be done with hard-coded data, using endpoint testing tools (SwaggerUI or Insomnia)
* Database schema for the backend defined and test tuples in database

We fully expected to accomplish this mVP when we started creating OBGAPI. Our team was satisfied with the requirements that we have defined as we believed at the time that completing the mVP would be well within our abilities. Given the product planning, design, and implementation, we also believe that future versions are now within our ability as well. We wanted to take on a challenge and this project certainly was. However, the mVP still gave us enough breathing room to focus on exploring the full set of tools, processes, and concepts learned in previous courses and required for project completion.

# Requirements

## Data Requirements:

The data to support this API is based on a FIGURE. FIGUREs consist of First & Last name, a Bio, Birth and Death dates, Birth City, a Category, a bio, and a unique Id. FIGUREs are returned in the /figure endpoint.

An INVENTION is created by a FIGURE. Each INVENTION has a database id number, Patent No./id No., name of the Invention, names of Inventors, a Category and Description

The /fact endpoint will show a black history fact of the day. This data will be represented as a FACT entity. A FACT has a Date, a Fact, a figure Id and name, and a Link (URL string to further reading)

The /song endpoint will send a random song released by a black artist. This data is organized into SONG entities. A SONG has an Id, Artist(s), song title, Year, name of the Album, name of the recording studio, year, and Link.

## Functional Requirements

* Role-Specific:
  + U.D all SONGS, FIGURES, INVENTIONS, and FACTS entitles as Administrator
* Client/End User:
  + Retrieve a FIGURE (all entity data)
  + Retrieve FIGUREs in specific categories
  + Retrieve FIGUREs from a birth city
  + Retrieve a random FACT
  + Retrieve a random FACT for any day of the year
  + Retrieve an INVENTION (all entity data)
  + Retrieve an INVENTION BY patent/id number.
  + Retrieve an INVENTION BY Category.
  + Retrieve a SONG (all entity data)
  + Retrieve SONGs from a Studio
  + Retrieve SONGs from an Artist

