Software Development

- Introduction to Docker
- Software Development Crash Course
- Introduction to Backend Development

Introduction to Docker

Click here for the Docker workshop slides: GBM #4_ Docker Workshop - Ezri Slides.pdf

What is Docker

Docker is a platform and tool for developing, shipping, and running applications by using containerization technology. ***Containers*** allow a developer to package up an application with all of the parts it needs, such as libraries and other dependencies, and ship it all out as one package. This ensures that the application will run on any other Linux machine regardless of any customized settings that machine might have that could differ from the machine used for writing and testing the code.

This provides consistency when working on a team where people run different operating systems.

In Blueprint we use Docker containers in several of our applications to manage different services and projects.

- Install Docker on Mac
- Install Docker on Windows
- Install Docker on Linux

Additional Resources

- Docker Docs
- Comprehensive Docker Breakdown
- Docker Crash Course (video)

Software Development Crash Course

This workshop covers some tools and best practices that will help you develop software more efficiently, both independently, but especially as a TEAM. We attached the following resources for anyone who missed the workshop or would like a refresher:

- Workshop Recording
- Slide Deck
- GitHub Actions Sample Activity
- Tech Team Onboarding Guide
- Project Management & Agile at Blueprint

Introduction to Backend Development

Controller-Service-Repository Pattern

The controller-service-repository (CSR) pattern is a pattern used to build software applications.

It divides the software into three distinct sections, here's a quick summary, see further below for more:

Controller: Processes incoming requests and returns a response (like a front desk job). This communication most likely uses HTTP.

Service: Processes data from controller, then, based on controller data, communicates with repository. For example, if the data from the controller is requesting specific information, the service layer can get information from the repository, process the retrieved information, and then return it to the controller.

Communication between the controller and service layer uses DTO (Data Transfer Objects) which is essentially nicely packaged data. The controller may receive an HTTP request in a JSON format, then convert it to a DTO which is sent to the service layer.

Repository: The service talks to the repository/data layer for persistence (making sure the data is maintained even after the program is run). It handles all the details of accessing and storing data.

Controller:

What does the Controller layer do?

The controller handles incoming requests and returns appropriate responses (usually using HTTP (Hypertext Transfer Protocol) and TCP (Transmission Control Protocol)).

REST, API, and HTTP

An API establishes an interface for two processes to communicate. This could be a way for an app on your phone to communicate with your phone’s operating system, it could also be a web API for communicating over the internet. Whenever you use the “Login with Google” feature, the website you are trying to log in to (using your Google account) will send an API request to Google’s API, which will send a response confirming your identity.

REST (Representational State Transfer) is a software architecture that has rules for how APIs should communicate. If an API follows the principles of REST, then it is a "RESTful API" (a.k.a. REST API).

In order to communicate between them, HTTP (Hypertext Transfer Protocol) methods (such as GET, POST, DELETE) are used.

Service:

What does the Service layer do?

The service layer receives data from the Controller layer and performs "business logic" by communicating with the Repository layer to fetch or store data.

Business Logic

It's a very broad term, but it generally means the logic that relates the software to the real-world--as opposed to the lower-level logic such as displaying something to your screen.

For example, this may include data validation, or making sure that the data entered is the correct type. Let's say a user was making changes to a database and wanted to enter a string where the cost of a cookie should be. We, as humans, understand that the cost of a cookie should be a dollar amount, not some blurb of text; business logic is our way of telling a program this so it will not allow the change to happen.

As another example, let's say the controller is asking for the number of cookies left in a jar. The service layer would communicate with the repository layer to get the number of cookies originally in the jar and the number of cookies taken out of the jar. Then, it would subtract to two for the number of remaining cookies and return it.

Repository:

What does the Repository layer do?

The repository layer is focused on handling data and modifying and retrieving data from the database.

SQL and PostgreSQL

Structured Querying Language (SQL) is a language used by Relational Database Management System (RDBMS) such as PostgreSQL in order to manage databases.

The repository will communicate with the PostgreSQL database to get or change information.

