44-563: Unit 09

Developing Web Applications and Services

Includes

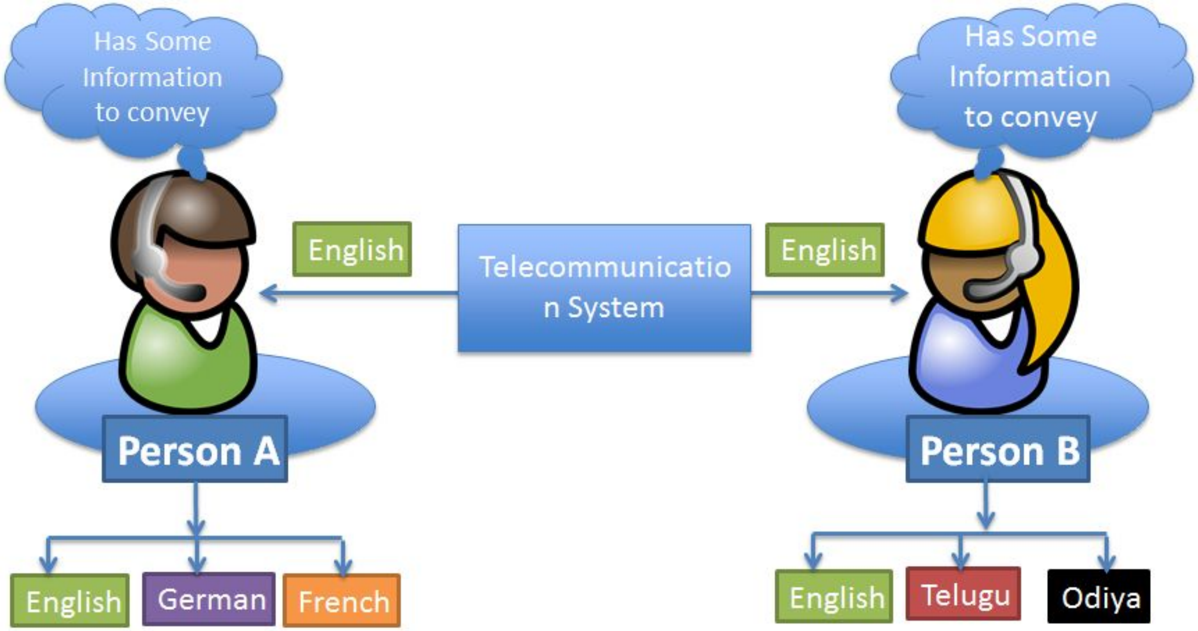
- Working with Data
- Data Serialization formats
- XML
- Postman
- JSON
- **Wed: React by Nick Larson 4PM**
- Fri: Project introduction

Common Problem

Web apps & services need to **exchange data** over the internet.

Communicating programs may:

- Be written in different **languages**
- Run on different **platforms**.



Common Solution

Represent data in a platform-independent way

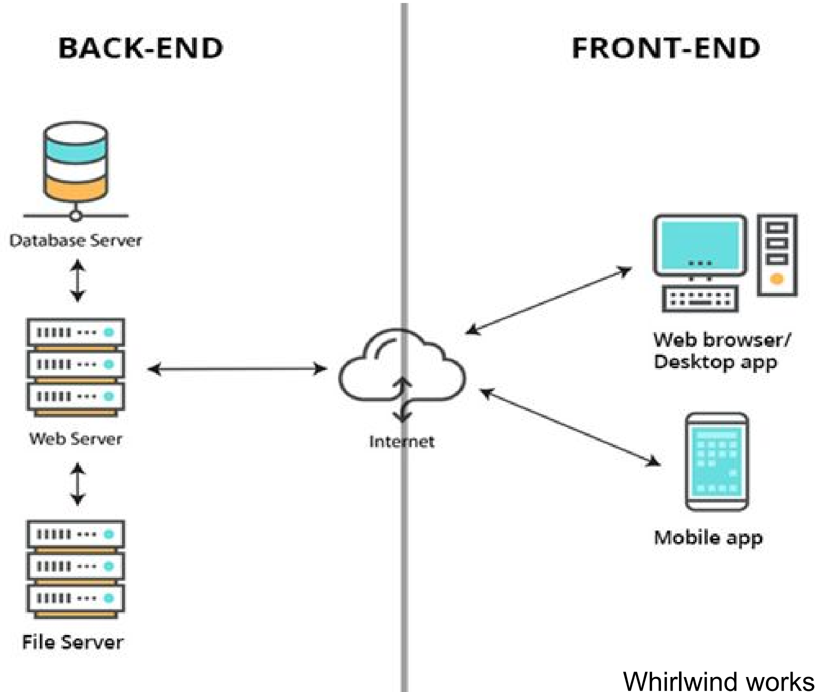
Serialization Formats

Several ways of exchanging data:

- **XML**
- **JSON**
- YAML
- *Apache Thrift*
- Many more...

Enable data exchange across languages and platforms

We will focus on XML and JSON



Serialization formats

- **XML** is a **markup language**, tree-based
- **JSON** is a data **format** for simple data
- **YAML** is a data **serialization format** that can handle cyclic data references the other two can't. Often used with Swagger … as it should be. ☻
- Apache **Thrift** is a **protocol** for cross-platform services (used in **big data**).

Many options

Very different things

Which is better?

Which is better?

