A REPORT

ON

GILHARI SIMPLIFYING EXCHANGING OF JSON DATA WITH AN RDBMS

BY

Naman Shah -2022B4A70534G

R.V.S Aashrey Kumar - 2022A7PS0160H

Nishit Mukesh Patel - 2022B3A70568P

Mitul Goyal - 2022B4A31623H

AT

(Software Tree, California)

A Practice School-I Station of

BIRLA INSTITUTE OF TECHNOLOGY & SCIENCE,

PILANI (June,2024)

A REPORT

ON

GILHARI SIMPLIFYING EXCHANGING OF JSON DATA WITH AN RDBMS

BY

Name(s)	Id No.(s)	Discipline(s)
Naman Shah	2022B4A70534G	M.sc Maths + C.S.E
R.V.S Aashrey Kumar	2022A7PS0160H	C.S.E
Nishit Mukesh Patel	2022B3A70568P	M.sc Economics + C.S.E
Mitul Goyal	2022B4A31623H	M.sc Maths + E.E.E

Prepared in partial fulfillment of the

Practice School-I Course Nos.

BITS C221/BITS C231/BITS C241

AT

(Software Tree, California)

A Practice School-I Station of

BIRLA INSTITUTE OF TECHNOLOGY & SCIENCE, PILANI (June,2024)

Acknowledgement

First of all, I am grateful to Software Tree, LLC (Company) for providing us the opportunity to work for them as Summer Interns under Bits Pilani PS-I program and creating a great environment to learn both soft and hard skills.

We would also like to express our sincere gratitude to Damodar Periwal Sir, founder of Software Tree for taking his valuable time to guide us and provide us with invaluable inputs wherever required. We would also like to thank our supervisor Dr. Girish Kant Garg for his help throughout the process. We are thankful for all this help and guidance throughout this period .

BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE PILANI

(RAJASTHAN)

Practice School Division

Station: Software Tree Centre : California,U.S.A

Duration : 2 months Date of Start : 28/05/2024

Date of Submission: 20/06/2024

Title of the Project: GILHARI SIMPLIFYING EXCHANGING OF JSON DATA WITH AN RDBMS

ID No. / Name(s) / Discipline(s) / of the student(s) :

Naman Shah 2022B4A70534G M.sc Maths + C.S.E

R.V.S Aashrey Kumar 2022A7PS0160H C.S.E

Nishit Mukesh Patel 2022B3A70568P M.sc Economics + C.S.E

Mitul Goyal 2022B4A31623H M.sc Maths + E.E.E

Name(s) and designation(s) of the expert(s) : Damodar Periwal, Founder of Software Tree

Name(s) of the PS Faculty : Dr. Girish Kant Garg

Key Words: Glihari , Software Tree, RDBMS, Postman , Github etc.

Project Areas: Json data Integration, Microservices, Database systems, etc

Abstract: The project involves creating a stand-alone Java application for relational database management of JSON data using the Gilhari microservice framework.

Signature(s) of Student(s) Signature of PS Faculty

Date Date:

Table of Contents :

Introduction

Software Tree is a Silicon Valley-based startup that was founded in 1997 with the goal of making data integration simpler. Software Tree is a leader in technology, offering superior software infrastructure that is revolutionizing the paradigm for application/database integration. Its patented Object Relational Mapping (ORM) products have a common, tested architecture and offer comparable user interfaces for the Java, .NET, and Android markets.

JDX™ for the Java platform, NJDX™ for the.NET platform, and JDXA™ for the Android platform are three of Software Tree's quick, adaptable, and

feature-rich ORM technologies. Following a few well-considered KISS (Keep It Simple and Straightforward) guidelines, these cutting-edge and lightweight products offer a basic data integration capability to facilitate and expedite the development of contemporary object-oriented frameworks, tools, and applications.