Java Persistence API (JPA)

Bridges the gap between object-oriented programming and relational databases. Basically, it maps Java objects to database tables.

"The Java Persistence API (JPA) is the Java API specification that bridges the gap between relational databases and object-oriented programming by handling the mapping of the Java objects to the database tables and vice-versa. This process is known as the Object Relational Mapping (ORM). JPA can define the way Java classes (entities) are mapped to the database tables and how they can interact with the database through the EntityManager, the Persistence context, and transactions."

JPA - Introduction - GeeksforGeeks

Relational Database

What is a Relational Database?

A relational database is--as the name suggests--a type of database. Data is stored in multiple tables, all with some relation to each other.

For any given table, each row has a unique ID called a "key".

Users:

id	first_name	last_name	account_balance
1	Jane	Doe	$2203
2	John	Doe	$1039
3	Ada	Lovelace	$5000

Orders:

order_num	customer	payment_amount
1	2	$50
2	3	$200
3	1	$80
4	2	$70

As shown in the example above, every row in both tables has a "key" (**id** and **order_num**). Additionally, they are related because every order has a customer **id** number associated with it

(ie., the "**customer**" column in the Orders table contains the "**id**" in the Users table).

For example, for order 1, the row has 2 listed as the customer, which relates back to the **id** of John Doe in the Users table.

Resources

- you need to learn SQL RIGHT NOW!! (SQL Tutorial for Beginners)
- What Is a Relational Database | Oracle: https://www.oracle.com/database/what-is-a-relational-database/

SQL

What is SQL?

SQL stands for Structured Query Language.

It is used by a Relational Database Management System (RDBMS) such as PostgreSQL, MySQL, etc., as a "language" to manipulate a Relational Database.

SQL vs NoSQL Database

A NoSQL database is a non-relational database meaning that the structure is different from that of an SQL database. Instead of tables, NoSQL databases typically store information as JSON objects. Overall, it is a much more dynamic system since there does not need to be a predefined schema that each item needs to conform to. In the end, NoSQL databases are better suited if you have unstructured data.

One prominent example of a NoSQL database is MongoDB. It uses JSON-like documents to store data, and it organizes these documents into collections to be queried. This collection method of organizing data can make data retrieval faster in some use cases, particularly when dealing with large amounts of data.

Resources

Videos

- you need to learn SQL RIGHT NOW!! (SQL Tutorial for Beginners) - Network Chuck: https://youtu.be/xiUTqnI6xk8
- Overview of Scaling: Vertical And Horizontal Scaling - GeeksforGeeks
- SQL vs. NoSQL: What's the difference? - IBM Technology: https://youtu.be/Q5aTUc7c4jg
- MongoDB in 100 Seconds - Fireship: https://youtu.be/-bt_y4Loofg