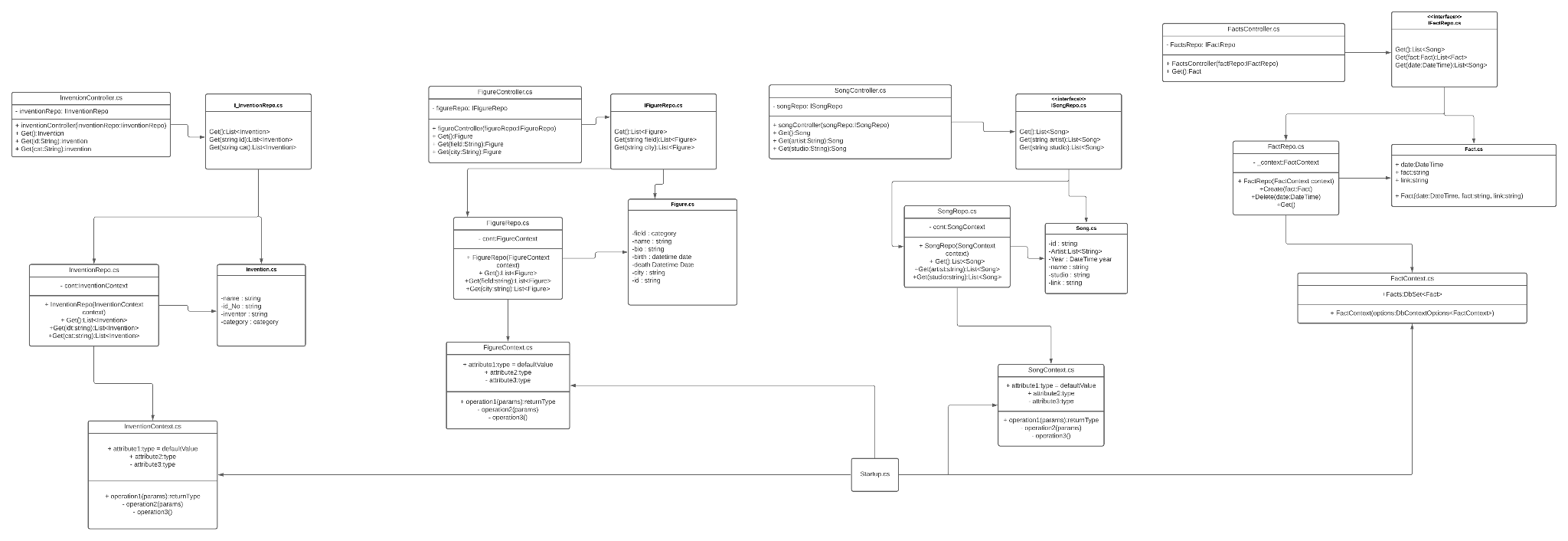
# Project Design

Project Architecture:  
Diagram

Description automatically generated

*Fig. 1: Version 3 of OBGAPI’s architecture diagram. OBGAPI is the entire server side. We want to create and maintain the DB as well as build & improve the API itself. Once the endpoint is hit, the API will generate a response from the controller. Also notably depicted here are the service layer as well as and repositories which are used to abstract access and functionality in similar ways.*

## UMLs:



*Fig. 2: The client is served by the controller of the respective endpoint which connects in various ways to other files we’ve created for this project.*

# API implementation

*\*\*Source code for this project can be found* [*in the project Github Repository*](https://github.com/roderick-bishop11/seniorProject)*\*\**

## Controllers

The API controllers are redundant in our design, as we wanted to have smaller and more modular code. There is one controller for each endpoint aptly named in the following scheme: “{Entity}Controller”. These controllers are the endpoint handlers for our API and specify the actions taken upon receiving an HTTP request from the client. The nature of this API is such that a client will query any endpoint and can expect to receive a response with a single payload, or “Gems”.

## Repositories

The repositories are abstractions of the data access layer in your API. With this abstraction, we can precisely define how we want to access our data on the server side using queries. In future versions, OBGAPI will specify queries using these repos and define methods using the repository interfaces.

## Data Models

The data models are the object-oriented representations of the entity data retrieved back from the repository query. Entity attributes are mapped to these fields and can be manipulated in any such way the programmer sees fit before these are sent as a payload to the client.

There are 4 data models defined in this early version of the API:

* Song (int id, List<string> artists, year, name, studio, link)
* Fact (DateTime date, string fact, string link)
* Figure (int id, string name, int temp, DateTime birthDate, DateTime deathDate, string bio, string city)
* Invention (int id, string patentId, string inventionName, string desc, Category category, List<Figure> inventors)

## Service

The service layer of any API is where most of the data manipulation and processing occurs prior to being sent as a payload from the controller. In the case of our API, the service layer is used to abstract and add functions before the payload is sent. This abstraction of the service layer allows us to change how we want the data to be sent to the client in different ways in the future and afford us some level of modularity and thus flexibility in the design.

In our current version (v0.0.1), the two services are:

* Selectsingle.cs- for selecting a random Song, Fact, Figure, or Invention from the list returned via query
* SelectCategory.cs- for selecting one of our enumerators given an int stored in the DB to represent it’s value.
* FormatFacts.cs- for formatting our strings that will print out for the /fact endpoint. This is mostly used in the mVP version but could be used in later versions as well.

## Tools used

**Git/GitHub:** GitHub was used for our versioning system. At all times a MAIN branch was kept as the current version and other branches were created, committed, and merged into the main branch upon approval from the team. Pushes directly to the main branch were blocked and required a merge. This was done to mimic the Agile workflow and concepts of CI/CD as well as Semantic Versioning to keep track of changes.

