POSSIBILI DOMANDE DI TEC WEB INVENTATE DA ME E RIPETIZIONE TEORIA

Emilia Napolano

1. **Cosa sono i Framework Web?**

Un framework è un set di componenti e strumenti predefiniti che fungono come fondamento per lo sviluppo software. Sono progettati per semplificare lo sviluppo di applicazioni web, fornendo un modo strutturato di costruirle e organizzarle, riducendo il bisogno degli sviluppatori di iniziare un nuovo progetto da zero e re-inventare un qualcosa che è stato già inventato.
Includono funzionalità di routing, parsing delle richieste, validazione degli input, gestione dei cookie e delle sessioni; gestiscono i template engine permettendo il rendering di contenuti dinamici e includono meccanismi di ORM (Object-Relational Mapping) che semplificano le interazioni con il database permettendo agli sviluppatori di lavorare con oggetti piuttosto che con query SQL.

2. **Che differenze ci sono tra i Framework Web e le librerie software?**

Le librerie software sono insiemi di funzioni che possono essere utilizzate per certi scopi, ogni chiamata fa qualche lavoro e poi rilascia il controllo al chiamante, ovvero il codice dell'utente è in controllo del flusso, mentre per i framework non è così, bensì il codice scritto dall'utente può essere inserito all'interno del framework, il quale è in controllo del flusso di esecuzione.

3. **Qual è il principio dell'inversione di controllo?**

L'IoC è una caratteristica molto importante dei framework: il controllo di flusso è inverso rispetto alla tradizionale programmazione imperativa, ovvero la responsabilità di controllare il flusso è spostata dallo sviluppatore al framework.

4. **Qual è la differenza tra framework opinionated e unopinionated?**

I framework opinionated sono rigidi in quanto le decisioni sono fatte dai creatori del framework, impongono modi specifici per fare determinate cose e danno meno spazio di manovra agli sviluppatori. Assicurano livelli alti di consistenza su una moltitudine di progetti e possono velocizzare lo sviluppo riducendo così la fatica di prendere decisioni, d'altro canto invece possono avere una curva d'apprendimento più ripida e potrebbero non essere abbastanza flessibili per certi progetti.
I framwork unopinionated invece non impongono alcun modo specifico di fare le cose.

5. **Parlami del framework Express.**

Express è un framework veloce, unopinionated e minimalista per Node.js, si installa attraverso il comando `npm install express`. Il routing (processo per il quale si determina come gestire una data richiesta) avviene attraverso l'oggetto app con il seguente comando:
`app.<METHOD>(<PATH>, callback)` dove il metodo e il percorso identificano un endpoint e la callback è una funzione da eseguire per gestire richieste su quel determinato endpoint.
I percorsi di indirizzamento possono essere stringhe, pattern di stringhe o espressioni regolari.
I percorsi possono anche contenere parametri dichiarati nel percorso usando `:nomeParametro`, i parametri catturati sono disponibili nell'oggetto `req.params.`
È possibile, inoltre, specificare più gestori di funzioni per un percorso, queste funzioni agiscono come una sorta di middleware, e lavorano con l'ausilio della callback next per far eseguire la prossima funzione in lista. È possibile anche creare percorsi concatenabili con `app.route()`.
Esiste anche il concetto di modularità per i percorsi importando il modulo quando serve.

6. Cosa succede nel modulo di routing se ci sono più match del percorso?

In questo caso si applica la prima regola che fa matching.

7. Come si fa render di un template con Express?

Il rendering di un template con Express è possibile attraverso `app.set("view engine", "pug")` e `app.get(<PATH>, callback)` e tramite gli oggetti `res.locals` (simile a localStorage) e `res.render.`

8. Cos'è il ciclo di richiesta-risposta in Express?

Il ciclo di richiesta e risposta in Express è una sequenza di passaggi che avvengono ogni volta che il server riceve una richiesta da un client e invia una risposta corrispondente. Inizialmente, l'applicazione Express viene inizializzata creando un'istanza del server. Quando una richiesta arriva al server, essa passa attraverso una serie di middleware, ognuno dei quali può manipolare la richiesta o la risposta in modi diversi. Successivamente, la richiesta viene instradata verso una specifica funzione di gestione, nota come "route handler", che contiene la logica per gestire la richiesta. Questa logica può coinvolgere l'accesso al database, l'elaborazione dei dati o altre operazioni necessarie per soddisfare la richiesta del client. Infine, una volta che la logica della richiesta è stata completata, il server invia una risposta al client. Questa risposta può essere un semplice messaggio di testo, un documento HTML, un file JSON o qualsiasi altro tipo di dati appropriato. Una volta inviata la risposta, il ciclo di richiesta e risposta si completa e il server torna in attesa di una nuova richiesta da gestire.

9. Cos'è un middleware?

Con il termine middleware si intende un software che funge da intermediario tra applicazioni, strumenti e database per fornire agli utenti servizi unificati. È il raccordo che collega tra loro piattaforme software e dispositivi diversi.

10. Cos'è un middleware in Express?

I middleware in Express sono funzioni che possono manipolare la richiesta corrente e l'oggetto di risposta o eseguire un qualsiasi codice, sono chiamati così perché sono eseguiti tra la ricezione della richiesta e l'invio della risposta. Vengono eseguiti nello stesso ordine in cui sono dichiarati e prendono in input due argomenti: `req` (l'oggetto della richiesta HTTP) e `next` (una callback che chiama il prossimo middleware nello stack).

11. Come si gestiscono gli errori in Express?

La gestione degli errori è possibile lanciando un errore durante l'esecuzione di un codice sincrono che verrà poi gestito dall'error handler di default di Express che stampa l'intero stack trace solo nella modalità sviluppo. Gli errori ritornati dalle funzioni asincrone invocate dai gestori dei percorsi e i middleware devono essere passati alla funzione `next().`

12. Quali sono i middleware principali di Express visti nel corso?

Abbiamo visto `express.static` che si utilizza per servire file statici, `express.urlencoded` che analizza il corpo delle richieste in arrivo con carichi urlcodificati, `express.json` che si comporta quasi come il precedente ma è progettato per analizzare corpi di richiesta JSON, `express-session` che si abilita attraverso `npm` e semplifica la gestione delle sessioni conservate dal server, `cookie-parser` che si abilità attraverso `npm` e analizza gli header cookie delle richieste in arrivo e popola `req.cookies` con un oggetto le cui proprietà sono i nomi dei cookie.

13. Cosa sono i messaggi flash e a che servono?

I messaggi flash sono strumenti che forniscono un feedback visivo di ciò che sta accadendo nella pagina. La parte difficile della gestione dei messaggi flash sta nel fatto che, dopo il reindirizzamento, inizia un nuovo ciclo di richiesta-risposta. In Express esiste il middleware `connect-flash` che si abilita attraverso `npm`.

14. Che modi di rappresentare i dati conosciamo generalmente?

Il primo modo è quello di avere gli oggetti che popolano l'heap (una regione di memoria dinamica utilizzata da un programma durante l'esecuzione) e un altro modo è quello di conservare i dati nelle tabelle RDBMS (Relational Database Management System).