- MySQL - How to run SQL file or script from the terminal | sebhastian

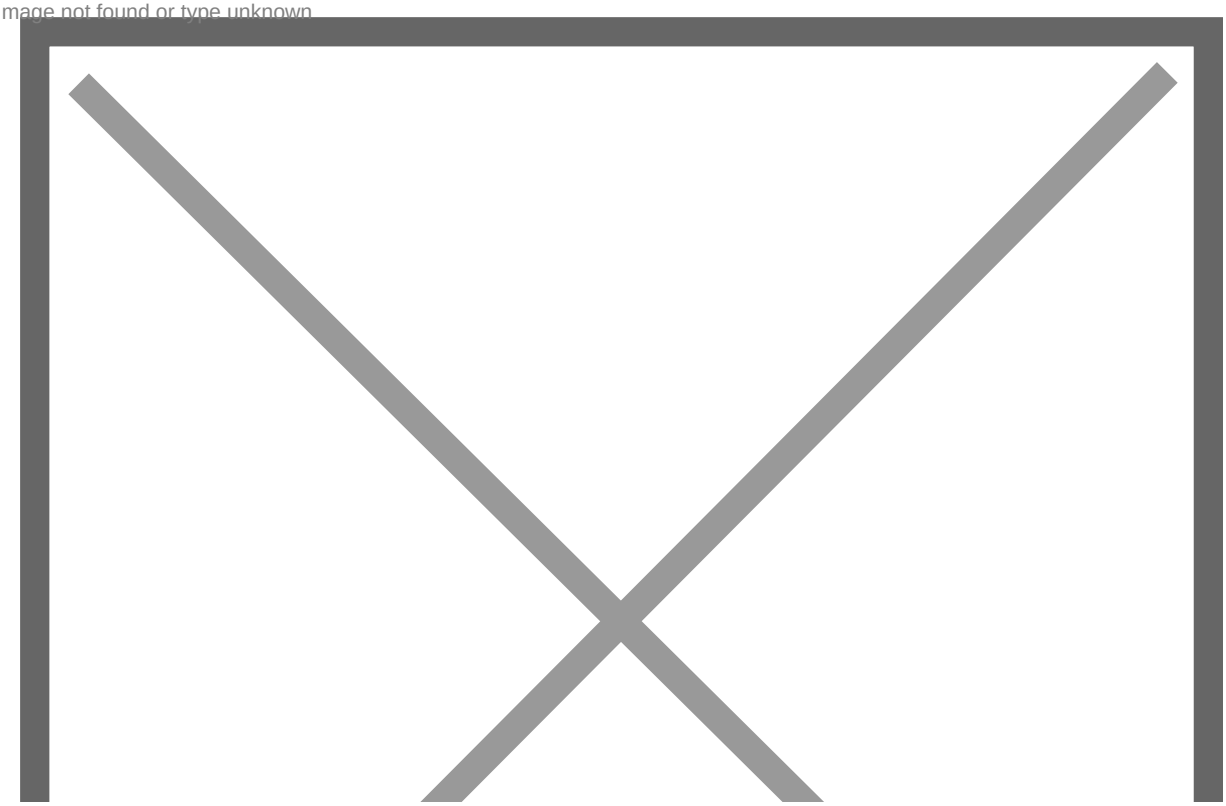
HTTP

What is HTTP

HTTP or Hypertext Transfer Protocol is a protocol (as the name suggests) that is used to communicate between client and server on the world wide web.

This is probably used in your everyday life! Whenever you search a URL for example, an HTTP request is sent to a server which sends back an appropriate response; most of the time, it's the webpage you were looking for.

Image not found or type unknown



GET, POST, DELETE

GET

A GET request is used to retrieve information from a server. For example, if we want to access a webpage, we can use a GET request to get information from the webserver.

POST

A POST request is used for creating new data. If you were to make a new account, a POST request would be used to create the account on the server

DELETE

This is used to delete data from a server such as removing a user profile.

What Does HTTP Look Like?

HTTP is based on a number of methods, however the most basic ones are GET and POST .

Sample Request:
The snippet below asks for a resource at example.com/contact:

```
GET / HTTP/1.1
Host: example.com
User-Agent: curl/8.6.0
Accept: */*
GET - HTTP | MDN
```

In this case, we are doing GET / indicating that we are getting the root of example.com. User-Agent and Accept are examples of headers which specify additional arguments to include in the request. User-Agent specifies information such as browser information (or in this case the version of the curl command). Accept indicates the type of response that the request will accept (which is all in this case).

You can try this request yourself in the terminal (use Command Prompt, not PowerShell for this if you are on windows). The -i flag simply signals to include response headers.

Without headers:

```
curl -i -X GET example.com
```

With headers:

```
curl -i -X GET http://example.com -H "User-Agent: curl/8.6.0" -H "Accept: */*"
```

> In this scenario, -X GET is redundant since it is the default method.

Sample Response
In our case, we sent a request to the webpage, so we will get the HTML for that webpage back. However, we will also receive the response headers. It will look something like this:

```
HTTP/1.1 200 OK
Content-Type: text/html
ETag: "84238dfc8092e5d9c0dac8ef93371a07:1736799080.121134"
Last-Modified: Mon, 13 Jan 2025 20:11:20 GMT
Cache-Control: max-age=1405
Date: Thu, 16 Jan 2025 17:46:37 GMT
Content-Length: 1256
Connection: keep-alive
```

The first line includes our status code, in this case 200, which means OK. Others include 404 (Not Found), 403 (Access Denied), 503 (Service Unavailable) among others.