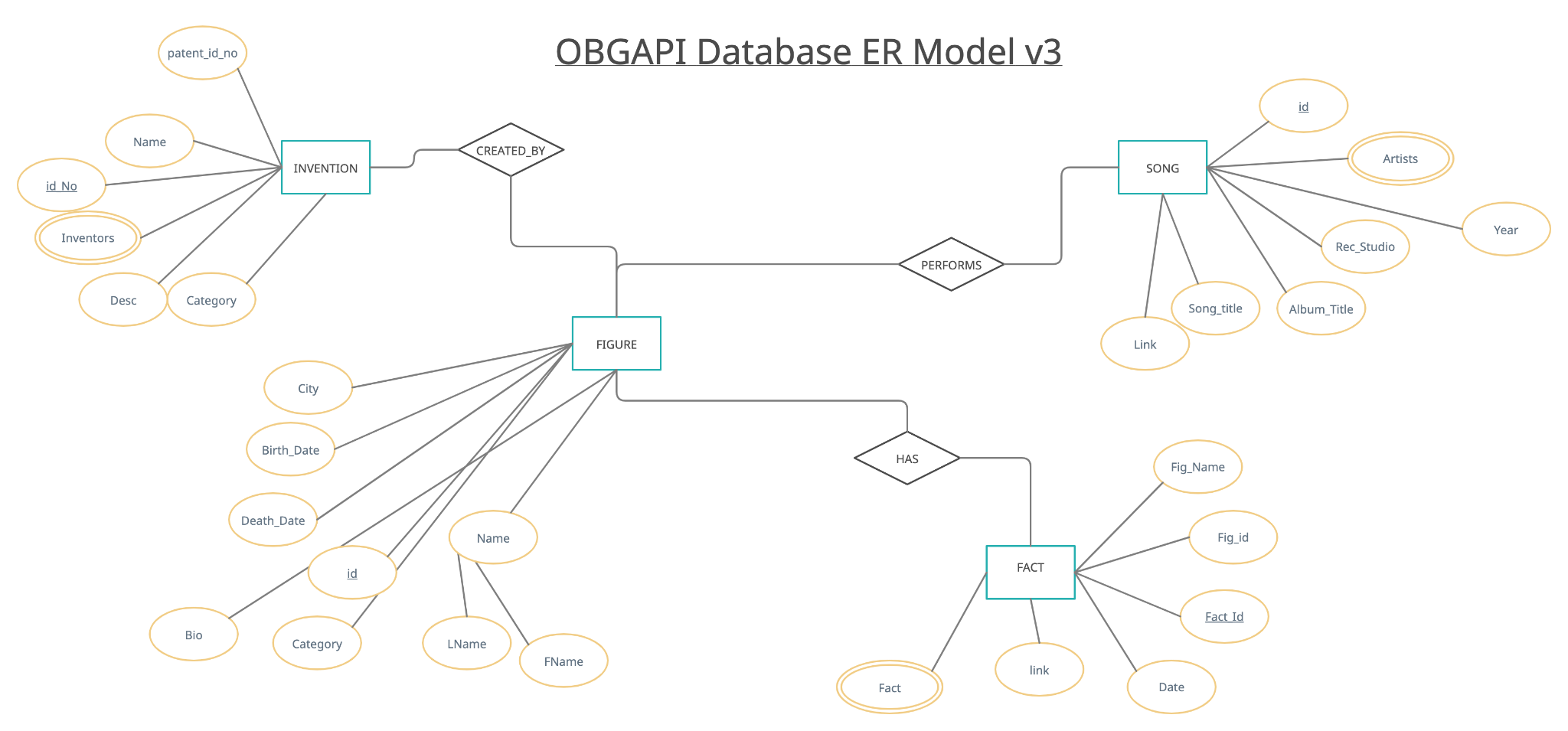
The GitHub Repository(seniorProject) can be found via this [link](https://github.com/roderick-bishop11/seniorProject) and is referenced in the resources section of this report. All of the project diagrams and code can be found there as well as the README.md and changelog.md for the project giving a full account of the project and its build history. The code can also be downloaded and spun up locally by cloning the repo or accessed via live URL in version 1.0.0 of the API.

**ASP.NET:** ASP.NET is the framework for building API’s using the .NET stack of HTML, JS, C#, and SQL. We used every technology in this stack to create OBG API and found the stack to be straightforward as well as useful for every portion of the process.

**DBeaver Lite:** DBeaver Lite is a free and open-source DB tool that we used to visualize and make queries. While we did complete a lot of our DB work via command line, we also used DBeaver Lite for ease of use when it came to inserting, querying, etc. with our DB.