15. Cos'è l'ORM?

L'Object-Relational Mapping è un modo per allineare le due rappresentazioni viste prima dove i cambiamenti agli oggetti di dominio devono essere mappati allo schema dell'RDBMS e i cambiamenti allo schema RDBMS potrebbero impattare il codice esistente. Le librerie ORM supportano gli sviluppatori nel mantenere le rappresentazioni basate sugli oggetti e quelle relazionali allineate, permettendo di lavorare con astrazioni di alto livello invece che con pure query SQL, aumentando la produttività riducendo il bisogno di codice ripetitivo e rendendo più semplice il cambio su differenti RDBMS. Tramite queste librerie è possibile definire in maniera dichiarativa come le tabelle del database vengono mappate in oggetti, è possibile creare ed aggiornare automaticamente lo schema RDBMS al bisogno ed eseguire facilmente le operazioni CRUD (Create, Read, Update, Delete), è possibile comporre ed eseguire query complesse in maniera dichiarativa, si possono gestire le transazioni e sono supportate strategie di caching.

16. Parlami di Sequelize.

Sequelize è un ORM per Node.js che supporta svariati database (Oracle, Postgres, MySQL, SQLite, ecc), è un API elegante basato su promesse, si installa tramite `npm install sequelize` e c'è bisogno di installare anche i driver per il RDBMS, ad esempio, `npm install sqlite3` per SQLite.

17. Fai un esempio di connessione a database con sequelize.

```
import { Sequelize } from "sequelize";
//create connection
const database = new Sequelize("sqlite:mydb.sqlite");
try {
    await database.authenticate();
    console.log('Connection has been established successfully.');
} catch (error) {
    console.error('Unable to connect to the database:', error);
}
```

Da notare che `await database.authenticate()` ritorna una promessa.

18. Cosa sono i modelli di un ORM?

I modelli in un ORM sono classi che estendono Model, ci sono due modi equivalenti per definire modelli: chiamando il metodo `define` sulla connessione al database o creando una classe che estende Model e poi chiamando il suo metodo `init`.
Quando definiamo un modello diciamo essenzialmente a Sequelize informazioni sui nostri dati, bisogna comunicare a Sequelize di allineare lo schema del database con i modelli che abbiamo definito, questo allineamento si ottiene attraverso il model synchronization, più precisamente

tramite il metodo `model.sync()` che è una funzione asincrona che ritorna una promessa.
È possibile avere tre modi per sincronizzare, il primo è `model.sync()` che crea una tabella se non esiste (e non fa niente se esiste), il secondo è `model.sync({force: true})` che ricrea la tabella e la droppa se esiste (quindi è distruttivo), e l'ultimo è `model.sync({alter: true})` che controlla lo stato della tabella e nel caso fosse necessario la cambia per matchare il modello.

19. Cos'è il meccanismo di pluralizzazione di Sequelize?

Quando definiamo un modello, Sequelize deduce un nome per la tabella dal nome del modello utilizzando un nome pluralizzato (es. User diventa Users, Child diventa Children).

20. In che modo è possibile creare entità?

Si possono creare entità in due modi:
Nel primo modo, creiamo un'istanza del modello e poi la salviamo nel database con il metodo `save()`:

```
let janet = new User({name: "Janet", age: 22 });
await janet.save();
```

Nel secondo modo invece, utilizziamo il metodo statico create() che crea l'istanza e la salva:

```
let riff = await User.create({name: "Riff Raff", age: 26 });
```

21. È possibile definire associazioni tra modelli, in che modo?

Sequelize supporta le associazioni uno-ad-uno, uno-a-molti e molti-a-molti attraverso i metodi:
`A.hasOne(B);A.belongsTo(B);A.hasMany(B);A.belongsToMany(B,{ through: 'C' });`
Sequelize crea di default una colonna chiamata ModelloDiRiferimento+ChiavePrimariaDiModelloDiRiferimento.

22. Quali altre operazioni sono possibili per la manipolazione dei dati?

Con Sequelize è possibile recuperare i dati ad esempio utilizzando la classe Op e i metodi `findAll()` e `findByPk(pk)` che non sono altro che un modo dichiarativo di scrivere query SQL, inoltre il metodo `save()` può anche effettuare un update dell'entità sul database se ad esempio prima sono stati settati valori con `setDataValue(…)` mentre il metodo `destroy()` cancella l'entità dal database.

23. Quali sono i pro e i contro degli ORM?

Le librerie ORM lavorano con oggetti e modelli invece di SQL (astrazione), c'è bisogno di meno replicazione di codice e inoltre la generazione degli schemi è automatica (produttività), è facile cambiare RDBMS (portabilità) ed è più leggibile rispetto al puro SQL, forzando a pattern di codice specifici e a determinate convenzioni (leggibilità e consistenza).
D'altro canto invece, ha una curva di apprendimento più ripida, alcune operazioni potrebbero essere più complesse (complessità), potrebbe essere difficile passare ad un altro ORM (vendor lock-in) e gli ORM potrebbero introdurre ad un sovraccarico per alcune operazioni, soprattutto quando viene utilizzato in maniera impropria (performance).

24. Com'è convenzionalmente organizzata una web app con Express?

Tra le principali cartelle di una web app con Express troviamo: views (contenente i tempate), routes (contenente la definizione dei Routers, questo codice è responsabile di indirizzare le richieste al controller appropriato), controllers (contenente il codice che implementa la logica di business per la gestione delle richieste), models (contenente la definizione dei modelli di dati e le connessioni al database), middleware (contenente i middleware personalizzati) e public

(contenente i file statici organizzati in cartelle).
Inoltre è buona prassi avere un file di configurazione (.env) contenente informazioni sensibili (chiavi, password, credenziali) e impostazioni di ambiente specifiche (URL di produzione/sviluppo del database, ecc). Questi tipi di file non dovrebbero avere più versioni ma piuttosto è possibile avere più versioni di una "dummy version". Per leggere il file .env su Node.js è possibile usare pacchetti come dotenv che fornisce l'oggetto `process.env` per poter utilizzare e leggere il contenuto del file di configurazione.

25. Quali sono le pecche delle web app tradizionali?

Le web app tradizionali non sono abbastanza flessibili, altri software intenzionati ad accedere ai dati devono scaricare le pagine HTML e analizzarle per estrarre i dati rilevanti, questo è un approccio non molto efficiente (vengono trasferite grandi quantità di dati di cui non si ha bisogno) e non è abbastanza robusto (qualsiasi cambiamento nelle pagine HTML potrebbe non far funzionare correttamente l'estrapolazione dei dati).

26. In che modo sono stati risolti i problemi relativi alle app tradizionali?

Il modo attuale è quello di separare le applicazioni web in due componenti: backend (che è responsabile dei dati e della logica, server-side) e frontend (responsabile per l'interfaccia grafica, client-side), queste due componenti comunicano attraverso Internet.

27. Cosa sono le APIs?

Le APIs (Application Programming Interfaces) sono modi per far comunicare programmi tra di loro, attualmente l'83% del traffico di rete è relativo a chiamate API. È possibile gestire la comunicazione su internet manualmente (definendo un protocollo su TCP/UDP e aprendo le socket, ecc) oppure con strumenti come CORBA, Java RMI, SOAP, o anche semplicemente utilizzando protocolli web (come HTTP).

