

Lecture 6: Backend Development

IST 3108 Application Development

Dr. Peter Khisa Wakholi

Dept of Information Systems, Makerere University

Agenda

- Backend Programming
 - Why Back-end
 - Node.Js
 - Firebase
- Back-end Programming practices
 - APIs
 - Secuirty
 - States
 - Microservices
 - Deployment

Reference List

- **Node.js:**
 - [Node.js Official Documentation](#): The official documentation covers all aspects of Node.js.
 - [Node.js Best Practices](#): A comprehensive guide to Node.js best practices on GitHub.
- **API Design:**
 - [REST API Design Best Practices](#): A guide to RESTful API design principles.
 - [Understanding RESTful APIs](#): A Mozilla Developer Network (MDN) guide to RESTful APIs.
- **State Management:**
 - [JWT Introduction and Examples](#): Learn about JSON Web Tokens for state management.
 - [Authentication in Express.js](#): Express.js guide on user authentication.
- **Privacy & Security:**
 - [Express.js Security Best Practices](#): Best practices for security in Express.js.
 - [OWASP \(Open Web Application Security Project\)](#): A comprehensive resource for web application security.
- **Microservices Architecture:**
 - [Microservices Architecture: What Is It and How to Build It?](#): NGINX guide on microservices architecture.
 - [Microservices Patterns](#): A collection of microservices design patterns.
- **Backend Deployment:**
 - [Heroku Dev Center](#): Heroku's official documentation for deployment.
 - [AWS Deployment Guide](#): Amazon Web Services (AWS) deployment guide.

Back-end Programming

Part 1

Why we need backends

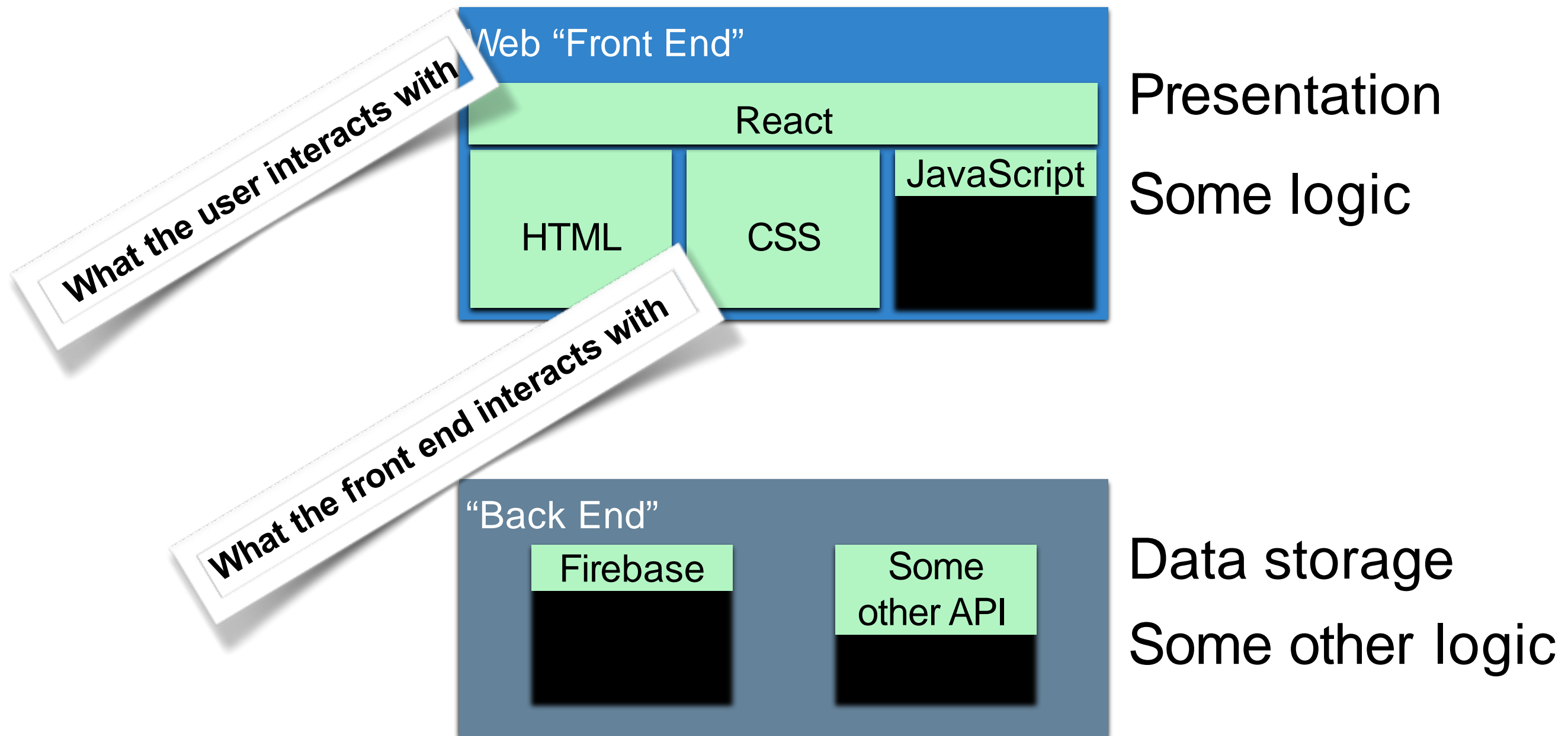
Performance:

- Do heavy computation on more powerful machines Do data-intensive computation “nearer” to the data
- Avoid duplicating computation (do it once and cache)

Security: *SOME* part of our code needs to be “trusted”

Validation, security, etc. that we don't want to allow users to bypass

Dynamic Web Apps



Front-end Vs Back-end

- **Frontend**

- Pros

- Very responsive (low latency)

- Cons

- Security Performance
- Unable to share between front-ends

- **Backend**

- Pros

- Easy to refactor between multiple clients
- Logic is hidden from users (good for security, compatibility, and intensive computation)

- Cons

- Interactions require a round-trip to server

Divide Between Front and Back-end

Decision need to be made on what to do at the back-end Vs Front-end

```
firebase.initializeApp(config);

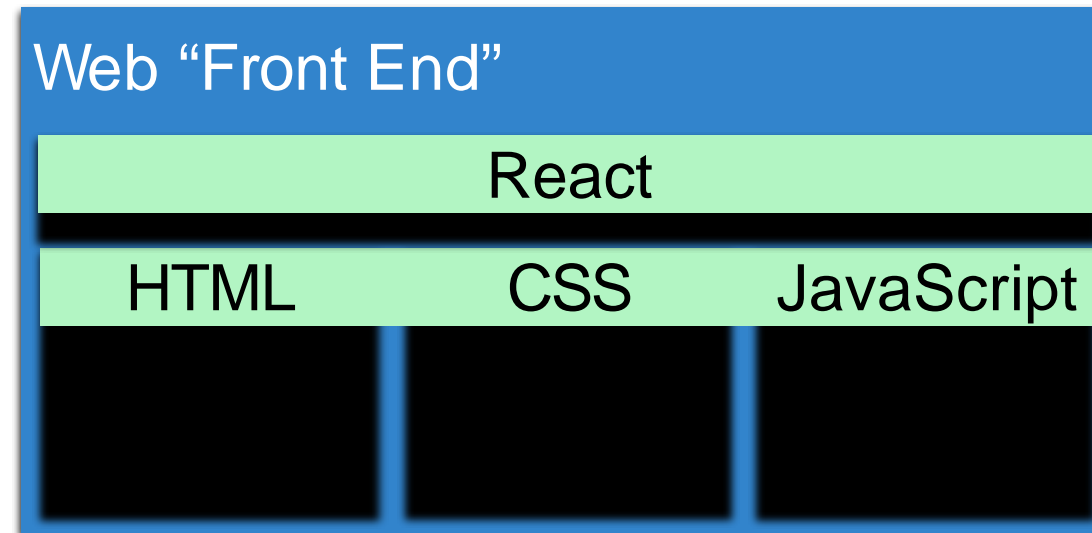
app.use(bodyParser.json());

app.post('/api/updateBalance', (req, res) => {
  const { username, amountToAdd } = req.body;
  if (typeof amountToAdd !== 'number' || amountToAdd <= 0) return res.status(400).json({ error: "Invalid amount" });
  const user = getUserByUsername(username);
  if (!user) return res.status(401).json({ error: "Unauthorized" });
  const updatedBalance = user.balance + amountToAdd;
  firebase.database().ref(`users/${username}/balance`).set(updatedBalance);
  res.status(200).json({ message: "Balance updated successfully" });
});

app.listen(3000, () => console.log("Server is running on port 3000"));
```

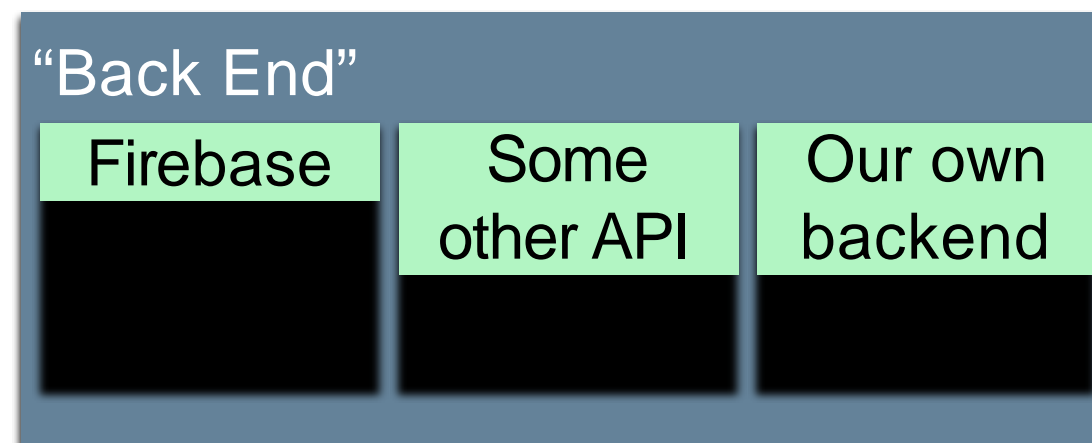
```
// Front-end code (React, for example)
function updateUserBalance(user, amountToAdd) {
  if (typeof amountToAdd !== 'number' || amountToAdd <= 0) return;
  fetch('/api/updateBalance', {
    method: 'POST',
    headers: { 'Content-Type': 'application/json' },
    body: JSON.stringify({ username: user.username, amountToAdd })
  })
  .then(response => {
    if (response.ok) console.log("Balance updated successfully");
    else console.error("Failed to update balance.");
  })
  .catch(error => console.error("Error updating balance:", error));
}
```


