Università degli Studi di Modena e Reggio Emilia

Dipartimento di Ingegneria “Enzo Ferrari”

Corso di Laurea Triennale in Ingegneria Informatica

Sviluppo di un’applicazione per visualizzare gli spostamenti tra i reparti di un ospedale

Relatore:
Prof.ssa Sonia Bergamaschi

Candidato:
Gabriele Ruini

Correlatore:
Prof. Giovanni Simonini

Sommario

CAPITOLO 1 - Introduzione

Il seguente elaborato tratta del mio personale contributo al progetto che l'azienda Coopservice S. Coop. P.A. ha commissionato al DBGroup dell'Università di Modena e Reggio Emilia.

L'oggetto di questa relazione, pertanto, sarà la presentazione delle fasi di studio, progettazione e sviluppo di un'applicazione per la visualizzazione degli spostamenti dei pazienti del Policlinico S. Orsola-Malpighi, tra i suoi reparti.

Più precisamente, nel primo capitolo verrà introdotto il contesto generale, con lo scopo di far comprendere quali sono state le motivazioni che hanno fatto nascere il progetto, e quali sono i suoi obiettivi.

Nel secondo capitolo verranno descritte le più note tra le principali tecnologie utilizzate nella realizzazione del progetto, mentre nel terzo sarà presentato un approfondimento sulla libreria di JavaScript D3, il cui studio ha rappresentato una parte consistente del mio lavoro.

Il quarto capitolo è dedicato alla presentazione del lavoro da me svolto.

Infine, il quinto capitolo contiene la descrizione del lavoro futuro necessario a terminare il progetto.

CAPITOLO 2 – Progetto e Contesto

In questo primo capitolo presenterò il progetto, contestualizzandolo all'interno di Coopservice xe nell'ambito in cui l'azienda opera.

Nel primo paragrafo parlerò dell'azienda e di alcuni servizi da essa erogati, che hanno portato all'esigenza a cui vuole rispondere questo progetto.

Il secondo paragrafo è dedicato alla descrizione di quello da cui siamo partiti: i dati che Coopservice ha raccolto nel primo anno di pandemia nel Policlino S. Orsola-Malpighi.

L'ultimo paragrafo servirà a fornire una delucidazione riguardo agli obiettivi del progetto.

2.1 Coopservice e contesto generale

Coopservice è un'azienda che opera in ambito sanitario, civile ed industriale, offrendo servizi di facility management come:

- Sicurezza e vigilanza
- Movimentazione merci e logistica
- Pulizia e sanificazione

Nasce nel 1991 a Reggio Emilia dalla fusione di due aziende attive nel campo della vigilanza e della pulizia. In questi 30 anni, incorporando diverse realtà di tutto il territorio nazionale e non (nel 2015 acquisisce il 50% di un'azienda di San Paolo in Brasile) si è ampliata enormemente, tanto che ad oggi conta 25000 dipendenti e un miliardo di euro di fatturato.

In ambito sanitario Coopservice eroga servizi importantissimi per il perseguimento del Total Quality Management (o TQM, è un approccio alla gestione di un'organizzazione basato sul coinvolgimento di tutto il personale che mira ad ottenere il successo a lungo termine tramite la cura della qualità di ogni aspetto aziendale).
A partire da marzo 2020 questi servizi sono diventati fondamentali anche per contrastare l'avanzata del virus **garantendo le attività essenziali** di igiene e sanificazione, logistica, vigilanza e sicurezza negli ospedali.
Tali servizi permettono di avere una garanzia sulla salubrità degli ambienti e sulla sicurezza dell'attività sanitaria, contribuendo a salvaguardare la salute dei pazienti, degli ospiti e degli operatori e consentendo di intervenire tempestivamente in caso di eventi critici per la sicurezza. Inoltre, consentono di mantenere ad un alto livello di efficienza gli edifici, i locali e gli impianti, curando tutti quegli aspetti legati ai trasporti e alla logistica interna ed esterna.

