1. **FIRST WEEK**

**OOP**

CLASSE

Una classe è un tipo (o modello base) di un oggetto, che ha degli attributi (ovvero delle caratteristiche) e dei metodi (delle funzionalità).

OGGETTO/ISTANZA:

Implementazione concreta creata a partire da una classe. L’oggetto è un’istanza della classe.

**ATTRIBUTI E METODI**

Gli attributi sono delle variabili che definiscono delle proprietà di una classe, i metodi sono le funzionalità che definiscono il comportamento degli oggeti.

PROPERTY

Sono un insieme di metodi get e set.

Le property servono per dichiarare un attributo di una classe che sarà privato ma che avrà i metodi get e set (che serviranno per leggerlo e modificarlo) che saranno pubblici e quindi accessibili dall’esterno.

**PROPRIETA OOP**

**INCAPSULAMENTO, ASTRAZIONE, POLIMORFISMO, EREDITARIETA’**

* **INCAPSULAMENTO:**

strategia attraverso la quale raggruppiamo le informazioni e funzionalità di un oggetto. consiste nel nascondere i dettagli interni di un oggetto e permettere l'accesso solo attraverso metodi o proprietà pubbliche. Gli oggetti espongono metodi pubblici che permettono di interagire con i dati interni in maniera controllata.

**INFORMATION HIDING:**

Attraverso l’information hiding possiamo nascondere dettagli implementativi di una classe con l’utilizzo di modificatori di accesso (ad esempio private e protected).

* **ASTRAZIONE**

L’astrazione consiste nella modellazione di concetti reali attraverso la creazione di classi che ne rappresentano i comportamenti utilizzando gli attributi e i metodi.

Inoltre nasconde la logica complessa che c’è dietro al programma e mostra solo le funzionalità necessari.

* **POLIMORFISMO**

Il polimorfismo permette agli oggetti di essere trattati come istanze della loro classe base, eseguendo comportamenti diversi a seconda del tipo effettivo.

**VIRTUAL**

La keyowrd virtual si utilizza per permettere ai metodi una superclasse di essere sovrascritti in una sottoclasse.

**OVERRIDE**

La keyword override permette la ridefinizione di un metodo di una superclasse.

**CLASSE ASTRATTA**

E’ una classe che non permette la sua istanziazione.

Può contenere metodi astratti, che sono implicitamente virtual e che devono necessariamente essere ridefiniti, mentre quelli non astratti li utilizziamo così come sono implementati nella superclasse.

**UML (Unified Modeling Language)**

E’ un linguaggio universale di modellazione standard utilizzato per documentare in maniera grafica i comportamenti di un programma.

Esempi: class diagram, use case diagram e sequence diagram.

**Quali sono le tecniche di debugging più utilizzate in Visual Studio?**Breakpoints, esecuzione step-by-step (F10, F11), il visualizzatore di variabili

**Come si crea un progetto Console Application in Visual Studio?**  
Dalla schermata iniziale, si seleziona "Crea nuovo progetto", si sceglie "Console Application" come tipo di progetto, si specifica il linguaggio C# e si segue il wizard per configurare il progetto.

**Eccezioni**

Meccanismo per la gestione degli errori che si verificano durante l’esecuzione di un programma. Solitamente vengono utilizzate all’interno di un blocco try..catch, nel catch.

1. **SECOND WEEK - SQL**

**Diagramma ER**

Strumento grafico per rappresentare strutture dati e le relazioni tra di esse.

**RELAZIONI**

**One to many:** associazioni tra due entità in cui un’istanza di una può essere associata a molteplici istanze dell’altra .

ES. CLIENTE-ORDINI

**One to one:** associazioni tra due entità in cui un’istanza di una entità è associata a una istanza dell’altra entità

ES. PERSONA-PASSAPORTO

**Many to many:** associazioni tra due entità in cui un’istanza di una entità è associata a molteplici istanze dell’altra E ***VICEVERSA.***

ES. STUDENTI-CORSI

**DDL, DML, QL**

DDL: Data Definition Language, è il linguaggio che si occupa della creazione degli oggetti principali del database che formeranno lo SCHEMA, ovvero la parte invariante del database (CREATE, ALTER, DROP)

DML: Data Manipulation Language, è il linguaggio che si occupa della manipolazione dei dati (INSERT, UPDATE, DELETE)

QL: Query Language, è il linguaggio che si occupa di interrogare i dati (SELECT)

**FUNZIONI DI AGGREGAZIONE:**

Sono funzioni che si occupano di raggruppare e riordinare i dati in base a determinate condizioni.

