ImageDots

1 Introduzione 3

1.1 Informazioni sul progetto 3

1.2 Abstract 3

1.3 Scopo 3

2 Analisi 4

2.1 Analisi del dominio 4

2.2 Analisi e specifica dei requisiti 4

2.2.1 Spiegazione elementi tabella dei requisiti: 5

2.3 Use case 6

2.4 Pianificazione 6

2.5 Analisi dei mezzi 6

2.5.1 Software 6

2.5.2 Hardware 6

3 Progettazione 7

3.1 Design dell’architettura del sistema 7

3.1.1 Struttura dell’applicativo WEB 7

3.2 Design dei dati e database 8

3.3 Design delle interfacce 8

3.4 Design procedurale 9

4 Implementazione 11

4.1 Struttura base 11

4.2 main.js 11

4.3 layers.js 12

4.4 dots.js 13

4.5 drawing\_manager.js 17

5 Test 17

5.1 Protocollo di test 17

5.2 Risultati test 19

5.3 Mancanze/limitazioni conosciute 19

6 Consuntivo 19

7 Conclusioni 19

7.1 Sviluppi futuri 19

7.2 Considerazioni personali 19

8 Glossario 19

9 Bibliografia 20

9.1 Bibliografia per articoli di riviste: 20

9.2 Bibliografia per libri 20

9.3 Sitografia 20

10 Allegati 20

# Introduzione

## Informazioni sul progetto

## Abstract

È una breve e accurata rappresentazione dei contenuti di un documento, senza notazioni critiche o valutazioni. Lo scopo di un abstract efficace dovrebbe essere quello di far conoscere all’utente il contenuto di base di un documento e metterlo nella condizione di decidere se risponde ai suoi interessi e se è opportuno il ricorso al documento originale.

Può contenere alcuni o tutti gli elementi seguenti:

* **Background/Situazione iniziale**
* **Descrizione del problema e motivazione**: Che problema ho cercato di risolvere? Questa sezione dovrebbe includere l'importanza del vostro lavoro, la difficoltà dell'area e l'effetto che potrebbe avere se portato a termine con successo.
* **Approccio/Metodi**: Come ho ottenuto dei progressi? Come ho risolto il problema (tecniche…)? Quale è stata l’entità del mio lavoro? Che fattori importanti controllo, ignoro o misuro?
* **Risultati**: Quale è la risposta? Quali sono i risultati? Quanto è più veloce, più sicuro, più economico o in qualche altro aspetto migliore di altri prodotti/soluzioni?

Esempio di abstract:

*As the size and complexity of today’s most modern computer chips increase, new techniques must be developed to effectively design and create Very Large-Scale Integration chips quickly. For this project, a new type of hardware compiler is created. This hardware compiler will read a C++ program, and physically design a suitable microprocessor intended for running that specific program. With this new and powerful compiler, it is possible to design anything from a small adder, to a microprocessor with millions of transistors. Designing new computer chips, such as the Pentium 4, can require dozens of engineers and months of time. With the help of this compiler, a single person could design such a large-scale microprocessor in just weeks.*

## Scopo

Lo scopo del progetto (scopi didattici/scopi operativi). Dovrebbe descrivere il mandato, ma non vanno ricopiate le informazioni del quaderno dei compiti (che va invece allegato).

# Analisi

## Analisi del dominio

L’applicativo verrà utilizzato principalmente da genitori, da maestri dell’asilo o delle scuole elementari.

Si vuole dare la possibilità di generare le proprie immagini con i puntini da collegare, per evitare la ricerca di immagini già con i puntini che potrebbero non rispettare l’idea originale dell’utente.

## Analisi e specifica dei requisiti

|  |  |
| --- | --- |
| **ID: REQ-001** | |
| **Nome** | Interfaccia |
| **Priorità** | 1 |
| **Versione** | 1.0 |
| **Note** | Caricamento immagine |
| **Sotto requisiti** | |
| **001** | I formati dell’immagine caricata sono webp, jpg, jpeg, png |
| **002** | L’Immagine caricata deve essere ridimensionata |
| **003** | L’immagine creata deve essere salvabile |
| **004** | L’immagine creata deve essere stampabile |
| **005** | Si deve poter scartare l’immagine |

|  |  |
| --- | --- |
| **ID: REQ-002** | |
| **Nome** | Layer |
| **Priorità** | 1 |
| **Versione** | 1.0 |
| **Note** | Ci devono essere più layer: disegni, puntini |
| **Sotto requisiti** | |
| **001** | Si può cambiare la visualizzazione dei layer |

|  |  |
| --- | --- |
| **ID: REQ-003** | |
| **Nome** | Puntini |
| **Priorità** | 1 |
| **Versione** | 1.0 |
| **Note** | Bisogna essere in grado di aggiungere, eliminare, spostare i puntini |
| **Sotto requisiti** | |
| **001** | Il numero dei puntini aggiunti aumenta ogni volta che vengono aggiunti |
| **002** | Il numero dei puntini deve poter essere modifcato |
| **003** | Si può vedere il risultato dei puntini collegati |

|  |  |
| --- | --- |
| **ID: REQ-004** | |
| **Nome** | Disegno |
| **Priorità** | 1 |
| **Versione** | 1.0 |
| **Note** | Ci devono essere i strumenti di disegno come penna, gomma, linee, rettangolo, cerchio, ovale |
| **Sotto requisiti** | |
| **001** | Unione del disegno insieme ai puntini |

|  |  |
| --- | --- |
| **ID: REQ-005** | |
| **Nome** | Stampa |
| **Priorità** | 1 |
| **Versione** | 1.0 |
| **Note** | L’immagine creata deve essere stampabile |
| **Sotto requisiti** | |
| **001** | Ricerca stampanti |

### Spiegazione elementi tabella dei requisiti:

**ID**: identificativo univoco del requisito

**Nome**: breve descrizione del requisito

**Priorità**: indica l’importanza di un requisito nell’insieme del progetto, definita assieme al committente. Ad esempio, poter disporre di report con colonne di colori diversi ha priorità minore rispetto al fatto di avere un database con gli elementi al suo interno. Solitamente si definiscono al massimo di 2-3 livelli di priorità.