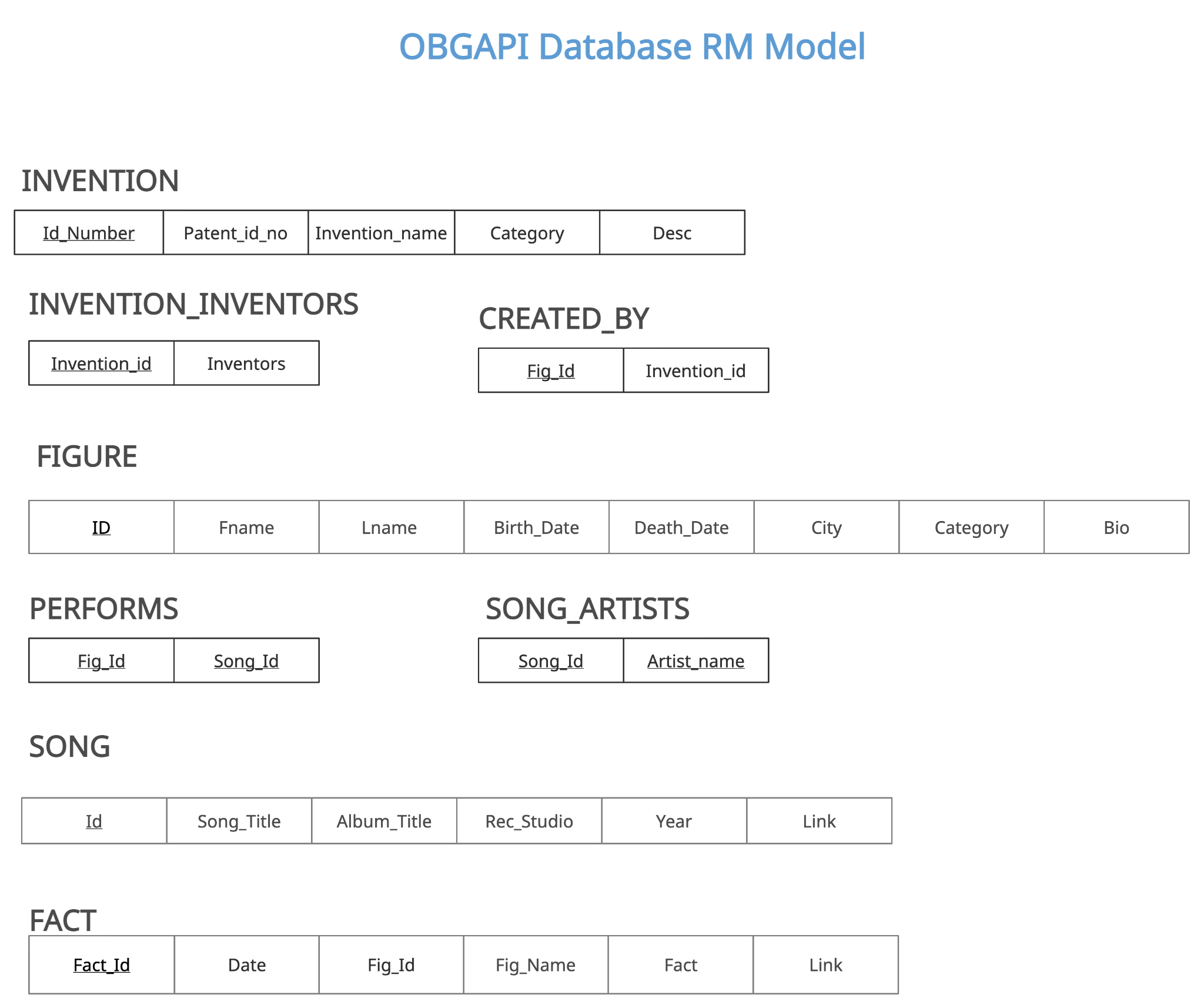
# Database implementation

## Database Overview and Design:

****

*Fig. 3: ER Data Model for our backend.*

As seen in Figure 3, our DB is organized into 4 entities, one to support each endpoint. Each entity will be queried and represented as objects of a class with the same name. Each attribute is mapped to an appropriate class member and represented as such to the client.