Very different things

It depends on your application

Let's take a quick look at these 4

XML Example

```xml
<?xml version="1.0"?>
<book id="123">
  <title>Object Thinking</title>
  <author>David West</author>
  <published>
    <by>Microsoft Press</by>
    <year>2004</year>
  </published>
</book>
```

160 characters

http://www.yegor256.com/2015/11/16/json-vs-xml.html

JSON Example

```
{
   "book": {
      "id": 123,
      "title": "Object Thinking",
      "author": "David West",
      "published": {
         "by": "Microsoft Press",
         "year": 2004
      }
   }
}
```

100 characters

http://www.yegor256.com/2015/11/16/json-vs-xml.html

YAML Example

```yaml
---
id: 123
title: Object Thinking
author: David West
published:
  by: Microsoft Press
  year: 2004
```

80 characters; whitespace matters (indenting must be exact)

http://www.yegor256.com/2015/11/16/json-vs-xml.html

Thrift Example (partial)

```
namespace java edu.nwmissouri.csis

struct Book{
 1: i64 id
 2: string title
}

service BookService {
 User createBook(1: string title)
}
```

If you like "big data", you may see Apache Thrift

https://github.com/twitter/scrooge/blob/develop/demos/scrooge-maven-demo/src/main/thrift/user.thrift

XML

XML

```
<?xml version="1.0"?>
<book id="123">
  <title>Object Thinking</title>
  <author>David West</author>
  <published>
    <by>Microsoft Press</by>
    <year>2004</year>
  </published>
</book>
```

- Extensible Markup Language
- Powerful **language**
- Looks like HTML markup language, uses angle brackets & closing tags.
- HTML defines **vocabulary** (head, body, p, footer, etc.)
- XML is **syntax** - can represent any data

XML

```xml
<?xml version="1.0"?>
<book id="123">
  <title>Object Thinking</title>
  <author>David West</author>
  <published>
    <by>Microsoft Press</by>
    <year>2004</year>
  </published>
</book>
```

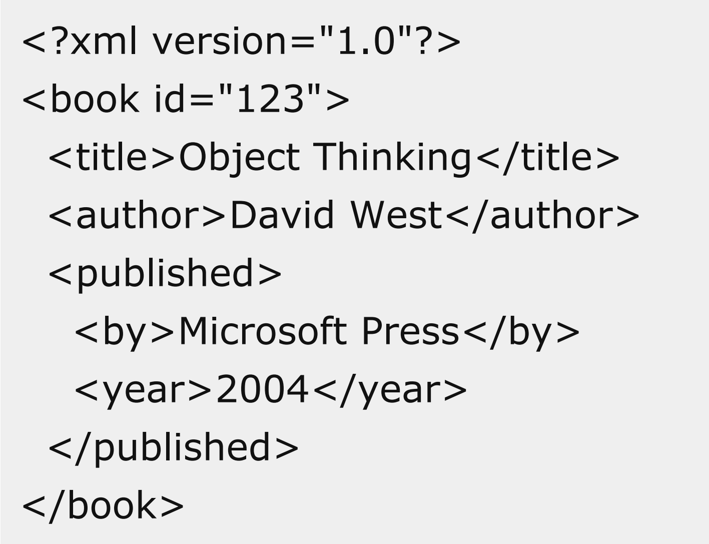
Advantages

- Plain text
- Easily parsed
- Hierarchical (tree-based)
- Represents data but nothing about display.
- Can be transformed via XSL.
- Powerful and lots of tools to support it.

Disadvantages

- Verbose

XML



```
<?xml version="1.0"?>
<book id="123">
  <title>Object Thinking</title>
  <author>David West</author>
  <published>
    <by>Microsoft Press</by>
    <year>2004</year>
  </published>
</book>
```

- Add **XML comments** with
  - <! COMMENT>
- A **self-closing** tag is **empty** (i.e., there is no inner html)
  - <dog age="1"/>

XML Prolog

```
<?xml version="1.0" encoding="UTF-16"?>

<book id="123">
  <title>Object Thinking</title>
  <author>David West</author>
  <published>
    <by>Microsoft Press</by>
    <year>2004</year>
  </published>
</book>
```

Prolog is optional.

If exists, must be **first**.

XML docs can contain international characters.

To avoid errors, specify the encoding used.

UTF-8 is default character encoding for XML (and is not needed).

XML Documents

XML documents consist of **text content marked up with tags**.

Unlike HTML, there are **no predefined tags.**

Markup tags are defined to represent data in particular application domains.

```xml
<?xml version="1.0"?>

<book id="123">
  <title>Object Thinking</title>
  <author>David West</author>
  <published>
    <by>Microsoft Press</by>
    <year>2004</year>
  </published>
</book>
```

Is the highlighted portion **tags** or **text**?

XML Root Element

One root element contains all other elements.

The document is a tree.

- Enclosing element is the single **parent**.
- Subelements are **children**.
- **Root** has no parent.

```
<?xml version="1.0"?>

<book id="123">
  <title>Object Thinking</title>
  <author>David West</author>
  <published>
    <by>Microsoft Press</by>
    <year>2004</year>
  </published>
</book>
```

Book is the root element

XML Elements

Elements describe data.

An **element** consists of:

- A start tag
- An end tag
- Content
    - Everything between these tags
    - Could be data or more elements

```
<?xml version="1.0"?>

<book id="123">
  <title>Object Thinking</title>
  <author>David West</author>
  <published>
    <by>Microsoft Press</by>
    <year>2004</year>
  </published>
</book>
```

Where is the book content?

XML is Case Sensitive

XML is case-sensitive.

<book> is not the same as <Book>

```
<?xml version="1.0"?>
<book id="123">
  <title>Object Thinking</title>
  <author>David West</author>
  <published>
    <by>Microsoft Press</by>
    <year>2004</year>
  </published>
</book>
```

Another XML Example

```xml
<?xml version="1.0" encoding="UTF-16"?>
<dogList>
   <dog>
      <dogName>Fido</dogName>
      <dogAge>10</dogAge>
   </dog>
   <dog>
      <dogName>Fudge</dogName>
      <dogAge>12</dogAge>
   </dog>
</dogList>
```

root element: **dogList**

The content of each **dog** is two subelements: **dogName** and **dogAge**

The content of **dogAge** is the data value **12**

Element attributes

XML **elements** can have **attributes**, just like HTML.

You choose how to store information.

```
<dog>
      <name>Fido</name>
      <age>10<age>
</dog>

----- OR ---------


<dog name="Fido" age="10" />
```

Well-Formed

```
<?xml version="1.0"?>
<book id="123">
  <title>Object Thinking</title>
  <author>David West</author>
  <published>
    <by>Microsoft Press</by>
    <year>2004</year>
  </published>
</book>
```

1. Starts with a **prolog**
2. Every opening tag has a **closing** tag.
3. All tags are completely **nested**.

An XML file is **valid** if:
well-formed, links to XML schema, and valid according to the schema.