**Versione**: indica la versione del requisito. Ogni modifica del requisito avrà una versione aggiornata.

Sulla documentazione apparirà solamente l’ultima versione, mentre le vecchie dovranno essere inserite nei diari.

**Note**: eventuali osservazioni importanti o riferimenti ad altri requisiti.

**Sotto requisiti**: elementi che compongono il requisito.

## Use case

I casi d’uso rappresentano l’interazione tra i vari attori e le funzionalità del prodotto.

## Pianificazione

Prima di stabilire una pianificazione bisogna avere almeno una vaga idea del modello di sviluppo che si intende adottare. In questa sezione bisognerà inserire il modello concettuale di sviluppo che si seguirà durante il progetto. Gli elementi di riferimento per una buona pianificazione derivano da una scomposizione top-down della problematica del progetto.

La pianificazione può essere rappresentata mediante un diagramma di Gantt:

|  |
| --- |
| Figura 1: Esempio di diagramma di Gantt. |

Se si usano altri metodi di pianificazione (p.es. scrum), dovranno apparire in questo capitolo.

## Analisi dei mezzi

Elencare e descrivere i mezzi disponibili per la realizzazione del progetto. Ricordarsi di sempre descrivere nel dettaglio le versioni e il modello di riferimento.

### Software

* Visual Studio Code 1.78.2
* Html 5
* Css 3
* Javascript 1.5
* Print.js 1.5.0

### Hardware

* 1 PC: Le componenti del PC che verrà utilizzato per lo sviluppo dell’applicativo sono:
  + I7-9700
  + 32 GB RAM
  + SSD 512 GB
  + NVIDIA GeForce RTX 2060

# Progettazione

## Design dell’architettura del sistema

### Struttura dell’applicativo WEB

La struttura delle cartelle è la seguente e all’interno sono presenti i seguenti elementi:

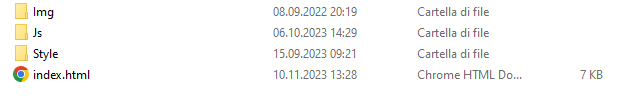


Figura 2 – Struttura cartelle

* **Img:** Dentro questa cartella sono presenti le immagini utilizzate dall’applicativo.
* **Js:** Contiene tutti gli script Javascript.
* **Style:** Contiene tutti i file CSS, che modificano la veste grafica dell’applicativo

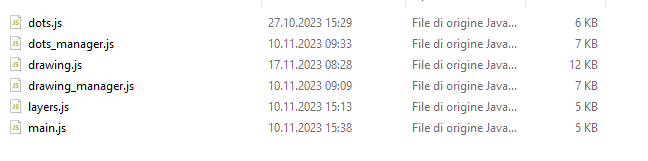
****

Figura 3 – File cartella Js

* **File dots.js e drawing.js:** contengono le funzioni essenziali per il disegno dei punti.
* **File dots\_manager.js e drawing\_manager.js**: fungono da intermediari tra l'HTML e il JS, e includono anche funzioni semplici e di dimensioni ridotte.
* **File layers.js:** gestisce il cambio della visualizzazione dei layer.
* **File main.js:** comprende il resto delle funzioni, come il salvataggio, stampa, inizializzazione di funzioni.

## Design dei dati e database

## Design delle interfacce

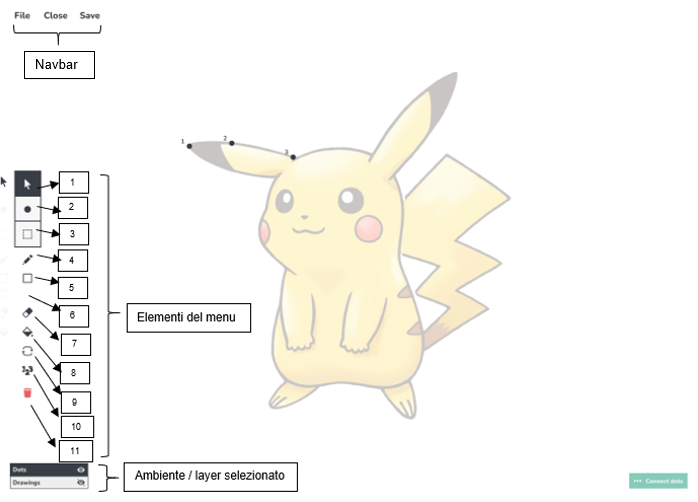


Figura 4 – Pagina principale

L’applicativo lavora principalmente su questa pagina, le differenze principali dal design al codice sono che:

* L’elemento del menu 1 è disponibile in entrambi gli ambienti.
* Gli elementi del menu 2, 3, 9, 10 sono disponibili solo quando siamo nell’ambiente dei puntini.
* Gli elementi del menu 4, 5, 6, 7, 8 sono disponibili solo quando siamo nell’ambiente dei disegni.
* L’elemento del menu 10 e 11 appare quando un puntino viene selezionato.
* L’elemento del menu 11 appare anche quando rettangoli ed elissi vengono selezionati.
* Il pulsante “Connect dots” è disponibile solo nell’ambiente dei puntini.
* Alla fine degli elementi c’è uno slider per cambiare le dimensioni di puntini o delle linee delle forme disegnate.
* Al posto dell’elemento “File” nella navbar c’è l’elemento

I vari elementi del menu rappresentano le seguenti funzionalità:

1. La selezione di un puntino o disegno.
2. L’aggiunta di un puntino.
3. La selezione di tutti i puntini.
4. Disegno di linee a mano libera.
5. Disegno di rettangoli.
6. Disegno di elissi.
7. Cancellazione di linee a mano libera.
8. Se selezionato le forme disegnate saranno riempite.
9. Inverte il numero dei puntini.
10. Cambia il numero del puntino selezionato.
11. Elimina il puntino o la forma selezionata.

Gli elementi della navbar rappresentano le seguenti funzionalità:

* File: diventano poi “Open”, prende l’immagine dall’utente.
* Close: chiude l’immagine e cancella tutte le cose che l’utente ha fatto.
* Save: apre il menu di salvataggio.

Pulsante “Connect dots”:

* Compare se siamo nell’ambiente dei puntini.
* Se schiacciato fa vedere tutti i puntini collegati.

Menu layer:

* Si seleziona l’ambiente che si vuole utilizzare.
* Si può visualizzare l’altro ambiente che non è selezionato. Es. l’utente è nell’ambiente puntini, vuole vedere come stanno i disegni insieme, per farlo schiaccia l’occhio accanto a “Drawings”.

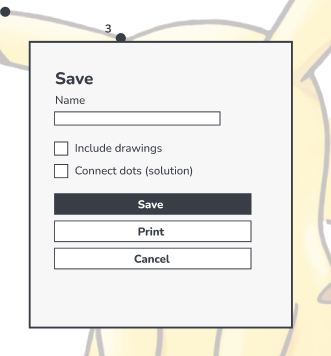


Figura 5 – Menu di salvataggio

* **Name:** Rappresenta il nome del file scaricato, che verrà scaricato in formato png.
* **Include drawings:** Se schiacciato oltre ai puntini, saranno inclusi anche i disegni nell’immagine finale.
* **Connect dots (solution):** Stesso funzionamento del bottone “Connect dots”, per l’immagine scaricata.
* **Save:** Scarica l’immagine.
* **Print:** Apre il menu di stampa del browser.
* **Cancel:** Esce dal menu.

## Design procedurale

Descrive i concetti dettagliati dell’architettura/sviluppo utilizzando ad esempio:

* Diagrammi di flusso e Nassi.
* Tabelle.
* Classi e metodi.
* Tabelle di routing
* Diritti di accesso a condivisioni …

Questi documenti permetteranno di rappresentare i dettagli procedurali per la realizzazione del prodotto.

# Implementazione

## Struttura base

Ho diviso il codice nelle seguenti categorie:

* **File dots.js e drawing.js:** contengono le funzioni essenziali per il disegno dei punti.
* **File dots\_manager.js e drawing\_manager.js**: fungono da intermediari tra l'HTML e il JS, e includono anche funzioni semplici e di dimensioni ridotte.
* **File layers.js:** gestisce il cambio della visualizzazione dei layer.
* **File main.js:** comprende il resto delle funzioni, come il salvataggio, stampa, inizializzazione di funzioni.

## main.js

Come detto in precedenza, main.js comprende il resto delle funzioni, come il salvataggio, stampa, inizializzazione di funzioni.



Figura 6 – main.js, getImageData()

getImageData() serve a usare i dati dell'immagine presi dall'input, dopodiché modifica le proprietà dei vari canvas e l'immagine viene disegnata.

Questa parte inizialmente mi ha dato un po’ di problemi, soprattutto per il resize dell’immagine, ma “backgroundSize = ‘contain’” fa tutto il lavoro.

## layers.js

Come detto in precedenza gestisce il cambio della visualizzazione dei layer.

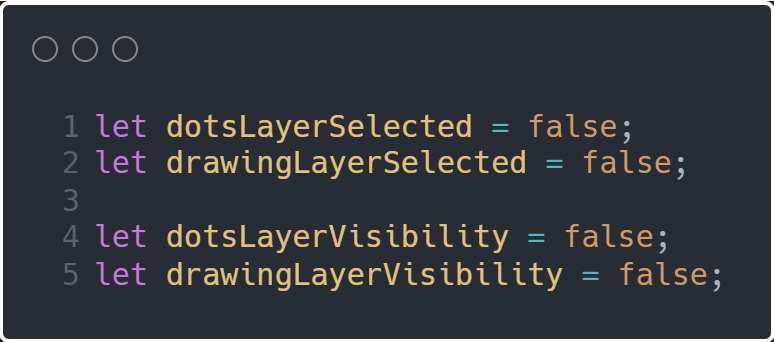


Figura 7 – layers.js, variabili globali

* **dotsLayerSelected e drawingLayerSelected:** servono dopo nella selezione della visibilità dei layer, perché se selezioni il layer dei puntini non puoi selezionare anche la visibiltà del layer dei puntini.
* **dotsLayerVisibility e drawingLayerVisibility:** se true usati in drawing.js e dots.js.