28. Parlami di REST.

REST (Representational State Transfer) non è un protocollo standard ma è uno stile architetturale, ovvero una serie di regole e linee guida che definiscono come gli standard web dovrebbero essere usati nelle APIs. È basato su HTTP e URIs e fornisce un'interfaccia comune e consistente basata su un uso appropriato di HTTP: ci permette di interagire con le risorse che sono associate ad un unico URI ed esse corrispondono tipicamente a oggetti persistenti del dominio. I metodi HTTP dovrebbero essere usati per recuperare o manipolare le risorse. Le REST API dovrebbero essere stateless. Il trasferimento dei dati avviene utilizzando rappresentazioni appropriate come JSON, XML, YAML, ecc. Quando si sviluppa un'API è possibile definire risorse innestate (comunque rispettando la regola che ogni risorsa dovrebbe avere un solo URI) un esempio è `GET /seller/{id}/reviews` che permette di fare una lista di tutte le reviews fatte da un determinato venditore identificato tramite un id.
L'implementazione di una REST API è abbastanza semplice: basta rimanere in ascolto di richieste HTTP e gestirle, una volta che la richiesta viene gestita bisogna inviare una risposta HTTP, ma invece di rispondere con una rappresentazione HTML utilizziamo una rappresentazione più machine-friendly (JSON).

29. Qual è la differenza tra autenticazione e autorizzazione?

Con l'autenticazione vogliamo che solo gli utenti legittimi possano accedere alle risorse, mentre con l'autorizzazione vogliamo che alcuni utenti possano accedere solo a determinate risorse (es. non vogliamo che un impiegato possa modificare il suo salario).

30. Come rendiamo le REST API sicure?

L'idea che c'è dietro alla resa sicura delle REST API si fonda sul fatto che il client mandi una richiesta con un username ed una password all'API, l'API valida l'username e la password e genera un token, il token viene ritornato al client che deve passarlo all'API per ogni richiesta successiva che richiede la sua autenticazione e l'API verifica il token prima di ogni risposta.

31. Parlami di JWT.

JWT (JSON Web Token) è uno standard ampiamente adottato che permette di scambiare sicuramente richieste tra le due parti, è una stringa composta da tre parti e divise tra loro da un punto: la prima parte è l'header contenente informazioni su tipo di token, è codificato in Base64Url; la seconda parte è il payload, anch'esso codificato in Base64Url, queste prime due parti sono facilmente invertibili; la terza ed ultima parte è la firma che è ottenuta applicando una funzione di hashing crittografica alla stringa ottenuta concatenando l'header, il payload del token e una chiave segreta conosciuta solo dal server che genera il token.
Questa funzione di hashing crittografica è priva di collisioni e la stringa ottenuta viene calcolata facilmente ma non può essere invertita.

32. Parlami di OpenAPI.

OpenAPI è uno standard formale per descrivere le API HTTP: assicura una documentazione consistente (standardizzazione), puo' essere usato per generare automaticamente documenti comprensibili (documentazione), permette la generazione automatica di librerie del client e stub del server (generazione di codice), permette scoperte automatizzate (leggibile sia dalle macchine che dall'uomo) ed è lo standard industriale più ampiamente adottato. Le sue specifiche sono scritte come documenti di testo strutturati, ogni documento rappresenta un oggetto JSON in formato JSON o YAML. Questi file di specifiche descrivono la versione della specificazione OpenAPI (openapi), l'API (info) e gli endpoints (paths). Per ogni percorso c'è un metodo supportato e per ogni metodo ci sono i parametri di input attesi, il corpo della richiesta, le possibili risposte, ecc. I pacchetti utilizzati da noi sono swagger-jsdoc che genera automaticamente specifiche OpenAPI basate sulle annotazioni presenti nel codice sorgente (si installa con `npm install swagger-jsdoc`) e swagger-ui-express che fornisce documentazione generata automaticamente dell'interfaccia grafica Swagger da Express (si installa con `npm install swagger-ui-express`)

33. Parlami dell'approccio OpenAPI-driven development.

L'OpenAPI-driven development è un approccio che si basa sul lavorare con un'API definendo la sua specifica prima della scrittura di qualsivoglia codice effettivo. Questa modalità di lavoro promuove l'indipendenza tra i diversi team di sviluppo in un progetto.

34. Cosa sono i Transpilers e perché sono stati introdotti?

Nel momento in cui JavaScript è stato utilizzato per programmi sempre più complessi, si è sentito il bisogno di notare alcuni bug prima dell'esecuzione del codice, visto che l'IDE dava poco supporto e il codice iniziava a diventare sempre meno manutenibile e difficile da capire.
Dati questi problemi sono stati introdotti i Transpilers, strumenti software che traducono il codice sorgente scritto in un linguaggio di programmazione in un altro linguaggio di programmazione. Di solito, vengono utilizzati per convertire il codice scritto in un linguaggio in un altro linguaggio che è compatibile con un ambiente diverso.

35. Parlami di TypeScript

TypeScript è un sovrainsieme di JavaScript che aggiunge ad esso tra le varie cose la tipizzazione statica, l'enumerazione, i generics e le interfacce. Viene compilato in codice JavaScript e può essere installato come pacchetto npm attraverso il comando `npm install -g typescript` che rende il compilatore `tsc` disponibile globalmente. L'estensione di un file TypeScript è di solito `.ts`.

36. Cos'è il Downleveling?

Il Downleveling è il processo per il quale TypeScript riscrive il codice da versioni più nuove a versioni più vecchie di JavaScript, per mantenere un livello di retrocompatibilità (ad esempio con ES3). Quando si compila con tsc è possibile specificare nel target la versione che vogliamo usare, ad esempio `tsc -target es6 hello.ts` .

37. Come funziona la dichiarazione dei tipi in TypeScript?

In TypeScript i tipi primitivi più comuni sono string, number e boolean. È possibile dichiarare un tipo con lo statement `let nomeVariabile: nomeTipo`. Nel caso degli array lo statement diventa `let nomeArray: tipoArray[]` . Esiste un altro tipo che include qualsiasi tipo speciale: any, si può fare qualsiasi cosa con questo tipo di dato e il compilatore non farà storie. Gli oggetti sono definiti come `nomeOggetto: {nomeAttributo: tipo, nomeAttributo: tipo}.` Anche in TypeScript esiste il meccanismo di optional chaining con ?
In aggiunta ai tipi come string e number, è possibile specificare stringhe e numeri in tipi di posizioni, con i type literals, come ad esempio `let alignment: "center".`
Altri tipi da conoscere sono void (il tipo di ritorno di funzioni che non ritornano alcun valore), unknown (rappresenta qualsiasi valore plausibile, simile ad any ma più sicuro perché non permette alcuna operazione su di esso) e never (rappresenta i valori che non sono mai osservati).

38. Cosa sono le union types?

Le union types in TypeScript sono ottenute combinando due o più tipi usando il separatore | .
Lavorando con le union types è buona norma captare il tipo di dato con cui stiamo lavorando usando `typeof`, inoltre TypeScript può dedurre il tipo specifico per particolari rami.

39. Parlami dell'inferenza automatica di tipo in TypeScript.

In TypeScript esiste il meccanismo di inferenza automatica di tipo che permette di ricavare automaticamente i tipi dal contesto in cui si trovano. Quando non si specifica un tipo e TypeScript non riesce ad utilizzare questo meccanismo il compilatore setterà quella variabile ad any di deafult. Si può utilizzare il flag noImplicitAny al compilatore per segnalare qualsiasi any implicito come un errore.

