CS 4220

- Current Trends in Web Development -

Application Design and Development with Node.js

Prof. Cydney Auman

AGENDA

01 Command Line Interfaces

02 Web APIs

03 HTTP Responses

04 Integrating with Web APIs

Timeline

Week 07 - 3/08

- Introduction to CLI and APIs
- Midterm Assigned

Week 08 - 3/15

- Adding features to CLIs
- Homework 2 Review

Week 09 - 3/22

- Midterm Due - Sunday 3/26 at 11:59 PM
- Remote Class Wednesday 3/22 - focused on Q&A for Midterm

Week 10

- Spring Break (https://www.calstatela.edu/academicresources/academic-calendar)

Command Line Interfaces

Command Line Interfaces (**CLIs**) or Command Line Applications, are programs designed to be used from the terminal/command-prompt. Command Line Applications usually accept inputs as arguments. Additionally, they can accept options, often referred to as flags.

Examples:

- npm - (CLI for managing JS packages)
- git -(CLI for Github)
- aws - (CLI for Amazon Web Services)
- webpack - (CLI for bundling JS files)

```
$ git --help
usage: git [--version] [--help] [-C <path>] [-c <name>=<value>]
           [--exec-path[=<path>]] [--html-path] [--man-path] [--info-path]
           [-p | --paginate | -P | --no-pager] [--no-replace-objects] [--bare]
           [--git-dir=<path>] [--work-tree=<path>] [--namespace=<name>]
           <command> [<args>]

These are common Git commands used in various situations:

start a working area (see also: git help tutorial)
   clone      Clone a repository into a new directory
   init       Create an empty Git repository or reinitialize an existing one

work on the current change (see also: git help everyday)
   add        Add file contents to the index
   mv         Move or rename a file, a directory, or a symlink
   reset      Reset current HEAD to the specified state
   rm         Remove files from the working tree and from the index

examine the history and state (see also: git help revisions)
   bisect     Use binary search to find the commit that introduced a bug
   grep       Print lines matching a pattern
   log        Show commit logs
   show       Show various types of objects
   status     Show the working tree status

grow, mark and tweak your common history
   branch     List, create, or delete branches
   checkout    Switch branches or restore working tree files
   commit     Record changes to the repository
   diff       Show changes between commits, commit and working tree, etc
   merge      Join two or more development histories together
   rebase     Reapply commits on top of another base tip
   tag        Create, list, delete or verify a tag object signed with GPG

collaborate (see also: git help workflows)
   fetch      Download objects and refs from another repository
   pull       Fetch from and integrate with another repository or a local branch
   push       Update remote refs along with associated objects
```

Building a CLI

Typically these Command Line Interfaces (**CLIs**) or Command Line Applications are designed and developed to be very efficient and focus on a single purpose.

Command Line Applications are not intended to be monoliths - meaning an application that does everything . Essentially, CLIs are like a microservice. They should be designed as simple, concise applications that can perform designated tasks proficiently.

CLIs can be designed to interact directly with the information on your computer. They can also be designed to reach out to a Database. And the most common form of CLIs are ones that reach out and connect to APIs on the internet.

Why Build a CLI?

There are many reasons why people decide to build CLIs.

Developer Productivity

CLIs can aid in developer productivity. Often times as Developers we perform the same routine tasks, by building a CLI we are able to automate this process.

Integrating Processes

Often times two separate processes need to be integrated. Building a CLI can streamline this concept. For example, we might have a deployment process and then we might want to post the error or success into a team chat application like Slack. This is something that could be built using a CLI.

Interacting with APIs

APIs offer up alot of data. By building a CLI we can create a user friendly interface between the user and the data that is return from an API.

Running a CLI

In Node.js the typical process to run a CLI application is the following.

Use **node** followed by the **filename**. Typically, the first String after the filename is often referred to the as the **command**. The command can optionally accept arguments.

Often times CLIs will also take **options** or **flags**. Flags are specified using **--** for full words or a simplified form of **-** plus a letter.

```
node cli.js run backend_tests -e dev --logging true
```

```
node                                    filename
command                                 command argument (optional)
short form flag (using single -)        flag argument (optional)
long form flag (using double --)        flag argument (optional)
```

Accepting args from a CLI

The built-in way you retrieve arguments is by using the **process object** that Node.js provides.

It has an argv property, which is an array that contains all the command line invocation arguments.

- The first element is the full path of the node command.
- The second element is the full path of the file being executed.
- All the additional arguments and flags are present from the third position going forward.

```
// this returns the array described above
const argv = process.argv

// remove the first and second elements to access the arguments/flags
const args = argv.slice(2);
```

yargs

yargs is a Node Module that can be found on NPM
https://www.npmjs.com/package/yargs

Yargs helps you build interactive command line tools, by **parsing arguments and options** while generating an clean user interface with features.