****

*Fig. 4: Relational DB Schema*

## Data definition:

-- OBG\_API.figure definition

CREATE TABLE `figure` (

`Id` int NOT NULL AUTO\_INCREMENT,

`Fname` varchar(100) NOT NULL,

`Lname` varchar(100) DEFAULT NULL,

`Birth\_Date` date DEFAULT NULL,

`Death\_Date` date DEFAULT NULL,

`City` varchar(100) DEFAULT NULL,

`Category` int DEFAULT NULL,

`Bio` text,

PRIMARY KEY (`Id`)

) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4\_0900\_ai\_ci;

-- OBG\_API.invention definition

CREATE TABLE `invention` (

`Id` int NOT NULL AUTO\_INCREMENT,

`Patent\_Id\_No` varchar(255) DEFAULT NULL,

`Invention\_name` text NOT NULL,

`Category` int DEFAULT NULL,

`Description` text,

`Inventors` varchar(255) DEFAULT NULL,

PRIMARY KEY (`Id`)

) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4\_0900\_ai\_ci;

-- OBG\_API.song definition

CREATE TABLE `song` (

`Id` int NOT NULL AUTO\_INCREMENT,

`Artist\_Fname` varchar(100) DEFAULT NULL,

`Artist\_Lname` varchar(100) DEFAULT NULL,

`Rec\_Studio` text,

`Song\_Title` text,

`Album\_Title` text,

`Link` text,

`Year` int DEFAULT NULL,

PRIMARY KEY (`Id`)

) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4\_0900\_ai\_ci;

-- OBG\_API.fact definition

CREATE TABLE `fact` (

`Id` int NOT NULL AUTO\_INCREMENT,

`Date` date DEFAULT NULL,

`Fact` text,

`Fig\_name` text,

`Fig\_Id` int DEFAULT NULL,

`Link` text,

PRIMARY KEY (`Id`),

KEY `Fig\_Id` (`Fig\_Id`),

CONSTRAINT `fact\_ibfk\_1` FOREIGN KEY (`Fig\_Id`) REFERENCES `Figure` (`Id`) ON DELETE CASCADE ON UPDATE CASCADE

) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4\_0900\_ai\_ci;

## Operations & violation resolution:

**FIGURE:**

* On Delete: Set Default Name will be replaced by a default string “our Hero”
* On update: Cascade

**FACT:**

* On Delete: Cascade. There should be at least one fact for each day. Restrict delete if there is only one fact within the Fact attribute.
* On Update: Cascade. If a figure name is updated, then the name should be cascaded into this table as well.

**INVENTION:**

* On insert: Cascade. User will be prompted to add tuple for Figure with that name. Corresponds to INVENTION.Inventor.
* On Update: Cascade. Update corresponding Figure name through update of the tuple with FK (Inventor)