XML in specific domains

MathML - XML for math

SVG - XML for Scalable Vector Graphics

```
<math
xmlns="http://www.w3.org/1998/Math/MathML">
   <mrow>
   <apply>
      <minus/>
      <ci>a</ci>
      <ci>b</ci>
   </apply>
   </mrow>
</math>

<svg xmlns="http://www.w3.org/2000/svg">
  <!-- more tags here -->
</svg>
```

Specify the **namespace** (much like Java or C#)

DTD

Document Type Definition

DTD is a document/content model.

It specifies the elements, attributes, and structure of the XML document.

Document models can enforce rules regarding structure (provide validation)

```
<?xml version="1.0" ?>
<!DOCTYPE dogs SYSTEM
    "dogs.dtd" >
<dogs>
<title>My Dogs</title>
 <dogList>


  <dog name="Fudge" age="12" />
  <dog name="Audrey" />


 </dogList>
</dogs>
```

XML document with DTD

```
<!ELEMENT dogs (title, dogList)>
<!ELEMENT title (#PCDATA)>
<!ELEMENT dogList (dog+)>
<!ELEMENT dog EMPTY>
   <!ATTLIST dog
       name ID #REQUIRED
       age CDATA #IMPLIED>
```

dog is an empty element - there is no inner content

Associated DTD

```
<?xml version="1.0" ?>
<!DOCTYPE dogs SYSTEM
    "dogs.dtd" >
<dogs>
<title>My Dogs</title>
 <dogList>


  <dog name="Fudge" age="12" />
  <dog name="Audrey" />


 </dogList>
</dogs>
```

XML document with DTD

```
<!ELEMENT dogs (title, dogList)>
<!ELEMENT title (#PCDATA)>
<!ELEMENT dogList (dog+)>
<!ELEMENT dog EMPTY>
   <!ATTLIST dog
      name ID #REQUIRED
      age CDATA #IMPLIED>
```

Dog is an empty element - there is no inner content

Associated DTD

A Few Factoids about DTDs

- Terms in ( ) are child elements
- title and dogList can only occur exactly once inside dogs, since they lack a + or *
- + ⇒ dog can appear at least 1 time
- * ⇒ dog can appear at least 0 times
- #PCDATA = Parsed Character Data, the text between tags, that will be parsed (so it could contain other tags, and they will be interpreted) #CDATA = Character Data: the text between tags will not be parsed.
- IMPLIED really means *optional*
- See w3schools.com for more

DTD Limitations

Not written in full XML (XML comment notation)

No namespace support

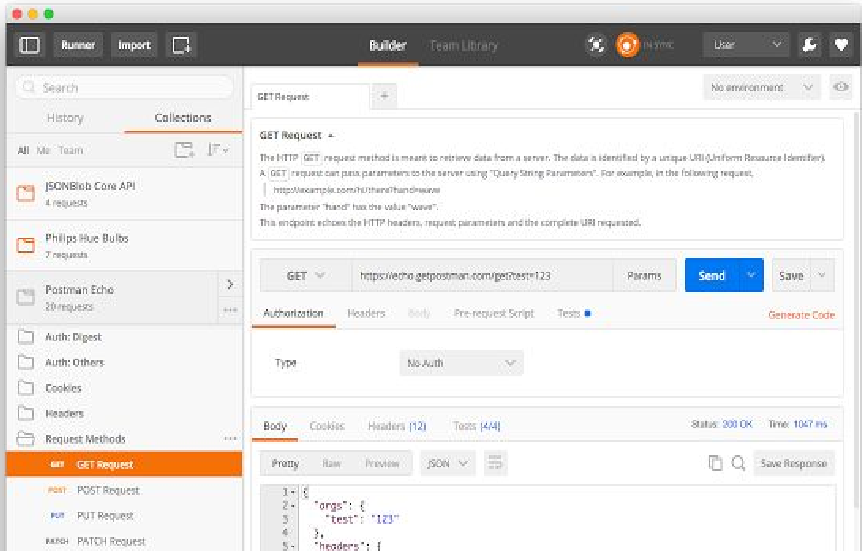
No data types

Not object-oriented

M09 - Postman

API Developer Tool

- Postman
- Developer tool for Chrome
- HTTP request builder.
- Allows us to:
  - Create & send any HTTP request.
  - Explore response.
  - Test APIs.



Build APIs faster

GET Request

The HTTP GET request method is meant to retrieve data from a server. The data is identified by a unique URI (Uniform Resource Identifier).
A GET request can pass parameters to the server using "Query String Parameters". For example, in the following request,

http://example.com/hi/there?hand=wave

The parameter "hand" has the value "wave".
This endpoint echoes the HTTP headers, request parameters and the complete URI requested.

