Android, iOS and Hybrid Applications

Mobile-Development

Overview

- SQLite
- HTTP
- Error Handling

SQLite

- File 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’re 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);
```

SQLite – Create a Table

- 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 IDatabase and SQLiteDatabase
- Don’t forget to initialise/create your database!
- It’s a simple engine…

EF

- You can use it
- There’re limitations…

https://docs.microsoft.com/en-us/ef/core/providers/sqlite/limitations

- NuGet: Microsoft.EntityFrameworkCore.Sqlite
- UseSqlite("Data Source=Path”)
- There’s no “official” support…

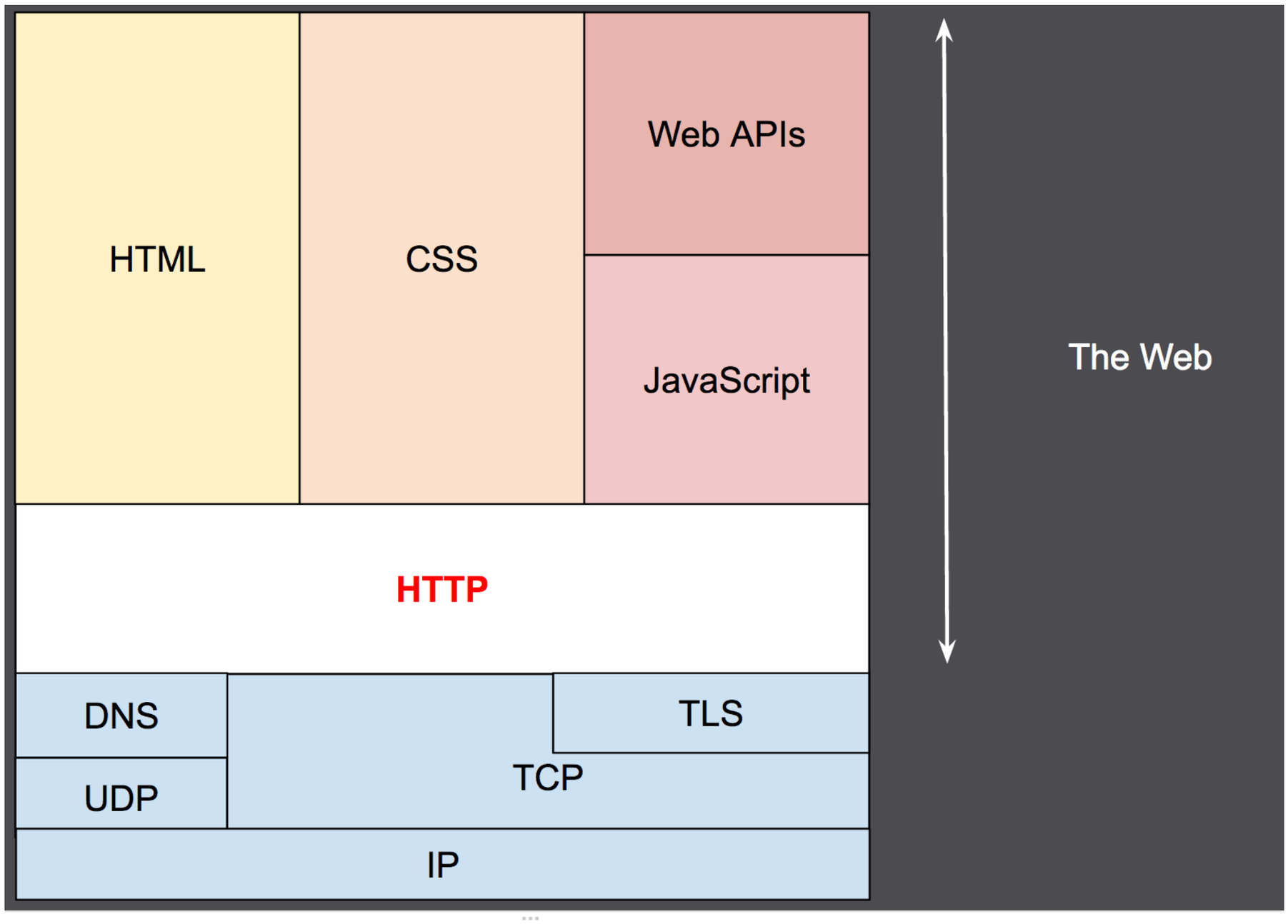
https://docs.microsoft.com/en-us/ef/core/providers/sqlite/?tabs=dotnet-core-cli

Questions?

Tasks

- Persist your data over application restarts

HTTP OVERVIEW



HTTP Protocol

- The top layer under JS/HTML/CSS
- Client - Server
- Request - Response
- HTTP Headers as a key concept for extensibility

https://www.w3.org/Protocols/

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

HTTP Headers

- Can be unidirectional
  - Request & Response
  - Request or Response only
- Key & Value
  - Content-Type : application/json

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

HTTP Request Headers

- Authorization
  - Send authentication information
- Cache-Control
  - Control caching of a request
- Accept
  - Tell the server what sort of result we expect

HTTP Response Headers

- Content-Type
  - What type of content is sent
- Cache-Control
  - Manage caching of resources

Example Request

GET / HTTP/1.1

Host: developer.mozilla.org

Accept-Language: fr

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... (here comes the 29769 bytes of the requested web page)

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

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

https://docs.microsoft.com/en-us/xamarin/xamarin-forms/data-cloud/web-services/rest

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

```
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 them per HttpClient instance

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

HTTP Client Base Address

- Set it once
- Alter your request to only include the path

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

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

JSON

- Use Newtonsoft to deserialise values
- Add the package to your project

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

https://www.newtonsoft.com/json/help/html/Introduction.htm

Debugging APIs

- Use Postman
- Create requests and tests them
- Useful if you're trying out an API and don't know the exact behaviour

https://www.postman.com/product/api-client/

Questions?

Tasks

- Work on your app
- Use Databases and Http-Clients where necessary

AUTHENTICATION

- Initial Setup:
  - Generate a symmetric key (Key A)
  - Encrypt (Key A) with the password (Key B)
  - Encrypt (Key A) with biometrics (Key C)
  - Use (Key A) to decrypt your database

AUTHENTICATION

- Startup:
  - Check if (Key A) exists
  - Try with Biometrics
  - Fallback to password
    - Optional: Setup Biometric again
  - Log in

Questions?