**SONG:**

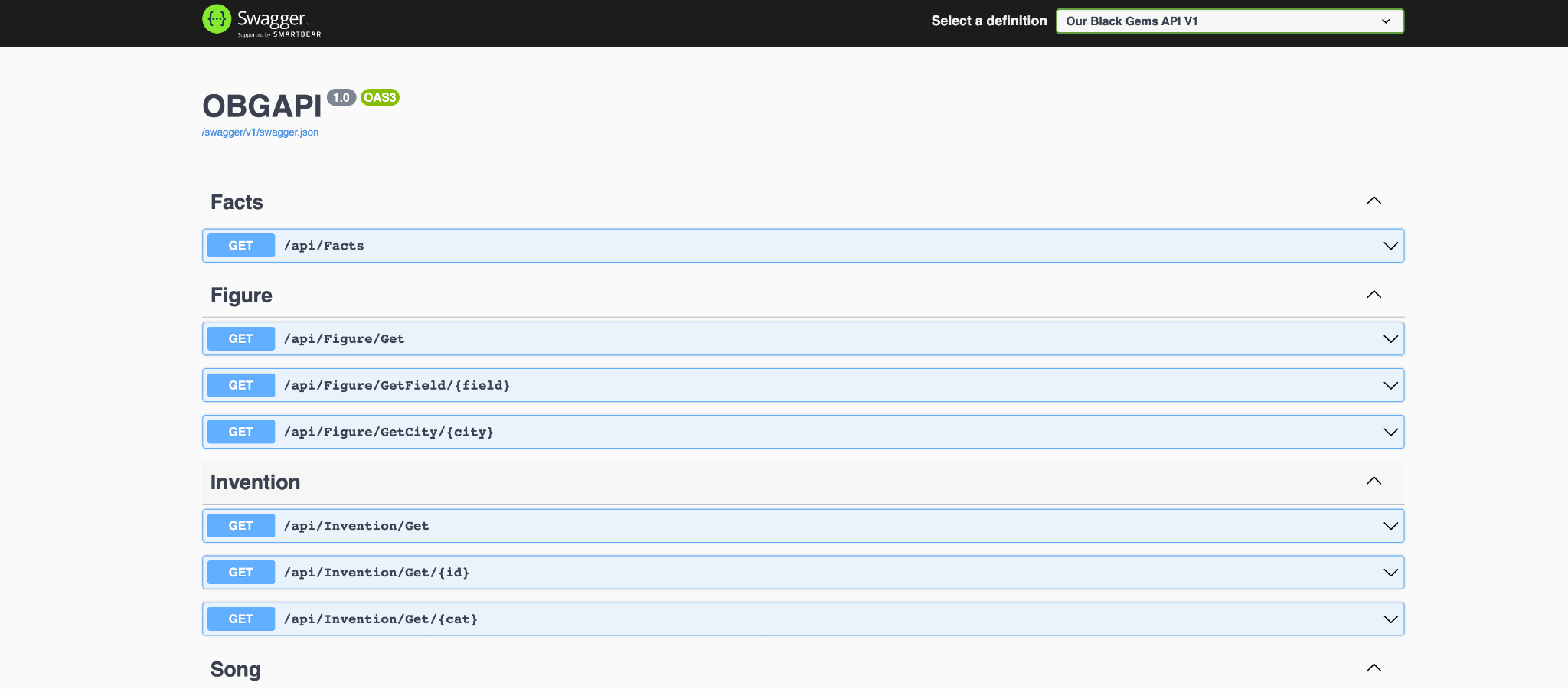
* On insert: Cascade. User will be prompted to add tuple for Figure with that name. Corresponds to SONG.Artist.
* On Update: Cascade. Update corresponding Figure name through update of the tuple with FK (Artist)

# Final Project Outcomes

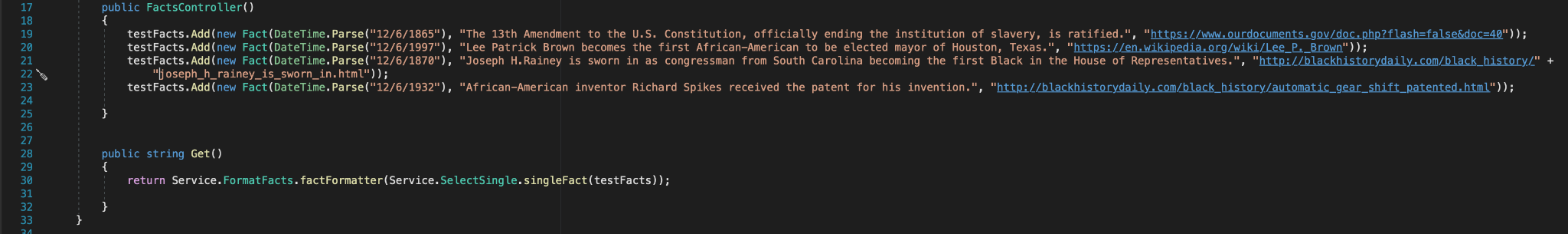
## Project Measures of Success Revisited:

Accomplishing the API mVP was a happy moment for us. Our API can send data to localhost, via an HTTP GET method that returns a list of facts. Using hard coded data, we were able to hit the endpoint and select a random single Fact out of the list of Facts. This is what we wanted for the function of the API, given that we want the user to get one fact at a time. Because the HTTP GET method is run even on reloads, we are also able to choose a random one out of the list each reload since we have an entire list that is loaded.

