Android, iOS and Hybrid Applications

Mobile-App Development

OVERVIEW

▸ Databases

  ▸ Using *SQLite* and *Entity Framework* to store data

▸ External data

  ▸ Using HTTP to request data

Databases: SQLite & Entity Framework

SQLITE

▸ File-based Database

▸ Easy to use

▸ **Local** Storage to persist data

https://docs.microsoft.com/en-us/xamarin/xamarin-forms/data-cloud/data/databases

https://github.com/praeclarum/sqlite-net/wiki

SQLᴵᵀᴱ

▸ There are two NuGet packages:

   ▸ sqlite-net-pcl

   ▸ sqlite-net-sqlcipher

▸ The cipher version is encrypted

SQLITE – USAGE

▸ Use the SQLiteConnection

    ▸ For the unencrypted version you can omit the key

```
var options = new SQLiteConnectionString(DatabasePath, true, key: key);
_connection = new SQLiteConnection(options);
```

SQL**ITE** – C**REATE A** T**ABLE**

▸ Ensure the table does not already exist

▸ Creates the table via reflection

```
if (_connection.TableMappings.All(x => !x.TableName.Equals("TodoItem",
StringComparison.InvariantCultureIgnoreCase)))
{
  _connection.CreateTable<TodoItem>();
}
```

SQLITE – ANNOTATIONS

▸ Similar to EntityFramework

▸ PrimaryKey, AutoIncrement

```
public class TodoItem
{
    [PrimaryKey, AutoIncrement]
    public int Id { get; set; }
}
```

SQLite – Annotations

▸ Ignore

▸ Indexed

▸ MaxLength

▸ Unique

▸ Column

▸ Table

SQLITE – CRUD

▸ Create, Read, Update, Delete

▸ Async Versions exist

```
_connection.Table<T>().ToList();

_connection.Update(obj);

_connection.Insert(obj);

_connection.Delete<T>(id);
```

SQLITE - SQL

▸ Execute arbitrary SQL statements

```
_connection.Execute("Select * from [TodoItem]");
```

SQLITE

▸ Have a look at the SQLiteDataStore<T>

▸ Don't forget to initialize/create your database!

▸ It's a simple engine…

EF

▸ It's usable, but...

▸ ...there are limitations

▸ There's no "official" support...

▸ NuGet: Microsoft.EntityFrameworkCore.Sqlite

▸ Call UseSqlite("Data Source=Path")

Questions?