The company's products enable application developers to focus on business logic and avoid wasting time on low-level infrastructure coding by elevating the level of data access abstraction. The key advantages of using our ORM products include:

- Increase developer productivity ... up to 70% by eliminating endless lines of complex JDBC/ADO.NET/SQL code
- Leverage legacy data ... avoids costly data conversion or transformation needs

- Shorten development cycles ... saves time, money, and frustration
- Create better performing and more flexible applications

About Gilhari

Gilhari™ is a microservice framework designed to provide persistence for JSON objects in relational databases. Available as a Docker image, Gilhari is highly configurable to fit specific application objects and relational models. It exposes a REST (REpresentational State Transfer) interface, offering APIs for CRUD (Create, Retrieve, Update, and Delete) operations on JSON objects tailored to the application.

Named after the Hindi word for squirrel, Gilhari efficiently transfers JSON data between an application and a database. This framework is particularly useful for leveraging existing data in legacy relational databases.

In terms of architecture, a Gilhari microservice focuses on exchanging JSON data with a relational

database and typically operates behind other application components, such as additional microservices or an API gateway, which handle business logic, authentication, and authorization. For scalability, a Gilhari microservice can be deployed in a container orchestration system like Kubernetes.

Few simple steps to install Gilhari

- First, compile the Java container class and save the resulting .class file in the bin folder. This requires using a specific command that includes necessary dependencies such as MySQL connector, JX classes, and JSON libraries.
- Since MySQL is used for the project, it is important to update the .jdx and .config files to reflect this.
- Next, create a Dockerfile that will download the base Gilhari image and apply the specific requirements for the application.
- Then, build the Docker image. This involves executing a command that specifies the Dockerfile and assigns a tag to the image.
- Finally, run the Docker image by executing a command that specifies the port mapping and platform.
- Once these steps are completed, the setup will be ready to go. It's a straightforward process.

Compelling Benefits of using Gilhari

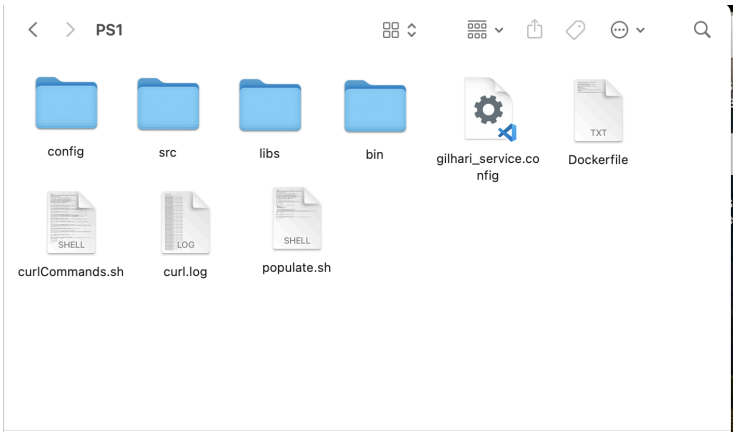
- Facilitate rapid development of new cloud and on-premises applications by simplifying the creation of flexible microservices for data integration.
- Streamline the creation of data pipelines from SaaS sources that export data in JSON format into JDBC-compliant databases and data warehouses.
- Enable easy retrieval of data in JSON format from any JDBC-compliant database for downstream processing.
- Support mobile clients in exchanging JSON data with relational databases on remote servers.
- Seamlessly ingest IoT (Internet of Things) data for storage, analysis, reporting, and device control.