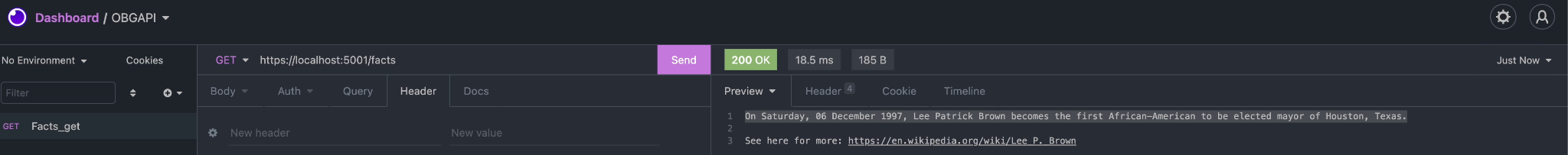
## mVP:



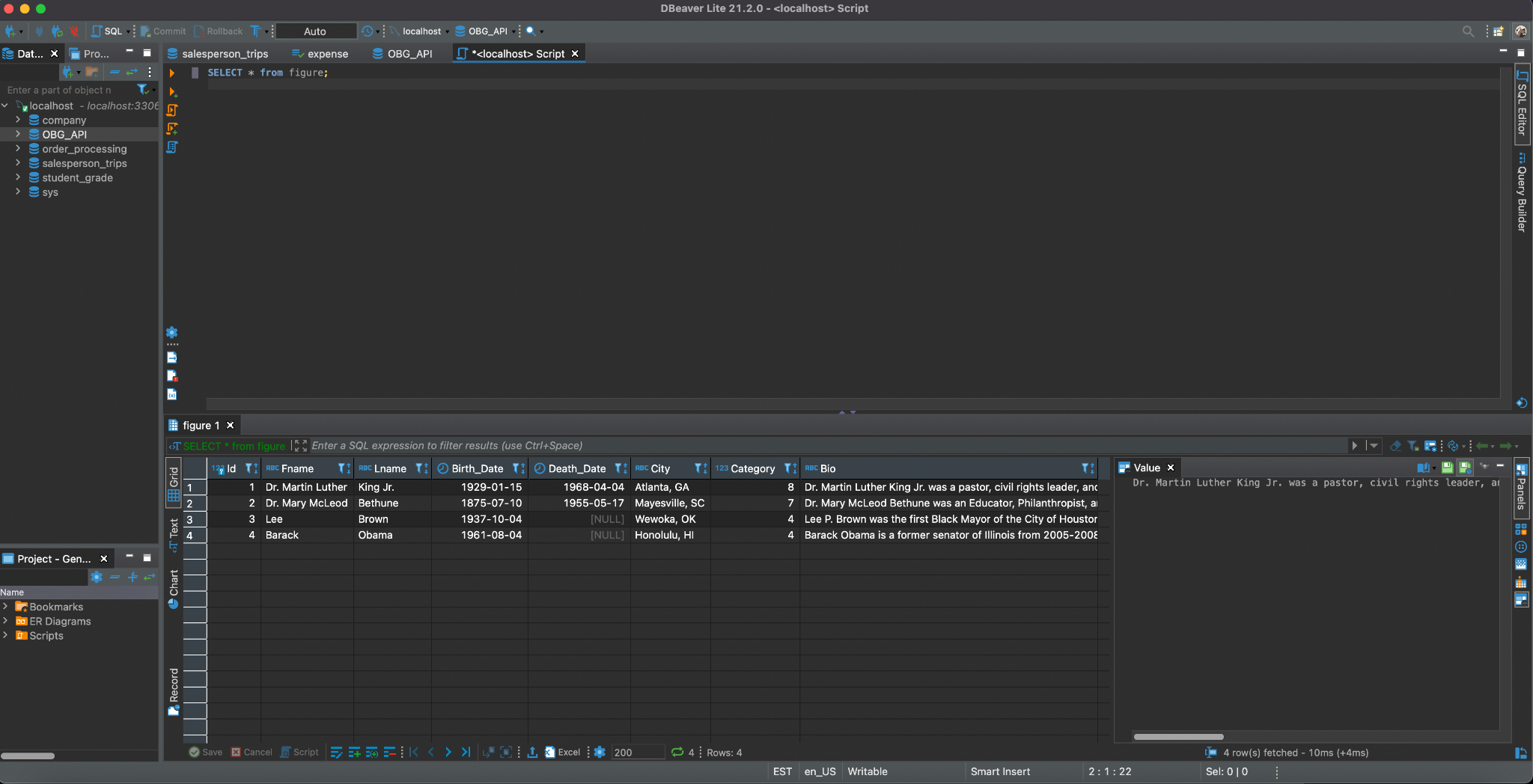
*Fig. 5: In this figure, we can see SwaggerUI loading all of our endpoints. The operational endpoint, api/Facts/ is the endpoint that is demoed for the mVP.*



