***Introduzione***

L’utente interagisce con l’applicazione tramite i propri sensi

* Fotorecettori per la ricezione dei colori (il blu è poco riconoscibile)
* Luminanza è il rapporto tra la luce emessa e l’area di emissione
* Il contrasto è la differenza tra la luce e il suo sfondo
* Per migliorare l’acuità visiva (capacità di percepire i dettagli) basta avere una alta luminanza (scritte scura su sfondo bianco)
* Se frequenza di refresh non è sufficientemente alta si percepisce uno sfarfallio

***Interazione***

in sistema interattivo l’utente cerca di raggiungere un obiettivo (goal) in un determinato dominio applicativo (domain)

* Dominio - area dove l’utente lavora
* Intenzione (intention) – azione per raggiungere obiettivo
* Attività (task) – operazione che consente di manipolare i concetti di un dominio
* Obiettivo – output desiderato

**Esecution-evaluation cycle (Norman)**

Stabilire il goal 🡪 formulare l’intenzione 🡪 specificare la sequenza di azione 🡪 eseguire l’azione 🡪 percepire lo stato di sistema 🡪 interpretare lo stato 🡪 valutare lo stato rispetto obbiettivi e intenzioni

Problemi

* Utente e sistema linguaggi diversi
  + Task language 🡪 la parte di computazione dell’utente
  + Core language 🡪 la parte di computazione del sistema
* Golfo d’esecuzione
  + Differenza tra il task language e core language (es. interfaccia grafica minimizza)
* Golfo di valutazione
  + Se l’utente riesce a valutare velocemente lo stato del sistema allora la valutazione è piccola

Errori Utente

* Slips 🡪 l’utente preme sul pulsante sbagliato (golfo d’esecuzione)
* Mistakes 🡪 un utente che clicca sul bottone della lente pensando che sia “cerca” .invece è zoom (valutazione)

Soluzioni

* Slips 🡪 Maggior spazio tra i bottoni
* Mistakes 🡪 riprogettare l’interfaccia

**Legge di Jakob (sulla Internet User Experience) -** Utenti vogliono la stessa funzionalità che trovano negli altri siti

**8 regole d’oro sul design delle interfacce (Shneiderman)**

Consistenza 🡪 Abilitare scorciatoie 🡪 Offrire feedback 🡪 progettare conversazioni con una conclusione🡪 gestione e prevenzione di errori 🡪 reversibilità delle azioni 🡪 punto centrale di controllo 🡪 ridurre carico memoria a breve termine

**7 principi dell’Interaction Design (secondo Norman)**

Informazioni Visibili e facili da memorizzare 🡪 Semplificare la struttura dei task 🡪 Effetto di un azione visibile 🡪 costruire con chiarezza le associazioni (quindi le intenzioni dell’utente devono riflettersi nel sistema) 🡪 sfruttare i vincoli (solo azioni corrette nel modo corretto) 🡪 progettare tenendo in mente gli errori 🡪 usare gli standard quando non funziona qualcosa (non reinventare la ruota)

**Stili di interazione**

* Cmd line(Utenti già abituati /Adatto Accesso remoto /Apprendimento Curva ripida)
* Interfaccie menu (Punta al riconoscimento/Ordinate secondo una logica/Gerarchico)
* linguaggi naturali (Fraintendibile/Basato su grammatiche ( difficile)/Linguaggio libero (problemi privacy))
* domande/risposte - query (es. Google Search/Facile da apprendere/Funzionalità limitata/Difficoltà in linguaggi specifici (es SQL))
* form e fogli di calcolo(Facile da apprendere/Necessitano validazione/Input e output sono rappresentati allo stesso modo)
* WIMP (windows, icons, menus, pointer) (Es windows)
* punta e clicca (browser)(Usato nelle pagine web/Connesso con interfacce WIMP/Usato nei videogiochi)
* Interfaccie 3D(Usato anche nelle interfacce WIMP/Videogiochi, interfacce AR, VR/Non facili da usare)

**Legge di Fitts 🡪** Il tempo necessario ad acquisire un bersaglio è funzione della distanza e delle dimensioni del bersaglio

**Legge di Hicks 🡪** Il tempo necessario a prendere una decisione cresce con il numero e la complessità delle scelte

**Eventi**

* Event Capturing
  + In questa prima fase l’evento si propaga dagli elementi parent a tutti i discendenti. Per default questa fase non avviene
* Event bubbling
  + In questa fase l’evento si propaga dal target verso i nodi esterni. Permette event delegation
* Event Delegation
  + permette di aggiungere lo stesso listener a più elementi che condividono un antenato
* Aggiunta di listener (da non usare)
  + Si basa su uso di variabili globali
  + <div onevent=”doSomething()”> </div>
  + target.onclick = doSomething; // solo se necessario
* Come aggiungerli o rimuoverli?
  + target.addEventListener(tipo, listener[, opzioni]); // IE > 11
  + target.addEventListener(tipo, listener[, useCapture]); // IE > 9
  + target.removeEventListener(tipo, listener[, opzioni]); // IE > 11
  + target.removeEventListener(tipo, listener[, useCapture]); // IE > 9