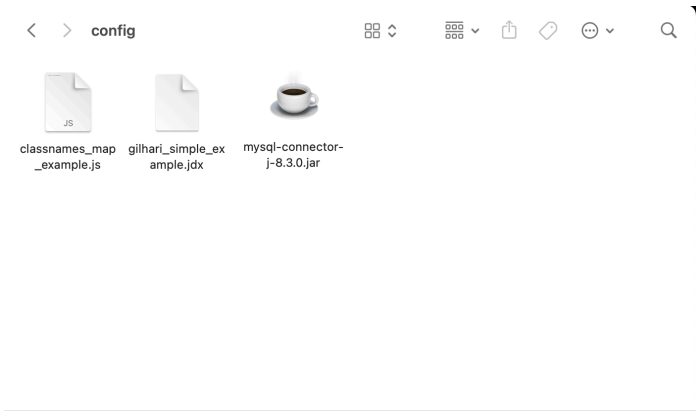
- Utilize existing schemas and data without relying on non-standard or absent native JSON data types in databases.
- Provide out-of-the-box RESTful data integration components tailored for standard data models across various industries, including banking, finance, energy, government, manufacturing, healthcare, insurance, telecommunications, and transportation.
- Simplify the integration of desktop and mobile applications with cloud data using Software Tree's lightweight Object Relational Mapping (ORM) products for Java and Android.
- Accelerate the introduction of new cloud services.

Main Text :

Detailed guide to configure Gilhari

Step1: Make a project directory with the following sub-folders.





- Here, ./config will contain files we acquired from Gilhari SDK downloaded from the Software Tree website.
- ./libs will contain all the jar files we are going to use to compile the .java container class.

- Once we compile the java file, .class file will be populated in the ./bin folder.

src/org/emp contains the java file we are going to compile.

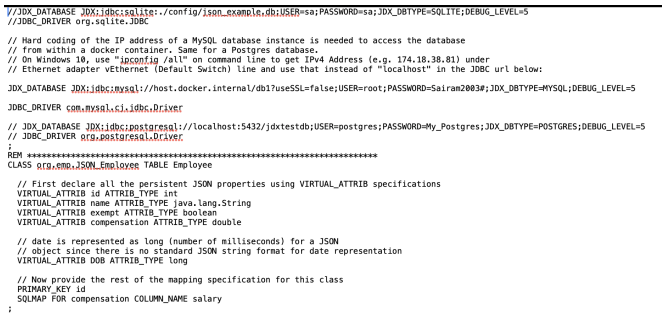
- Config directory contains classnames_map_example.js which contains the class mapping.
- The .jdx file in config specifies the database we will be using eg., MySQL and the JDBC specs along with ORM mapping.
- Config also contains the jdbc driver jar file as you can see.

We will be going into the details now.

Step 2: Updating the files according to system requirements

We have copied many files from the SDK but since every system will have different requirements, let us first update them accordingly.

gilhari_service.config file



.jdx ORM specification file

- As you can see here, the .config file contains the paths to the ORM spec file and the jdbc driver. It also mentions the path to the classnames_map_file,db_username and password.
- The .jdx file mentions the Database URL,JDBC driver path and the ORM mapping.

This is the container class that we are going to compile, when we compile we should make sure to include the following jar files in the classpath:

- jxclasses.jar: All the classes that do the magic and are provided with the SDK.
- json-202403030.jar: This will allow us to import JSONException and JSONObject in our container class.
- JDBC Driver

Now, we can go ahead and compile the java class in the terminal using the command: javac -cp config/mysql-connector-j-8.3.0.jar:libs/jxclasses.jar:libs/json-20240303.jar -d bin src/org/emp/JSON_Employee.java
Since we have mentioned the destinations(using -d tag) as the bin folder, we will get the .class file in the bin folder.
Step3: Build and run the Docker image

```
# Use the base image with gilhari already installed
FROM dperiwal/st_repo:gilhari

# Set the working directory inside the container
WORKDIR /opt/gilhari_simple_example

# Add necessary files and directories to the container
ADD bin ./bin
ADD config ./config
ADD gilhari_service.config .

# Expose port 8081 (assuming your application listens on this port)
EXPOSE 8081

# Specify the command to run when the container starts
CMD ["node", "/node/node_modules/gilhari_rest_server/gilhari_rest_server.js",
"gilhari_service.config"]
```

Dockerfile

As we can see, we are importing the base Gilhari image from dperiwal/st_repo and we imposed the app-specific

requirements by specifying ./bin,./config and the .config file here.