Figura 8 – layers.js, selezione layer

Quando viene selezionato un layer, in entrambi i casi si cambiano le variabili di prima, si fa il refresh del canvas e si cambiano gli elementi del menu per rispettare l’ambiente in cui siamo.

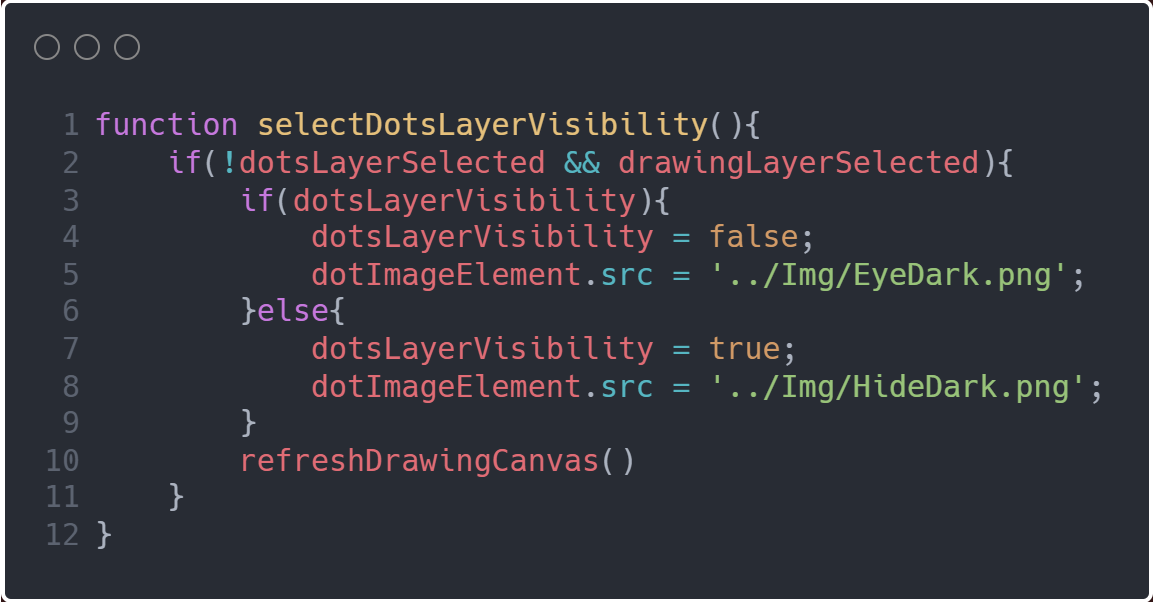


Figura 9 – layers.js, selezione visibilità

Quando viene selezionata la visibilità, in entrambi i casi le cose che succedono sono cambiare lo stile e le variabili.

## dots.js

Come detto in precedenza questo file serve contiene le funzioni di base dei puntini.

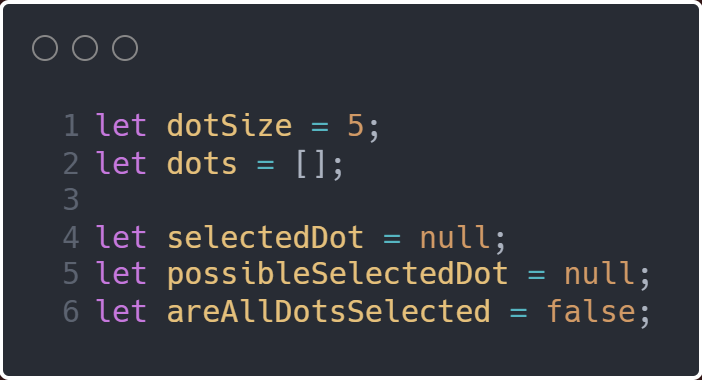


Figura 10 – dots.js variabili

Le funzioni dei puntini utilizzano principalmente queste variabili:

* **dotSize:** rappresenta il raggio del puntino.
* **dots:** contiene i puntini aggiunti dall’utente presenti nel canvas.
* **selectedDot:** il puntino selezionato dall’utente.
* **possibleSelectedDot:** il puntino che se l’utente schiaccia viene selezionato.



Figura 11 – dots.js, classe Dot

La classe Dot è la più usata ovviamente per il funzionamento dei puntini. Contiene solo un metodo oltre al costruttore che è draw(context). Le parti più interessanti sono:

* **Riga 23:** qua viene disegnato concretamente il puntino:
  + **pointX:** la coordinata X di dove viene disegnato il puntino.
  + **pointY:** la coordinata Y di dove viene disegnato il puntino.
  + **this.size:** rappresenta il raggio del puntino.
  + **Gli ultimi due parametri:** rappresentano l'angolo di inizio e l'angolo di fine dell'arco. In questo caso, l'arco si estende da 0 a 360 gradi, ovvero un giro completo.
* **Riga 30:** viene calcolato quanto deve essere la grandezza del testo del label, il numero del puntino. Ho deciso che la grandezza che più sta bene è 16px / 5 px del raggio del puntino.