40. Cos'è il meccanismo di Type alias?

Con il Type alias è possibile assegnare un nome specifico ad un tipo con la sintassi type nomeAlias = nomeTipo.

41. Cosa sono le interfacce in TypeScript?

Le interfacce in TypeScript sono un altro modo di nominare tipi di oggetti, ad esempio:

```
interface nomeInterfaccia {
     nomeAttributo: tipo;
     nomeAttributo: tipo;
}
```

È possibile estendere le interfacce utilizzando la parola chiave `extends`.

42. Qual è la differenza tra i Type alias e le interfacce?

I type alias non possono essere riaperti per aggiungere nuove proprietà, mentre le interfacce sono sempre estendibili. Per sviare al problema della poca estendibilità dei type alias si possono utilizzare gli intersection types che combinano più tipi in uno utilizzando il carattere &.
Un esempio pratico è il seguente:

```
type Pet = {
     name: string
}
type Bird = Pet & {
     flies: boolean
}
```

43. Qual è la differenza tra linguaggio tipizzato in maniera nominale o in maniera strutturale?

Tipizzazione per nome (o nominale): In questo tipo di tipizzazione, il tipo di un dato viene determinato in base al suo nome. Questo significa che due tipi con nomi diversi, anche se hanno la stessa struttura interna, sono considerati distinti. Ad esempio, se hai due tipi denominati Persona e Studente, entrambi con gli stessi campi come nome, cognome e età, in una tipizzazione per nome, questi sarebbero considerati due tipi distinti. Anche se la struttura interna è la stessa, il compilatore o l'interprete tratteranno Persona e Studente come tipi diversi.
Tipizzazione per struttura (o strutturale): In questo tipo di tipizzazione, il tipo di un dato viene determinato in base alla sua struttura interna, indipendentemente dal nome del tipo. Ciò significa che due tipi con la stessa struttura interna sono considerati compatibili e intercambiabili, anche se hanno nomi diversi. Ad esempio, se hai due tipi di dati che hanno gli stessi campi (nome, cognome, età), in una tipizzazione per struttura, questi sarebbero considerati dello stesso tipo, indipendentemente dai loro nomi. TypeScript ha una tipizzazione strutturale, mentre Java ha una tipizzazione nominale.

44. Cosa sono le Type assertions?

In TypeScript, la type assertion (o "type cast") è una funzionalità che consente ai programmatori di specificare esplicitamente il tipo di una variabile, quando il TypeScript compiler non è in grado di determinarlo in modo automatico. Un esempio è const nameInput = document.getElementById(“name”) as HTMLInputElement. Attenzione! La type assertion non ha alcun impatto sul valore effettivo della variabile a tempo di esecuzione; è solo un'indicazione per il TypeScript compiler sul tipo della variabile durante la fase di compilazione.

45. Parlami degli enums in TypeScript.

Gli enums in TypeScript permettono agli sviluppatori di definire un insieme di costanti:

```
enum Priority{
     Low,
     Medium,
     High
}
```

46. Parlami delle function type expressions in TypeScript.

In TypeScript le funzioni possono essere descritte utilizzando il meccanismo di function type expression: `type nomeFunzione = (a: tipo, b: tipo) => tipoRitornato.`

47. Parlami dei generics in TypeScript.

I generics in TypeScript sono un modo per rendere il codice riutilizzabile lavorando su una moltitudine di tipi contemporaneamente. Il costrutto è del tipo `function nomeFunzione<Type>(a: Type): Type {…} .`
Questo tipo di approccio non è la stessa cosa di utilizzare il tipo any: in questo caso preserviamo l'informazione sul tipo di input. E' possibile utilizzare più Type diversi elencandoli sempre tra < >, è possibile utilizzare array di tipo parametrico con `nomeArray: T[]` e anche classi di tipo parametrico con `class CustomCollection<T> { … }.`

48. A cosa serve strictNullChecks?

strictNullChecks è un flag che può essere usato per assicurare che i valori null o undefined siano gestiti esplicitamente.

49. Cosa sono i pre-/post-processori di CSS?

I pre-/post-processori di CSS sono strumenti essenziali nello sviluppo web moderno e permettono agli sviluppatori di creare stylesheets robusti, scalabili e ottimizzati in maniera efficiente.
I pre-processori sono compilatori speciali che possono genere codice CSS a partire da file scritti nella loro sintassi specifica, questa sintassi può essere vista come una estensione del CSS di base, introducendo nuove caratteristiche (ad esempio le variabili); un esempio è Sass.
I post-processori possono processare e trasformare automaticamente stylesheets esistenti utilizzati tipicamente per il downleveling, l'autoprefixing, ottimizzazione (bundling e minification), imporre convenzioni e rivelare gli errori; un esempio è PostCSS.

50. Parlami di Sass.

Sass è il pre-processore di CSS attualmente più popolare, si installa con `npm install -g sass`, utilizza due sintassi: la prima con estensione `.scss`, che è un sovrainsieme di CSS e la seconda con estensione `.sass`, basata sull'indentazione come Python. Supporta le variabili utilizzate come un modo per riutilizzare valori su tutti gli spreadsheets, quando compiliamo un file Sass le variabili sono sostituite dal loro valore, utilizzando il comando `sass file.scss file.css`.
Sass supporta il nesting delle regole CSS, rendendo il codice più leggibile, ma senza esagerare altrimenti si ottiene l'effetto contrario.
Un gruppo di dichiarazioni CSS/Sass possono essere riutilizzate apponendo prima la regola `@mixin`, che può essere inclusa utilizzando la regola `@include`, mixin può anche includere opzionalmente uno o più parametri.
Sass supporta anche la modularità includendo un file esterno .scss attraverso la regola `@use`.
I file Sass che iniziano per trattino basso si chiamano partials e sono utilizzati solo per includere altri file, in questo caso Sass non genera un codice CSS a sé stante, un esempio di utilizzo è quello di avere un file partial che includa tutte le variabili.
Sass supporta anche liste e mappe, inoltre è possibile avere regole di controllo di flusso (con regole del tipo `@if/@else, @each, @for, @while`).

51. Parlami dei "framework" CSS.

I framework CSS forniscono un insieme di stili a scopo generale che possiamo utilizzare così come sono fatti o possiamo modificarli ad hoc. Dobbiamo solo importare i file CSS nelle nostre pagine e usare le classi fornite dal framework, possiamo scaricare i file CSS e utilizzarli nel nostro server oppure possiamo utilizzare i Content Delivery Networks che hostano i file CSS per noi (convenienti e veloci ma non sicuri). La maggior parte dei framework CSS sono costruiti utilizzando Sass o simili, un modo per personalizzare il framework CSS è quello di cambiare le variabili di costruzione

in base alle nostre necessità, oppure un altro modo è quello di definire stili CSS personalizzati che vadano a fare l'override di alcuni degli stili del framework.
Uno dei framework più popolari è Bootstrap, possiamo importare il file principale di bootstrap Sass attraverso la regola `@import`.

52. Cos'è il Bundling?

Il Bundling permette di combinare più file CSS o JavaScript in un unico file, in questo modo i browser devono fare meno richieste HTTP per caricare la pagina, in quanto richieste aggiuntive potrebbero introdurre latenza, ma attenzione: l'ordine in cui ogni singolo file appare nel bundle ha importanza!