Yargs also has a robust API that allows us to create CLIs tools with a variety of options and flags which allows for clean interface and controlling the flow of our code. Yargs also provides built in validation and type coercion

yargs

Define the commands in our application using

`.command(String, String, Builder Function, Handler Function)`

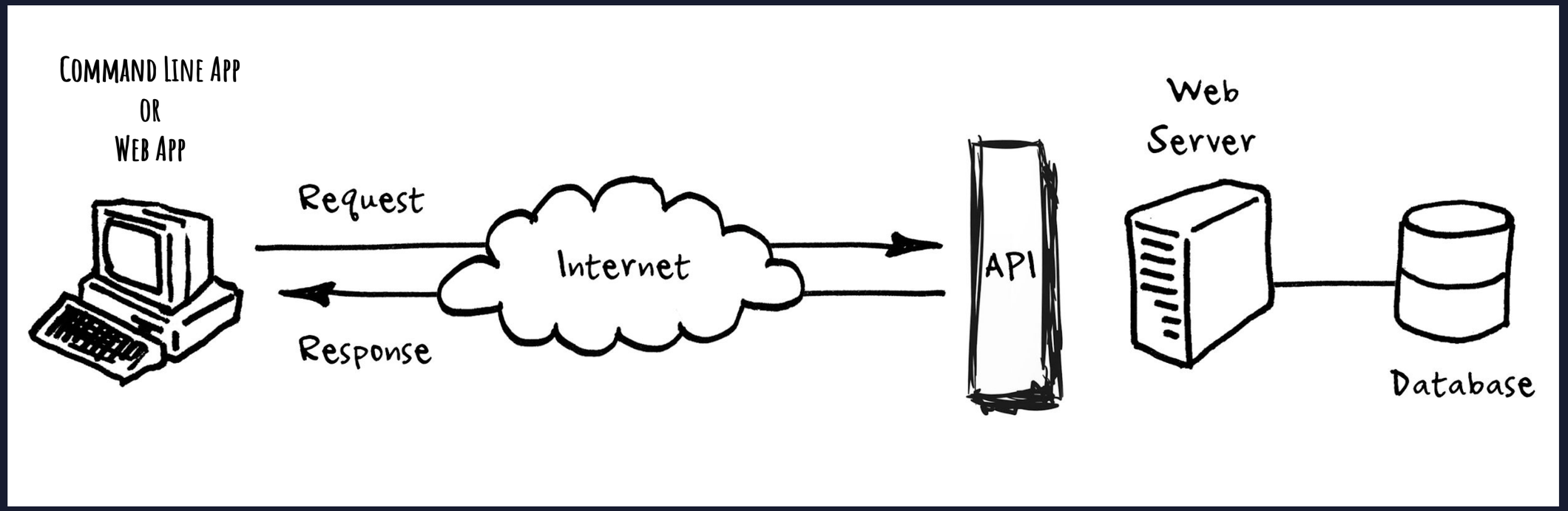
The first string is the command name. The second string is the description for the command.

Builder Function
This function receives the yargs instance as an argument and is used to provide command-specific configurations.

Handler Function
This function receives the parsed arguments and is used to handle how our application should be executed based on the arguments provided.

What is an Web API?

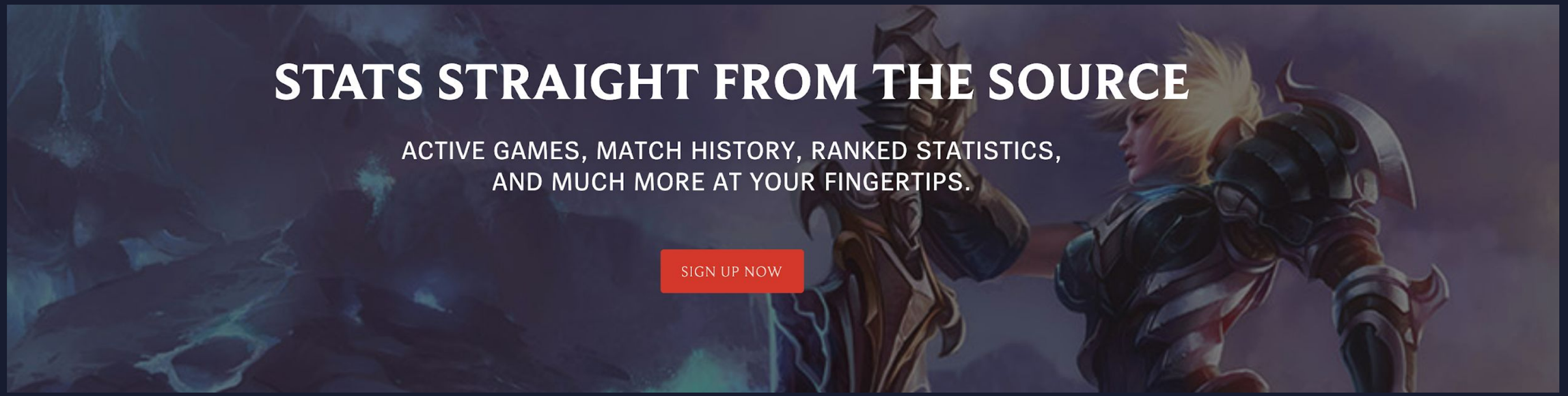
Web APIs (Application Programming Interface) are a large part of web applications. It is an intermediary software that allows applications to communicate with each other. Essentially, an API is the messenger that delivers your request to the provider of some data and then delivers the response back to you.