Gilhari will be listening on the port number 8081.

After creating the text file, next step will be to build the docker image on the terminal using the command:
docker build -t my_app_gilhari12 -f ./Dockerfile .

Then we will run the image using:
Run the docker image using: docker run -p 80:8081 --platform linux/amd64 my_app_gilhari12.

That’s it! You’re good to go.

Here is a high-level summary:

- **Update and modify files according to system requirements.**
- **Compile the container class.**
- **Make the docker image and run it.**

Populating the database

```
echo "** BEGIN OUTPUT **" > curl.log

echo "** Delete all Employee objects to start fresh" >> curl.log
curl -X DELETE "http://localhost:80/gilhari/v1/Employee" >> curl.log
echo "" >> curl.log

for i in $(seq 1 50)
do
  # Create a unique ID, name, compensation, and DOB
  ID=$i
  NAME="John$ID"
  COMPENSATION=$((54000 + i))
  DOB=$((381484800000 + i))

  # Add the curl command to the log file and execute it
  echo "** Inserting Employee $ID" >> curl.log
  curl -X POST "http://localhost:80/gilhari/v1/Employee" -H 'Content-Type: application/json' -d "{\"entity\":{\"id\"
  echo "" >> curl.log
done

echo "** Completed inserting Employees" >> curl.log
echo "" >> curl.log

echo "** Query all Employee objects" >> curl.log
curl -X GET "http://localhost:80/gilhari/v1/Employee" -H 'Content-Type: application/json' >> curl.log
echo "" >> curl.log

cat curl.log
```

- This shell script creates 50 sample JSON data and sends it to gilhari.
- Gilhari then internally converts these requests into SQL queries and communicates with the database using the JDBC driver we have mentioned in the classpath.

- This picture below demonstrates the high-level overview of the working of Gilhari microservice.