53. Parlami del Tree Shaking.

Il meccanismo di Tree Shaking permette di analizzare gli import e gli export per poter potare via tutto il codice inutilizzato dal bundle finale. In questo modo aumentiamo la performance visto che abbiamo un impatto significativo sulla grandezza del bundle.

54. Parlami della Minification.

Il meccanismo di Minification è il processo per il quale si ottimizza il trasferimento di file rimuovendo gli spazi bianchi non necessari e accorciando il nostro codice sorgente senza impattare la sua funzionalità. Avviene una notevole compressione dei dati e quindi risparmio di memoria.

55. Parlami di Vite.

Vite è uno strumento a supporto degli sviluppatori composto da due componenti chiave: un server di sviluppo con una serie di funzionalità avanzate per semplificare lo sviluppo e uno strumento di build per creare assets statici altamente ottimizzati, pronti per la produzione. Un progetto Vite si inizia con il comando `npm create vite@versione`. La sua impalcatura è formata da: dev che inizializza il server dev, qui Vite controlla i cambiamenti dei file sorgenti e fa un reload automatico della web app quando è necessario; build che genera un bundle ottimizzato, pronto per la produzione (qui i bundle finali hanno dei nomi particolari dove l'ultima parte del nome è un hash del loro contenuto attuale, in questo modo si risolvono problemi correlati alla cache dei browser) e preview che è un semplice server statico che fornisce i file generati durante la fase di build.

56. Che differenza c'è tra le multi-page web applications e le single-page web applications?

Nelle multi-page web application quando una nuova richiesta è mandata al server la nuova pagina è ritornata come risposta e mostrata dal browser (navigare su pagine differenti triggera sempre un full-reload della pagina web).
Una single-page web application reagisce in base all'input dell'utente aggiornando il contenuto della pagina corrente dinamicamente (invece di caricare una nuova pagina dal server), l'intera applicazione è caricata solo nella richiesta iniziale (la complessità del rendering delle view è passato dal server al codice del client).
Le single-page web application utilizzano il client-side routing andando ad aggiornare l'url del browser (router), andando a caricare e a renderizzare il contenuto appropriato sulla pagina web (view rendering) e interagendo con l'API History del browser per permettere agli utenti di utilizzare i tasti indietro e avanti del browser per poter navigare tra le diverse view (history management).

57. Cosa sono i components?

I components sono decomposizioni della UI, ogni component incapsula una funzionalità specifica e dovrebbe essere responsabile di renderizzare la sua UI (HTML) e di gestire il suo stato interno, le views che utilizzano un component non dovrebbero preoccuparsi di questi dettagli (modularity ed encapsulation); i component possono essere facilmente riutilizzati in più views (reusability).

58. Quali sono i vantaggi nell'utilizzo delle single-page web applications?

Le single-page web application offrono una user experience più dinamica, anche se il primo caricamento sarà più lento i seguenti saranno sempre più veloci e questo approccio permette di ridurre il carico di lavoro dal server in quanto solo i dati richiesti vengono recuperati e solo quando vengono effettivamente richiesti, inoltre non c'è bisogno di trasferire gli interi documenti HTML di continuo, ottenendo così una riduzione del carico e di larghezza di banda.

59. Quali sono gli svantaggi nell'utilizzo delle single-page web applications?

Nelle single-page web applications c'è bisogno di tener traccia dello stato e di avere l'interfaccia grafica sempre aggiornata in modo da aggiornare in maniera corretta le views dopo un qualsiasi cambio di stato, la manipolazione del DOM richiede la stesura di molti codici ripetitivi, il routing del client con routes parametriche potrebbero complicare le cose e il tutto non è il massimo dell'efficienza in termini di costruzione del documento.

60. Cosa si usava prima dell'API fetch introdotta i ES6?

Prima dell'introduzione dell'API fetch in ES6, le richieste di rete venivano inviate usando AJAX (Asynchronous JavaScript And XML) implementato usando l'interfaccia XMLHttpRequest.

61. Parlami di Angular.

Angular è un framework opinionato per il frontend sviluppato da Google, ha una forte concentrazione sugli strumenti per sviluppatori e produttività, include dependency injection, routing, ecc, riducendo così l'affaticamento decisionale e assicurando la consistenza su più progetti. Supporta la migrazione su versioni più nuove e impone TypeScript.
Si installa tramite il comando `npm install -g @angular/cli`, un nuovo progetto si crea tramite `ng new angular-app –create-application`. L'applicazione si avvia tramite il comando `ng serve`.
I componenti sono parti essenziali delle applicazioni Angular, ogni componente è responsabile nel definire la funzione e l'apparenza di un elemento (il suo codice, il layout HTML e lo stile).
Un componente può contenere altri componenti, basti pensare ad un'app Angular come un albero di componenti.
Nella pratica un componente non è altro che una classe TypeScript, annotati con il metadato passato come oggetto al decoratore @Component. I metadati specificano, tra le altre cose, il layout HTML e lo stile per il componente. Il campo selector è utilizzato per dire ad Angular dove renderizzare il componente, Angular creerà un'istanza di un componente per ogni elemento HTML che si abbina.
Un componente può anche essere generato attraverso il comando `ng generate component nome_componente`, in questo modo Angular genera il codice di base per noi (il filme HTML, i file TypeScript e il file di stile).
È possibile comporre i componenti aggiungendoli al campo `imports: [nome_componente].`
Angular supporta il controllo di flusso nei template attraverso i comandi `@if/@else` e `@for` (con la direttiva `track` obbligatoria per la performance).

62. Come si gestiscono gli eventi in Angular?

Gli eventi in Angular sono vincolati agli handlers utilizzando la sintassi con le parentesi, ad esempio: `<button (click)=”increment()”>Click me</button>`, dove increment() è un metodo del componente.

63. Cosa sono i componenti standalone?

I componenti dove il metadato standalone è settato a true sono componenti che possono importare direttamente altri componenti o direttive utilizzate nei template.

64. Come funziona la comunicazione tra componenti?

I componenti che hanno bisogno di scambiare dati con altri componenti possono utilizzare i decoratori @Input() e @Output().
Nel componente figlio, le proprietà che dovrebbero essere passate dal padre sono decorate utilizzando @Input(). Il componente padre potrà poi passare dati al figlio settando una proprietà sull’elemento HTML, potremmo avere ad esempio `<app-counter buttonText=”CLICK HERE”/>` con app-counter child selector (all’interno del template del padre), buttonText come target (ovvero la proprietà @Input() dal figlio) e “CLICK HERE” come sorgente (il valore dal padre).
Per l’output bisogna usare il decoratore @Output() su una proprietà della classe ed assegnare ad esso un valore di tipo EventEmitter, il componente figlio invoca il metodo emit() sulla proprietà @Output per inviare l’evento sul canale, in questo modo il componente padre può reagire agli eventi sul canale di comunicazione andando a definire un gestore di eventi, ad esempio `<app-counter (counterClickedTenTimes)=addClicks($event)/>` con app-counter child selector (all’interno del template del padre), counterClickedTenTimes come evento (ovvero la proprietà @Output() dal figlio) e addClicks($event) come gestore dell’evento (sul componente padre).