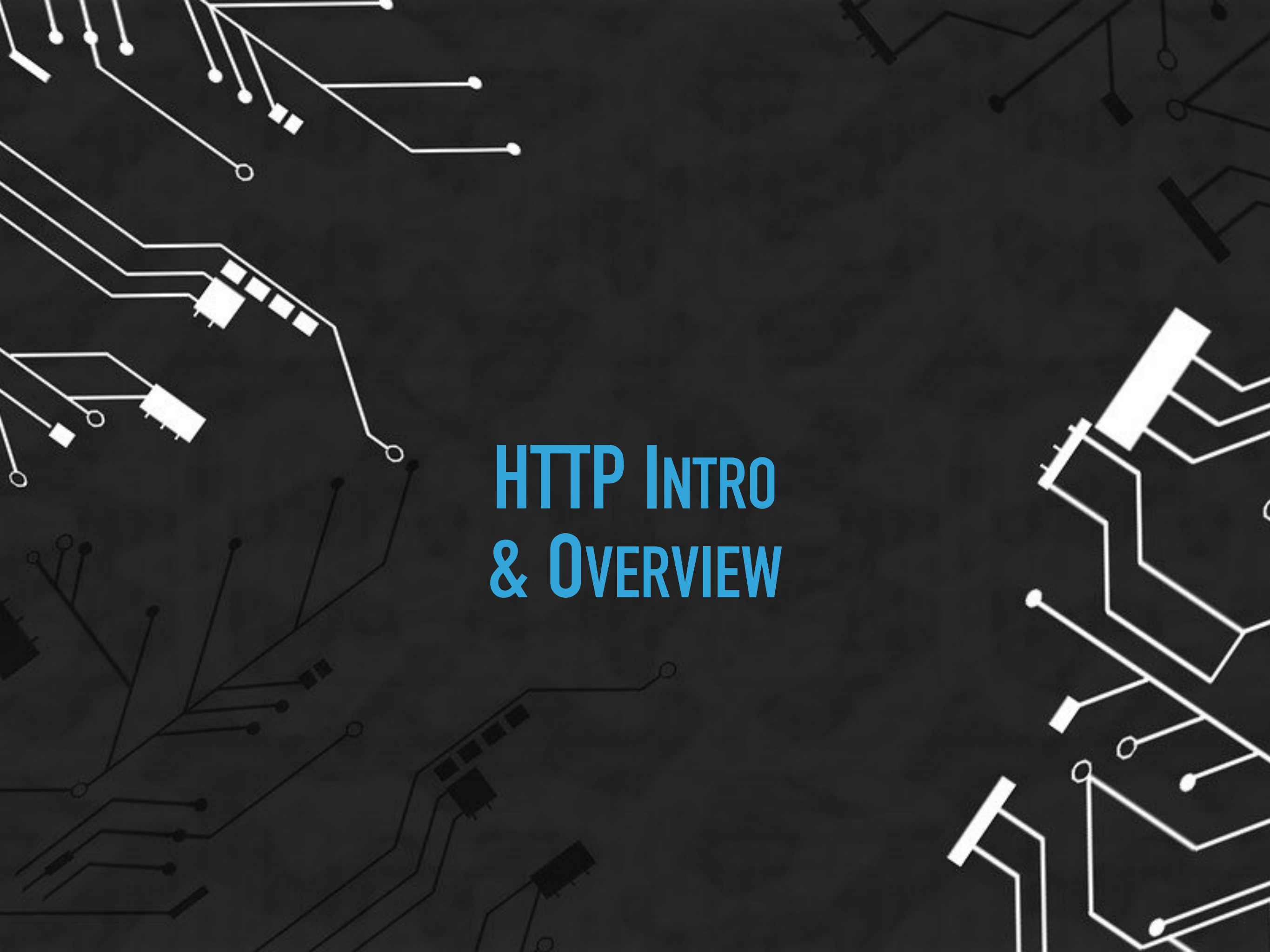# TASKS

‣ Persist your data between application restarts

HTTP Intro & Overview
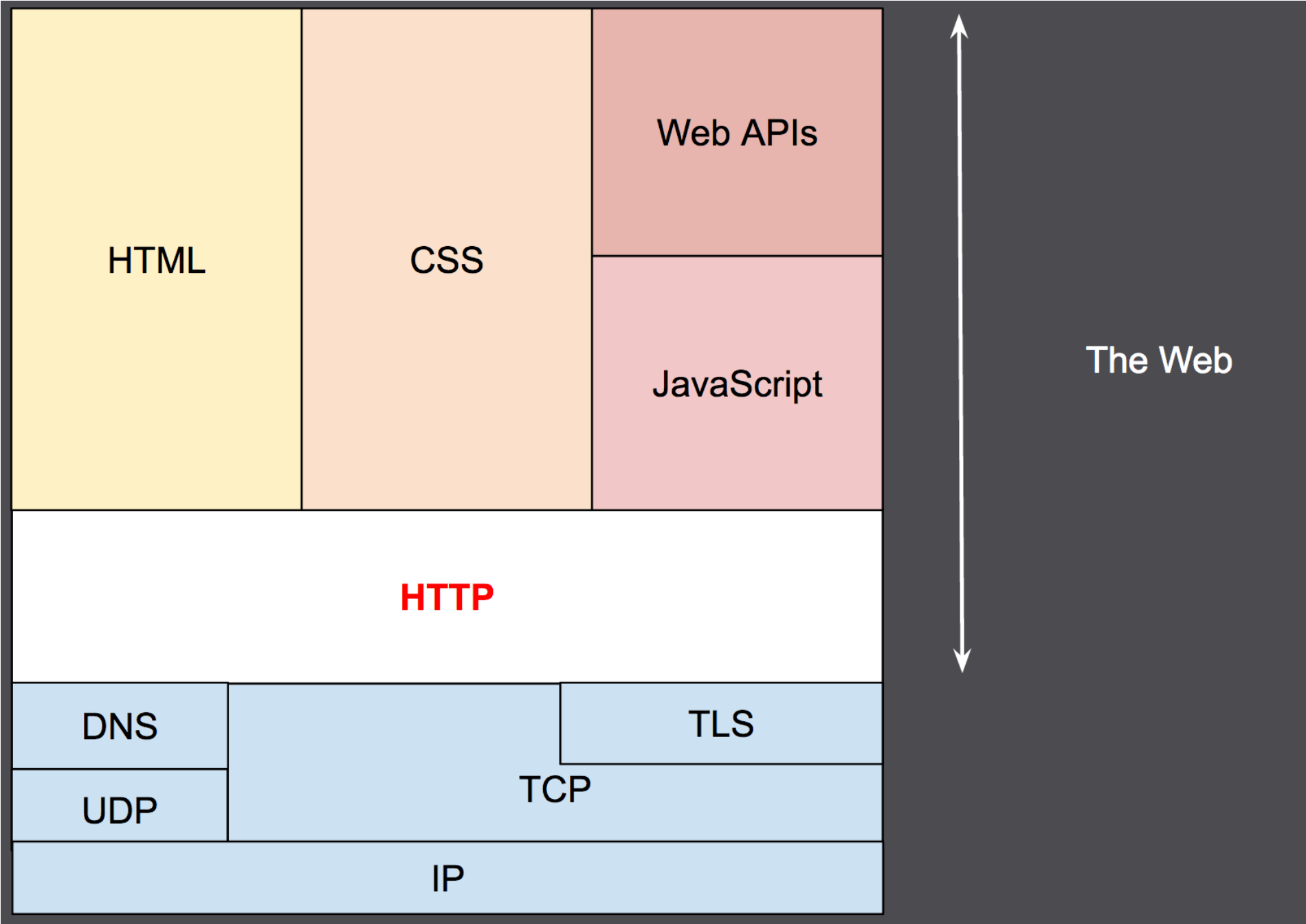
HTTP Protocol

▸ The networking [protocol](#) layer under JS/HTML/CSS

  ▸ Client <-> Server

  ▸ Request <-> Response

HTTP OVERVIEW

HTTP Headers

▸ HTTP Headers are a key concept for extensibility

▸ *Can* be unidirectional

▸ Some are limited to Request or Response only

▸ Key & Value

▸ E.g.: Content-Type : application/json

▸ See documentation for more information

https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers

HTTP REQUEST HEADERS: EXAMPLES

▸ Authorization

   ▸ Send authentication information

▸ Cache-Control

   ▸ Control caching of a request

▸ Accept*

   ▸ Tell the server what sort of result we expect

      ▸ E.g. accept-language

      ▸ E.g. accept-encoding

HTTP Response Headers

▸ Content-Type

  ▸ The format of the response body

▸ Cache-Control

  ▸ Manages caching of resources

Example Request & Response

Request:

GET / HTTP/1.1

Host: developer.mozilla.org

Accept-Language: fr

Response:

HTTP/1.1 200 OK

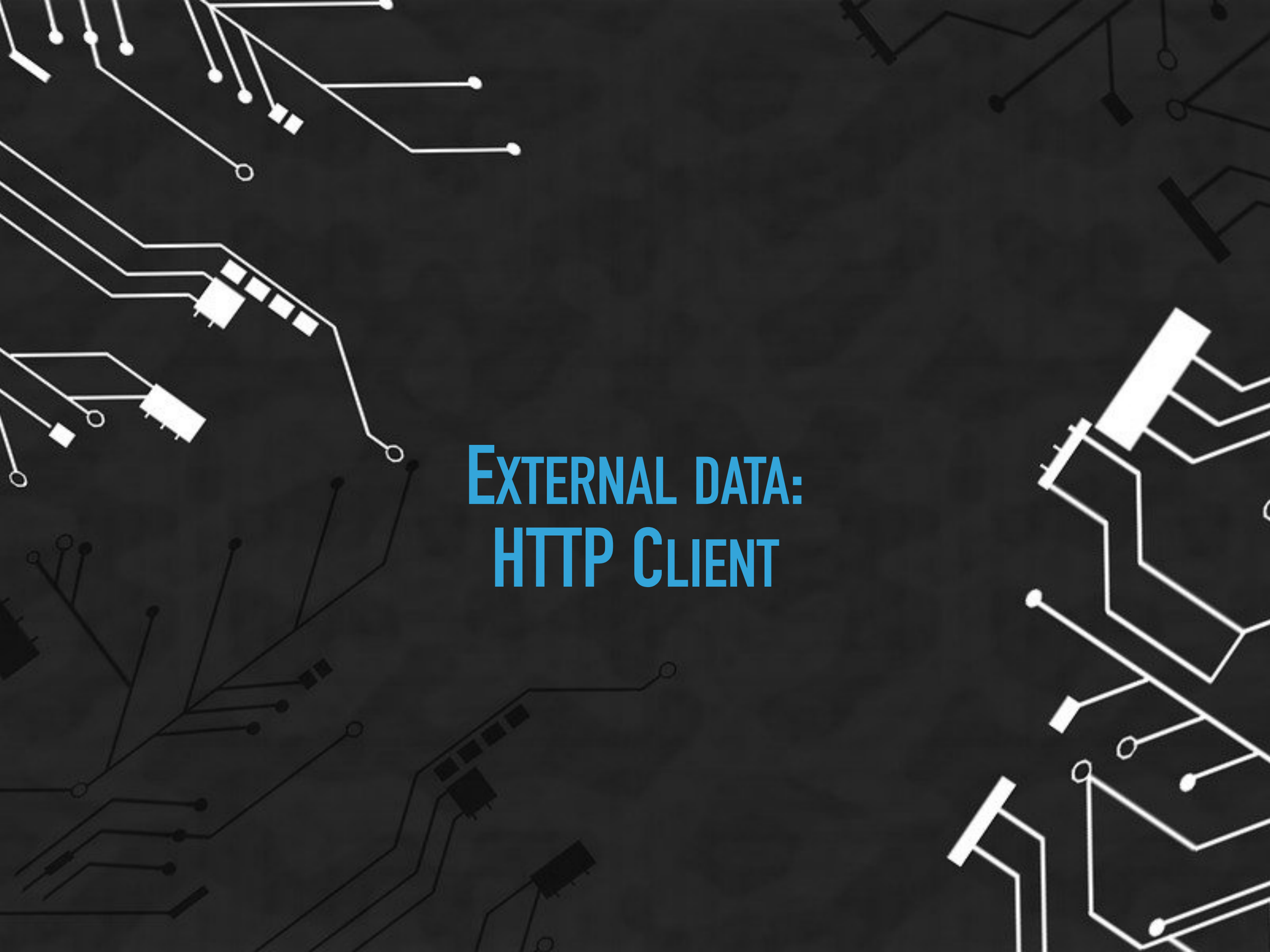Date: Sat, 09 Oct 2010 14:28:02 GMT

Server: Apache

Last-Modified: Tue, 01 Dec 2009 20:18:22 GMT

Accept-Ranges: bytes

Content-Length: 29769

Content-Type: text/html

<!DOCTYPE html... [rest of the 29769 bytes of the requested web page]

EXTERNAL DATA: HTTP CLIENT

HTTP CLIENT

▸ Register it as *transient* in your IoC

   ▸ Each service that needs one will receive a new one

   ▸ But not each request will have a new one

▸ See the <u>documentation</u> for more information

```
Services.Register(() => new HttpClient());
```

HTTP Client GET

```
public async Task<IEnumerable<T>> Get<T>()
{
  var result = await _client.GetAsync("https://google.com");

  if (result.IsSuccessStatusCode)
  {
    // Do something with the result...
    var stringResult = await result.Content.ReadAsStringAsync();
    return JsonConvert.DeserializeObject<IEnumerable<T>>(stringResult);
  }

  // You might want to throw an exception here since the request was not successful.
  return new List<T>();
}
```

HTTP CLIENT POST

```csharp
public async Task<int> Post<T>(T toPost)
{
  var serializedObject = JsonConvert.SerializeObject(toPost);
  var content = new StringContent(serializedObject, Encoding.UTF8, "application/json");
  var result = await _client.PostAsync("https://google.com", content);

  _client.DefaultRequestHeaders.Authorization = new AuthenticationHeaderValue("Bearer",
"JWTToken");

  // Same handling as with get. Check the status code and read out the result.
}
```

HTTP CLIENT HEADERS

▸ Set headers on each *HttpClient* instance

```
_client.DefaultRequestHeaders.Authorization = new AuthenticationHeaderValue("Bearer",
"JWTToken");
```

HTTP CLIENT BASE ADDRESS

▸ Set it once per instance

▸ Requests include only the relative path

```
_client.BaseAddress = new Uri("https://myapi.com");

var result = await _client.GetAsync("/api/");
```

JSON

▸ Add the Newtonsoft.Json NuGet package

  ▸ serialize JSON

  ▸ deserialize JSON

▸ For example:

```
JsonConvert.DeserializeObject<TodoItem>(result);
```

POSTMAN: API TESTING

▸ <u>Postman</u> is an HTTP Rest client

  ▸ Create requests and tests them

  ▸ Useful if you're trying out an API and don't know the exact behavior

QUESTIONS?

TASKS

▸ Work on your app

▸ Use Databases and Http-Clients where appropriate