**GROUP BY**

Raggruppa righe che hanno gli stessi valori in colonne specificate, si usa per le funzioni di aggregazione (avg, sum, max, min)

**HAVING:**

Viene usato insieme al group by. Mentre where filtra riga prima dell’aggregazione, having lo fa dopo che l’aggregazione è stata eseguita.

**JOIN:**

Funzione per combinare righe di due tabelle che hanno colonne in comune.

INNER JOIN: restituisce solo le righe che hanno corrispondenza in entrambe le tabelle. Se non ci sono saranno null.

LEFT/RIGHT JOIN: Restituisce tutte le righe della tabella di sx e quelle corrispondenti di quelle di destra.

**VIEW:**

Tabella virtuale il cui contenuto è definito da una query. I dati vengono generati dinamicamente ogni volta che viene chiamata e non vengono salvati fisicamente.

**STORED PROCEDURE:**

Una stored procedure è un blocco di istruzioni sql salvate nel db. Al suo interno possiamo innestare vero e proprio linguaggio di programmazione, chiamato Transact SQL (TSQL).

Al contrario della view, con la stored procedure possiamo interagire attraverso l’utilizzo di variabili.

**TRANSACTION:**

Blocco di istruzioni che deve obbligatoriamente essere eseguito in maniera unica (atomica) e in caso vada a buon fine viene eseguito il COMMIT, altrimenti si esegue un ROLLBACK e si torna allo stato antecedente e quindi non viene eseguita nessuna istruzione che era al suo interno.

**ATOMICITA’:** le operazioni vengono eseguite in maniera completamente o per nulla

**CONSISTENZA:** consistenza garantisce che una transazione porti il database da uno stato valido a un altro stato valido. Dopo una transazione, il database deve soddisfare tutte le regole di integrità, come vincoli di chiave primaria e referenziale, per garantire la correttezza dei dati.

**ISOLAMENTO:** l'Isolamento significa che le transazioni devono essere eseguite indipendentemente l'una dall'altra. Gli effetti di una transazione in corso non devono essere visibili ad altre transazioni fino a quando la transazione non viene completata con successo (es conto bancario)

**DURABILITA’**: la Durabilità assicura che, una volta che una transazione è stata completata con successo, i suoi effetti siano permanenti nel database.

**THIRD WEEK**

**CONNESSIONE C# A DB**Si utilizza il pacchetto System.Data.SqlClient o l'Entity Framework per la connessione. Si crea una stringa di connessione che specifica il server, il database, l'utente e la password, e poi si utilizza la classe SqlConnection per aprire la connessione.

**SQL INJECTION**

SQL Injection è una vulnerabilità che consente agli utenti malintenzionati di eseguire comandi SQL non autorizzati attraverso input non sanitizzati. Per prevenirla, è importante utilizzare query parametrizzate e stored procedures invece di concatenare stringhe di query.

**SINGLETON**

Il pattern Singleton garantisce che una classe abbia una sola istanza in tutto il programma e fornisce un punto di accesso globale a tale istanza. Si implementa rendendo il costruttore privato e fornendo un attributo statico che rappresenta l’istanza e un metodo statico pubblico per ottenere tale istanza.

**DAO (Data Access Object)**

I DAO sono delle classi che si occupano di effettuare query al database e sono utili per separare la business logic dalla logica di accesso ai dati. Viene utilizzato per standardizzare l'accesso ai dati e per ridurre la duplicazione del codice.

**ENTITY FRAMEWORK**

Entity Framework è un ORM (Object-Relational Mapper) che permette di lavorare con i database utilizzando oggetti C# anziché scrivere query SQL. In particolar modo consente di mappare tabelle del db in classi c# facilitando l’interazione con i dati trattandoli come oggetti. Questo approccio rende il DBMS indipendente, lavorando su un modello concettuale indipendente dal database fisico.

**ORM**

E’ una tecnica con cui le tabelle del db vengono trasformate in oggetti.