65. Che differenza c’è tra le espressioni con la template interpolation {{ }} e il property binding [] ?

Le espressioni all’interno di {{ }} nei template Angular sono valutate e il risultato viene renderizzato nel template (ad esempio `<img src={{imageUrl}}`), il property binding invece permette di settare dinamicamente valori per attributi di elementi HTML e componenti all’interno di template, vincolando questi valori alle proprietà del componente corrente (ad esempio `<img [src]=”imageUrl”>`).
La differenza risiede essenzialmente nel fatto che la template interpolation valuta l’espressione data e converte il risultato, nel momento in cui bisogna passare un oggetto o un array come una proprietà dell’elemento bisogna per forza utilizzare il property binding.

66. Come funziona il routing in Angular?

Il routing in Angular abilita la navigazione tra più viste (interpretando l’URL come una istruzione per cambiare la vista). Nell’entry point `./src/main.ts` ci occupiamo di caricare il componente radice, facendo così forniamo un oggetto di configurazione definito in `./app/app.config.ts`, questo oggetto `app.config` specifica che l’applicazione utilizza il Routing.
Il `./src/app.config.ts` esporta un oggetto `appConfig` di tipo `ApplicationConfig`, la funzione `provideRouter()` prende in input una descrizione delle route dell’applicazione di tipo Routes, definito in `./src/app.routes.ts` e setta i providers necessari per far sì che il modulo Router funzioni. L’oggetto `Routes` è definito in `./src/app.routes.ts`.
Dopo aver fatto tutti questi passaggi, possiamo usare il componente `RouterOutlet` all’interno dei nostri template. Questo componente analizza l’URL corrente e renderizza il componente associato al path che fa match.

67. Come funzionano i link con i Router Angular?

Se usassimo i link tradizionali nei nostri template finiremmo per rendere vano l'obiettivo delle single-page web application andando ad effettuare ogni volta una nuova richiesta HTTP.
Per navigare in maniera appropriata tra le viste, bisogna utilizzare la direttiva `RouterLink` sugli elementi che triggerano la navigazione.
Alcune volte vogliamo che lo stile del link attivo su cui stiamo sia diverso rispetto a quelli non attivi, per ricordarci quale pagina stiamo navigando, la direttiva `RouterLinkActive` permette di specificare una lista di classi CSS che dovrebbero essere applicate ai RouterLink corrispondenti al Route che si sta navigando.

68. In che modo posso creare forms in Angular?

In Angular posso creare forms in due modi: template-driven forms (le associazioni tra i dati dei Component e i form sono definiti implicitamente) e reactive forms (le associazioni tra i dati dei Component e form sono definiti esplicitamente nel componente). Nel primo caso si utilizza la notazione "banana in a box" che rappresenta un tipo di binding bidirezionale: property binding ed event binding; la direttiva [(ngModel)]="pass" viene utilizzata per specificare che il valore del form è legato alla poprietà, in questo caso pass, del componente.

69. Che differenza c'è tra i Component e i Services in Angular?

I Component in Angular dovrebbero essere solo responsabili nel definire la user experience: quindi template, stili, e la logica che media tra le viste e la logica dell'applicazione. I Services d'altro canto sono classi che forniscono funzionalità che possono essere usate in più parti dell'applicazione, essi non sono associati direttamente alla user experience (solo la logica che può essere invocata al bisogno).
I Services sono ottimi per scopi come il recupero dei dati dalle REST API, fare logging, validare dati, ecc. Il decoratore `@Injectable` si utilizza per marcare il Service come qualcosa che può essere iniettato all'interno di un component come una dipendenza. Le dipendenze possono essere istanziate utilizzando il metodo `inject()` oppure dichiarando le dipendenze nei costruttori della classe.

70. Come funziona il Dependency Injection in Angular?

Angular fa leva su pattern di Dependency Injection per rendere più semplice per il codice dell'applicazione ottenere istanze delle sue dipendenze (ad esempio i Services di cui ha bisogno). L'obiettivo della Dependency Injection è quello di separare la preoccupazione del costruire le dipendenze ed usarle, migliorando il disaccoppiamento. Ci sono due ruoli: il dependency consumer (il codice che scriviamo) e il dependency provider (solitamente gestito da Angular).

71. Parlami del Route Guarding.

I Route Guards servono a gestire il Routing per utenti autorizzati, sono funzioni invocate da Angular per determinare quale utente ha il permesso di fare certe azioni (canActivate, canMatch, canLoad, canDeactivate) su un determinato percorso. Sono specificate come proprietà degli oggetti Route. Più Guards possono essere specificate per un Route, Angular le esegue tutte in parallelo e l'azione può essere performata solo se tutte le Guards ritornano true.
Una Guard si crea con il comando `ng generate guard nome_guard`. Le funzioni Guard possono ritornare un valore booleano o un oggetto UrlTree: true significa che la guard è soddisfatta e la navigazione può procedere, false significa che la guard non è soddisfatta quindi l'azione non può essere performata, un oggetto UrlTree significa che il Router dovrebbe ridirezionare a quell'URL

ritornato. Quando più guards sono settate, si dovrebbe sempre ritornare un UrlTree per ridirezionare piuttosto che utilizzare il metodo Router.navigate() direttamente.

72. In che modo è possibile gestire le richieste HTTP in Angular?

Un modo per performare richieste HTTP in Angular è attraverso l'API implementata nella classe service HttpClient, dopo averlo configurato utilizzando la dependency injection, può essere istanziato come una dipendenza nei nostri services e component (con inject() o nel costruttore della classe). Esempio:

```
export class CatFactsComponent {
     constructor(private http: HttpClient){}
     catFact: string = "";
          loadCatFact(){
               let url = "https://cat-fact.herokuapp.com/facts/random";
               this.http.get(url).subscribe(data => {
                   this.catFact = data.text;
           });
      }
}
```

HttpClient fornisce metodi che corrispondono ai diversi metodi utilizzati dal protocollo HTTP per fare le richieste, ogni metodo ritorna un Observable dalla libreria RxJS, questi Observable sono utilizzati da Angular per gestire molte operazioni asincrone.

73. Che differenza c'è tra gli observables e le promise?

Gli observable possono emettere in maniera asincrona più valori col tempo, mentre le promise possono risolvere (o rigettare) solo una volta, ovvero emettono in maniera asincrona un solo valore (o un errore).

74. Parlami degli Interceptors.

È possibile personalizzare le caratteristiche di HttpClient quando configuriamo i suoi dependency injection providers: si può usare il metodo `withFetch()` per usare l'API Fetch invece di XMLHttpRequest, possiamo anche configurare gli Interceptors che sono una sorta di middleware supportati da HttpClient, che si applicano alle richieste in uscita: non sono altro che funzioni che prendono in input la richiesta corrente in uscita HttpRequest e la funzione next che rappresenta il passo successivo nella catena di interceptors.
Gli Interceptors possono essere dunque usati per aggiungere gli header di autenticazione alle richieste (come i JWT tokens), conservare le risposte per un certo periodo di tempo e per registrare le richieste in uscita.

75. Parlami del change detection in Angular.

Quando una proprietà di un componente cambia, il suo template è aggiornato automaticamente da Angular e aggiornare automaticamente le pagine solo se necessario in maniera efficiente è un punto cardine delle single-page web applications. In Angular, quando succede qualcosa nell'applicazione (ad esempio un evento DOM) il meccanismo di change detection si attiva, e quando ciò accade, Angular naviga su tutti i componenti nella gerarchia dei componenti e per ognuno di essi controlla se il suo stato è cambiato e se lo stato influenzi la vista, se così accade allora la porzione del DOM corrispondente al template del componente è aggiornata.