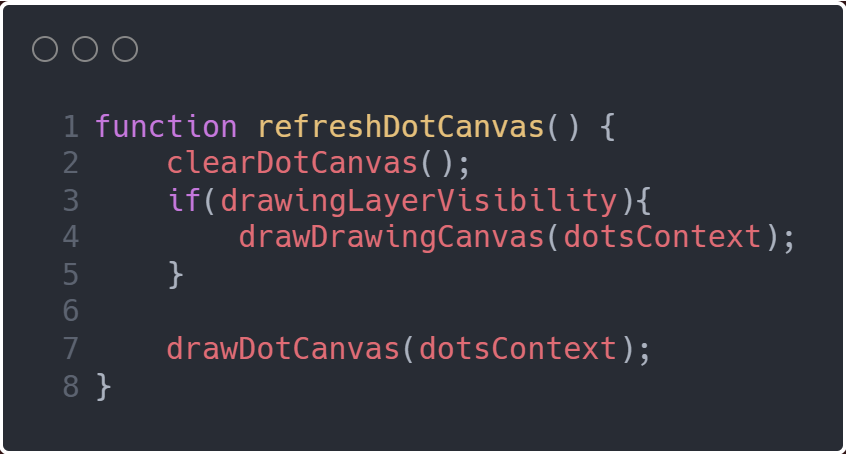


Figura 12 – dots.js, refreshDotCanvas()

Un metodo molto usato che vale la pena far vedere è refreshDotCanvas(), è usato continuamente nel codice. Prima viene pulito il canvas, poi se la visibilità del layer dei disegni, descritta precedentemente, è attiva aggiunge i disegni, infine usa drawDotCanvas(dotsContext) che disegnerà nel context dei puntini i puntini.

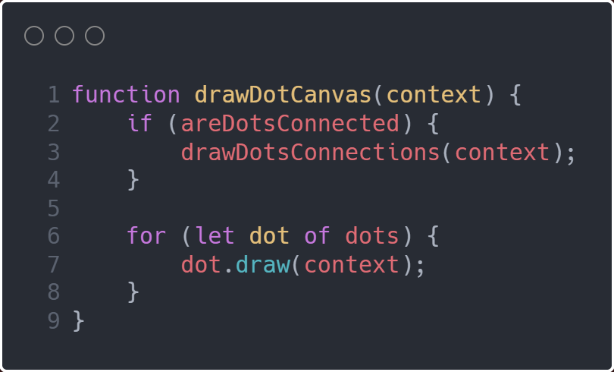


Figura 13 – dots.js, drawDotCanvas(context)

Come metodo drawDotCanvas(context) è molto semplice, la maggior parte del lavoro la fa la nostra classe Dot, se areDotsConnected è true, saranno disegnate le linee tra i puntini.

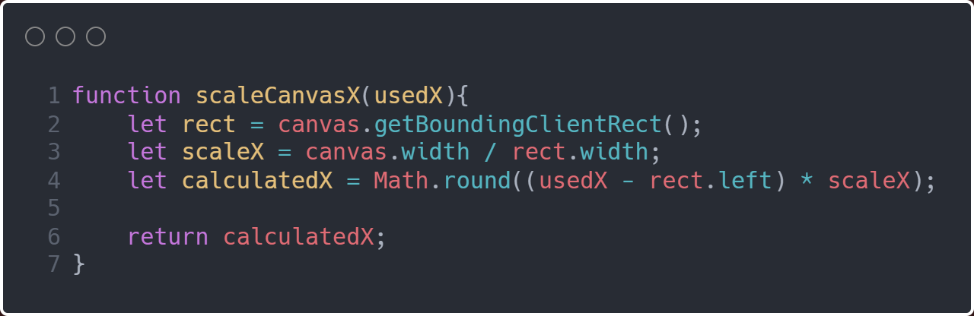


Figura 14 – dots.js, scaleCanvasX(usedX)

Il metodo scaleCanvasX(usedX) e scaleCanvasY(usedY) vengono usati spesso, senza di essi, le funzioni di aggiunta, spostamento e selezione dei puntini non sarebbero precise e potrebbero non funzionare.

* **usedX:** la X data dagli eventi, perché gli eventi prendono la X e la Y dello schermo e non del canvas.
* **getBoundingClientRect:** per ottenere il rettangolo che rappresenta l'elemento canvas nell'area di visualizzazione del browser. Questo rettangolo fornisce informazioni come l'altezza, la larghezza, la posizione sinistra e la posizione superiore dell'elemento canvas rispetto all'area di visualizzazione.
* **scaleX:** che rappresenta la scala orizzontale del canvas. Viene calcolata dividendo la larghezza del canvas per la larghezza del rettangolo ottenuto in precedenza.
* **calculatedX:** utilizzando la formula (usedX - rect.left) \* scaleX. Questo calcola la posizione orizzontale corrispondente all'input usedX rispetto all'area di visualizzazione del browser.

