# ENSF 381
# Full Stack Web Development

## Lecture 25: Backend

**Slides: Ahmad Abdellatif, PhD**

**Instructor: Novarun Deb, PhD**

UNIVERSITY OF CALGARY

# Outline

- Overview of the Backend.
- Importance of Backend.
- HTTP Request and Response.
- HTTP Methods.

# What is backend development?

- Backend development refers to the **server-side** of an application.

- While frontend focuses on the user interface and client-side interactions, backend handles the server-side logic, ensuring the application's functionality.

- Backend and frontend development work collaboratively to create a seamless user experience.

# Importance of backend in web development

- Business Logic: the core functionality and rules of the application. Also, it processes data on the server.
- Data Management: manages and organizes data storage efficiently, ensuring data integrity and security.
- Authentication and Authorization: handles user authentication and authorization processes.
- Server-Side Validation: verifies data submitted by users before storing it in the database.

# Backend tools and technologies

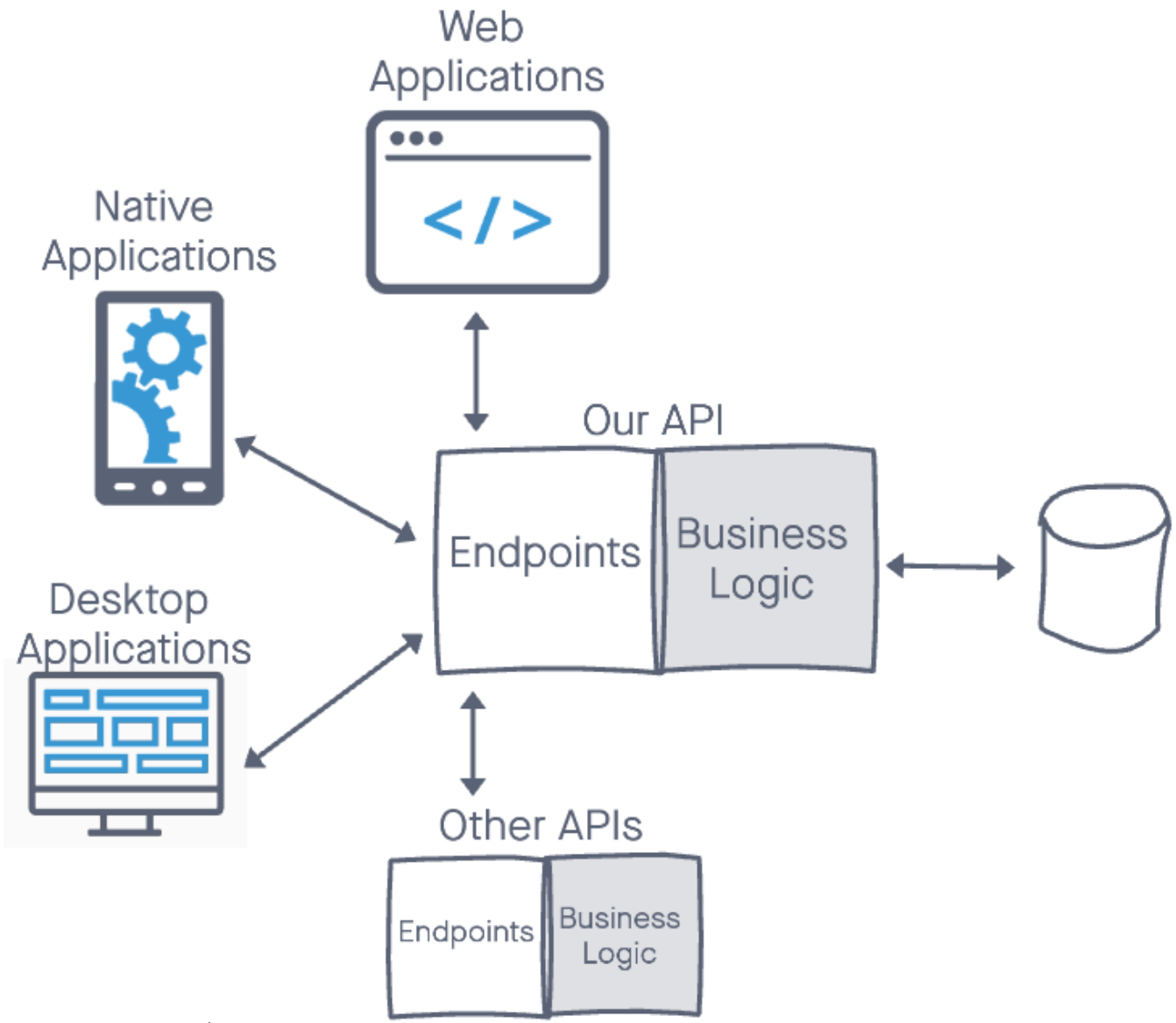
Backend developers employ various tools and technologies in their daily tasks:

- PHP
- Ruby
- Java
- Node.js
- Django
- Flask
- Express
- Ruby on Rails

# What is Application Programming Interface (API)?

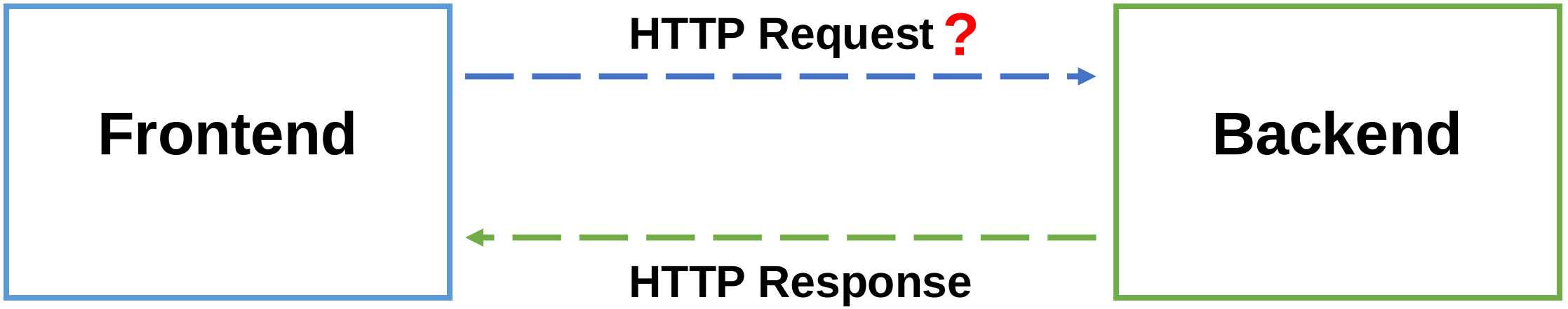
- A set of rules and protocols for building software applications that allow them to communicate with each other over the internet.
- It defines the methods and data formats that applications can use to request and exchange information.
- Web APIs are commonly used to enable the integration of different software systems, allowing them to work together and share data.