GET https://echo.getpostman.com/get?test=123 Params Send Save

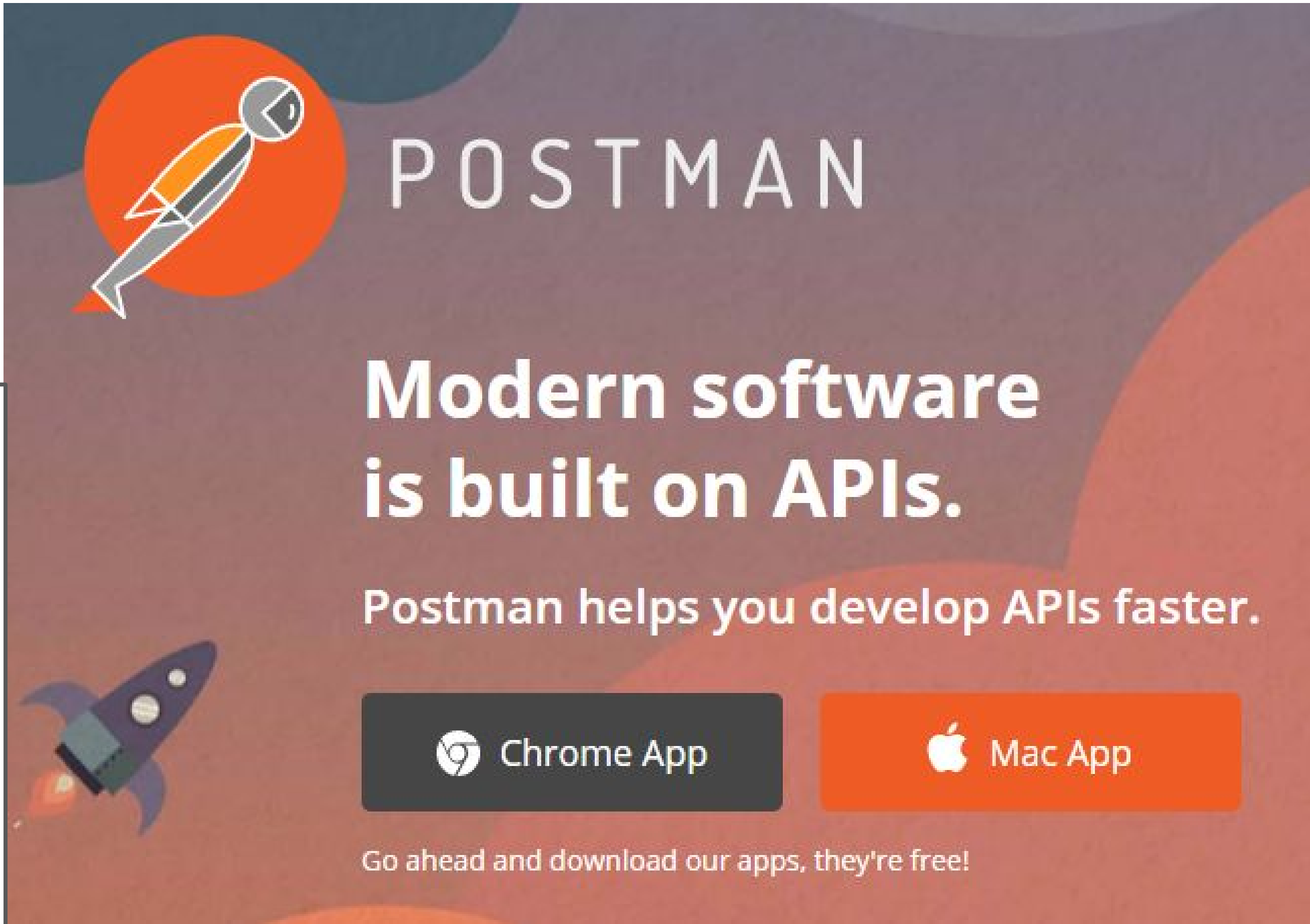
Type: No Auth

Status: 200 OK Time: 1047 ms

```
{
  "args": {
    "test": "123"
  },
  "headers": {
```

Install Postman

1. https://www.getpostman.com/
2. Choose your OS
3. Install
4. Sign up
5. Sign in

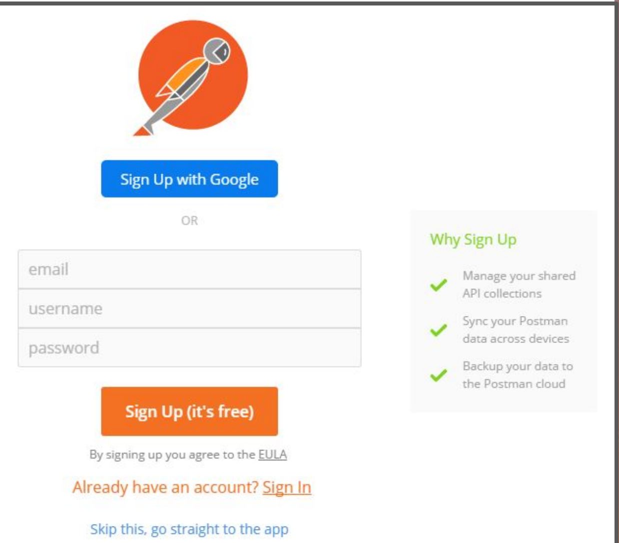


POSTMAN

Modern software is built on APIs.

Postman helps you develop APIs faster.

Go ahead and download our apps, they're free!



Why Sign Up

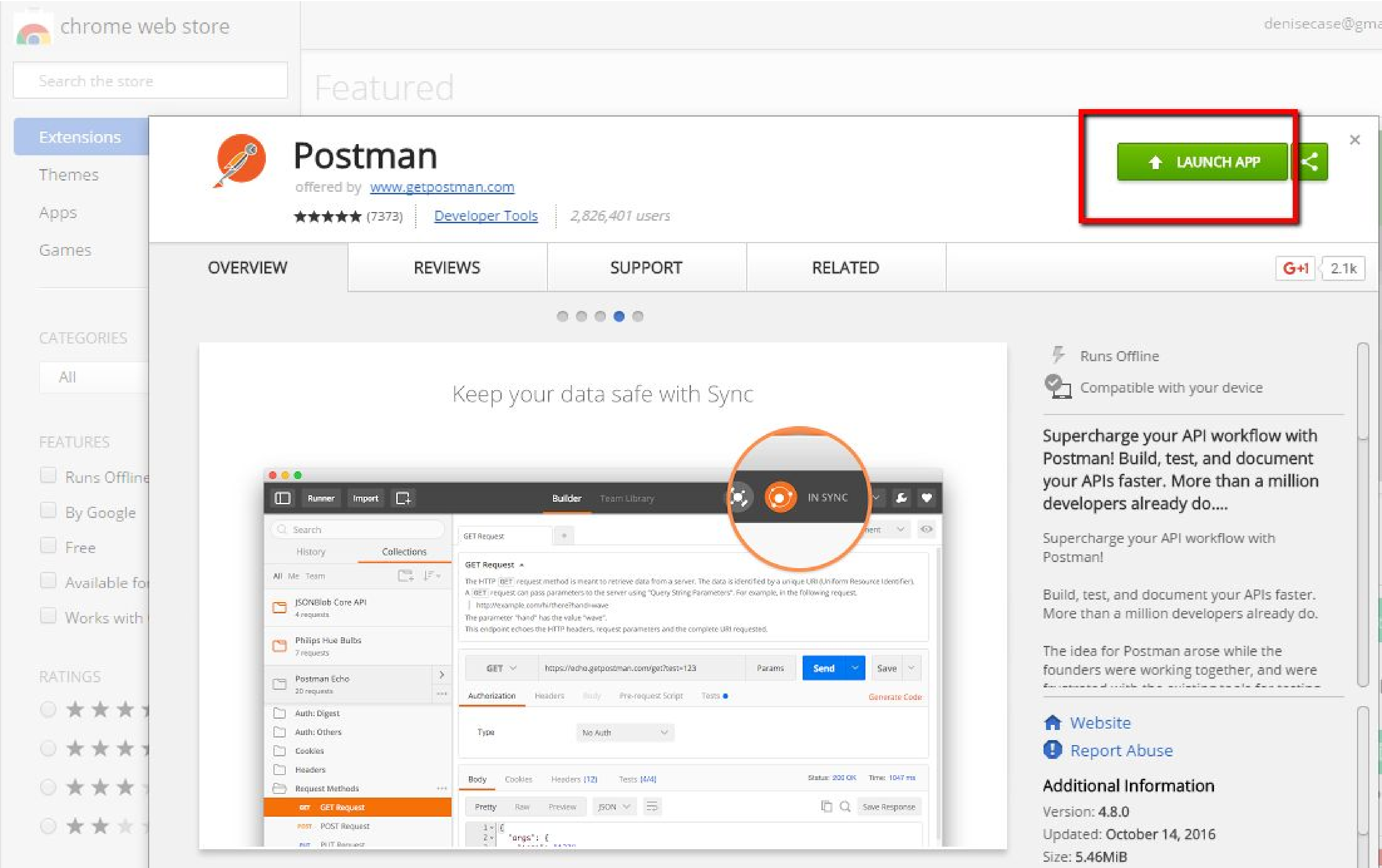
- ✓ Manage your shared API collections
- ✓ Sync your Postman data across devices
- ✓ Backup your data to the Postman cloud