76. Cosa sono i signals in Angular?

I Signals tracciano come e dove viene utilizzato lo stato in tutta l'applicazione, permettendo ad Angular di ottimizzare in seguito gli aggiornamenti del rendering. Sono come delle variabili che possono anche notificare un qualsiasi cliente quando il loro valore cambia. Sono un modo di ottenere una programmazione reattiva e permettono ad Angular di evitare controlli inutili sul change detection.
I signals possono essere writeable o read-only, i primi si creano utilizzando la funzione `signal(),` passando un valore iniziale ad essa. I valori possono essere aggiornati utilizzando i metodi `.set()` per settare un nuovo valore o il metodo `.update()` per calcolare un nuovo valore a partire da quello precedente.
I segnali calcolati sono segnali read-only che derivano il proprio valore dai valori degli altri segnali, si definiscono utilizzando la funzione `computed()`, Angular li mantiene aggiornati automaticamente in una modalità reattiva (.set() e .update() qui non si possono utilizzare).
Un effect è una funzione che si esegue ogni qual volta che un segnale cambia, esse vengono eseguite almeno una volta e quando si eseguono tengono traccia di qualunque segnale che leggono, non appena uno qualsiasi di questi segnali cambia, la funzione effect viene eseguita nuovamente.

77. Su che livelli è possibile gestire la sicurezza delle web app?

È possibile gestire la sicurezza delle web app su livello rete (con le comunicazioni tra l'app e internet: si focalizza sulla crittografia dei dati, filtraggio di richieste HTTP malevole con firewalls; più specificamente assicura la protezione dei canali di comunicazione e l'infrastruttura, protegge contro gli attacchi che prendono di mira il livello di rete, come lo sniffing o il man in the middle) oppure su livello applicazione (con misure di sicurezza implementate all'interno della web app: si focalizza sull'autenticazione e l'autorizzazione, validazione di input per assicurarsi l'integrità dei dati, gestione della sessione, assicurarsi la riservatezza dei dati; più specificamente mitiga i rischi associati con l'applicazione stessa dove gli attacchi potrebbero essere nascosti all'interno della richiesta HTTP apparentemente valida così da sfruttare le vulnerabilità nel modo in cui l'app è implementata in maniera da raggiungere obiettivi malevoli).

78. Cos'è OWASP?

OWASP è un progetto no profit volto a collezionare dati vulnerabili, inoltre pubblica report periodici sulle classi di vulnerabilità più frequenti.

79. Quali sono i tipi più comuni di vulnerabilità in app?

I tipi più comuni di vulnerabilità sono: Cross-site Scripting (XSS), Cross-site Request Forgery (XSRF, CSRF), SQL Injections, Session Attacks/Hijacking e Attacks on data integrity tramite la insufficiente validazione di input.

80. Parlami del Cross-site Scripting.

Il Cross-site Scripting punta ad iniettare codice malizioso dal lato del client in siti non affidabili, colui che effettua l'attacco sfrutta i difetti nella web app (come input non affidabili) per inviare codice client-side malevolo agli altri utenti. Questo processo è molto pericoloso in quanto è possibile accedere e manipolare i cookies, accedere al localStorage e al sessionStorage, manipolare la pagina web e performare azioni illecite. Per evitare questi attacchi bisogna effettuare validazioni corrette dell'input o escaping.
Non bisognerebbe affidarsi ad approcci di filtering a meno che non sia strettamente necessario, il modo migliore per affrontare il problema è di utilizzare in maniera corretta l'escaping per gli input

quando li inseriamo nelle pagine web, come convertire simboli pericolosi in innocue entità HTMLEntities oppure usare soluzioni o librerie dedicate come sanitize per Node.js.
Angular "sanitizza" automaticamente tutti i dati non affidabili che sono legati o interpolati in un template; di default tutti i dati non sono affidabili, è possibile settare un dato come affidabile utilizzando DomSanitizer.

81. Parlami del Cross-site Request Forgery.

Questi tipi di attacchi mirano a forzare un utente finale ad eseguire azioni non intenzionali (trasferimento di fondi, cambiamento dell'email associata a quell'account) su una applicazione web su cui sono autenticati e se la vittima è un utente con i permessi di amministratore l'attacco può risultare nella compromissione dell'intera web app. Può avvenire ad esempio tramite form nascosti.
Il miglior modo di evitare questo tipo di attacchi è quello di utilizzare un approccio basato su token CSRF che viene salvato nella sessione ed incluso in un campo nascosto nei form sensibili, ogni qualvolta c'è un invio del form, il token viene trasferito di nuovo alla web app che verifica se il token ricevuto è uguale a quello generato e conservato nella sessione.
In Express si usano molti middleware come tiny-csrf; Angular include contromisure CSRF.

82. Parlami di SQL Injection.

Consiste nell'iniettare codice malevolo che proviene da input utente non "igienizzato" in query di database, questi attacchi possono risultare in fuoriuscita di dati privati, modifica, creazione, aggiornamento o cancellazione dei dati del database, violazioni di controllo di accesso, e in alcuni casi anche l'esecuzione di codice arbitrario sul server del database.
Un modo per prevenire questo attacco è quello di usare statements preparati con query parametrizzate e usare procedure salvate.

83. Parlami del Session Hijacking.

È un tipo di attacco che mira a compromettere il session token, di solito performato rubando il token valido o facendo una predizione di esso.

84. Parlami della insufficiente validazione di input.

La validazione dell'input lato client può essere totalmente evitata dagli hacker che possono accedere e manipolare il codice dal lato client o generare direttamente richieste HTTP.
Non si dovrebbe mai tralasciare la validazione degli input lato client, pur esistendo la validazione lato server e viceversa!

85. In che modo i browser ci danno sicurezza?

I browser moderni rafforzano un modello di sicurezza rigoroso per migliorare la sicurezza e bloccare alcuni tipi di attacchi, uno di questi meccanismi è la policy same-origin.
Di default, un documento o uno script ha permesso di interagire solo con risorse dalla stessa origine, ovvero l'url ha lo stesso protocollo, porta e host.
Cross-origin resource sharing (CORS) è un meccanismo che permette al server di "uscire" dalla policy same-origin e specifica che il suo contenuto può essere raggiunto anche da altre origini oltre che dalla sua. I server possono includere headers specifici nelle risposte per dichiarare che i dati possono essere raggiunti da origini specifiche (`Access-Control-Allow-Origin: *` significa che qualsiasi origine ha i permessi per accedere).

86. Come posso assicurarmi di scrivere web app di alta qualità?

Il testing è fondamentale per catturare quanti più bug possibili prima che raggiungano la fase di produzione (dove possono costare molti soldi). La verifica può essere svolta in due modi: verifica dinamica (tramite l'esecuzione di programmi, software testing) e la verifica statica (che non coinvolge l'esecuzione di programmi, come la review del codice, analisi statiche o verifiche automatiche con model checking).

87. Parlami del software testing.

Un test software è una sequenza di azioni progettate per valutare un aspetto o una funzionalità particolare del software. Un test di solito consiste in: assicurare che le precondizioni sono soddisfatte (arrange), eseguire una o più azioni (act) e controllare che il software under test si usa comportato come richiesto (assert).