# Example on API

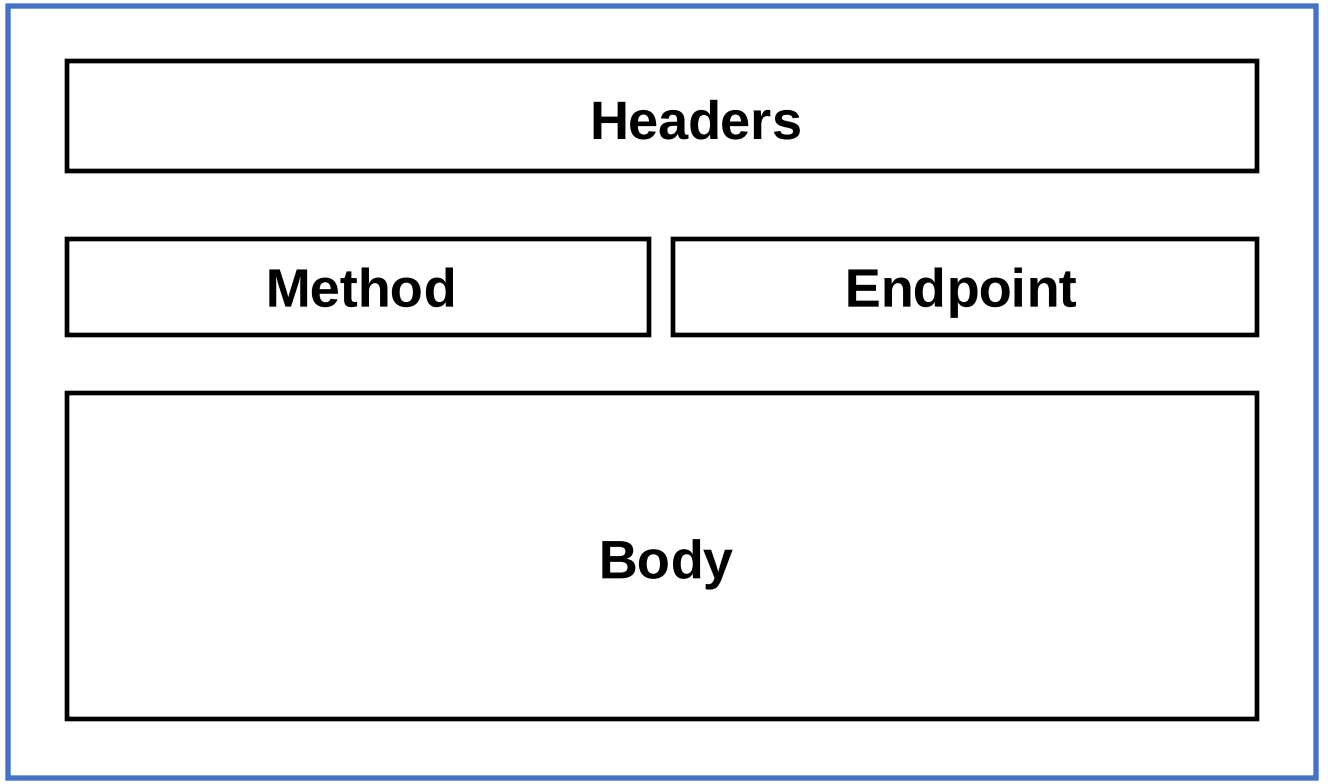



https://www.dotnetnakama.com/blog/designing-a-restful-web-api/

# HTTP - Example



# HTTP Request




**Headers**: contains information related to some authentication data, and browser type.

**Endpoint**: the URL of the API endpoint.

**Body**: contains data that needs to be sent to the server.

**Method**: Indicates the HTTP method used (e.g., GET, POST, PUT, DELETE).

# CRUD

- Represents basic operations that can be performed on data in a database or a persistent storage system.

- These operations are fundamental in the context of data management and are commonly associated with database systems and web applications.

- CRUD is an acronym that stands for Create, Read, Update, and Delete.
  - Create (C): involves creating new records or entries in a database.
  - Read (R): involves retrieving or reading data from a database.
  - Update (U): involves modifying existing records or entries in a database.
  - Delete (D): involves removing records or entries from a database.

# HTTP methods - GET

GET: retrieve data from a specified resource.

- Example: fetching a webpage, image, or any resource without modifying the server's state.
- GET is the **default** method.

```
fetch('https://api.example.com/data')
  .then(response => response.json())
  .then(data => {
    console.log('GET response:', data);
    // Handle the retrieved data
  })
  .catch(error => console.error('GET error:', error));
```

# HTTP methods - POST

POST: submit data to be processed to a specified resource.

- Example: submitting a form, uploading a file, or making a request that results in the creation of a new resource on the server.

```
const postData = { key: 'value' };

fetch('https://api.example.com/data', {
  method: 'POST',
  headers: {
    'Content-Type': 'application/json',
  },
  body: JSON.stringify(postData),
})
  .then(response => response.json())
  .then(data => {
    console.log('POST response:', data);
    // Handle the response data
  })
  .catch(error => console.error('POST error:', error));
```

# HTTP methods - PUT

PUT: update or modify a resource on the server.