La gestione della logistica gioca un ruolo sempre più importante nell'efficienza delle strutture sanitarie e ospedaliere. Avendo come propria missione la cura del paziente, esse debbono necessariamente affidare le attività non sanitarie ad aziende altamente

specializzate che ne condividano gli obiettivi, contribuendo al raggiungimento dei risultati prefissati.
L'outsourcing delle funzioni logistiche, oltre che ridurre i costi ed assicurare maggiore flessibilità, consente pertanto all'Azienda ospedaliera di concentrarsi sull'attività clinica, affidando ad un gestore professionale l'intera filiera del supply chain.
In particolare, si creano le condizioni sia per la migliore gestione e distribuzione di prodotti farmaceutici, dispositivi medico-chirurgici e prodotti economali, che per ottimizzare tutte le attività logistiche all'interno delle Aziende sanitarie (magazzini, archivi, trasporto dei pazienti, distribuzione di cibo e medicinali).
Per ricapitolare, i principali obiettivi che Coopservice si prefissa nell'ambito della logistica ospedaliera sono:

- Permettere all'Azienda ospedaliera di concentrarsi sulla propria mission di cura del paziente, affidando ad un gestore altamente professionale la filiera del supply chain.
- Consentire all'Azienda ospedaliera di ridurre i costi e incrementare la flessibilità delle attività logistiche interne.

In questo contesto si inserisce anche la relazione tra Coopservice e il Policlinico S. Orsola-Malpighi di Bologna.
L'area di Ricerca e sviluppo dell'azienda reggiana, durante questi mesi di pandemia, ha tenuto traccia degli spostamenti dei pazienti tra i diversi reparti e padiglioni del grande ospedale bolognese, accumulando una notevole quantità di dati in formato tabellare.
La collaborazione tra Unimore e Coopservice ha il fine di valorizzare questi dati, che rilegati all'interno delle loro tabelle risultano poco espressivi e poco comunicativi.
Per fare ciò è stato richiesto di sviluppare un'applicazione grafica per visualizzare il trasporto dei pazienti.

2.2 I dati

Il valore dei dati in possesso dell'area di *Ricerca e sviluppo* di Coopservice risiede nel fatto che gli spostamenti non hanno tutti lo stesso costo.
Ogni spostamento rappresenta un costo diverso a seconda dell'ubicazione dei reparti di partenza e di arrivo. In particolare, il costo risulta essere sensibilmente più alto negli spostamenti che avvengono tra due reparti che si trovano su piani distinti (spostamento verticale) rispetto a quelli che avvengono sullo stesso piano (spostamento orizzontale).
Inoltre, gli spostamenti di pazienti covid o sospetti tali, richiedono procedure che allungano notevolmente i tempi.

I dati sono mantenuti in un'unica tabella distribuita su 3 file Excel: il primo contiene gli spostamenti avvenuti tra il 1° marzo e il 1°agosto 2020, il secondo con quelli avvenuti tra il 2° agosto e il 31° dicembre 2020, e l'ultimo con gli spostamenti avvenuti a gennaio 2021.
I 3 file, che compongono la base di dati su cui si basa il progetto, contengono informazioni relative a 70.391 spostamenti.

Per ogni record sono sempre riportati alcuni campi, quali:

- **Data e ora di inserimento** all'interno del file. Spesso avviene qualche minuto dopo la chiusura delle operazioni.
- **Nominativo del paziente** che per motivi di privacy non è rappresentato dal nome e dal cognome completi, ma dall'iniziale del primo e le prime due lettere del secondo.
- **Modalità di trasporto,** che sono 7: barella, letto, carrozzina normale, deambulatore, barella obesi, carrozzina obesi e giro a vuoto.
- **Priorità** che può essere trasferimento, urgenza clinica, rientro, emergenza, urgenza organizzativa, non programmato, programmato, trasferimento con percorso programmato.
- **Tipologia di trasporto** che può essere trasferimento, esame strumentale, rientro, consulenza, multi-consulenza oppure altro.
- **Reparto / Unità operativa di partenza.**
- **Reparto / Unità operativa di arrivo.**
- **Trasportatore 1,** ovvero l'operatore sanitario addetto al trasporto del paziente.
- **Data e ora inizio** delle operazioni di trasporto (comprendenti la fase di preparazione del paziente).
- **Data e ora fine** operazione di trasporto.
- **Data e ora ingresso reparto Partenza** del trasportatore (o dei trasportatori).
- **Chiusura definitiva** delle operazioni di trasporto.
- **Orizzontale/verticale**

In aggiunta a queste colonne, che non hanno mai valore nullo nei record, sono presenti altri attributi che possono avere valore nullo:

- **Trasportatore 2**, presente solo in quegli spostamenti che appunto richiedo più di un operatore sanitario.
- **Note** varie, riferite ad aspetti atipici del paziente, dell'ambiente circostante o del trasporto.
- **Note centrale operativa,** ovvero provenienti dalla centrale operativa.
- **Data e ora presa in carico** del trasporto da parte degli operatori (a volte la registrazione della presa in cario può avvenire qualche minuto dopo l'ingresso nel reparto di partenza).
- **Data e ora uscita reparto Partenze** del paziente insieme ai trasportatori.
- **Data e ora ingresso reparto Arrivo** del paziente insieme ai trasportatori.
- **Data e ora uscita reparto Arrivo** dei trasportatori.