88. Su che livelli può essere svolto il testing?

Il testing può essere svolto su livelli differenti: unit testing (si testa una singola unità che può essere ad esempio una classe o un metodo; in questo caso bisogna trattare con codice asincrono e la dipendenza da servizi esterni), integration testing (si controlla se diverse unità lavorano insieme come richiesto), system testing (mira l'intero sistema), end-to-end testing (mira il software under test dal punto di vista dei suoi utenti finali a cui è destinato; in questo caso bisogna trattare con la fragilità e la flakiness). Nell'ordine in cui li ho scritto vanno dal più veloce ed economico (più vicino al codice), al più lento e costoso (più vicino agli utenti finali).

89. Come puoi effettuare testing su Angular?

Di default il codice di test è scritto nei file .spec, su Angular il testing è reso possibile tramite Jasmine. I file stub spec sono generati di default da Angular e di solito vengono piazzati nella stessa cartella del componente o servizio che bisogna testare.
Gli strumenti principali sono: `describe()` che crea un insieme di test correlati, `TestBed` fornisce metodi per creare componenti e servizi nei test di unità, `it()` definisce un singolo test.
Il comando ng test compila l'applicazione e poi lancia il Karma Test Runner che raccoglie qualsiasi file spec e lo esegue.

90. Come funziona il testing in isolation?

Nel testing in isolation il risultato del test non dovrebbe dipendere da unità di codice esterne (come le dipendenze). La soluzione è quello di usare "doppioni" dei test (Test Doubles) che sono rimpiazzamenti per gli oggetti di produzione che sono di solito usati nei test: a supporto di questo approccio è possibile utilizzare il meccanismo di dependency injection, così facendo possiamo configurare quale injector risolve dipendenze utilizzando i Test Duoubles invece degli oggetti reali. Jasmine utilizza il concetto di Spies dove una Spy è un test che può "imitare" qualsiasi funzione.

91. Come si controlla il code coverage?

È possibile controllare il code coverage attraverso il comando `ng test –no-watch –code-coverage`.

92. Parlami dell'end-to-end testing.

L'obiettivo dell'end-to-end testing è quello di testare l'intero sistema dal punto di vista dei suoi utenti finali dove ogni test replica un flusso di utilizzo realistico e dove è possibile interagire con le interfacce esterne del sistema (proprio come fanno gli utenti finali).
Quando la web app è una REST API, gli endpoints REST sono le interfacce esterne, gli utilizzatori

finali sono gli altri programmi che mandano le richieste HTTP e i test E2E dovrebbero mandare le richieste HTTP ed accertarsi che le risposte sono corrette rispetto alla specifica.
Quando la web app è una app tradizionale o una single-page web application, le pagine web sono le interfacce esterne, gli utilizzatori finali sono gli umani che usano il browser web e i test E2E dovrebbero interagire con le pagine web e verificare che cambiano correttamente come risultato delle interazioni.
I web test E2E dovrebbero essere eseguiti in isolation, in modo da non avere fallimenti a catena, da non dare importanza all'ordine dell'esecuzione dei teste in modo che ogni test possa essere eseguito in maniera indipendente.

93. Quali sono gli step chiave del testing web in E2E?

Il testing E2E segue i seguenti passi:

```
execute(insieme di test):
   foreach test scenario nell'insieme di test:
      foreach step in test scenario:
         select l'elemento con cui interagire (bottone, campo di testo…)
         interact con esso
```

94. Che differenza c'è tra test E2E manuale e automatizzato?

Nel test E2E manuale i test umani hanno uno script con una lista dettagliata degli step da seguire per replicare gli scenari utente, seguono e replicano lo script step by step facendo un report di qualsiasi errore; in questo caso non c'è bisogno di conoscenza di programmazione, è un'attività che richiede tempo, tediosa ed incline ad errori, inoltre non è facilmente scalabile.
Nel test E2E automatizzato i tester sviluppano software test che possono automaticamente simulare gli scenari utente, questi test possono essere rieseguiti più volte; in questo caso è richiesta la conoscenza di programmazione e di testing, c'è un costo iniziale più alto ma di seguito i test possono essere rieseguiti più volte con uno sforzo addizionale minimo, il codice di test ha bisogno di manutenzione e potrebbe essere fragile o flaky.
Nel secondo caso si fa leva su librerie dedicate al controllo remoto di browser web per navigare sugli URL e per localizzare e interagire con gli elementi della pagina web.

95. In che modo è possibile selezionare elementi nella pagina web?

Ci sono tre approcci principali: coordinate assolute (identifica l'elemento in base alle sue coordinate; sono i più fragili), localizzatori basati sulla visuale (identifica l'elemento basato sulla sua apparenza, utilizzando algoritmi di associazione per immagine; sono robusti ai cambi di layout ma fragili ai cambiamenti di apparenza), e localizzatori basati su layout (identifica l'elemento basato sulle proprietà del layout, come l'utilizzo dei selettori CSS; sono robusti per i cambiamenti visuali ma sono più fragili ai cambiamenti dei layout, nonostante ciò sono i più utilizzati).

96. Parlami della fragilità del web test E2E.

Anche i cambiamenti più insignificanti della pagina web o dell'ambiente del test possono "rompere" i localizzatori, come risultato i test diventano incapaci di interagire con gli elementi corretti, in questo caso c'è una manutenzione che richiede più tempo per riparare i localizzatori rotti. Buoni localizzatori non dovrebbero essere né troppo generici né troppo specifici.

97. Parlami della flakiness del web test E2E.

I web test E2E tendono a comportarsi in maniera inconsistente in quando lo stesso test può superare con successo o fallire in maniera intermittente sotto le stesse condizioni, questa imprevedibilità è causata dal non determinismo correlato ai tempi di caricamento o al prelievo dei dati. Questo aspetto può essere mitigato usando strategie di attesa appropriate.

98. Parlami di Playwright.

Playwright è un framework cross-browser, cross-platform e open-source per la browser automation e il testing sviluppato da Microsoft. Utilizza un approccio event-driven e asincrono ed è scritto in TypeScript. Si installa tramite il comando `npm init playwright@versione`. I test vengono definiti nella funzione `test()` che prende in input un titolo e una funzione di test asincrona contenente il codice del test. Fornisce APIs per controllare in maniera remota i browser web, localizzare gli elementi nella pagina web ed interagire con essi. Il test si esegue con il comando `npx playwright test` ed è anche possibile visualizzare un report tramite il comando `npx playwright show-report`.
Playwright ottiene l'isolation iniziando da zero, dove ogni test viene eseguito in un ambiente pulito e non influenzato da altri test, utilizzando un nuovo `BrowserContext` per ogni test.
L'ambiente per ogni test è definito utilizzando `Test Fixtures`.

99. Quali sono gli aspetti fondamentali della scrittura di test E2E con Playwright?

Le tre parti cruciali nello scrivere test automatizzati E2E con Playwright sono: localizzare gli elementi (`.getByText(), .getByLabel(), …`), performare le azioni (`.clear(), .fill(), .click(), …`) e affermare lo stato rispetto alle aspettative (`expect(…).toBeDefined()/toBeEmpty()/toBeVisible()/…`).

100. Parlami del Capture & Replay.

Questo tipo di approccio permette di generare automaticamente codici di test E2E registrando le interazioni con l'utente, questo codice però potrebbe essere non ottimale ed essere soggetto a fragilità e flakiness.