Dynamic Web Apps - Firebase



Presentation

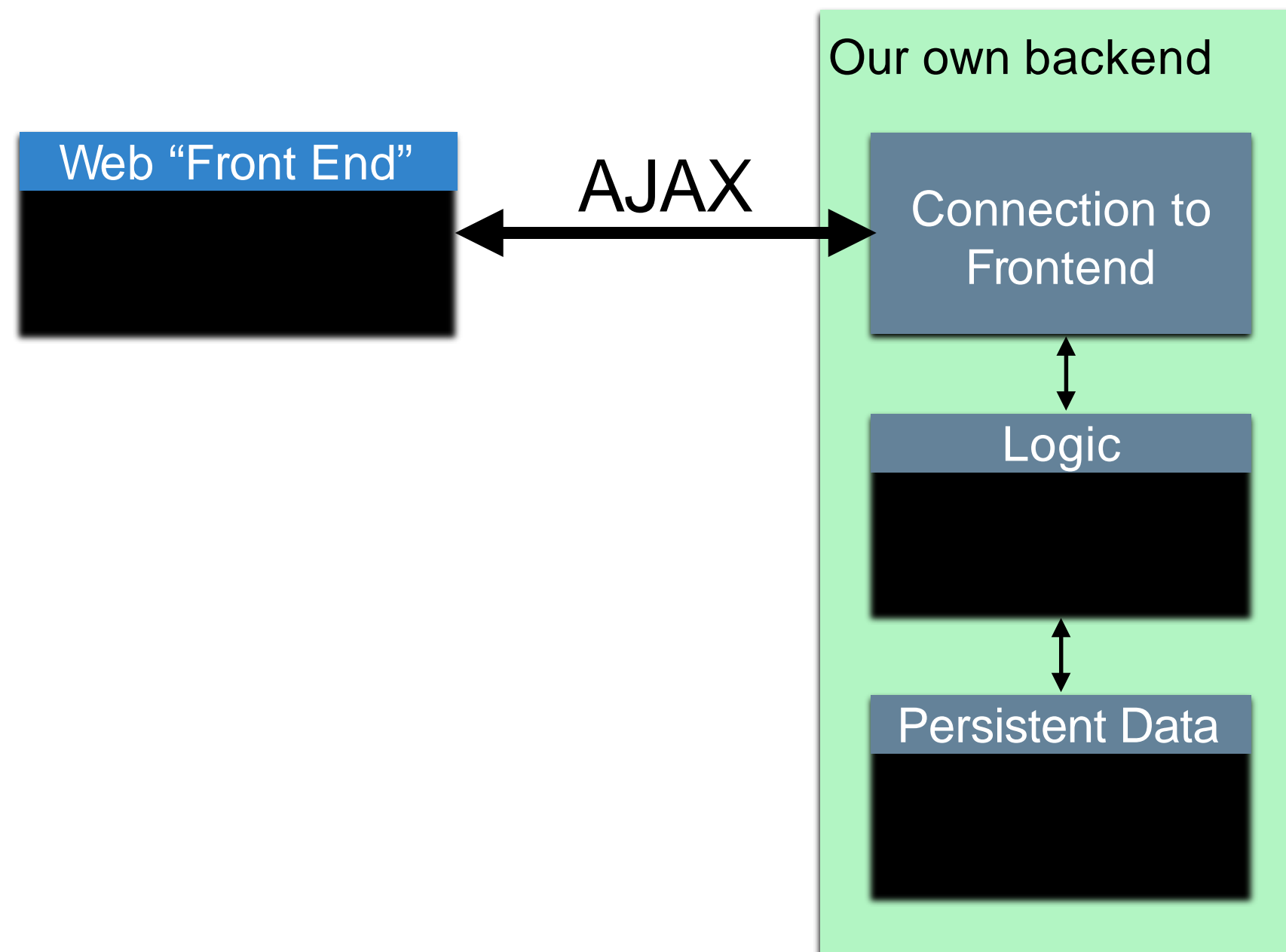
Some logic



Data storage

Some other logic

What does our backend look like?



Evolution of of Backend Development

In the beginning, you wrote whatever you wanted
using whatever language you wanted and whatever
framework you wanted