- Example: updating user profile information, uploading a new version of a file.

```
const putData = { updatedKey: 'updatedValue' };

fetch('https://api.example.com/data/123', {
  method: 'PUT',
  headers: {
    'Content-Type': 'application/json',
  },
  body: JSON.stringify(putData),
})
  .then(response => response.json())
  .then(data => {
    console.log('PUT response:', data);
    // Handle the response data
  })
  .catch(error => console.error('PUT error:', error));
```

# HTTP methods - DELETE

DELETE: delete a specified resource.

- Example: deleting a user account, removing a file from a server.

```
fetch('https://api.example.com/data/123', {
  method: 'DELETE',
 })
  .then(response => {
    if (!response.ok) {
      throw new Error('DELETE request failed');
    }
    console.log('DELETE request successful');
    // Handle success
  })
  .catch(error => console.error('DELETE error:', error));
```

# HTTP methods

- GET: retrieve data from a specified resource.
  - Example: fetching a webpage, image, or any resource without modifying the server's state.
- POST: submit data to be processed to a specified resource.
  - Example: submitting a form, uploading a file, or making a request that results in the creation of a new resource on the server.
- PUT: update a resource or create a new resource if it does not exist.
  - Example: updating user profile information, uploading a new version of a file.
- DELETE: delete a specified resource.
  - Example: deleting a user account, removing a file from a server.

# Question…

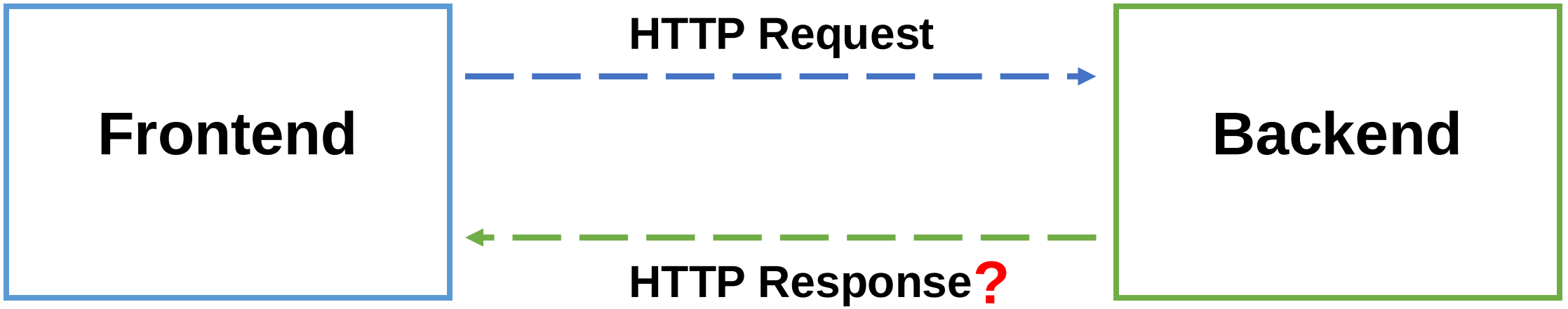
**Is it possible to send data using the GET method?**

Yes. While it is technically possible to include data with a GET request using query parameters, **it is not recommended**, especially for sensitive or large amounts of data.