Per quanto riguarda le celle che rappresentano Data e ora, il formato dei numeri è "personalizzato" e i valori compaiono nella forma "gg/mm/aaaa hh/mm/ss".
Le altre celle sono stringhe.

2.3 Obiettivo del progetto

La richiesta avanzata a Unimore da Coopservice è stata quella di permettergli di visualizzare in modo compatto ed immediato tutti i dati che hanno registrato nei primi undici mesi di pandemia.
L'idea è quella di poter selezionare la data (o l'insieme di date) e visionare tutti gli spostamenti avvenuti all'interno di quel periodo, uno ad uno.

Per fare ciò si è deciso di raffigurare i pazienti e i loro trasportatori come pallini, che si muovono, in un dato momento della giornata selezionata, all'interno di uno spazio che deve rappresentare i diversi reparti del Policlinico S. Orsola-Malpighi.

In seguito ad un incontro con il Signor. Giorgio Zucchi, responsabile dell'area Ricerca e Sviluppo, è emerso che per l'azienda fosse interessante visualizzare anche altri aspetti degli spostamenti oltre al momento in cui avvengono e ai reparti di arrivo e di partenza.
Ad esempio, il tempo medio di lavoro effettivo degli operatori sanitari, ovvero quello in cui durante l'orario di servizio sono impegnati in un trasporto.

Il grande numero di reparti presenti nel Policlinico (circa 340) rende difficile rappresentarli tutti contemporaneamente all'interno di un unico spazio. Ci è stato quindi richiesto di mostrare di default i 4 padiglioni in cui è suddiviso, e permettere all'utente di selezionare quello di interesse e di decidere il piano all'interno del quale osservare gli spostamenti avvenuti. In questo modo è possibili anche accorgersi di quali sono gli spostamenti orizzontali e quali quelli verticali.

CAPITOLO 3 – Tecnologie Utilizzate

In questo capitolo verrà presentata una panoramica generale delle tecnologie più conosciute che ho utilizzato in questo progetto.
Le brevi descrizioni che seguiranno sono volte a presentare soltanto gli aspetti più significativi, con lo scopo di rendere più chiaro quanto descritto nel capitolo inerente al progetto.

Nel primo paragrafo sarà presentato Visual Studio Code, l'editor di codice sorgente che ho utilizzato.
I restanti paragrafi sono dedicati alla descrizione di alcuni aspetti rilevanti di tre linguaggi fondamentali nella tecnologia Web: *HTML, CSS e JavaScript.*

3.1 Visual Studio Code

Visual Studio Code è un editor di codice sorgente appartenente alla famiglia di prodotti Visual Studio di Microsoft, che lo ha rilasciato nel 2015.

Si tratta di un editor molto leggero e al tempo stesso potente, disponibile per Windows, Linux e MacOS. Può essere usato con diversi linguaggi di programmazione, tra cui quelli della famiglia C (C, C++, C#), Java, HTML, CSS, JavaScript, Python e molti altri.

Mette a disposizione funzionalità molto utili come il debugging, grazie a un supporto integrato per JavaScript e ad altri debugger facilmente installabili dal *market-place.*
Include inoltre servizi di *syntax highlighting* (o colorazione della sintassi), di *IntelliSense* e di *refactoring* del codice.

3.2 Html

L'*HTML* è il linguaggio che permette di formattare ed impaginare i documenti ipertestuali presenti sul Web.
È stato sviluppato da Tim Berners-Lee nel 1990, al CERN di Ginevra. Il suo nome è l'acronimo di *HyperText Markup Language,* ed è il linguaggio di *markup* più utilizzato.
I linguaggi di *markup* forniscono le regole e le linee guida generali che descrivono i meccanismi di rappresentazione del contenuto dei documenti Web, e permettono di visualizzarne le informazioni correttamente su tutte le tipologie dei calcolatori connessi alla rete.

Nel corso degli anni sono state rilasciate nuove versioni dotate di sempre più funzionalità, fino ad arrivare all'attuale quinta versione, rilasciata dal *World Wide Web Consortium* nel 2014.