Then... PHP and ASP

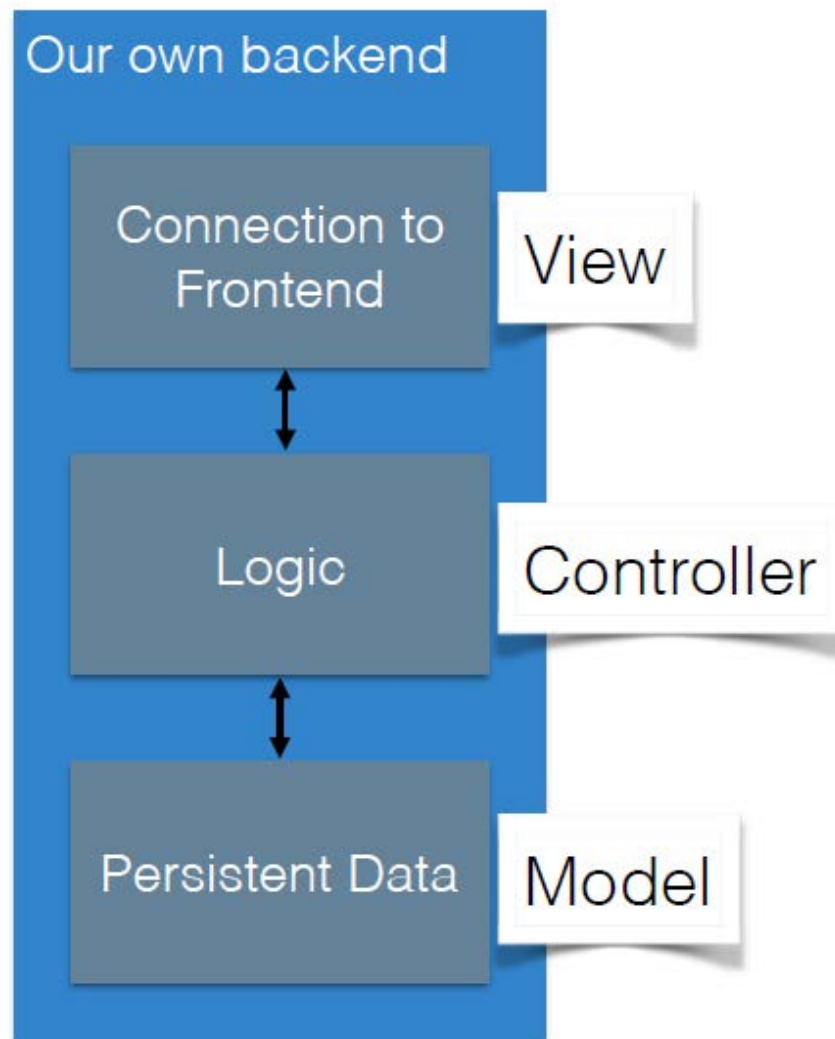
Languages “designed” for writing backends
Encouraged spaghetti code

A lot of the web was built on this

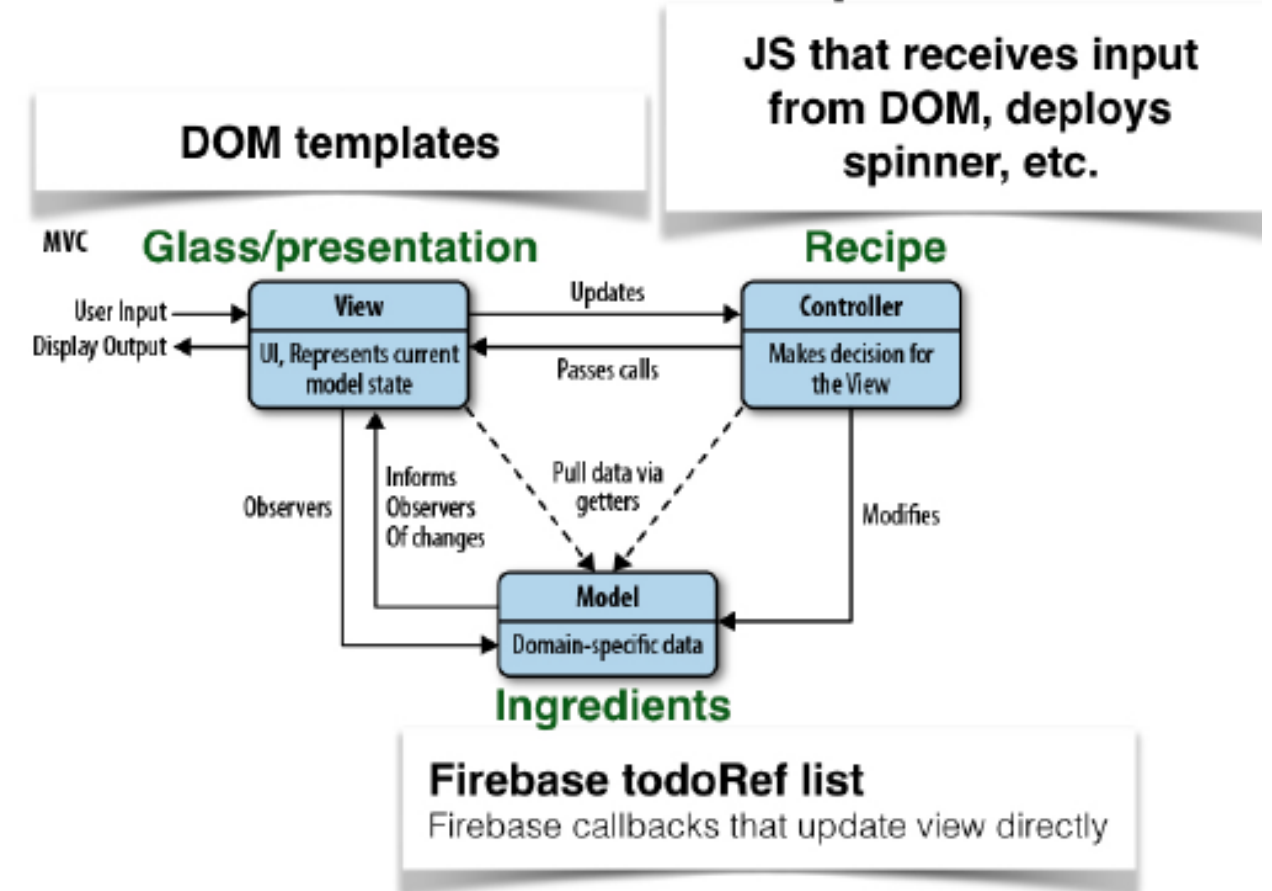
A whole lot of other languages were also springing up
in the 90's