*Fig. 6: In the demo, we used a hard coded testfacts List. On line 30, a formatted string is returned from our service factFormatter() that takes in the output from another service singleFact().*



*Fig. 7: Using Insomnia, when the /facts endpoint is hit, we receive a 200 from the localhost server. We also see our formatted output string that was built using our service factFormatter() in the preview pane on the right.*



*Fig. 8: Test tuples in the database. After running a total selection query in the OBG\_API DB, we can see that four test tuples are inserted in our figures table in DBeaver Lite.*

## Contribution:

As with every group project, each group member must contribute to the group's content. Our group, although small, was able to get the work done through teamwork. We had no issues working together and we shared all responsibilities in an equitable fashion. All group members contributed to the development of source code, design elements, as well as the presentation and report to make sure that we both had the full experience of developing this solution.

## Learning Statements:

* **Roderick Bishop-** This project is very special and dear to my heart. Originally a very rough personal idea, I pitched it and the group agreed to pursue its creation. Together, we decided how this API should work and what we wanted out of this project for CS460 as well as continuing OBGAPI’s build out and eventual launch. This project was more than we had asked for at times, and it was a lot of experiential learning along the way. Together we learned so much about C#, SQL, the application & employment of development processes & concepts that we’ve learned in a previous course at SCSU (CS405), and a lot more about API’s as a whole. Overall, I am glad that we were able to build our mVP and present it to complete the course. Going forward, I’d love to see OBGAPI fully realized and used in many mobile or web apps, websites, etc. It’s my hope that this will be an open-source project for young Black Engineers to contribute as well as learn about the wonderful people that helped shape the world that they know and love.
* **Taighlor Moultrie -** This project was something that my partner wanted to create for a long time. Starting this project was very difficult, I didn't know most of the materials my partner wanted me to do. I asked a lot of questions about the project that helped us get it up and running. This project definitely was a challenge, but it taught me a lot. Together we learned a lot about databases, the application, & API. Overall, I am glad I took CS460. It was definitely a big learning experience. I would recommend this course to anybody because it gives you the opportunity to think outside the box and create something cool

# Future for Project

OBGAPI will be an open-source project that developers can contribute to in the future. While v1.0.0 is a long way from being realized, here are a few things we would like to have in our first version of the API.

* Refactored backend. Logically speaking, the organization of the backend is not optimal. Refining and refactoring how we want the data stored will help us realize this project.
  + Photos for all four endpoints
  + Embedded links for /song endpoint (YouTube, Spotify, etc.)
* All four endpoints live and services
* A connected & hosted backend using MySQL
* A UI for a better user experience using HTML, CSS, & JS.

# References & Resources

**Tutorials & docs used to aid in project development**

* **Video Tutorials**
  + [IAmTimCorey- Intro to C# Web API’s](https://www.youtube.com/watch?v=vN9NRqv7xmY)
  + [TechWIthPat- How to create a web API with ASP.NET Core](https://www.youtube.com/watch?v=sWJayOop4k8)
* **Documentation**
  + [SwaggerUI Documentation](https://swagger.io/tools/swagger-ui/)
  + [MySQL Reference Manual for Version 8.0](https://dev.mysql.com/doc/refman/8.0/en/)
* **Other Tutorials** 
  + [FreeCodeCamp.org- How to create a REST API with .NET and C#](https://www.freecodecamp.org/news/create-a-rest-api-with-dot-net-5-and-c-sharp/)