Resources

- What is HTTP ? - GeeksforGeeks
- HTTP request methods - HTTP | MDN
- How to use curl on Windows – 4sysops

REST API

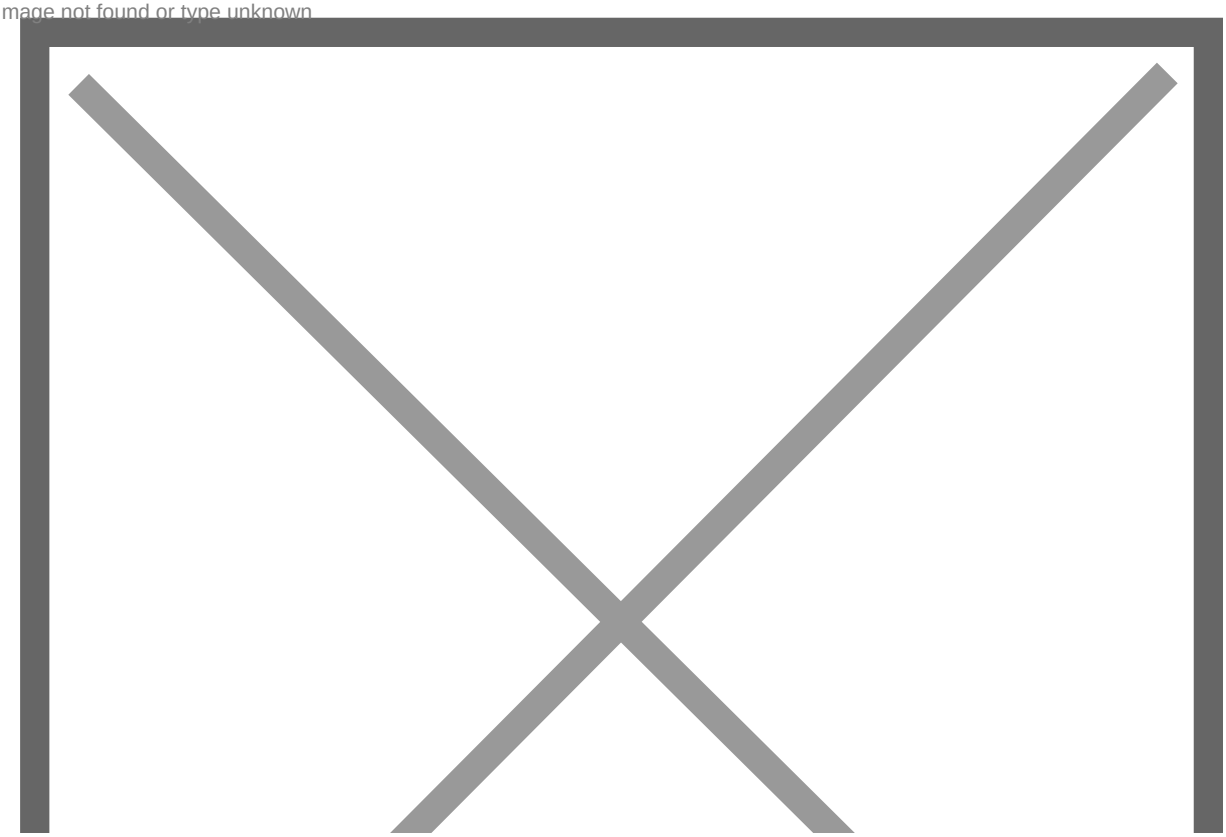
What is REST?

REST stands for Representational State Transfer. It is a software architecture that has rules for how APIs should communicate. If an API follows the principles of REST, then it is a "RESTful API" (a.k.a. REST API).