Ruby, Python, JSP

Re-Organising Code

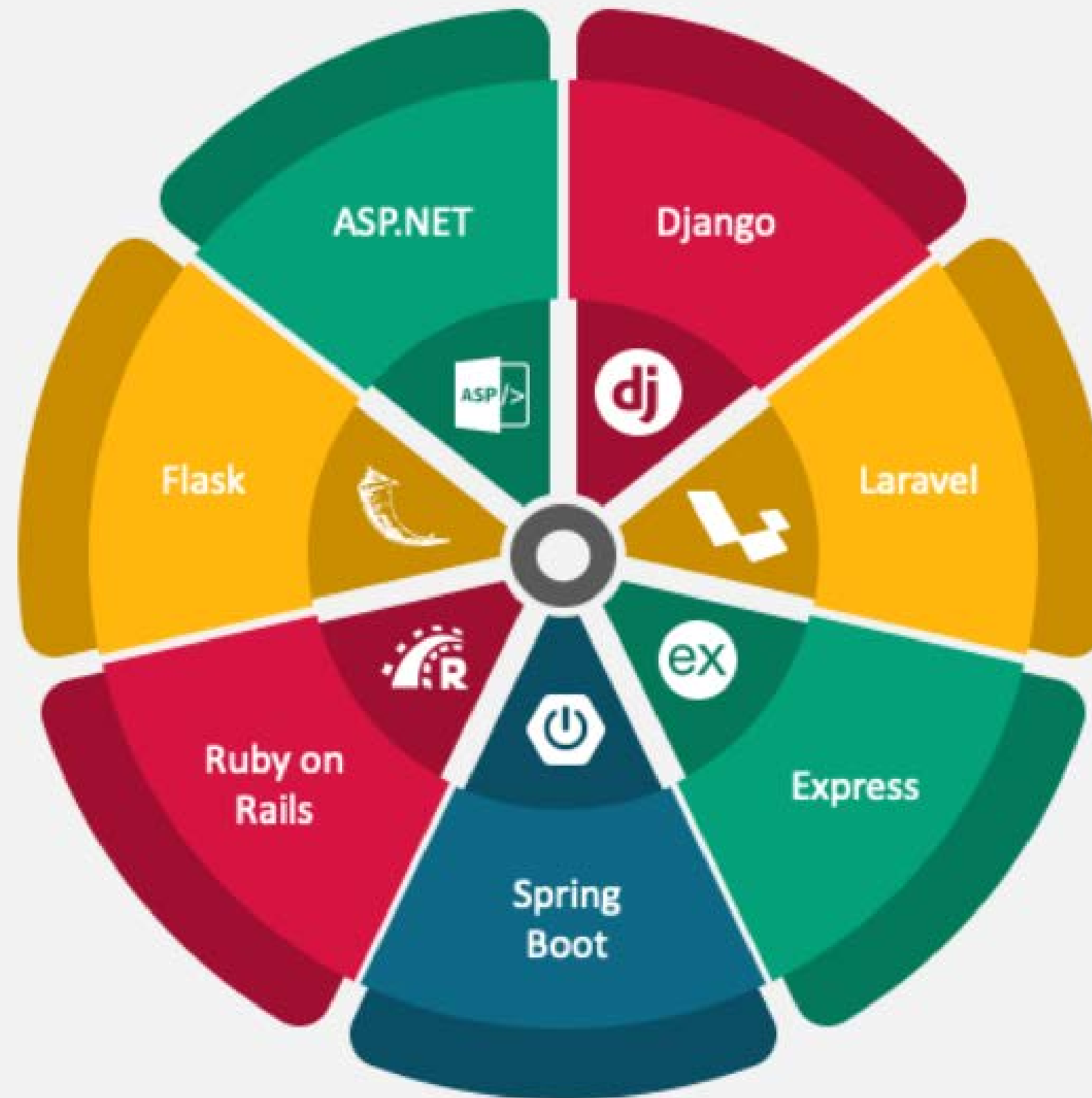


MVC & JavaScript



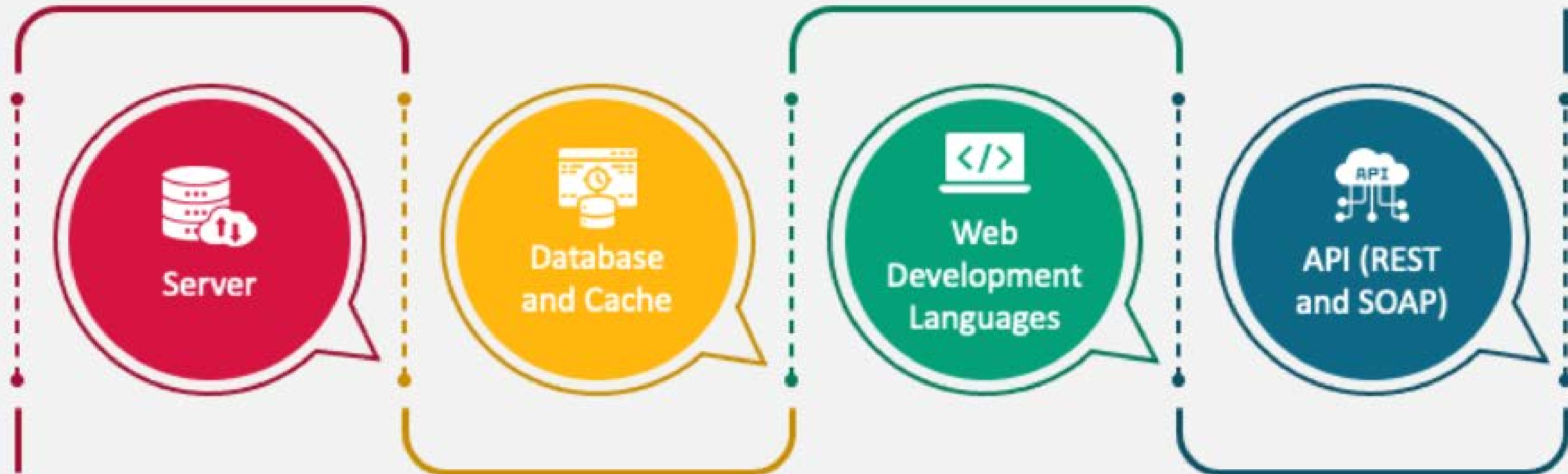
BACKEND DEVELOPMENT

Top 7 Frameworks for Backend Development



BACKEND DEVELOPMENT

Backend Development Skills



Node.JS