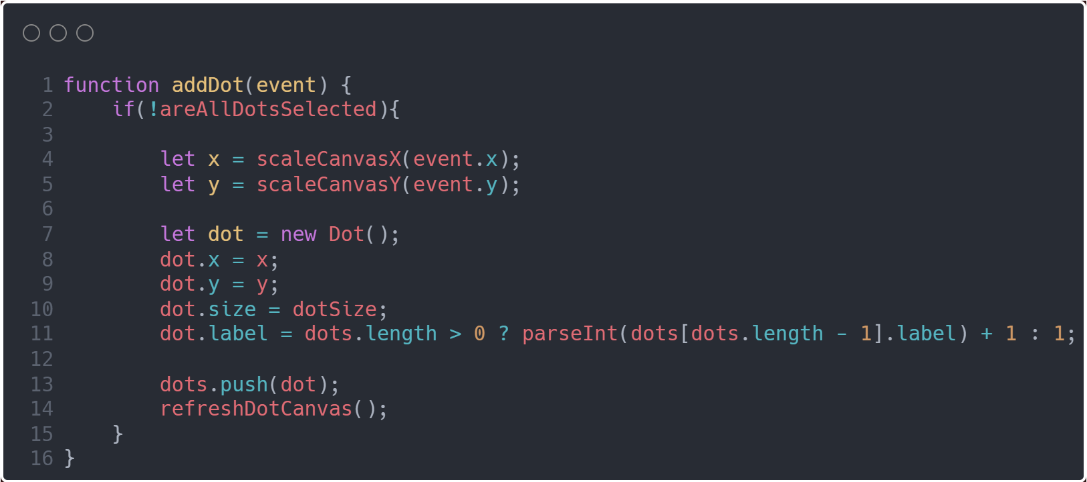


Figura 15 – dots.js, addDot()

addDot() viene usato come il nome fa intuire, per aggiungere i puntini. Il funzionamento è molto semplice, quando si schiaccia sul canvas, viene creato un nuovo puntino ed aggiunto all’array dei puntini, per il numero del puntino si prende il numero dell’ultimo puntino creato, si aggiunge 1.



Figura 16 – dots.js, selectDot

selectDot() viene usato per la selezione di un puntino, che potrà dopo essere cancellato, spostato, modificato, ecc.

Quando viene schiacciato il canvas, si controllano tutti puntini, per vedere se la posizione schiacciata corrisponde alla posizione di un puntino, se corrisponde si cambia il colore del puntino, viene salvato come puntino selezionato. Infine vengono visualizzati gli elementi del menu che servono al puntino selezionato.

Se la posizione schiacciata non corrisponde alla posizione di un puntino, vengono deselezionati tutti i puntini e spariscono gli elementi del menu che servirebbero al singolo puntino.



Figura 17 – dots.js, moveDot

Per muovere il puntino ci sono tre metodi, stopMovingDot, startMovingDot, moveDot.

* **stopMovingDot:** imposta la variabile isMouseDown a false, facendo smettere lo spostamento di un puntino.
* **moveDot:** sposta il puntino, per lo spostamento di un puntino ovviamente almeno un puntino deve essere selezionato, cambia la posizione del/dei puntino/i con quella del mouse.

## dots\_manager.js

Come detto in precedenza questo file funge da intermediario tra l'HTML e dots.js, include anche funzioni semplici e di dimensioni ridotte.



Figura 18 – dots\_manager.js, removeAllDotsEvents e resetOptionElements

removeAllDotsEvents e resetOptionElements servono ad evitare la ripetizione di codice, quando viene selezionata una modalità.



Figura 19 – dots\_manager.js, selectAddMode

Quando c’è un cambio di modalità il codice si struttura più o meno nella stessa maniera, prendendo d’esempio selectAddMode(), che serve ad aggiungere nuovi puntini:

* Si tolgono gli eventi delle altre modalità.
* Si modifica lo stile degli elementi del menu.
* Si usano funzioni di dots.js.



Figura 20 – dots\_manager.js, selectDeleteMode

selectDeleteMode() è la funzione utilizzata dal cestino, se tutti i puntini sono selezionati, rimuove tutti i puntini, altrimenti:

* Scorre l’array e rimuove il punto selezionato.
* Controlla se c’è una lacuna nel numero dei punti.
* Per ogni puntino controlla se è stata trovata una lacuna, decrementa il numero del punto. Altrimenti, controlla se c'è una lacuna tra i numeri dei punti.