***Elementi e principi di design***

Gli elementi di design sono i mattoni elementari che possono essere usati per creare il design di una interfaccia

Hue (Colore, tonalità)

* È il colore, hue in hsl da 0 a 360. 0 è rosso, 120 è verde, 240 è blu.
* Il significato delle culture
* Scelta dei colori tramite color wheel (ricordati primario e complementare)

Value (Brightness, luminosità, valore, value)🡪Varia aggiungendo bianco (tinte pastello) o nero (tinte più serie/pesanti)

Texture (Trama) 🡪(Poco usata si predilige flat designPuò dare sensazioni quasi tattili)

Shape (Forma) 🡪 (Forma delimitata da linee, colori, tonalità (logo google esempio)/positive o negative/forme ripetute senso di unità/coesione)

Form (struttura)🡪(L’insieme di forme del design e le relazioni/Elementi relazionati tra di loro)

Spazio (Space, white space o negative space)

* Inteso come spazio negativo, non occupato da altri elementi o forme
* Lo spazio è fondamentale per ogni design: l’occhio deve avere spazio per riposare fra un elemento e l’altro

Line (Linea)

* Utilizzate per marcare confini/separare elementi (danno ordine)
* Non necessariamente linee fisiche(dritte o curve) ma anche colori

Type (Carattere)

* Font utilizzato, l’interlinea (line-height) influenza leggibilità default 1.2
* Font-size privilegaiare misure relative alle dimensioni di default (rem)
* Per leggibilità è meglio sans-serif

**Principi di design**

Contrast 🡪 per dare interesse nelle varie aree (tramite colori, luminosità, direzioni, linee,..)

Emphasis 🡪 dare l’attenzione in una singola area, es con dimensioni font, spazii

Balance 🡪 allineare tutto il possibile, la disposizione (privilegiare simmetrie e asimmetrie) e usare grid

Unity (harmony) 🡪 disposizione piacevole delle parti (colori, forme e strutture consistenti)

Pattern 🡪 usare ripetizioni di elementi per mantere organizzato il design

Movement 🡪animazioni e movimento per attirare l’attenzione

Rhythm and repetition 🡪 suddividere per dare un ritmo visivo nel design (es. tabelle)

Proportion 🡪 dare delle proporzioni in base alla relazione

Simplicity 🡪 domandarsi se ogni elemento è essenziale

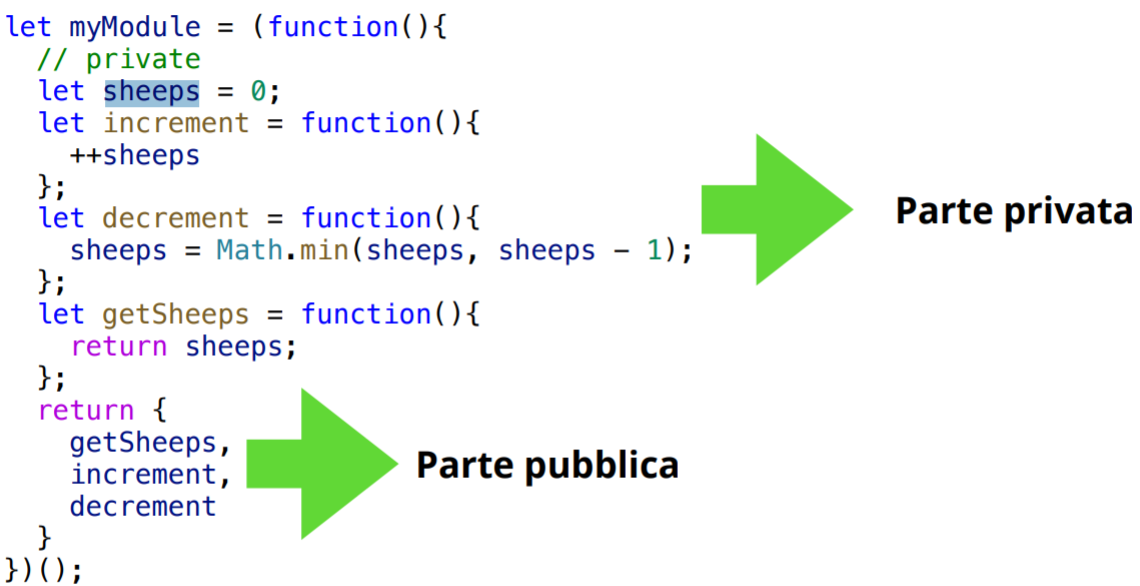
Gradation 🡪 ogni cambiamento devono essere facilmente osservabili e guidare l’occhio