**DATABASE-FIRST**

Nel Database-First, il database esiste prima e il modello di oggetti viene generato dal database esistente. Questo può essere effettuato tramite una tecnica chiamata **SCAFFOLDING** che genera automaticamente del codice c# rendendo le tabelle delle classi model**.**

**CODE-FIRST**

Nel Code-First, si scrive prima il codice C# per le entità e il database viene generato dal modello del codice tramite **MIGRATIONS.** Esse permettono di gestire l’evoluzione dello schema del database quando vengono apportate modifiche al modello Code-First, senza perdere i dati esistenti. Il db tiene traccia di tutte le migration e quindi di tutte le modifiche apportate.

**LINQ (Language Integrated Query)**

È un insieme di tecnologie che permettono di eseguire query direttamente su collezioni di dati utilizzando una sintassi simile a SQL.

ESEMPIO: var result = myList.Where(item => item.Age > 30).OrderBy(item => item.Name).Select(item => item);

**HASHTABLE**

E’ una collezione di dati non tipizzati composta da elementi formati da una coppia chiave-valore.

**DICTIONARY**

E’ una collezione di dati FORTEMENTE TIPIZZATI composti da una coppia chiave-valore.

**WPF:**

E’ una libreria di classi del framework dot net progettata per creare interfacce grafiche per applicazione desktop.

**PROVIDER SQL:**

Fornisce librerie e strumenti per la connessione al database

DRIVER DI CONNESSIONE:

**DBCONTEXT:**

Il DBContext rappresenta una libreria che mi fornisce gli strumenti per comunicare con il database e al suo interno ci sono i dbset che rappresentano le tabelle.

**DBSET**

Gestisce l’accesso a un insieme di entità e rappresenta il corrispondente di una tabella del database.

**FOURTH WEEK**

**HTML (HyperTextMarkup Language)**

E’ un linguaggio di markup utilizzato per creare la struttura di una pagina web.

**CSS (Cascading Style Sheets)**

E’ un linguaggio che permette di applicare degli stili grafici agli elementi html.

**Layout Responsive**

Un layout responsive adatta il design di una pagina web alle diverse dimensioni dello schermo

**BOOTSTRAP**

E’ un framework CSS che ha una collezione di componenti e stili predefiniti

**Javascript**

E’ un linguaggio di scripting interpretato utilizzato principalmente per le pagine web. Per interpretato si intende che il codice viene tradotto riga per riga durante la sua esecuzione.

**Differenza tra LET, VAR E CONST**

var ha scope di funzione e può essere ridefinita o aggiornata; let ha scope di blocco, può essere aggiornata ma non ridefinita; const ha scope di blocco e non può essere né ridefinita né aggiornata.

**PROMISES**

Una **Promise** è un oggetto che rappresenta il completamento (o il fallimento) di una operazione asincrona e il suo risultato. Le Promises hanno tre stati: **Pending**, **Resolved** (completata con successo) o **Rejected** (fallita).

**ARROW FUNCTION**

Sono una sintassi compatta per dichiarare funzioni. Non hanno il this e sono utilizzabili come callback.

**CALLBACK**

Funzioni che vengono richiamate come argomento di altre funzioni che vien poi richiamata al termine di un operazione. Vengono utilizzate per gestire operazioni asincrone senza bloccare il flusso principale del programma.

**ASYNC/AWAIT**

Sono keyword utilizzate per gestire operazioni asincrone. **async** dichiara una funzione asincrona, mentre **await** sospende l'esecuzione della funzione finché una Promise non viene risolta.

**DOM (Document Object Model)**

Rappresentazione strutturata di un documento html sotto forma di albero di nodi. Ogni nodo rappresenta un tag.

**JQUERY**

Libreria javascript utilizzata per semplificare la manipolazione del DOM.

Si utilizza anche per eseguire chiamate ajax.

**AJAX**

Insieme di tecnologie di sviluppo usate per creare pagine web dinamiche, ciò permette alle pagine web di inviare e recuperare dati dal server in modo asincrono.

**WEB RENDERING**

Processo di trasformazione del codice in contenuti visivi e interattivi

**V8**

Js engine V8 è un motore JavaScript open source sviluppato da Google, attualmente incluso in Google Chrome. Esegue e compila JavaScript, ha un [garbage collector](https://it.wikipedia.org/wiki/Garbage_collection" \o "Garbage collection) altamente efficiente e gestisce in modo efficace l'allocazione di memoria per gli oggetti.