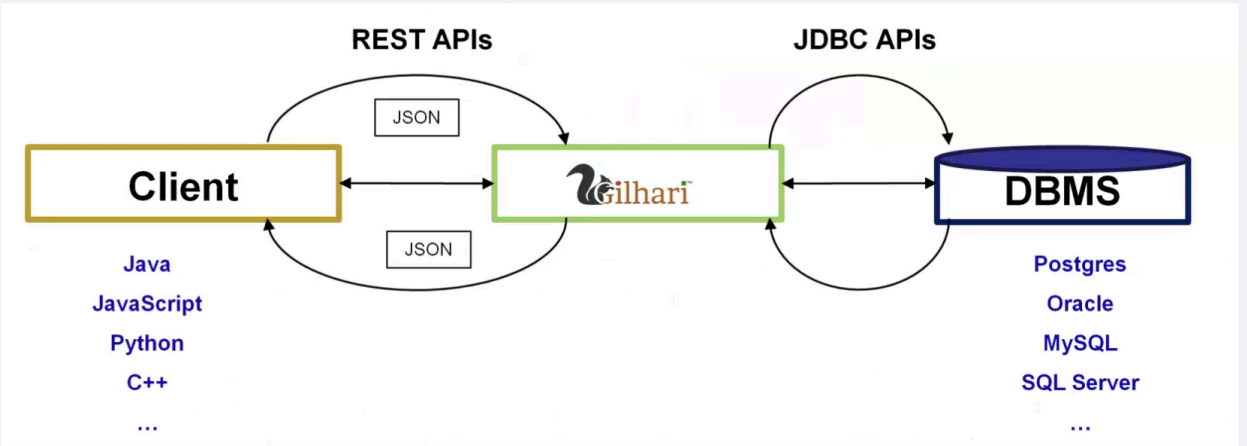
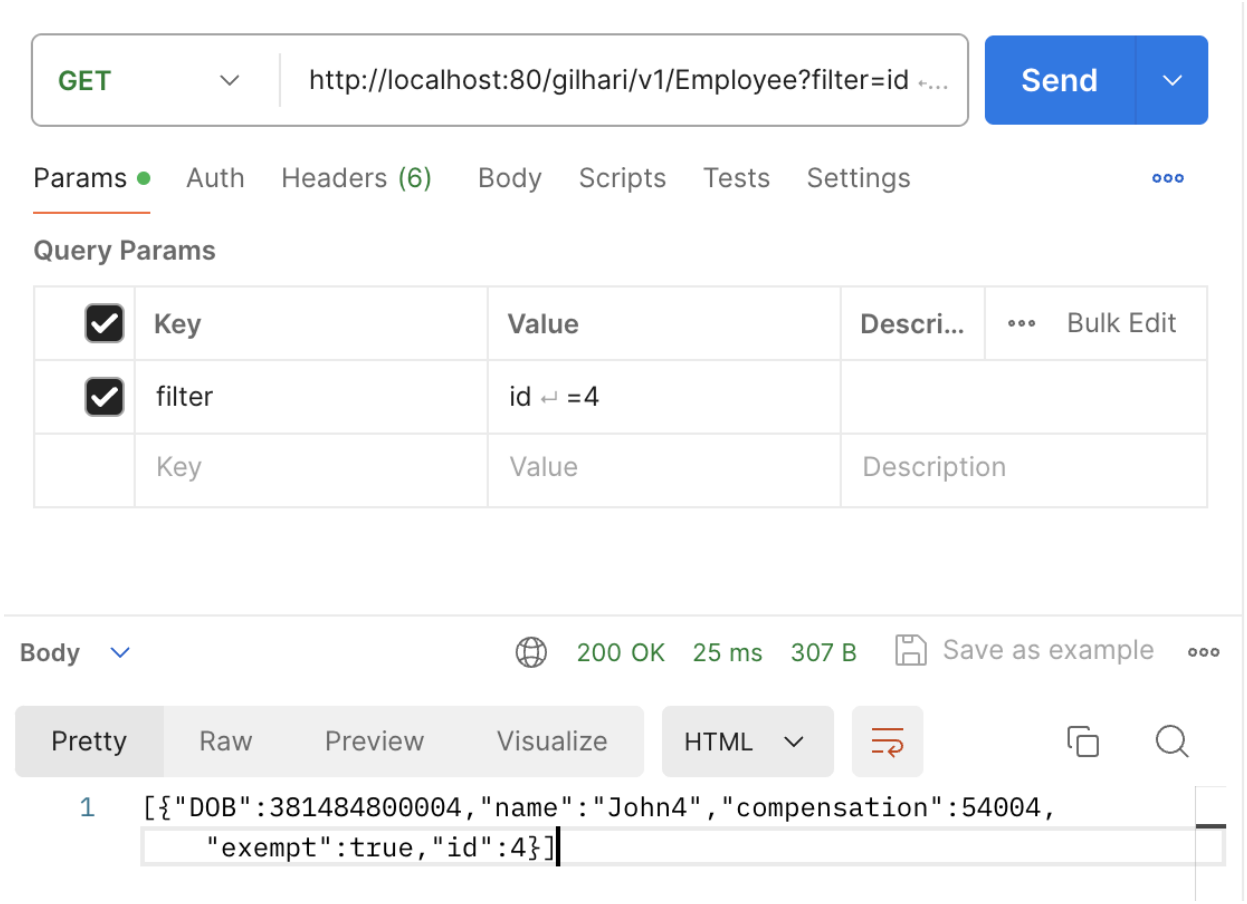


Table is populated as follows in the next slide:

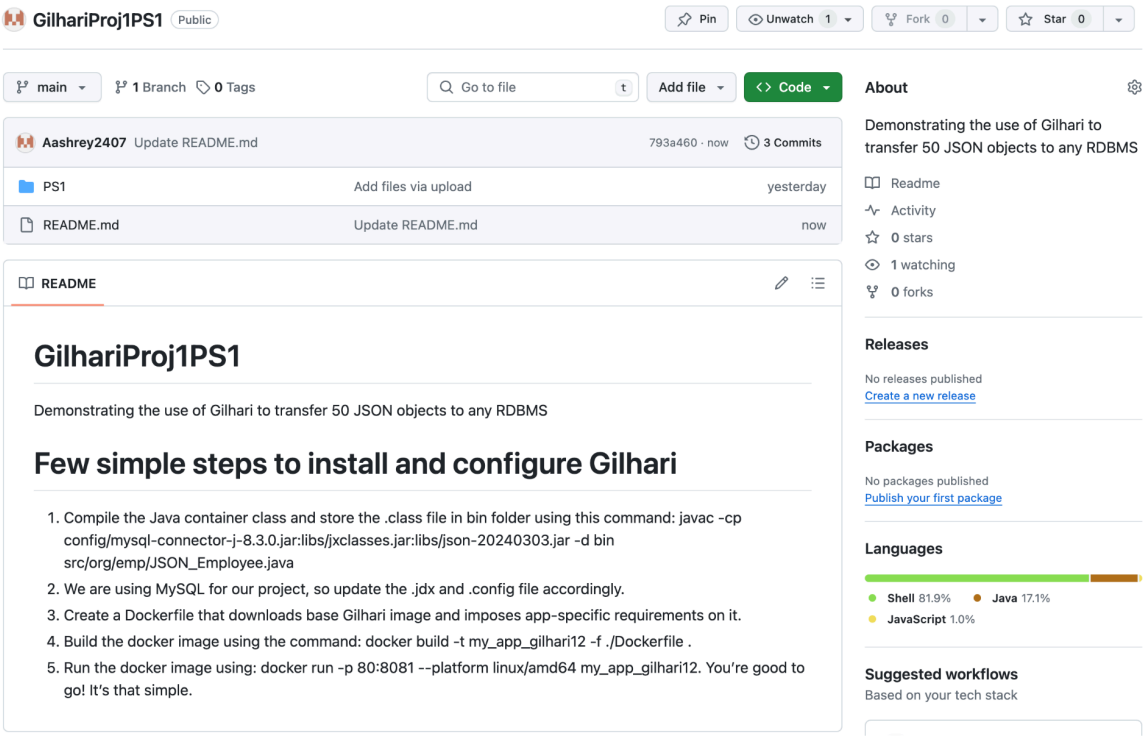
```
mysql> select * from Employee;
+--------+--------+--------------+--------+----+
| exempt | name   | DOB          | salary | id |
+--------+--------+--------------+--------+----+
|        | John1  | 381484800001 |  54001 |  1 |
|        | John2  | 381484800002 |  54002 |  2 |
|        | John3  | 381484800003 |  54003 |  3 |
|        | John4  | 381484800004 |  54004 |  4 |
|        | John5  | 381484800005 |  54005 |  5 |
|        | John6  | 381484800006 |  54006 |  6 |
|        | John7  | 381484800007 |  54007 |  7 |
|        | John8  | 381484800008 |  54008 |  8 |
|        | John9  | 381484800009 |  54009 |  9 |
|        | John10 | 381484800010 |  54010 | 10 |
|        | John11 | 381484800011 |  54011 | 11 |
|        | John12 | 381484800012 |  54012 | 12 |
|        | John13 | 381484800013 |  54013 | 13 |
|        | John14 | 381484800014 |  54014 | 14 |
|        | John15 | 381484800015 |  54015 | 15 |
|        | John16 | 381484800016 |  54016 | 16 |
|        | John17 | 381484800017 |  54017 | 17 |
|        | John18 | 381484800018 |  54018 | 18 |
|        | John19 | 381484800019 |  54019 | 19 |
|        | John20 | 381484800020 |  54020 | 20 |
|        | John21 | 381484800021 |  54021 | 21 |
|        | John22 | 381484800022 |  54022 | 22 |
|        | John23 | 381484800023 |  54023 | 23 |
|        | John24 | 381484800024 |  54024 | 24 |
|        | John25 | 381484800025 |  54025 | 25 |
|        | John26 | 381484800026 |  54026 | 26 |
|        | John27 | 381484800027 |  54027 | 27 |
|        | John28 | 381484800028 |  54028 | 28 |
|        | John29 | 381484800029 |  54029 | 29 |
|        | John30 | 381484800030 |  54030 | 30 |
|        | John31 | 381484800031 |  54031 | 31 |
|        | John32 | 381484800032 |  54032 | 32 |
|        | John33 | 381484800033 |  54033 | 33 |
|        | John34 | 381484800034 |  54034 | 34 |
|        | John35 | 381484800035 |  54035 | 35 |
|        | John36 | 381484800036 |  54036 | 36 |
|        | John37 | 381484800037 |  54037 | 37 |
|        | John38 | 381484800038 |  54038 | 38 |
|        | John39 | 381484800039 |  54039 | 39 |
|        | John40 | 381484800040 |  54040 | 40 |
|        | John41 | 381484800041 |  54041 | 41 |
|        | John42 | 381484800042 |  54042 | 42 |
|        | John43 | 381484800043 |  54043 | 43 |
|        | John44 | 381484800044 |  54044 | 44 |
|        | John45 | 381484800045 |  54045 | 45 |
|        | John46 | 381484800046 |  54046 | 46 |
|        | John47 | 381484800047 |  54047 | 47 |
|        | John48 | 381484800048 |  54048 | 48 |
|        | John49 | 381484800049 |  54049 | 49 |
|        | John50 | 381484800050 |  54050 | 50 |
+--------+--------+--------------+--------+----+
50 rows in set (0.01 sec)
```