Si tratta di un linguaggio gerarchico, ovvero in cui esistono dipendenze gerarchiche che rendono gli elementi l'uno antenato (o il genitore) dell'altro (discendente o figlio).
La descrizione dei contenuti avviene inserendo all'interno del testo alcune istruzioni dette *markup* o *tag* che producono le visualizzazioni e le azioni specificate.
Ogni *tag* viene convertito in un elemento che ha, come detto, dei rapporti gerarchici con gli altri, ed una posizione all'interno della struttura della pagina. Il modello che descrive come i vari elementi della pagina siano collegati tra di loro viene chiamato *DOM*, o *Document Object Model.*

La sintassi dell'*HTML* si basa proprio sul *tag html*, che ne rappresenta la struttura fondamentale: è una *keyword* racchiusa tra parentesi angolari, ed ha il compito di specificare il ruolo che ogni elemento all'interno della pagina *html* deve avere.

Alcuni esempi di funzionalità che gli elementi di una pagina html possono avere, e che ho utilizzato in questo progetto, sono:

- Collegamenti verso file di servizio esterni.
- Inserimento di contenuti interattivi (*script* o applicazioni esterne).
- Inserimento di immagini o pannelli *SVG.*
- Titolo, paragrafo, testo semplice, ecc.

3.3 CSS

Il *CSS*, acronimo di *Cascading Style Sheets*, è un linguaggio di programmazione usato per definire la resa grafica dei documenti scritti in HTML o XML.

Si è sviluppato parallelamente proprio all'*HTML*, che nelle intenzioni del *World Wide Web Consortium,* doveva essere estraneo alle problematiche legate allo stile e alla presentazione delle pagine web.
La tecnologia destinata a rispondere a queste esigenze è appunto il *CSS,* la cui prima regolamentazione risale al 1996.

Le classi CSS sono usate per specificare attributi grafici come font, dimensione, colore, spaziatura, bordo e posizione degli elementi all'interno di una rappresentazione web.
I fogli di stile CSS si applicano a cascata: tutte le regole sono valide, ma prevale l'ultima regola, la più specifica.

3.3 JavaScript

Se HTML rappresenta la struttura di una pagina web e CSS ne rappresenta lo stile, JavaScript è la tecnologia che permette di aggiornare dinamicamente il contenuto, gestire file multimediali o immagini animate, e "*pretty much everything else*", come sostengono alcuni programmatori.

JavaScript, o JS, è stato ideato nel 1995 da Netscape, con lo scopo di fornire dinamicità alle pagine HTML, grazie alla possibilità di manipolare i documenti senza dover coinvolgere il server.
La necessità crescente di rendere il web sempre più interattivo ha fatto nascere negli stessi anni tecnologie concorrenti, che in un primo momento sembrava dovessero imporsi su JavaScript in virtù della loro maggiore potenza: è il caso di Flash e degli Applet Java.
La chiave di volta che ha permesso a JS di spopolare è stata Ajax: si tratta di una tecnologia multi-piattaforma che consente di comunicare con il server in modo asincrono, tramite lo scambio di dati in background, permettendo l'aggiornamento dinamico della pagina Web senza che l'utente debba ricaricarla esplicitamente.

Atri due aspetti che hanno permesso a JS di avere successo sono la grossa mole di API a disposizione dei programmatori, e soprattutto l'avvento di Node.js, ovvero un ambiente che ha permesso a JavaScript di girare anche sui server, rendendolo utilizzabile anche per la programmazione server-side, oltre che per quella client-side per la quale è stato originariamente progettato.

JavaScript è un linguaggio interpretato, quindi lento ma agilmente modificabile.
Gli interpreti più moderni, per migliorarne le prestazioni, utilizzano una tecnica chiamata *"just-in-time compiling*", che consiste nel compilare a *run time* il codice sorgente in *bytecode*, che viene eseguito più velocemente.

La sintassi del JavaScript è abbastanza simile a quella di Java, con alcune differenze volte a rendere il primo più flessibile.
Ad esempio, in JS esistono solo cinque tipi di dato primitivi, e inoltre le variabili non devono essere obbligatoriamente dichiarate (anche se dichiararle è una pratica consigliabile).

Per aggiungere del codice JavaScript all'interno di una pagina HTML esistono tre modalità. Tutte prevedono di utilizzare il tag HTML "*script*".

1. JavaScript interno: il codice JS viene inserito all'interno dei tag *script* di apertura e di chiusura.

```
<script>

// JavaScript goes here

</script>
```

2. JavaScript esterno: il codice è scritto in un file esterno con estensione "js". Il tag script, in questo caso, è utilizzato per includere nel file HTML tale file, specificando la sua posizione sul file system locale o sul Web. Ad esempio, se il file contenente il codice si chiama script.js, avremo:

```
<script src="script.js"> </script>
```

3. JavaScript *inline*: il codice JS è associato ai diversi elementi del DOM direttamente nei loro *tag,* dopo la *keyword* "*onclick*". Il codice può trovarsi all'interno di un blocco script, o addirittura nel *tag* stesso.