Why are Web APIs Important?

APIs orchestrate access to an application in order to allow us to search and retrieve some portion of a companies data set.

Large tech companies, including social media companies make some of their data available to the public. APIs are also built by government organizations, startups, fan sites, gaming and eSports companies and much more.



Communicating with APIs

Communication with APIs centers around of the HTTP Request-Response Cycle.

An application sends the API a **request** to do something. The API delivers the request to the data provider and after it is processed the API sends the application a **response** with the requested data or an error if the request could not be processed.

The key features of requests/responses:

- Endpoint (aka the URL)
- Methods (GET, POST, PUT and DELETE)
- Status Codes (indicates type of completion for requests)
- Headers (Content Types, Authentication)
- Body (aka where the data is stored)

HTTP Path and Query Parameters

Path Parameters

Path Parameters are used to specify a particular resource and/or resources often times by using a unique id. They are always part of the full URL path.

Example:

`https://deckofcardsapi.com/api/deck/:deck_id/draw`

- where `:deck_id` is a variable and is replaced with the actual id

Query Parameters

Query Parameters are typically used to search, sort, limit, or apply a flag and etc when making requests. They are always at the end of the URL to the right of the ? and each query parameter is set equal to its value. Multiple query parameters are separated by an &.

Example:

`https://deckofcardsapi.com/api/deck/new/?deck_count=1&jokers_enabled=true`

API Methods

There are four basic **HTTP methods** used in making requests to interact with Servers.

- **GET** — Retrieving data. This can be used to retrieve a list of data with optional filtering. Or by using an id to get a specific piece of data.
- **POST** — Creating a new data entry by sending the details in the POST body.
- **PUT** — Updating a specific data entry by using an id and sending the details in the PUT body.
- **DELETE** — Removing a specific data entry by using an id. Typically the id is passed in the url parameters.

API Body

Web APIs typically respond with **JSON** data. **JSON** stands for JavaScript Object Notation and is a standard text-based format for representing structured data based on JavaScript object syntax.

This format is easy to read. Data is stored as key/value pairs. **JSON** differs from a JavaScript object literal as the *key* will always be a string whereas the values can be of various types - string, number, boolean, array, and even nested JSON.

```
// JSON
{
    "success": true,
    "deck_id": "3p40paa87x90",
    "shuffled": true,
    "remaining": 52
}
```

Integrating with APIs

Steps to Integrating with an API

- Find a public API that offers the type of information you are interested in accessing. Ensure that the API is well documented and maintained.
- Read over the documentation to make sure the API provides the data you need.
- Review the Endpoints, Methods and Response Body. Some APIs will require you to register to gain access to their data. Some will require Authentication Headers and your app will need to properly Authenticate with the API before data can be accessed.

Resources

Mozilla Developer Docs

- https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods
- https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/JSON

Command Line Interface

- https://yargs.js.org/docs/

APIs Simply Explained

- https://www.youtube.com/watch?v=OVvTv9Hy91Q

APIs

- https://github.com/public-apis/public-apis
- https://deckofcardsapi.com/

Review and Prep

Review

- Review Slides
- Review and Run Code Examples

Prep

Review Midterm on Canvas

- Opens Wednesday 3/08 at 10pm
- Due Sunday 3/26 at 11:59pm

API + Team Selection

- Pick a team of 2 - 3 for Midterm
- Pick a team captain
- Select an API
- Fill out the Google Form

```
console.log('Week 07');
console.log('Code Examples');
```