**SASS** Css + variabili, ereditarietà, calcoli, funzioni, css che non vengono renderizati), import senza altre chiamate http

***Gestione delle dipendenze***

Cattiva gestione, caricare dipendenze alla fine e cambiare variabili globali native. Si possono tollerare i polyfill (aggiungere funzionalità specifiche e/o correggere bug)

**revealing module pattern (simili alle classi)**



**Moduli e gestione delle dipendenze - divide et impera**

Dividere il software in piccoli moduli semplici

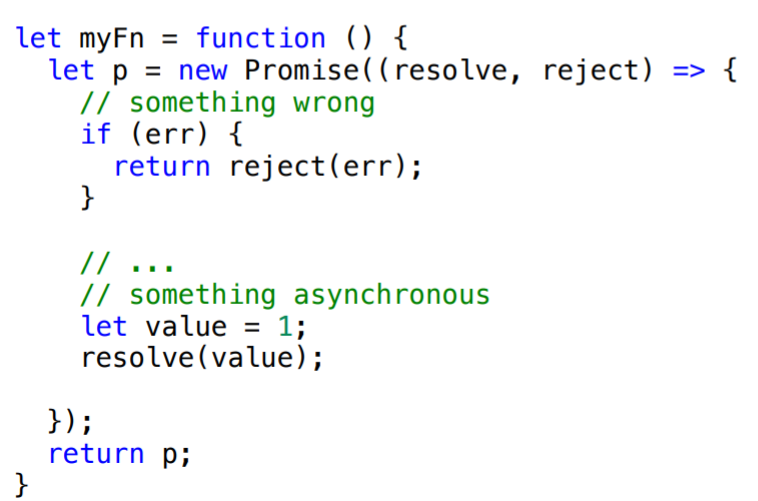
* AMD (il client è in grado di richiedere a runtime le dipendenze di cui ha bisogno in modo asincrono per offrire determinate funzionalità)
  + PRO Caricamento veloce
  + CONTRO quando viene richiesto il caricamento del nuovo modulo, l’utente deve aspettare
  + USO ad esempio la chat in GMAIL
* CommonJS (Le dipendenze vengono caricate in maniera sincrona, tramite un transpiler (webpack, browserify…))
  + PRO Dipendenze presenti al caricamento
  + CONTRO scaricamento anche di risorse che non verranno utilizzate
  + USO quando si vogliono fare poche chiamate http(s)
* ES2015 (simile a CommonJS) (quello che abbiamo usato nel progetto import/export)
  + Supportato da Node.js e transpiler)
* UMD (Permette il caricamento in entrambe le modalità, sincrona e on-demand (asincrona))
  + PRO unisce i pro dei metodi precendenti
  + CONTRO sintassi verbosa
  + USO in applicazioni molto complesse

Import on-Demand 🡪 Utile quando parte dell’applicazione può essere caricata successivamente all’avvio (lazy loading) per garantire un caricamento iniziale più veloce

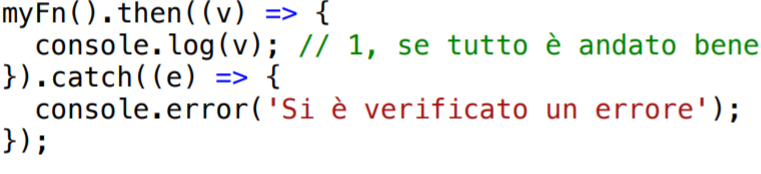
**Promises**

I caricamenti lazy devono essere eseguiti in maniera asincrona per non bloccare il thread principale (l’unico!)

L’esecuzione di then avviene quando il dato è pronto, come una callback, con la differenza che promises asincrone possono essere concatenate o eseguite contemporaneamente con estrema facilità



resolve deve essere invocata con il valore da restituire nella then() • reject() deve essere restituita in caso di errore, possibilmente con una spiegazione relativa all’eccezione



***SVG***

La viewport è la “finestra” attraverso la quale si osserva il documento SVG, è utile assegnalo come <object data=”ciao.svg”></object>

Asse x positivo, asse y verso il basso. Alto a sinistra (0,0).Dimensioni stabilite anche tramite css (width/height)

Attributo all’svg viewBox=”min-x min-y width height” simile ad un frame dove si può interagire tipo zoomare un svg

preserveAspectRatio = <align> <meetOrSlice>

* un valore “none” per align fa in modo che l’SVG venga scalato senza mantenere le proporzioni
* align può avere uno fra i nove valori definiti oppure none e viene utilizzato per allineare e scalare uniformemente la viewBox nella viewport (default xMidYMid)
* meetOrSlice: meet (default) si comporta come background-size:contain (css), l’immagine viene scalata in modo da stare tutta nella viewport, se la ratio è la stessa; se la ratio di viewport e viewBox è diversa, ci saranno delle parti non coperte dalla viewBox
* meetOrSlice: slice funziona come backgroundsize:cover: se le ratio non sono le stesse, la viewBox viene tagliata e la viewport resta tutta coperta in modo