**FORM**

Tag HTML che ci permette l’inserimento di dati per un successivo invio ad un server

**FIFTH WEEK**

**ARCHITETTURA CLIENT-SERVER**

L'architettura client-server è un modello di comunicazione in cui il client invia richieste a un server, e il server risponde con i dati o l'esecuzione di operazioni richieste. Il client e il server sono entità che comunicano attraverso una rete.

**THIN CLIENT E THICK CLIENT**

Un "Thin Client" dipende pesantemente dal server per l'elaborazione e l'archiviazione dei dati, mentre un "Thick Client" esegue molte delle elaborazioni localmente e richiede meno interazione con il server.

**HTTP**

**Protocollo di rete usato per la comunicazione tra client e server. Permette lo scambio di dati e la navigazione su internet.**

* http comunica con un ciclo di request e response tra client e server. Il client invia una http request e il server gli risponde con una http response.
* I metodi di richiesta http sono GET, POST, PUT E DELETE.
* E’ stateless , ossia ogni richiesta è indipendente dalle altre. Il server non mantiene lo stato delle richieste precedenti.

**HTTP REQUEST**

La request è una richiesta inviata dal client al server per leggere o manipolare dati.

Al suo interno deve avere il metodo (get, post….), l’url a cui fare la richiesta, l’headers che contiene informazioni aggiuntive sulla richiesta (tipo di contenuto) e il body che contiene dei dati da inviare al server (ovviamente solo post e put).

**HTTP RESPONSE**

La response è la risposta inviata dal server al client ed è composta da un body contenente i dati e uno status code che indica l’esito della richiesta (200 - ok, 400 – error client, 500 – error server)

**CORS POLICY**

È uno standard che permette a un server di indicare quali origini possono accedere alle risorse presenti su di esso

**ASP NET CORE**

E’ un framework sviluppato da Microsft utilizzato per la creazione di pagine web

**MVC (Model-View-Controller)**

E’ un pattern architetturale utilizzato per separare la logica di presentazione dei dati dalla logica di business. Grazie a questo pattern possiamo separare le responsabilità tra classi.

* **MODEL**: gestisce i dati e la logica di business
* **VIEW**: si occupa della presentazione dei dati all’utente e la usa interazione con essi
* **CONTROLLER**: si occupa di controllare ed elaborare gli input immessi dall’utente

**ACTION RESULT**

Le action sono dei metodi del controller utilizzati in scenari restful, che ci permettono di restituire delle risposte al client.

IActionResult si utilizza quando non deve tornare nessun dato, ma solo una http response.

ActionResult si utilizza quando devo tornare un dato (ad esempio un oggetto)

**CONTROLLER**

Il controller si occupa di gestire le richieste e gli input dell’utente, riceve le richieste, le interpreta e decide quali azioni intraprendere.

**SERVICE**

Il service si occupa della business logic, ovvero elaborare i dati ricevuti dal controller e li restituisce alla repo.

Solitamente si occupa del filtraggio dati in entrata/uscita attraverso il DTO.

**REPO**

La repository si occupa di accedere ai dati nel database. E’ responsabile di eseguire operazioni CRUD direttamente sul database , solitamente con sintassi LINQ.

**MODELS**

I model sono delle classi che rappresentano delle entità. Solitamente rappresentano una tabella e le sue proprietà rappresentano le colonne.

**API**:

Application Programming Interface, servono per scambiare informazioni tra applicazioni o tra client e server.

**RESTFUL**

Le API RESTFUL hanno un endpoint unico per tutti i metodi. Ad esempio per vedere la lista dei film faccio una request

**SIXTH WEEK**

**ANGULAR**

Un componente è una parte fondamentale di un progetto Angular dato che definisce sia il template (la view/struttura definita in html) che la logica, cioè le funzionalità che il componente stesso dovrà eseguire (scritta in typescript)

**COS’è UN COMPONENT**

Un component è un insieme di 4 file. 2 si occupano della logica e sono spec e ts, e altri 2 della presentazione (html,css)

**COS’è UN MODULO**

Un modulo è solitamente un file in cui viene definita una classe e tutte le sue funzionalità.