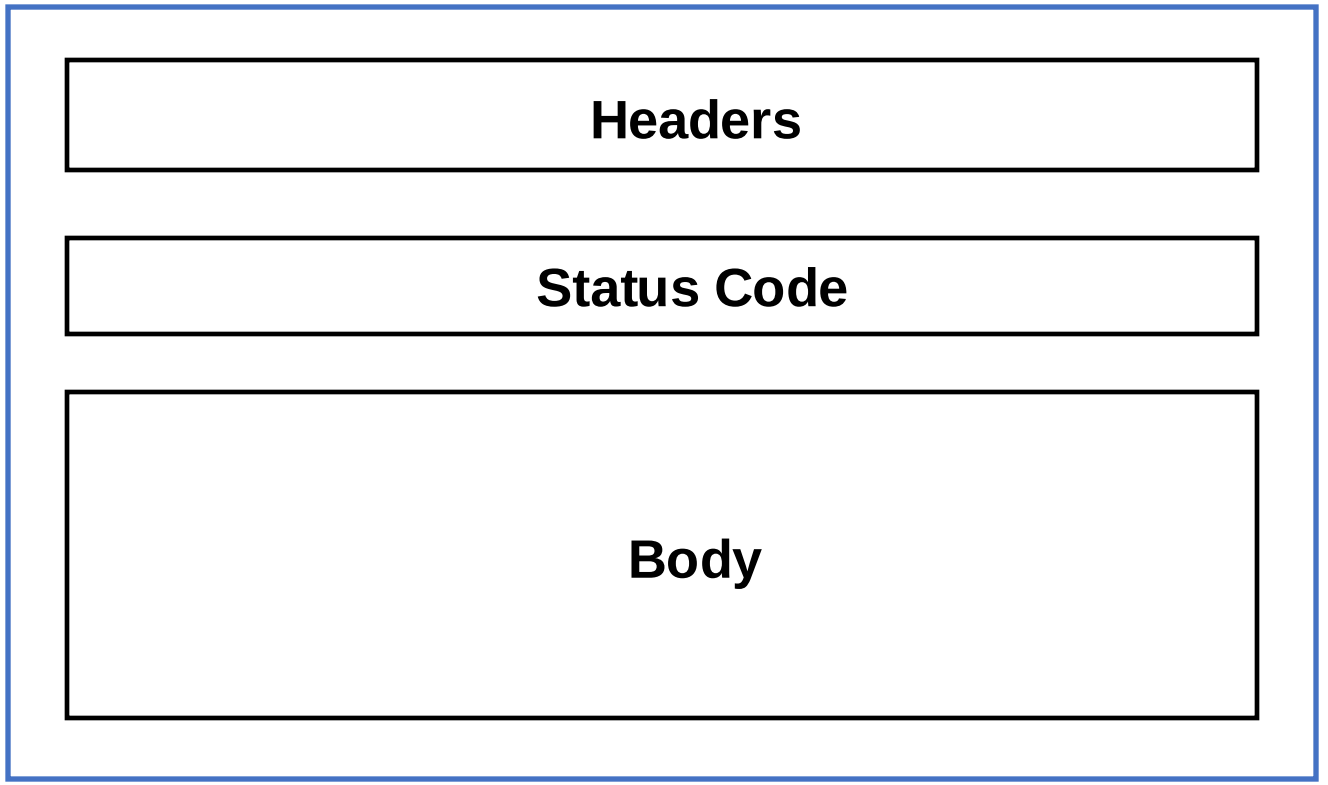
**Is it possible to retrieve data using the POST method?**

Yes, it is possible to retrieve data using the POST method. In some cases, using POST for data retrieval may be necessary, particularly when dealing with **large payloads or sensitive information**.

# HTTP - Example



# HTTP Response




**Headers**: Specifies the format of the data being sent (e.g., text/html, application/json), and the size of the response body in bytes.

**Status Code**: a three-digit numeric code indicating the outcome of the request such as 200 (OK), 404 (Not Found), and 500 (Internal Server Error).

**Body**: Contains the actual data or resource requested by the client.

# Common HTTP status code categories

- 200: OK
  - The request was successful, and the server has returned the requested data.
- 400: Bad Request
  - The server cannot process the request due to a client error, such as malformed syntax or invalid parameters.
- 401: Unauthorized
  - The user is not authorized to perform an operation.
- 404: Not Found
  - The requested resource is unavailable.
- 500: Internal Server Error
  - Unexpected condition was encountered on the server.
- 503: Service Unavailable
  - The server can not handle the request, often due to temporary overloading or maintenance.

# Authentication and authorization

- Authentication and authorization are critical components for ensuring the security of applications.

- **Authentication**: the process of verifying the identity of a user, ensuring they are who they claim to be.

- **Authorization**: granting or denying access to specific resources or actions based on the authenticated user's permissions.

# Questions

Reminder: The deadline for Assignment 3 is **today at 11:59 PM**.