What are the guidelines for REST?

What is REST?: REST API Tutorial

Image not found or type unknown



For our specific context, we will look at how HTTP (a method of communication for APIs) can be RESTful

1. **Uniform Interface**
   HTTP uses standard methods such as GET, POST, and DELETE.
2. **Client-Server**
   The UI is handled by the frontend while the business logic and data storage is handled by the backend.
3. **Stateless**
   HTTP requests are self-contained, so they don't need information from previous requests.
4. **Cacheable**
   HTTP responses can be "cached" meaning that they can be stored for later use. For example, webpages can be cached so they can be loaded faster.
5. **Layered System**
   Gateways and proxies are layers between the client and server. A proxy might redirect requests to different servers to prevent overloading one particular server.
6. **Code on Demand**
   When a server sends code to run client-side (locally) on your computer such as JavaScript. This is not too common due to security concerns; it's also why you'll sometimes get a popup saying, "Allow website to run JavaScript."

Dependency Injection

What is Dependency-Injection?

Dependency-Injection (DI) is a design pattern used in Object-Oriented languages, such as Java, as a way to decouple classes from dependencies, or make the classes more separated from their dependencies. This is done in order to make testing and maintaining the code easier.

What Does it Look Like?

Example from: Dependency Injection Made Simple with Java Examples | Clean Code and Best Practices | Geekific: https://youtu.be/GATSXm7WAxU

Before DI:

In order to better understand what DI is and the problem it is supposed to solve, let's see some code without DI.

```
public interface Food {}


public class Burger implements Food {}


public class Chef
{
	private Food burger;

    public Chef ()
    {
    	burger = new Burger();
    }

    public void prepareFood()
    {
    	//do something with burger
    }
}
```

Notice how the `Chef` class instantiates a `Burger` object (in the line `burger = new Burger()`). In this case, the `Burger` class is a dependency of the `Chef` class. The `Chef` class needs the `Burger` class in order to run.

This is okay if we only prepare burgers, but if we want other foods then it may look something like this. Let's say we introduce a `Pizza` class:

```
public class Pizza implements Food {}
```

Then our `Chef` will look something like this:

```
public class Chef
{
    private Food burger;
    private Food pizza;

    public Chef ()
    {
    burger = new Burger();
    pizza = new Pizza();
    }

    public void prepareBurger()
    {
        //do something with burger
    }

    public void preparePizza()
    {
        //do something with pizza
    }
}
```

There are other ways as well, such as having multiple instances of the `Chef` class. Either way, this makes things very interdependent; every time we change menus, we will need to update the `Chef` class as well.

After DI

Let's keep with the same example as before:

```
public interface Food {}


public class Burger implements Food {}


public class Pizza implements Food {}
```

However this time with the `Chef` class:

```
public class Chef
{
private Food food;
```

```
    public Chef (Food someFood)
    {
    this.food = someFood;
    }

    public void prepareFood()
    {
    //do something with burger
    }
}
```

Notice the changes to the constructor. When initializing the `Food` class, we will pass the dependency as a parameter, or **"inject"** the dependency.

We can then instantiate the `Chef` object as such:

```
Chef burgerChef = new Chef(new Burger());

Chef pizzaChef = new Chef(new Pizza());
```

Now, it doesn't matter what food is being added or removed, it (as the dependency) will simply be "injected" in.

Types of Dependency Injection

The example shown above is constructor injection, since the dependency is injected in the constructor.

There is also setter injection where a setter method takes a dependency as input and assigns it to an object variable.

Finally, there is field injection where the dependency is set outside of the class it is being injected into.

Generally, constructor injection is favored because the dependency is apparent in the constructor signature, which is not true for the other two methods.

Resources

- Example from: Dependency Injection Made Simple with Java Examples | Clean Code and Best Practices | Geekific: https://youtu.be/GATSXm7WAxU
- design patterns - What is dependency injection? - Stack Overflow
- Java Dependency Injection (DI) Design Pattern - GeeksforGeeks

- SQL Server Sample Database