By signing up you agree to the EULA

Already have an account? Sign In

Skip this, go straight to the app

Launch Postman



Postman

offered by www.getpostman.com

★★★★★ (7373) | Developer Tools | 2,826,401 users

GET Request

The HTTP GET request method is meant to retrieve data from a server. The data is identified by a unique URI (Uniform Resource Identifier).

A GET request can pass parameters to the server using "Query String Parameters". For example, in the following request,

http://example.com/hi/there?hand=wave

The parameter "hand" has the value "wave".

This endpoint echoes the HTTP headers, request parameters and the complete URI requested.

https://echo.getpostman.com/get?test=123

```
{
  "args": {
```

Supercharge your API workflow with Postman! Build, test, and document your APIs faster. More than a million developers already do....

Supercharge your API workflow with Postman!

Build, test, and document your APIs faster. More than a million developers already do.

The idea for Postman arose while the founders were working together, and were

Additional Information

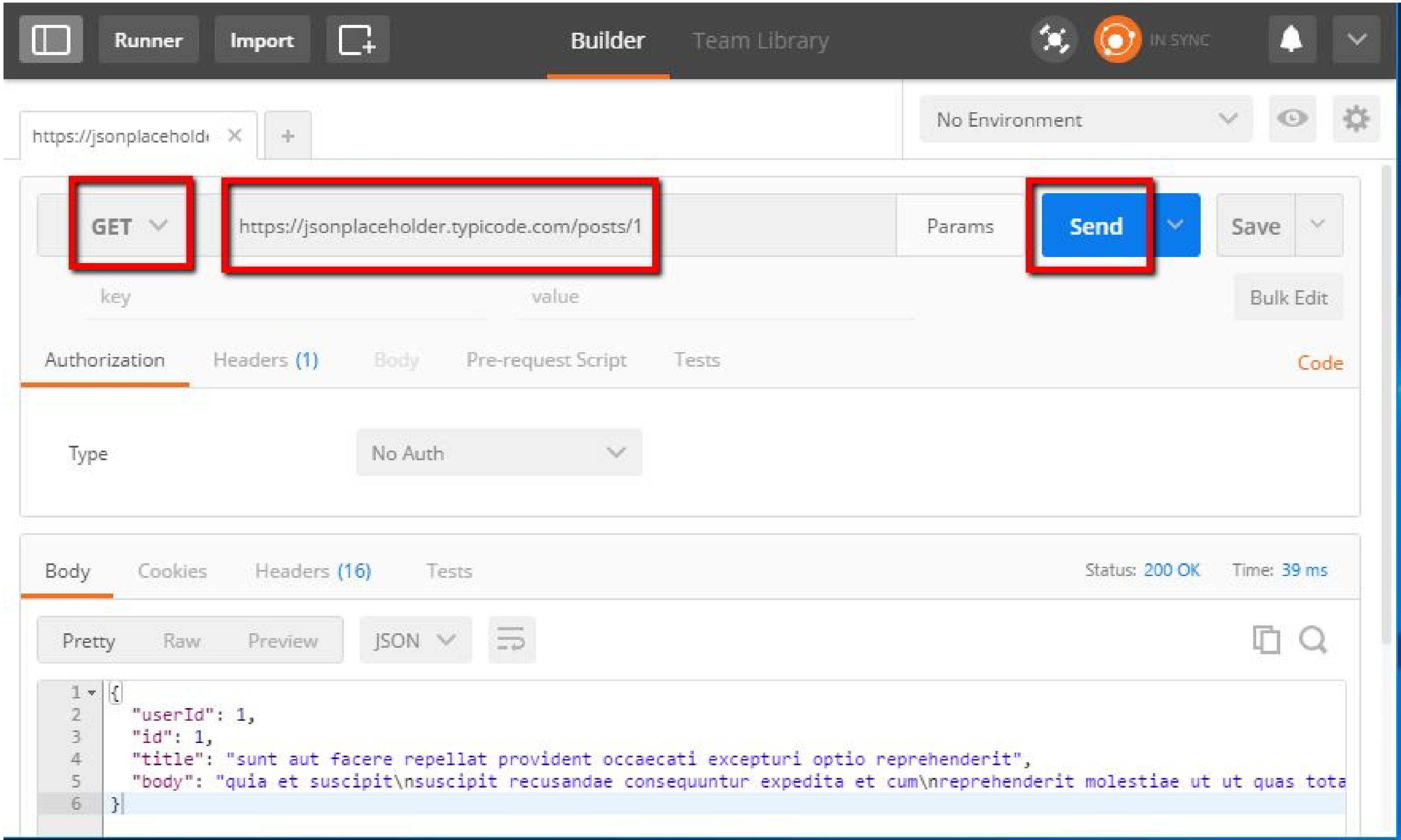
Version: 4.8.0

Updated: October 14, 2016

Size: 5.46MiB

Test GET

https://jsonplaceholder.typicode.com/posts/1



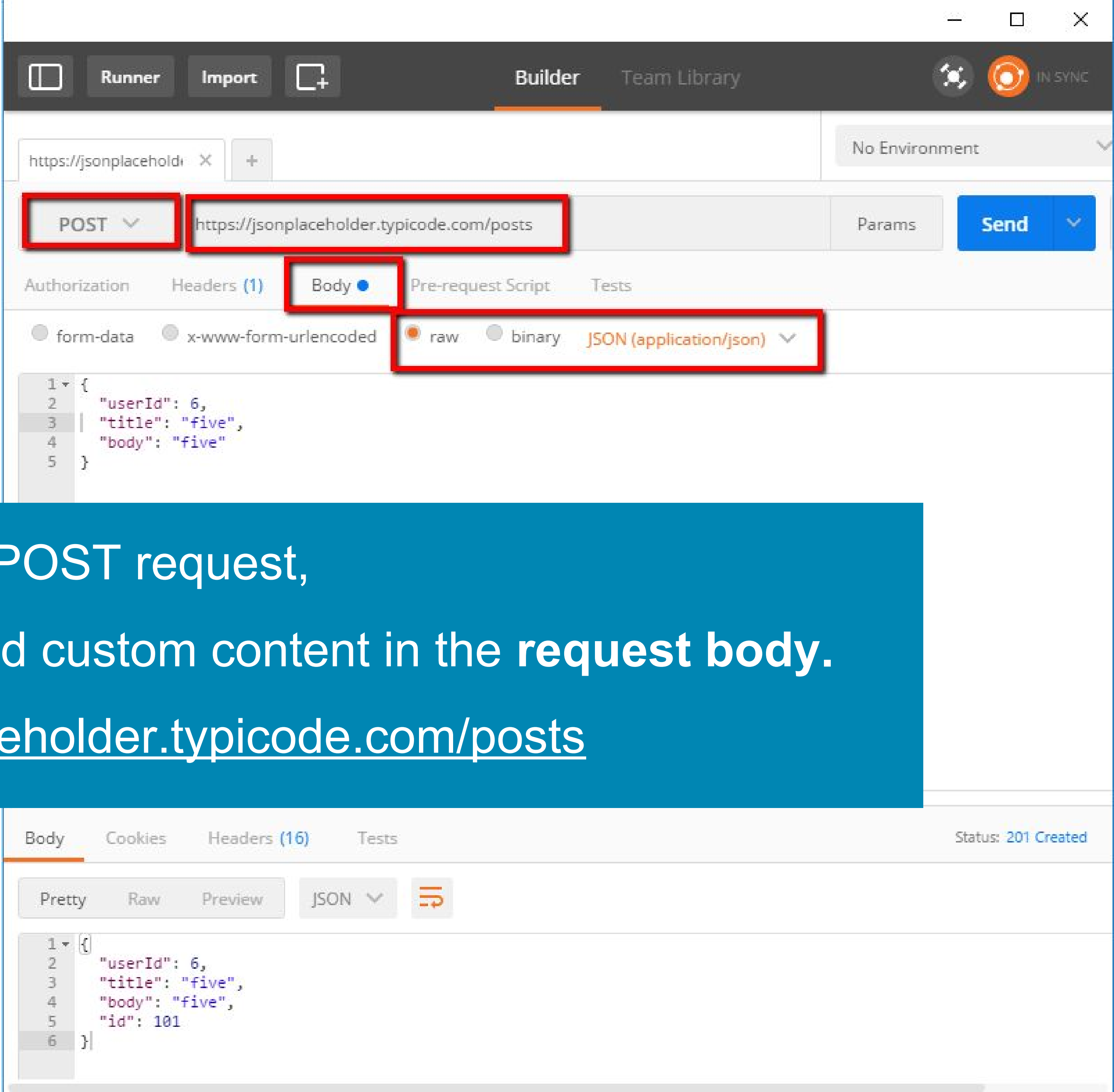
No Environment

GET https://jsonplaceholder.typicode.com/posts/1

Type: No Auth

Status: 200 OK Time: 39 ms

```
{
  "userId": 1,
  "id": 1,
  "title": "sunt aut facere repellat provident occaecati excepturi optio reprehenderit",
  "body": "quia et suscipit\nsuscipit recusandae consequuntur expedita et cum\nreprehenderit molestiae ut ut quas tota
}
```



POST https://jsonplaceholder.typicode.com/posts

Body: ● raw JSON (application/json)

```
{
  "userId": 6,
  "title": "five",
  "body": "five"
}
```

Status: 201 Created

JSON

```
{
  "userId": 6,
  "title": "five",
  "body": "five",
  "id": 101
}
```

With a PUT or POST request,

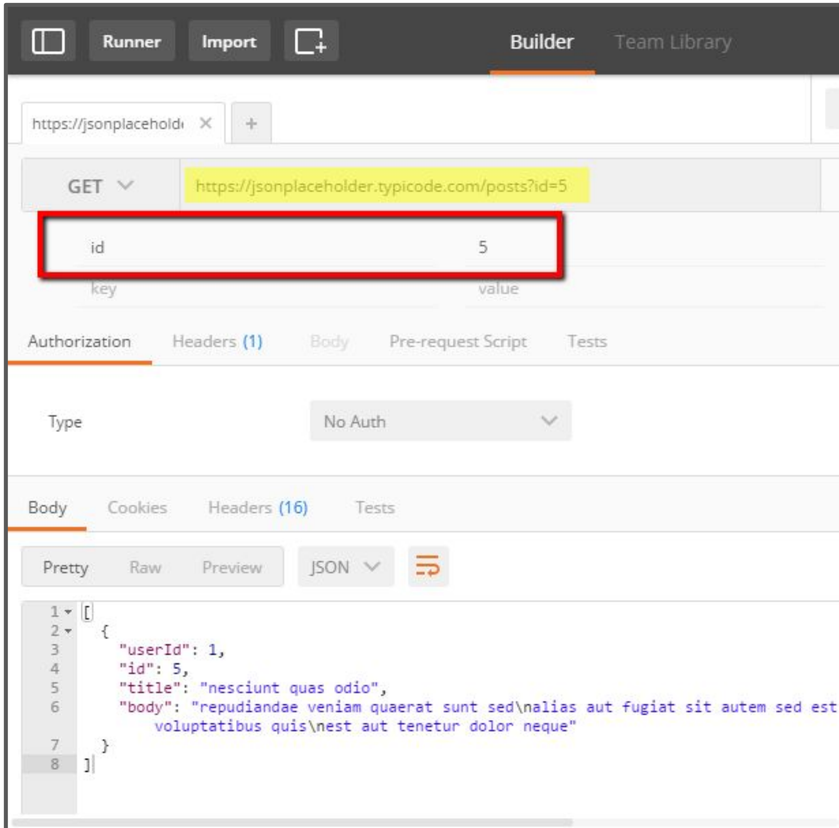
we need to send custom content in the **request body.**