   ```
   <button onclick=”exampleFunction()”>Click me!</button>
   ```

   Questa terza modalità è sconsigliata e considerata una *bad practice* di programmazione, in quanto "inquina" l'HTML con il JavaScript.

CAPITOLO 4 - La libreria D3.js

Questo capitolo è dedicato ad approfondire la libreria D3.js, la tecnologia meno conosciuta tra quelle che ho utilizzato in questo progetto, e che allo stesso tempo mi ha permesso di realizzare la parte a mio avviso più interessante: la visualizzazione dei dati.

Il primo paragrafo servirà a fornire un'introduzione alla libreria. Si parlerà di alcune delle sue caratteristiche e dei vantaggi che nell'utilizzarla.

I restanti paragrafi serviranno a descrivere brevemente le funzionalità messe a disposizione da D3 che ho maggiormente sfruttato.

4.1 Introduzione a D3.js

D3 è l'acronimo di *Data Driven Document,* ed è una libreria *open-source* di JavaScript, sviluppata Mike Bostock a partire dal 2011, che consente di manipolare documenti basati su dati.
Questa tecnologia permette di dare vita ai dati realizzando visualizzazioni interattive e personalizzabili, nei browser Web, utilizzando HTML, SVG e CSS.
Grazie all'uso degli standard del Web, sono garantite tutte le funzionalità dei browser moderni, senza la dipendenza da un *framework* proprietario.
D3 nasce dalla convinzione che la rappresentazione visiva dei dati sia la più efficiente, e la più adatta a veicolare le informazioni significative, ricavate da grosse quantità di dati.

Tra le caratteristiche che rendono popolare e utile questa libreria, queste sono le principali:

- Utilizza gli standard Web.
- È orientata ai dati: può interagire facilmente con dati statici, oppure con server remoti, in diversi formati.
- Consente di manipolare il DOM, anche dinamicamente, in base ai dati che vengono forniti.
- Consente di modificare le proprietà degli elementi del DOM dinamicamente: i dati possono determinare lo stile di ogni elemento della pagina.
- Agevola l'animazione e l'interazione: è infatti disponibile un importante supporto per le animazioni, grazie a funzioni che gestiscono le transizioni tra più stati, permettendo di controllare la durata, il ritardo, è l'attenuazione del movimento. Queste funzioni sono molto reattive agli input dell'utente.

In virtù delle sue caratteristiche, utilizzare D3 porta diversi vantaggi:

- Essendo una libreria in JavaScript, consente di essere utilizzata insieme a qualsiasi *framework* di JS.
- È *open-source,* quindi personalizzabile a seconda delle esigenze del programmatore, aggiungendo qualche riga di codice al sorgente.

- Lavorando con gli standard del Web non ha bisogno di *plugin* o particolari strumenti di debugging. Bastano un browser e i *tool* di debugging per JavaScript.
- Consente di avere un controllo quasi totale sulla visualizzazione dei dati, che può essere fortemente personalizzata.
- Essendo leggero, reattivo e molto veloce, consente di lavorare molto bene anche con grandi basi di dati.

4.2 Manipolazione degli elementi del DOM

Manipolare gli elementi della pagina è sicuramente una delle azioni che più spesso capita di dover compiere, in applicazioni di visualizzazione dei dati. D3 fornisce un modo molto utile e veloce per selezionare e modificare i diversi elementi del DOM.

Per poter selezionare gli elementi si utilizzano due metodi dell'oggetto "d3", che prendono come parametro un selettore che serve a specificare l'elemento. Sono:

- select(): che ritorna il primo elemento che fa match con il selettore passato come parametro
- selectAll(): che ritorna tutti gli elementi del documento che fanno match con il selettore.

Per riferirsi ad un elemento specifico si possono utilizzare selettori CSS, CSS Class Name, oppure gli Id degli elementi.

Una volta selezionato l'elemento che si desidera manipolare, è possibile utilizzare diversi metodi per modificarlo. Alcuni esempi sono:

- text(""): che setta il testo nell'elemento selezionato
- append("element name"): che inserisce un elemento appena prima la fine dell'elemento selezionato
- insert("element name"): che inserisce un elemento all'interno dell'elemento selezionato
- attr("name", "value"): che setta il valore dell'attributo specificato
- style("name", "value"): che setta lo stile dell'elemento.

Tra queste funzioni, ci sono anche quelle che permettono di animare i suoi elementi.
D3 semplifica il concetto di "animazione" con quello di "transizione" da una forma (o uno stato) ad un'altra. Infatti, il metodo che indica l'inizio di un'animazione è transition().