Why use Node?

- Easy to get into after learning JS (it's JS)
- Event based: really efficient for sending lots of quick updates to lots of clients

Why not use Node?

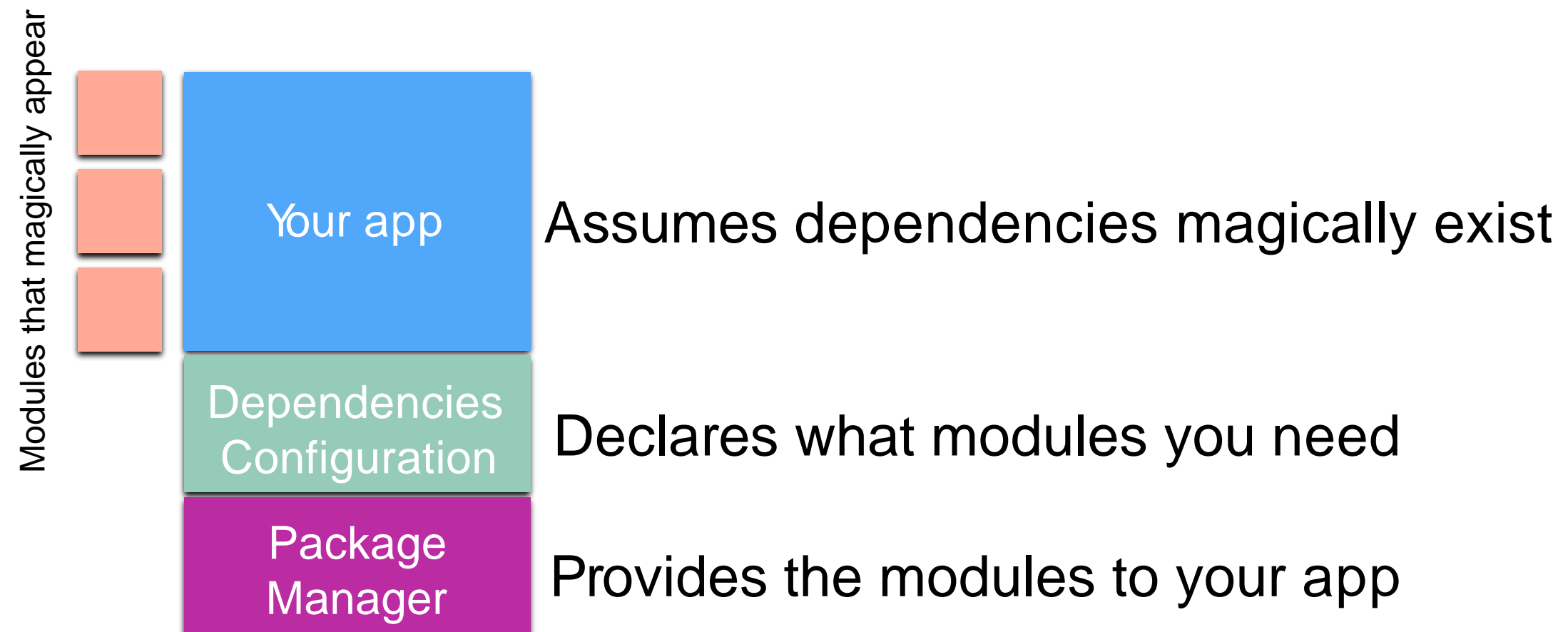
- Bad for CPU heavy stuff
- It's relatively immature