https://jsonplaceholder.typicode.com/posts

M09 - Params

1. Verb = GET & URI = https://jsonplaceholder.typicode.com/posts
2. Click **Params** button and set key-value param
3. Key: id
4. Value: 5

https://jsonplaceholder.typicode.com/posts?id=5



GET https://jsonplaceholder.typicode.com/posts?id=5

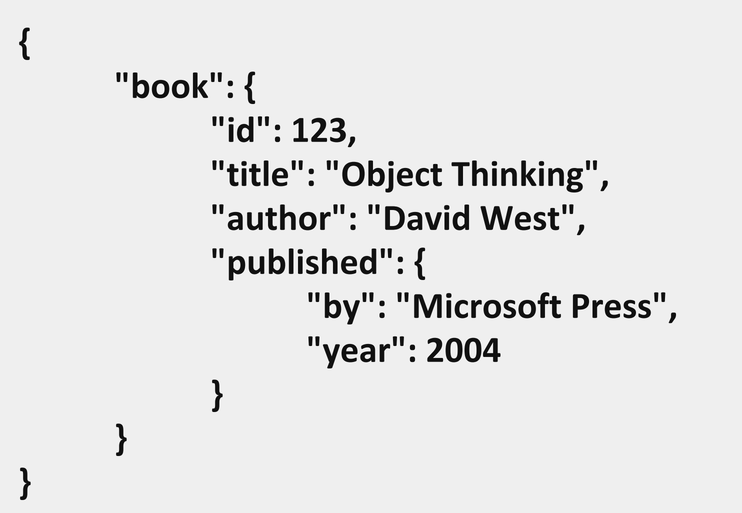
key	value
id	5

Type: No Auth

```
[
  {
    "userId": 1,
    "id": 5,
    "title": "nesciunt quas odio",
    "body": "repudiandae veniam quaerat sunt sed\nalias aut fugiat sit autem sed est
        voluptatibus quis\nest aut tenetur dolor neque"
  }
]
```

JSON

JSON

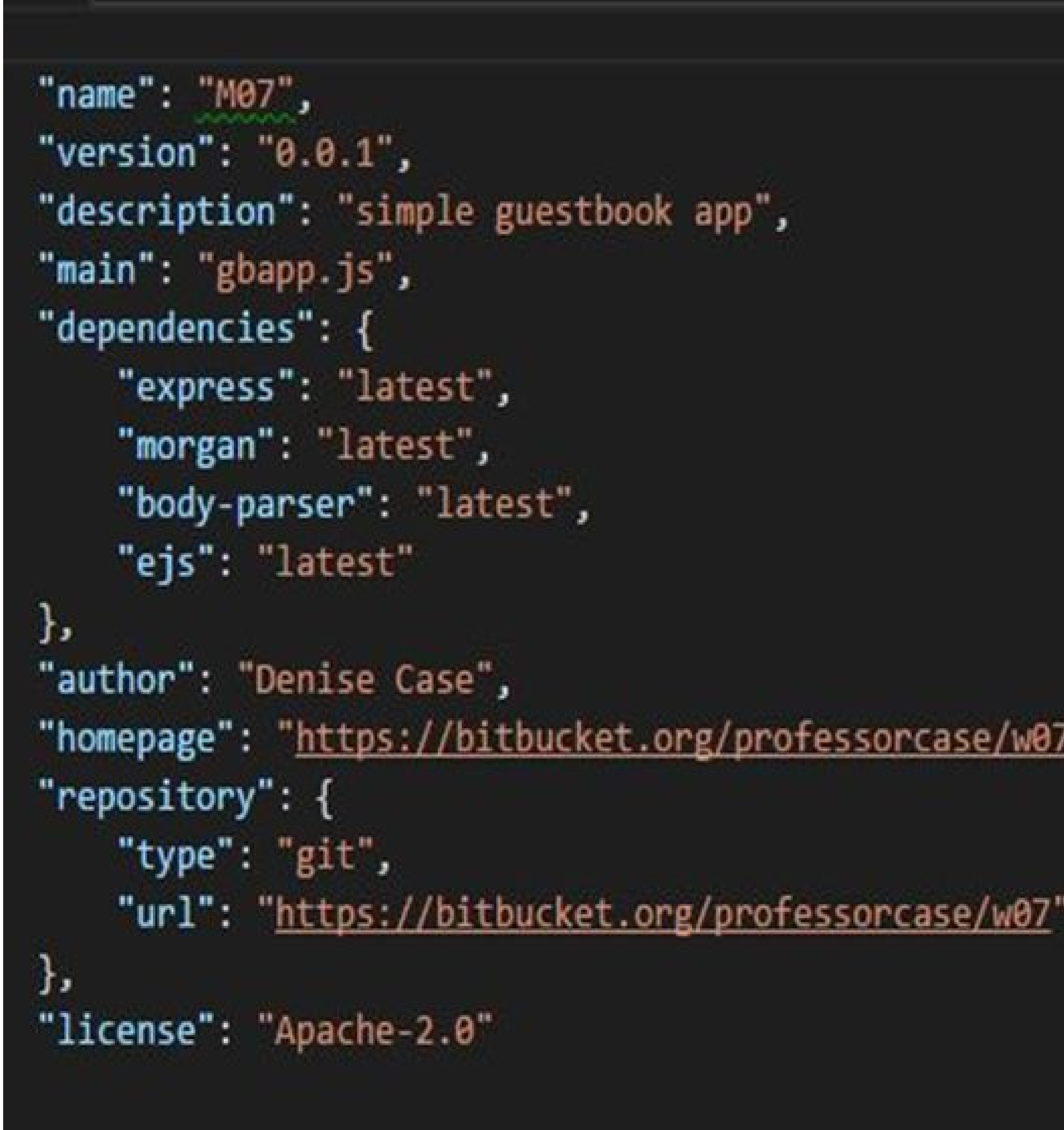


```
{
        "book": {
                "id": 123,
                "title": "Object Thinking",
                "author": "David West",
                "published": {
                        "by": "Microsoft Press",
                        "year": 2004
                }
        }
}
```