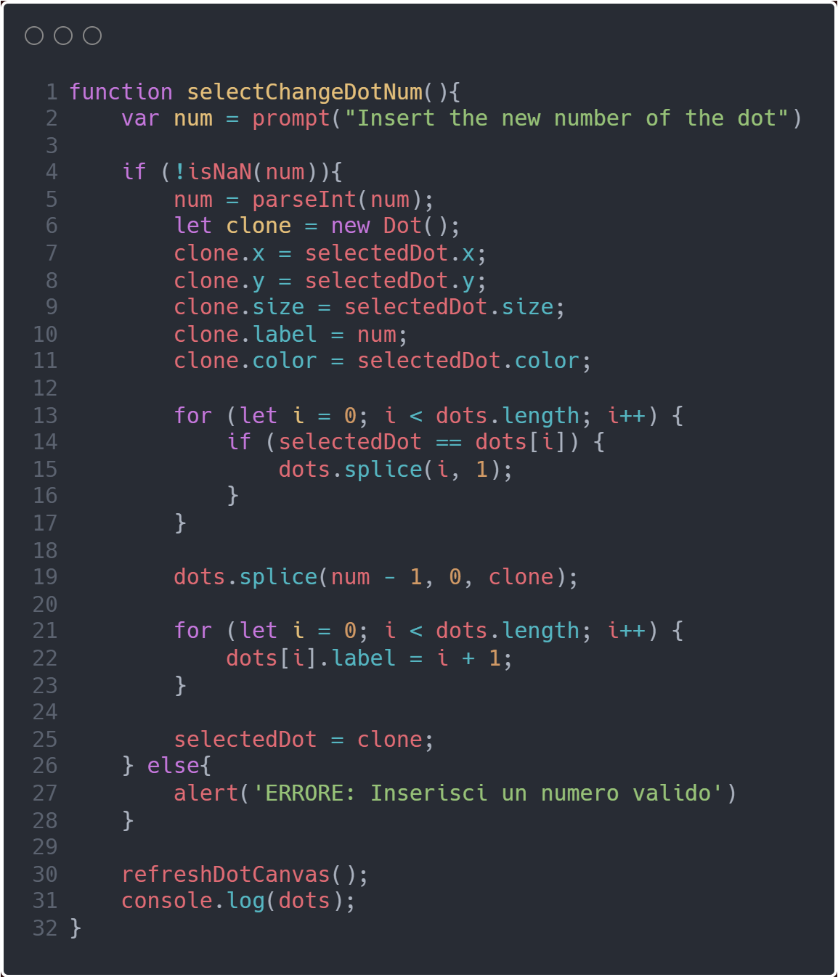


Figura 21 – dots\_manager.js, selectChangeDotNum

selectChangeDotNum chiede all’utente di inserire il nuovo numero del punto, verifica se il numero inserito è valido, crea un nuovo punto con le stesse proprietà del punto selezionato ma con il nuovo numero, rimuove il punto selezionato dall’array, inserisce il nuovo punto nell’array nella posizione specificata dal nuovo numero, aggiorna i numeri di tutti i punti nell’array, e infine aggiorna il canvas e stampa l’array dots sulla console.

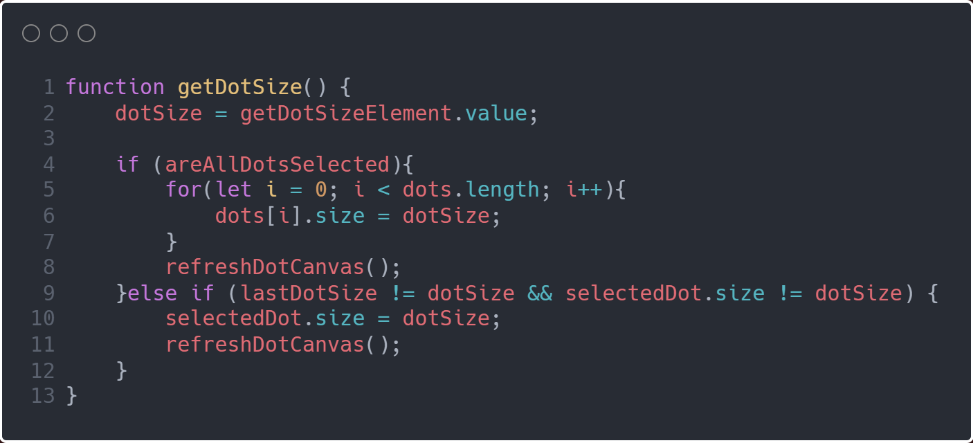


Figura 22 – dots\_manager.js, getDotSize

getDotSize() è il metodo utilizzato quando si usa lo slider, per cambiare la grandezza di un puntino. Se uno più puntini sono selezionati cambia la loro grandezza, se non ci sono puntini selezionati cambia la grandezza dei prossimi puntini che verranno aggiunti.

## drawing.js

Come detto in precedenza drawing.js contiene la logica del funzionamento dei disegni, contiene classi ed eventi che vengono usati in drawing\_manager.js e gli altri file.



Figura 23 – drawing.js, variabili

Le variabili più utilizzate sono le seguenti:

* **drawingSize:** spessore linee rettangoli, elissi e linee.
* **drawingColor:** colore usato nelle forme disegnate.
* **areShapesFilled:** se i rettangoli ed ellissi sono riempite con il colore selezionato.
* **lines:** tutte le linee disegnate nel drawingContext, create con la classe Line.
* **rects:** tutti i rettangoli disegnati nel drawingContext, creati con la classe Rect.
* **ellipses:** tutte le elissi disegnate nel drawingContext, create con la classe Ellipse.
* **selectedDrawing:** la forma selezionata che verrà eliminata, spostata, modificata.



Figura 24 – drawing.js, Line

La classe Line è utilizzata sia nel disegno delle linee a mano libera che quelle tra i puntini. I parametri necessari al funzionamento sono:

* **fromX:** coordinata X nel canvas da dove parte la linea.
* **fromY:** coordinata Y nel canvas da dove parte la linea.
* **toX:** coordinata X nel canvas dove arriva la linea.
* **toY:** coordinata Y nel canvas dove arriva la linea.

È importante la riga 12, altrimenti le linee vengono disegnate come se fossero un codice a barre.



Figura 25 – drawing.js, classe Rect

La classe Rect serve a disegnare i rettangoli, funziona principalmente come le altre classi per disegnare, i parametri sono:

* **x:** coordinata X nel canvas dove inizia la forma.
* **y:** coordinata Y nel canvas dove inizia la forma
* **width:** larghezza della forma.
* **height:** altezza della forma.
* **filled:** per riempire il rettangolo quando viene disegnato.
* **color:** colore della linea e del riempimento del rettangolo disegnato.
* **size:** spessore della linea del rettangolo disegnato.

Principalmente tutte le forme disegnate hanno questo tipo di parametri.



Figura 26 – drawing.js, drawDrawingCanvas

In drawDrawingCanvas il funzionamento è uguale a drawDotCanvas, ma ovviamente qua disegniamo le forme e non i puntini.