POSTMAN Functionality:



- GET,POST,PUT and DELETE REST API requests can also be made through POSTMAN giving us a user-friendly interface to work with.
- Here, we have applied a filter(id=4) and since id is a PRIMARY KEY, we have only a single record showing up below.

Our GitHub repository



- We have also made a simple README file containing the steps in brief to install and configure Gilhari on your computer.

- We have attached the root project directory in the repository and any new user is absolutely welcome to refer to this to get going with using Gilhari.
- Gilhari is a developer’s friend and we hope it becomes a norm in the software industry to use Gilhari as it bridges the gap between growing usage of JSON with the legacy usage of RDBMS softwares.

CONCLUSIONS

In summary, this project effectively demonstrated the effectiveness and user-friendliness of the Gilhari microservice architecture for relational database management of JSON data. We showcased Gilhari's ability to simplify complicated data integration processes with minimal configuration and coding effort by building a standalone Java application that showed how easy it is to integrate and manipulate JSON data.

Additionally, thorough documentation was produced to help novice users set up and operate the Gilhari framework. When developers encounter the framework for the first time, this documentation makes it more approachable and less daunting by

providing straightforward, step-by-step instructions and explanations. We have enabled new users to comfortably utilize Gilhari's potential by making its subtleties accessible, which has led to a greater acceptance and comprehension of this potent tool among the development community.

Overall, the project has succeeded in demonstrating the simplicity and potency of the Gilhari architecture, highlighting its usefulness as a workable answer to contemporary data management issues.