**TYPESCRIPT**

Typescript è un superset di javascript, che aggiunge tipizzazione forte e altre funzionalità (ad esempio le interfacce). Il codice typescript viene compilato in javascript.

**DEPENDENCY INJECTION**

La DI è un pattern che permette di gestire le dipendenze di una classe. Invece di creare ogni volta la classe da cui la mia classe dipende la posso iniettare direttamente nel suo costruttore. Questa classe da iniettare deve essere definita esternamente come iniettabile.

**Con SCOPED,TRANSIENT E SINGLETON dichiariamo in che modo una classe viene resa disponibile al programma:**

**SCOPED:** Un'istanza della classe viene creata una volta per ogni ambito o sessione

**TRANSIENT:** Un'istanza della classe viene creata ogni volta che viene richiesta.

**SINGLETON:** Un'istanza della classe viene creata una volta sola e viene riutilizzata per tutte le richieste successive durante il ciclo di vita dell'applicazione.

**MVVM**

E’ un pattern che separa model, view e view model. Separa la logica di presentazione dalla logica di business e tramite il model-view fa interagire model e view.

**TWO WAY BINDING [()]**

Con il two way binding possiamo sincronizzare automaticamente l’invio dei dati tra la view e il model. Questo significa che quando i dati nella view cambiano in automatico cambiano anche nel model.

**SEVENTH WEEK**

**COOKIE:**

Il cookie è un contenitore contenente una stringa, solitamente al loro interno vengono scritte informazioni sull’utente (tipo la lingua, dark mode etc).

Il cookie viene generato dal server e viene salvato nel nostro browser. La loro vulnerabilità è che può essere facilmente rubato o modificato.

**SESSIONE:**

La sessione è anch’essa un contenitore dati, generato dal server ma che al contempo crea un cookie lato client con un id che punta esattamente a quella sessione sul server.

La vulnerabilità della sessione c’è quando abbiamo più server su cui risiede il nostro sito e quindi la sessione non è condivisibile su più server.

**JWT:**

Jason web token, è una chiave che ci viene rilasciata dal server nella response all’accesso in un sito. Ci servirà per effettuare ogni successiva request al server in maniera autorizzata.

[ovvero una **stringa molto concisa**, di qualche centinaio di byte, che il server trasmette al client affinché possa provare la sua identità in ogni successiva richiesta HTTP.]

SESSION IJHACING:

furto del cookie

**MONGODB**

MongoDB è un database NOSQL, ovvero non si basa su tabelle e relazioni come nei tradizionali db.

Il database è organizzato in documenti (*record*) inseriti all’interno di collection (*tabelle*).

Questi documenti sono in formato BSON [Binary JSON], un formato molto simile al json ma più veloce e compatto.

BSON:

MongoDB non ha una struttura fissa, ovvero non ha uno SCHEMA.

Dato che non esistono relazioni tra documenti abbiamo solo due modi per creare una relazione tra un documento e un altro:

* RELAZIONE CON RIFERIMENTO: utilizzo l’ObjectId di un documento all’interno di un altro documento, creando un legame simile ad una relazione.

**{**

**"\_id": ObjectId("64a8cafe12b3da5d834e74e3"),**

**"product": "Laptop",**

**"quantity": 1,**

**"userId": ObjectId("648d4b23a8f6fc4b0a4c2131") // Riferimento all'ID dell'utente**

**}**

Con il FINDONE() possiamo effettuare la query:

Prima cerchiamo l’ordine

**const order = db.orders.findOne({ \_id: ObjectId("64a8cafe12b3da5d834e74e3") });**

Poi in base all’id dell’ordine cerchiamo l’utente

**const user = db.users.findOne({ \_id: order.userId });**

* RELAZIONE CON INCORPORAMENTO (EMBEDDING)

Con l’embedding includiamo tutti i dati di un documento all’interno di un altro documento.

**{ "\_id": ObjectId("648d4b23a8f6fc4b0a4c2131"),**

**"name": "Mario Rossi",**

**"email": "mario.rossi@example.com",**

**"orders": [**

**{ "product": "Laptop",**

**"quantity": 1,**

**"date": "2023-10-15" } ]**

**}**