Figura 27 – drawing.js, stopDrawingRect, startDrawingRect, drawRect

Tutti i metodi di disegno hanno sempre lo stesso funzionamento, tranne che per le righe a mano libera che sono ancora più semplici:

* Presenza di uno stopDrawing.
* Presenza di uno StartDrawing. 
* Controllo se la forma disegnata è la prima forma disegnata, altrimenti c’è un effetto visivo un po’ strano.
* lastForma, es. lastRect, serve a mantenere solo l’ultima forma disegnata, perché vediamo più forme quando trasciniamo, ma deve rimanerne solo una alla fine.
* Impostamento di lastForma a null, altrimenti a volte la forma viene duplicata quando usciamo dal canvas, perché stopDrawing risponde anche all’evento ‘mouseout’.

# Test

## Protocollo di test

Definire in modo accurato tutti i test che devono essere realizzati per garantire l’adempimento delle richieste formulate nei requisiti. I test fungono da garanzia di qualità del prodotto. Ogni test deve essere ripetibile alle stesse condizioni.

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Case:**  **Riferimento**: | TC-001  REQ-012 | **Nome:** | Import a card with KIC, KID and KIK keys, but not shown with the GUI |
| **Descrizione:** | Import a card with KIC, KID and KIK keys with no obfuscation, but not shown with the GUI | | |
| **Prerequisiti:** | Store on local PC: Profile\_1.2.001.xml (appendix n\_n) and Cards\_1.2.001.txt (appendix n\_n).  PIN (OTA\_VIEW\_PIN\_PUK\_KEY) and ADM (OTA\_VIEW\_ADM\_KEY) user right not set. | | |
| **Procedura:** | 1. Go to “Cards manager” menu,  in main page click “Import Profiles” link, Select the “1.2.001.xml” file, Import the Profile 2. Go to “Cards manager” menu,  in main page click “Import Cards” link, Select the “1.2.001.txt” file, Delete the cards,  Select the “1.2.001.txt” file, Import the cards 3. Research the “41795924770” Card, Click the imsi card link Check the card details 4. Execute the SQL: SELECT imsi, dir, keyset, cntr, rawtohex(kickey), rawtohex(kidkey), rawtohex(kikkey), rawtohex(chv), rawtohex(dap)FROM otacardkey a where imsi='340041795924770' ORDER BY keyset; | | |
| **Risultati attesi:** | Keys visible in the DB (OtaCardKey) but not visible in the GUI (Card details) | | |

## Risultati test

Tabella riassuntiva in cui si inseriscono i test riusciti e non del prodotto finale. Se un test non riesce e viene corretto l’errore, questo dovrà risultare nel documento finale come riuscito (la procedura della correzione apparirà nel diario), altrimenti dovrà essere descritto l’errore con eventuali ipotesi di correzione.

## Mancanze/limitazioni conosciute

Descrizione con motivazione di eventuali elementi mancanti o non completamente implementati, al di fuori dei test case. Non devono essere riportati gli errori e i problemi riscontrati e poi risolti durante il progetto.

# Consuntivo

Consuntivo del tempo di lavoro effettivo e considerazioni riguardo le differenze rispetto alla pianificazione (cap. 1.7) (ad esempio Gantt consuntivo).

# Conclusioni

Quali sono le implicazioni della mia soluzione? Che impatto avrà? Cambierà il mondo? È un successo importante? È solo un’aggiunta marginale o è semplicemente servita per scoprire che questo percorso è stato una perdita di tempo? I risultati ottenuti sono generali, facilmente generalizzabili o sono specifici di un caso particolare? ecc.

## Sviluppi futuri

Migliorie o estensioni che possono essere sviluppate sul prodotto.

## Considerazioni personali

Cosa ho imparato in questo progetto? ecc.

# Glossario

Inserite una semplice tabella con due colonne che spieghi i termini specifici del progetto (lista dei termini in ordine alfabetico A-Z)

Esempio:

|  |  |
| --- | --- |
| **Termine** | **Descrizione** |
| AJAX | **Asynchronous JavaScript And XML**: una tecnica che permette di eseguire richieste ed ottenere dati da una pagina web in modo asincrono. |
| CSS | **Cascading Style Sheets**: linguaggio che permette di definire il layout e la grafica di una pagina web. |

# Bibliografia

## Bibliografia per articoli di riviste:

1. Cognome e nome (o iniziali) dell’autore o degli autori, o nome dell’organizzazione,
2. Titolo dell’articolo (tra virgolette),
3. Titolo della rivista (in italico),
4. Anno e numero
5. Pagina iniziale dell’articolo.

## Bibliografia per libri

1. Cognome e nome (o iniziali) dell’autore o degli autori, o nome dell’organizzazione,
2. Titolo del libro (in italico),
3. ev. Numero di edizione,
4. Nome dell’editore,
5. Anno di pubblicazione,
6. ISBN.

## Sitografia

1. URL del sito (se troppo lungo solo dominio, evt completo nel diario),
2. Eventuale titolo della pagina (in italico),
3. Data di consultazione (GG-MM-AAAA).

**Esempio:**

* http://standards.ieee.org/guides/style/section7.html, *IEEE Standards Style Manual*, 07-06-2008.

# Allegati

Elenco degli allegati, esempio:

* Diari di lavoro
* Codici sorgente/documentazione macchine virtuali
* Istruzioni di installazione del prodotto (con credenziali di accesso) e/o di eventuali prodotti terzi
* Documentazione di prodotti di terzi
* Eventuali guide utente / Manuali di utilizzo
* Mandato e/o QdC
* Prodotto
* …