Di seguito all'invocazione di questo metodo, vengono specificati i parametri che rappresentano lo stato finale che avrà l'elemento al termine della transizione, e possono essere applicate altre funzioni che specificano alcune caratteristiche della transizione. Le più note sono:

- duration(): che specifica la durata della transizione.

- ease(): che specifica la tipologia di movimento durante la transizione.
- delay(): che specifica il ritardo dell'animazione in millisecondi.

4.3 Data Binding e Data Loading

Poiché D3 è una tecnologia orientata ai dati, infatti esistono diverse funzioni per la gestione delle informazioni.
In particolare, quando si gestiscono dati con D3, si parla di due tecniche a seconda che i dati si trovino all'interno di variabili locali o in file esterni: *Data Binding*, e *Data Loading*.

1. La prima riguarda il *binding* (legame) tra i dati presenti all'interno di variabili (spesso vettoriali) e i diversi elementi del DOM, che vengono aggiornati di conseguenza.
   Per il *data binding* esistono diversi metodi, il principale è data(), che crea una corrispondenza tra gli elementi selezionati e i dati passati come parametro.

2. Il *data loading* invece, consente di caricare all'interno del programma dati da file esterni di diversi formati. D3 mette a disposizione quattro metodi a seconda del formato del file in cui si trovano i dati:
   - d3.csv()
   - d3.json()
   - d3.tsv()
   - d3.xml()

   Tutte queste funzioni accettano due parametri: il primo rappresenta la risorsa che identifica il file (sul *file-system* o in rete), mentre il secondo rappresenta la funzione di *callback* che li gestisce.

Capitolo 5 – Il progetto

In questo capitolo parlerò del mio personale contributo alla realizzazione del progetto; infatti, non essendo l'unico studente coinvolto, il mio compito non era quello di terminarlo, ma piuttosto quello di costruire le basi che altri avrebbero poi potuto sfruttare per arrivare a realizzare il prodotto finale.
Nello specifico mi sono occupato di come visualizzare i dati nelle modalità richieste da Coopservice, ovvero come punti mobili all'interno di uno spazio che rappresentasse il Policlinico S. Orsola.

Una volta presentato il quadro generale da cui sono partito, sarà fatta una panoramica del lavoro che ho svolto.

Alla fine del capitolo verrà fatto qualche accenno al procedimento da me seguito per gestire i dati.

4.1 Approccio iniziale

Prima di poter iniziare il lavoro di stesura del codice vero e proprio, è stata necessaria una fase di studio delle componenti fondamentali per la realizzazione del progetto.
In particolare, mi riferisco al linguaggio di programmazione *Javascript* e alla libreria *d3.js*, con le quali non mi ero mai interfacciato prima.
L'indicazione di utilizzare queste tecnologie è stata fornita dal Professor. Giovanni Simonini che, conoscendole accuratamente ed essendo informato in modo puntuale sulle richieste di Coopservice, ha valutato che fossero particolarmente confacenti al progetto.

Per approfondire le mie conoscenze in merito, oltre a fare affidamento a corsi online e alla ricca documentazione trovata in rete, ho implementato alcuni programmi che simulavano e imitavano (in modo inizialmente molto semplificativo) l'applicazione finale, realizzando dei prototipi.
In particolare, lo scopo di questi esercizi era prendere confidenza con *SVG*, il formato delle planimetrie del Policlinico S. Orsola-Malpighi, su cui in principio sarebbero dovuti avvenire gli spostamenti dei punti.

L'*SVG* (Scalable Vector Graphics) è una tecnologia per il rendering di immagini sulle pagine web, a partire da un testo. La sua struttura è simile a quella dell'*HTML.*
All'interno di una pagina web, un pannello *SVG* risiede nel *DOM,* e le sue proprietà possono essere specificate come attributi.

4.2 Primi prototipi

Il primo step è stato quello di implementare un'applicazione *"Moving dots"* su un *SVG*.
I punti dovevano essere inseriti all'interno dell'*SVG* a partire da un array di semplici oggetti, che avevano come attributi soltanto le coordinate iniziali e finali.
Questo primo prototipo è stato realizzato sfruttando alcuni metodi della libreria *d3.js*:

```
d3.create("svg");
```

➔ che crea nel DOM un pannello SVG.

```
circle = d3.append("circle")
            .attr("r", 5)
            .attr("cx", points[i].x_in)
            .attr("cy", points[i].y_in);
```

➔ per aggiungere un elemento circle di raggio 5 all'*SVG*, nelle sue coordinate iniziali.

```
circle.transition()
                .duration(3000)
                .attr("cx", points[i].x_fin)
                .attr("cy", points[i].y_fin);
```

➔ Per far muovere I punti verso le loro coordinate finali, in un tempo di 3000 ms.