- Lightweight data **format**
- Easy-to-use, human-readable
- Syntax for **storing** & **exchanging** data
- Good for serialization (explicit)
- Subset of JavaScript **object notation** syntax (K-V pairs):
    - Data stored in **name/value pairs**
    - Records separated by **commas**
    - Field names & strings in **double-quotes**
    - Curly braces hold **objects**
    - Square brackets hold 0-based **arrays**
- Often used with **AJAX**

In what file do we already use JSON?

Package.json



```
"name": "M07",
"version": "0.0.1",
"description": "simple guestbook app",
"main": "gbapp.js",
"dependencies": {
    "express": "latest",
    "morgan": "latest",
    "body-parser": "latest",
    "ejs": "latest"
},
"author": "Denise Case",
"homepage": "https://bitbucket.org/professorcase/w07
"repository": {
    "type": "git",
    "url": "https://bitbucket.org/professorcase/w07"
},
"license": "Apache-2.0"
```

JSON Convention

property - a name/value pair inside a JSON object.

- property **name** - the name (or key) portion of the property (always a string always quotes)
- property **value** - the value portion of the property.

```
{    "propertyName": "propertyValue"    }
```

https://google.github.io/styleguide/jsoncstyleguide.xml

Limited Value Types

- object
- array
- number
- string
- true / false
- null

```
{
        "book": {
                "id": 123,
                "title": "Object Thinking",
                "author": "David West",
                "published": {
                        "by": "Microsoft Press",
                        "year": 2004
                }
        }
}
```

```
{   "propertyName": "propertyValue"   }
```

JSON object & array

This JSON syntax defines an **employees** object, with an **array** of 3 employee records (objects):

```
{"employees":[
    {"firstName":"John", "lastName":"Doe"},
    {"firstName":"Anna", "lastName":"Smith"},
    {"firstName":"Peter", "lastName":"Jones"}
]}
```

http://www.w3schools.com/js/js_json_intro.asp

XML object & children

```
<employees>
    <employee>
        <firstName>John</firstName> <lastName>Doe</lastName>
    </employee>
    <employee>
        <firstName>Anna</firstName> <lastName>Smith</lastName>
    </employee>
    <employee>
        <firstName>Peter</firstName> <lastName>Jones</lastName>
    </employee>
</employees>
```

JSON / XML Similarities

- plain text
- "self-describing" (human readable)
- hierarchical (values within values)
- can be fetched with an HttpRequest

JSON / XML Differences

- JSON doesn't use end tag
- JSON is shorter
- JSON is quicker to read and write
- JSON can use arrays
- The biggest difference is XML has to be parsed with an XML parser, JSON can be parsed by a standard JavaScript function.

For AJAX

JSON is often faster, easier than XML:

- Using XML
  - Fetch an XML document
  - Use XML DOM to loop through the document
  - Extract values and store in variables
- Using JSON
  - Fetch a JSON string
  - **JSON.Parse** the JSON string to an object (or value)

YAML

```
---
id: 123
title: Object Thinking
author: David West
published:
  by: Microsoft Press
  year: 2004
```

- stands for **YAML** **A**in't **M**arkup **L**anguage
- is a superset of JSON
- .yml files begin with '---', marking the start of the document
- key value pairs separated by colon
- lists begin with hyphen
- Supports comments and complex datatypes
- Use spaces not tabs - **whitespace** matters!
- Human readable and editable
- Used for **configuration** files
- Used with Swagger API framework

Wednesday

Wed: CH 3500 4 PM

- Nick Larson will be talking about **React**, a popular JavaScript library used for building client-side user interfaces.
- Pure JavaScript (with an optional built-in language JSX to make writing more concise)
- Not a framework (easy to add)
- React native (for mobile) and reactWindows available
- Simple to use, but can take a while to set up and learn best practices
- New version: React 16 (built on new core architecture Fiber) with async rendering (demo)

Wed: CH 3500 4 PM

- No regular classes will be held this Wednesday.
- All classes will meet Wed at 4 PM in CH 3500.
- Attendance is mandatory and will be taken during the presentation.
- We will finish any remaining lecture content on Friday.

Project

Goals

- Apply what we've learned
- Practice good sw engineering principles:
  - Separation of concerns
  - MVC (a common design pattern)
  - loosely-coupled components
  - following coding conventions (very important!)
- Practice with JavaScript, CSS, HTML, Bootstrap
- Practice designing and implementing a complex, client-server web application
- Practice collaborative coding
- Provide a useful, working application

MVC - next week

- Developers will build a unique application. We will create an Express app organized using the MVC pattern.
- Each developer will help build models. Models describe our data.
- Each developer will help build controllers. Controllers have the methods that handle web requests based on routing (GET + URL).
- Each developer will help build views. Views allow us to dynamically generate pages based on our data. We'll use the EJS view engine. React is a library for UI. You can use a little bit (or a lot) of React as you like.

Remaining Schedule

- We will start at the beginning - what is the project? What is the purpose?
- In Week 10, we will learn about organizing a fairly complex project using the popular Model-View-Controller design pattern (available in Node, in Java, in C# web apps and more).
- In Week 11 we will have Exam 2.
- Most of the rest of our semester will be practicing and applying what we have learned.
- Future programming assignments and some of the weekly activities will focus on the project.

Project Information Link

https://docs.google.com/presentation/d/1WIQfsEPhlG7S1CHRK-MHa4tCOPbee9ewHQOwOytbvmE/edit?usp=sharing