How to use Node.Js

- Node.JS is a *runtime* that lets you run JS outside of a browser
- Node.JS has a very large ecosystem of packages
 - Example: express (web server),
 - Nodemon (automatically restarts your server when it changes)
- Must be downloaded and installed
- <https://nodejs.org/en/>
- Get the latest stable version

A better way for modules

- Describe what your modules are
- Create a central repository of those modules
- Make a utility that can automatically find and include those modules



NPM: Node Package Manager

- Bring order to our modules and dependencies
- Declarative approach:
 - “My app is called helloworld”
 - “It is version 1”
 - You can run it by saying “node index.js”
 - “I need express, the most recent version is fine”
- Config is stored in json - specifically package.json

Generated by npm commands:

```
{
  "name": "helloworld",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "",
  "license": "ISC",
  "dependencies": {
    "express": "^4.14.0"
  }
}
```

Using NPM

- Your “project” is a directory which contains a special file, package.json
- Everything that is going to be in your project goes in this directory
- Step 1: Create NPM project
`npm init`
- Step 2: Declare dependencies
`npm install <packagename> --save`
- Step 3: Use modules in your app
`var myPkg = require("packagename")`
- Do NOT include node_modules in your git repo! Instead, just do
`node install`
 - This will download and install the modules on your machine given the existing config!

Demo Hello World Server

- 1: Make a directory, myapp
- 2: Enter that directory, type **npm init** (accept all defaults)
- 3: Type **npm install express --save**
- 4: Create text file app.js:

```
var express = require('express');
var app = express();
var port = process.env.port || 3000;
app.get('/', function (req, res) {
  res.send('Hello World!');
});

app.listen(port, function () {
  console.log('Example app listening on port' + port);
});
```

- 5: Type **node app.js**
- 6: Point your browser to <http://localhost:3000>

**Creates a configuration file
for your project**

**Tells NPM that you want to use
express, and to save that in your
project config**

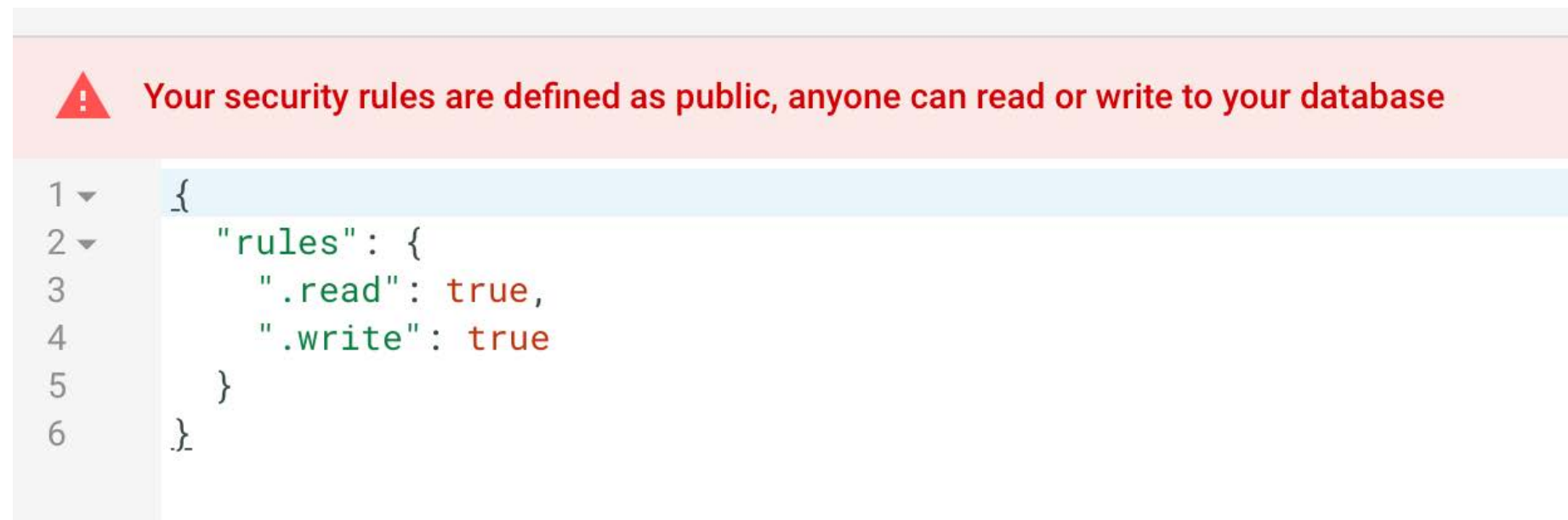
Runs your app

Learn More

- <https://www.w3schools.com/nodejs/>

Moving Firebase into Node

- If you set your database to be writeable by everyone... then make sure **NOBODY** has your private key.



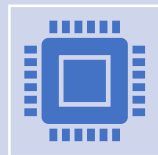
The screenshot shows a warning message from the Firebase console: "Your security rules are defined as public, anyone can read or write to your database". Below the warning, the security rules are displayed in a code editor. The rules are defined as follows:

```
1 {  
2   "rules": {  
3     ".read": true,  
4     ".write": true  
5   }  
6 }
```

Node.js + Firebase



Firebase provides real-time database storage, authentication, cloud functions, and more.