Per lo step successivo, ovvero mettere in relazione i movimenti dei pallini e il tempo, è stato aggiunto:

- un campo *time* agli oggetti *punti*, che fa riferimento al momento in cui avviene lo spostamento
- uno *slider temporale* all'*SVG*, per rappresentare il progredire del tempo.

Tramite una variabile che viene posta uguale al valore presente sulla barra temporale, si verifica ogni secondo che il campo time di tutti i dots non sia scaduto, e se il punto non è stato mosso (facilmente deducibile utilizzando una variabile booleana), si chiama la funzione *move()* o *moveBack()*, a seconda dei casi.

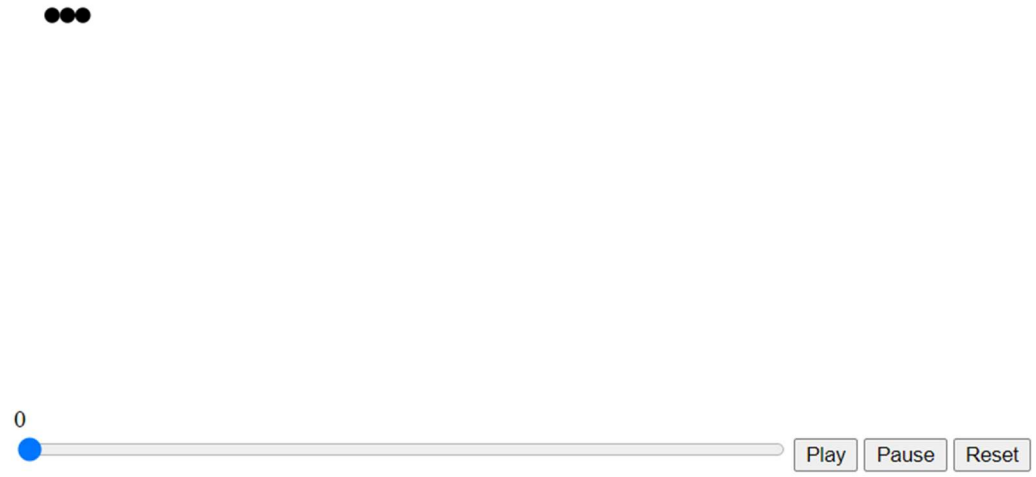

Figura 1. Al secondo 0 i pallini si trovano alla posizione iniziale

I tre button consentono all'utente di interagire con l'applicazione.
Il loro funzionamento è intuitivo, pertanto sottolineo soltanto che, una volta messo in pausa lo slider, è possibile modificare il valore numerico raffigurante il tempo trascinandolo.

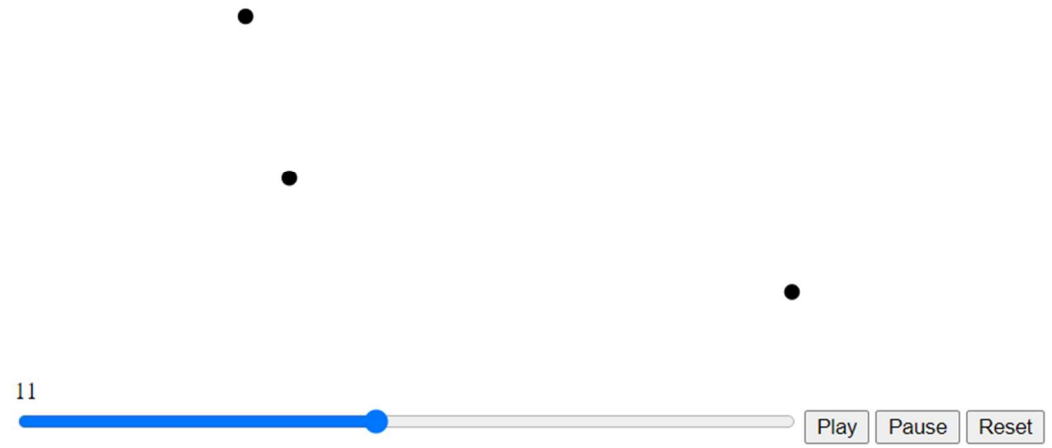

Figura 2. Dopo 11 secondi tutti i pallini si sono spostati alla posizione finale

4.3 Sviluppo del prototipo

Una volta realizzato il semplice prototipo sopra descritto, i next-step concordati con il prof. Simonini sono stati:

1. Creare una corrispondenza tra i reparti e le coordinate nell'*SVG*.
2. Inserire attributi più specifici, per consentire di raffigurare i pallini in maniera più esplicativa (ad esempio colorare i pallini in base all'operatore che ha effettuato lo spostamento).
3. Creare una rappresentazione distinta per la visualizzazione dello spostamento dei pazienti e degli operatori.
4. Inserire un date-picker.