**Trasformazioni tipo OpenGL (ricordarsi che si basano sul punto 0,0 del SVG)**

gli attributi fill, stroke e stroke-width cambiano rispettivamente il colore di riempimento, del bordo e la grandezza del bordo

**Path**

l’elemento <path> è il modo più versatile per disegnare delle forme in SVG. <path d=”M10 10”> indica di “spostare la penna” nel punto 10 10.

Mx y - spostamento assoluto • m dx - dy spostamento relativo • L x y - segmento dal punto corrente a quello di coordinate x, y (analogamente per l dx dy)

Potete utilizzare strumenti free come gravitDesigner, disegnare i path ed esportarli in SVG per poi includerli nelle interfacce. È possibile definire dei filtri che prendano in input un nodo SVG e forniscano in output il nodo modificato. Usando l’elemento clipPath è possibile definire una qualsiasi forma attraverso cui “tagliare” un altro elemento, per poi applicarla via css (non supportato da tutti) o via attributo clip-path. È possibile utilizzare l’elemento per raggruppare logicamente altri elementi e per poter applicare stili e trasformazioni a più elementi contemporaneamente. Si devono usare le varianti document.getElementsByTagNameNS e document.createElementNS rispettivamente per accedere e creare nuovi nodi

***Interfacce complesse*** facile dimenticare di aggiornare l’interfaccia al seguito di un cambiamento dei dati

**Vue.js**

* Vue è un framework per produrre interfacce utente che
* permette di sviluppare per componenti
* offre funzionalità di data binding
* è (relativamente) semplice da imparare
* ha un ricco ecosistema di plugin (eg vue router, vuex)

Componenti🡪 sono istanze riusabili di vue, permettono di definire nuovi tag, possono avere un proprio state o stateless, uno stato può essere propagato da padri a figli

**separation of concerns**

* scope variabili non globale, ma legato al componente
* gli handler di eventi vengono automaticamente rimossi da Vue quando serve
* viene usata (trasparentemente) l’event delegation
* la parte js dei componenti non è legata al DOM, quindi più “pulita” e facile da testare

**Tuttavia**

Sincronizzare diversi componenti che condividono lo stesso stato via eventi o messaggi diventa rapidamente poco manutenibile / instabile

**Soluzione**

Estrarre lo stato dai componenti dalla GUI e gestirlo come un singleton, in modo che possa fare da unico valore di verità per tutta l’interfaccia

**Vuex** è definita come pattern di gestione dello stato + libreria correlata

**Store** 🡪 mantiene lo stato dell’applicazione (può essere modificato tramite **mutations** tracciate) e fornisce facilitazioni (UNDO/REDO)

**State**🡪 Vuex è **reactive** quindi se si modifica lo stato l’interfaccia reagirà di conseguenza

**Getters** 🡪 metodi dello store che possono estrarre informazioni sullo stato

**Mutations** 🡪 devono essere sincrone (per poter essere tracciabili) Esse sono l’unico modo per modificare lo stato e possono essere invocate dai componenti o dalle actions

**Actions 🡪**sono all’interno di store e permettono di richiedere la modifica dello stato in maniera asincrona: un esempio può essere la chiamata a una URI remota

***Altro***

**Differenza let e var?**

La differenza è lo scope. Var è focalizzato sul blocco funzione (quindi vale per tutto il blocco) e let invece nel blocco di inclusione (quindi come java). Entrambi sono globali se fuori da qualsiasi blocco.

Inoltre le variabili dichiarate con let non sono accessibili prime che vengano dichiarate nel loro blocco .

**Cosa è Handler?**

**Cosa è rem?**

uello che possono fare è aiutarti a ottenere un design armonioso ed equilibrato

Uguale al valore calcolato della dimensione del font sull'elemento radice. Se specificato nella proprietà font-size dell'elemento root, le unità rem si riferiscono al valore iniziale della proprietà.

Ciò significa che 1rem è uguale alla dimensione del carattere dell'elemento html (che per la maggior parte dei browser ha un valore predefinito di 16px).

**Cosa è sezione aurea ?**

In generale l’uso della sezione aurea nel testo scritto serve per garantire una piacevole lettura. Ad esempio scegliendo una scala di proporzionalità aurea nei vari font-size utilizzati e magari utilizzando blocchi (dove possibile) con rapporto aureo fornisce un risultato esteticamente migliore.

**Cosa è dom?**

Document Object Model è un modello nel quale possiamo accedere a qualsiasi elemento nella pagina html, tipo un indice