When Node.js is used in conjunction with Firebase, it typically acts as a server or backend for your application.



Node.js and Firebase together provide a powerful combination for building real-time, serverless, and scalable applications.



Firebase handles the back end (database, authentication, and more), while Node.js provides a flexible and extensible server for custom logic and integration with other services.

Interacting with Firebase Services:

Once Firebase Admin is initialized in your Node.js application, you can interact with various Firebase services, such as:

Firebase Realtime Database: You can read and write data to the Firebase Realtime Database. You can also use Firebase's real-time capabilities for synchronized data updates.

Firebase Authentication: You can use Firebase Authentication to manage user sign-ups, logins, and user authentication in your Node.js application.

Firebase Cloud Firestore: If you're using Firestore (Firebase's NoSQL document database), you can perform read and write operations from your Node.js server.

Firebase Cloud Functions: You can deploy Firebase Cloud Functions, which are serverless functions written in Node.js that can respond to Firebase events, like database changes, authentication events, and HTTP requests.

Firebase Storage: You can use Firebase Storage to store and serve user-generated content like images, videos, and files.

Back-end Programming Practices

Part 2

APIs, States and Security

Part 3


```
// Example API endpoint structure
app.get('/api/users', (req, res) => {
  // Get a list of users
});
app.post('/api/users', (req, res) => {
  // Create a new user
});
```

*How do we
create
structured APIs?*

- Structured APIs are essential for maintaining a well-organized, predictable, and maintainable backend.
- Use of RESTful principles to structure APIs with clear endpoints.
- Implement HTTP methods (GET, POST, PUT, DELETE) for CRUD operations.

```
// Generating a JWT token in Node.js
const jwt = require('jsonwebtoken');
const token = jwt.sign({ userId: user.id }, 'secret-key', { expiresIn: '1h' });
```

How do we maintain
state between our
backend and
frontend?

- State management is crucial for preserving data and user sessions.
- Use technologies like JSON Web Tokens (JWT) for authentication and session management.

```
// Using a library like Helmet to enhance security in Express.js
const helmet = require('helmet');
app.use(helmet());
```

Privacy & Security

- Privacy and security considerations are critical for safeguarding user data and protecting your backend.
- Implement input validation, encryption, and user authentication.
- Utilize libraries and modules for secure coding.

```
// Microservices communication with HTTP requests  
const response = await axios.get('http://microservice2/api/resource');
```

Architecting Many Services Together

- Microservices architecture allows multiple services to work together efficiently.
- Each service focuses on a specific task.
- Communication between services using APIs.

```
# Deployment command using Heroku
heroku login
git add .
git commit -m 'Deployment update'
git push heroku master
```

Deploying Our Backend Services

- Deployment ensures that your backend services are accessible and performant in a production environment.
- Use platforms like Google Cloud, AWS, or Heroku for deployment.
- Configure environment variables for different stages (development, production).

Task 4 – Back-end Design in detail

Part 4

Task 4 in detail

-
- Task 4: Back-end Development is a crucial phase of the semester-long application development project.
 - In this task, students are responsible for building the system's back-end, which includes implementing the server, handling user requests, and interacting with the database.
 - This task ensures that the application has a robust and functional server-side component

1. Develop the System's Back-end

-
- Choose a suitable back-end technology stack or framework based on the project requirements and their preferences. Common choices include Node.js, Django, Ruby on Rails, Express.js, Flask, and others.
 - Create the back-end infrastructure, define routes, and establish the server that will handle incoming HTTP requests.

2. Implement the Server

-
- Set up the server, which acts as the core of the application's back end.
 - Define routes and endpoints that correspond to different functionalities of the application.
 - Implement routing logic to handle incoming requests, process data, and send responses.

Handle User Requests:

-
- Implement the necessary business logic to handle user requests effectively.
 - This includes user authentication, validation of incoming data, and executing operations based on user actions (e.g., creating, updating, or deleting data).
 - Ensure that the back end is designed to respond to different types of requests from the front end (e.g., GET, POST, PUT, DELETE) and provide appropriate responses.

4. Interact with the Database:

-
- Establish connections to the chosen database system (e.g., MySQL, PostgreSQL, MongoDB) and create the necessary schemas and models.
 - Implement database operations for storing, retrieving, updating, and deleting data.
 - Ensure that data is stored securely and that database queries are optimized for performance.

Secure and Optimize the Back End:

-
- Security is a critical aspect of back-end development. Students should implement security measures to protect against common threats like SQL injection, Cross-Site Scripting (XSS), and Cross-Site Request Forgery (CSRF).
 - They should also consider implementing user authentication and authorization to ensure that only authorized users can access specific resources.
 - Optimizing the back end involves improving the performance of the server and database queries. This may include caching, indexing, and minimizing resource-intensive operations.

6. Deliverable: Back-end Codebase and Functional Back-end System:

-
- The primary deliverable for this task is the back-end codebase, which should be well-structured, well-documented, and organized.
 - Provide a functional back-end system that can be integrated with the front-end components developed in earlier tasks.
 - The codebase should be hosted on a version control platform like GitLab, and students should provide a link to the repository.

7. Provide a URL Link to Your System:

-
- In addition to the codebase, students should deploy their back-end system to a server or cloud platform.
 - They should provide a URL link to the live system, allowing for interaction with and test the application.