REFERENCES

- **Gilhari Microservice Framework Documentation**
  - Software Tree. (2023). *Gilhari Microservice Framework*. Retrieved from Gilhari Introduction (softwaretree.com)
- **Java SE Documentation**
  - Oracle Corporation. (2023). *Java SE Documentation*. Retrieved from https://docs.oracle.com/javase/8/docs/
- **JSON Official Website**
  - JSON.org. (2023). *JSON (JavaScript Object Notation) Home Page*. Retrieved from https://www.json.org/
- **JDBC Driver Downloads**
  - MySQL. (2023). *MySQL Connector/J*. Retrieved from https://dev.mysql.com/downloads/connector/j/
- **IntelliJ IDEA Installation Guide**
  - JetBrains. (2023). *IntelliJ IDEA Download and Installation*. Retrieved from https://www.jetbrains.com/idea/download/

- **MySQL Installation Guide**
  - MySQL. (2023). *MySQL Installer for Windows*. Retrieved from https://dev.mysql.com/downloads/installer/
- **Software Tree Overview**
  - Software Tree Inc. (2023). *About Software Tree*. Retrieved from http://www.softwaretree.com
- **JSON Syntax and Data Formats**
  - Crockford, D. (2006). *The application/json Media Type for JavaScript Object Notation (JSON)*. Retrieved from https://www.json.org/json-en.html
- **RDBMS Overview**
  - Elmasri, R., & Navathe, S. B. (2015). *Fundamentals of Database Systems* (7th ed.). Pearson.
- **Java Database Connectivity (JDBC) Tutorial**
  - Oracle Corporation. (2023). *JDBC Basics*. Retrieved from https://docs.oracle.com/javase/tutorial/jdbc/
- **Eclipse IDE Installation Guide**
  - Eclipse Foundation. (2023). *Eclipse IDE for Java Developers*. Retrieved from https://www.eclipse.org/downloads/

- **NetBeans IDE Installation Guide**
  - Apache NetBeans. (2023). *Apache NetBeans IDE Download*. Retrieved from https://netbeans.apache.org/download/index.html
- **Relational Databases and JSON Integration**
  - O'Neil, P., & O'Neil, E. (2020). *Database: Principles, Programming, and Performance*. Morgan Kaufmann.
- **Java Best Practices for JSON Data Handling**
  - Bloch, J. (2018). *Effective Java* (3rd ed.). Addison-Wesley.
- **Software Engineering Principles**
  - Pressman, R. S. (2019). *Software Engineering: A Practitioner's Approach* (9th ed.). McGraw-Hill Education.
- **Gilhari Framework Integration with RDBMS**
  - Software Tree Inc. (2023). *Gilhari Integration Guide*. Retrieved from http://www.softwaretree.com/gilhari/integration
- **Java Programming for Beginners**
  - Schildt, H. (2018). *Java: The Complete Reference* (11th ed.). McGraw-Hill Education.

- **MySQL Database Setup and Management**

  - DuBois, P. (2018). *MySQL Cookbook* (4th ed.). O'Reilly Media.

- **Comprehensive Guide to JSON in Java**

  - Fette, I., & Melnikov, A. (2011). *The WebSocket Protocol*. Retrieved from https://tools.ietf.org/html/rfc6455

- **Gilhari Framework Overview and Benefits**

  - Software Tree Inc. (2023). *Why Use Gilhari?*. Retrieved from http://www.softwaretree.com/gilhari/why

Glossary

- ORM-Object-Relational Mapping (ORM) is a technique that allows developers to interact with a relational database using an object-oriented approach
- JDX- typically refers to a file extension or format used in database management or data exchange.
- JSON-stands for JavaScript Object Notation. It is a lightweight data-interchange format that is easy for humans to read and write and easy for machines to parse and generate.
- JDBC-stands for Java Database Connectivity. It is a Java API that enables Java applications to interact with databases, allowing developers to execute SQL queries, retrieve and update data, and manage database connections from within Java programs
- REST interface- also known as a RESTful interface, refers to a set of principles for designing networked applications that use HTTP or HTTPS for communication.
- API-stands for Application Programming Interface. In brief, an API is a set of rules and protocols that allow different software applications to communicate with each other.


Thank You