1. Per quanto riguarda il primo punto, seguendo le indicazioni del professore, ho accantonato l'idea di inserire una planimetria dell'ospedale, optando invece per una raffigurazione dei vari reparti tramite etichette disposte in cerchio, e contenenti il nome dei reparti.
   Questo tipo di rappresentazione prende spunto da "*A Day in the life of Americans*", un progetto di Nathan Yau, uno statistico americano.
   In questa fase non ho considerato il problema di dover rappresentare la totalità dei reparti poiché abbiamo deciso che l'utente non li avrebbe mai visualizzati tutti contemporaneamente, ma avrebbe scelto il piano e il padiglione, e visualizzato soltanto quelli relativi. Mi sono pertanto concentrato sulla realizzazione di un'applicazione in cui si possono visualizzare una dozzina di etichette.

   I reparti sono rappresentati da oggetti che hanno tre attributi:
   - indice
   - nome completo
   - nome breve

   Tali oggetti sono contenuti all'interno di un array popolabile in base alla parte dell'ospedale che si desidera visionare.

   Le coordinate vengono assegnate a questi oggetti, con una funzione che divide una circonferenza in tante parti uguali quante sono le etichette da rappresentare, ed infine assegna ad essi ascissa ed ordinata con banali calcoli trigonometrici.
   La rappresentazione di questi dati all'interno del DOM è stata realizzata grazie a una delle funzionalità più interessante di *d3.js*: il *data binding*, già presentato nel capitolo di approfondimento sulla libreria.
   Il nome del reparto riportato non è quello completo: dal momento che alcuni nomi risulterebbero troppo lunghi, si utilizza una notazione abbreviata che non contiene riferimenti al piano o al padiglione in cui si trova.

Ad esempio, se il nome intero è “PAD2 P5 - REP. MALATTIE INFETTIVE”, ciò che verrà visualizzato sarà “Malattie Infettive”.
Oltre al nome, abbiamo deciso di riportare la percentuale, aggiornata progressivamente, dei trasporti indirizzati verso quel reparto rispetto alla totalità dei trasporti visualizzati.

2. Per quanto riguarda il secondo punto, sono stati aggiunti negli oggetti raffiguranti i pallini due fields:
   - colore
   - nome del trasportatore

   Il vantaggio dell’aggiungere il primo attributo sta nel fatto che diventa possibile rappresentare un’informazione aggiuntiva in maniera immediata, focalizzando l’attenzione di chi guarda su un qualsiasi dato che si vuole evidenziare. Ad esempio, per quanto riguarda il movimento dei pazienti, abbiamo deciso di colorare di rosso i pallini che si spostano verso un reparto covid, e di nero tutti gli altri.
   Aggiungendo il nome del trasportatore, invece, è possibile raggruppare tutti i record che lo hanno coinvolto, utilizzando un unico pallino per tutti i suoi spostamenti.
   Nella visualizzazione attinente ai trasportatori, abbiamo deciso di combinare tra di loro i due nuovi attributi, e di utilizzare un colore diverso per ogni operatore sanitario.
   Per capire a quale operatore si riferisce ogni colore, ho inserito una legenda che riporta il nome di ogni trasportatore in servizio, coerentemente colorato.

3. Il terzo step è stato facilmente realizzato, riscrivendo il codice e modificandone solo alcune parti. Si noti che questa non è la soluzione ottimale, e non deve pertanto essere quella definitiva: dal punto di vista ingegneristico, infatti, la duplicazione del codice è una pratica da evitare dal momento che rende più difficile e costosa la manutenzione, aumentando la dimensione del codice e richiedendo modifiche ripetute in diverse parti del programma nel caso in cui una parte del codice duplicato debba essere modificata. Nello specifico, le parti che cambiano a seconda delle due visualizzazioni sono quelle inerenti alla logica che sta dietro alla colorazione dei pallini, e alla rappresentazione dei reparti.

4. Per quanto riguarda il date-picker, ho trovato una libreria open source online all’indirizzo https://www.daterangepicker.com/#example1 ed ho modificato alcuni aspetti dello style per renderlo più conforme a quanto già esistente del programma.

Arrivati a questo punto il prototipo appare così:

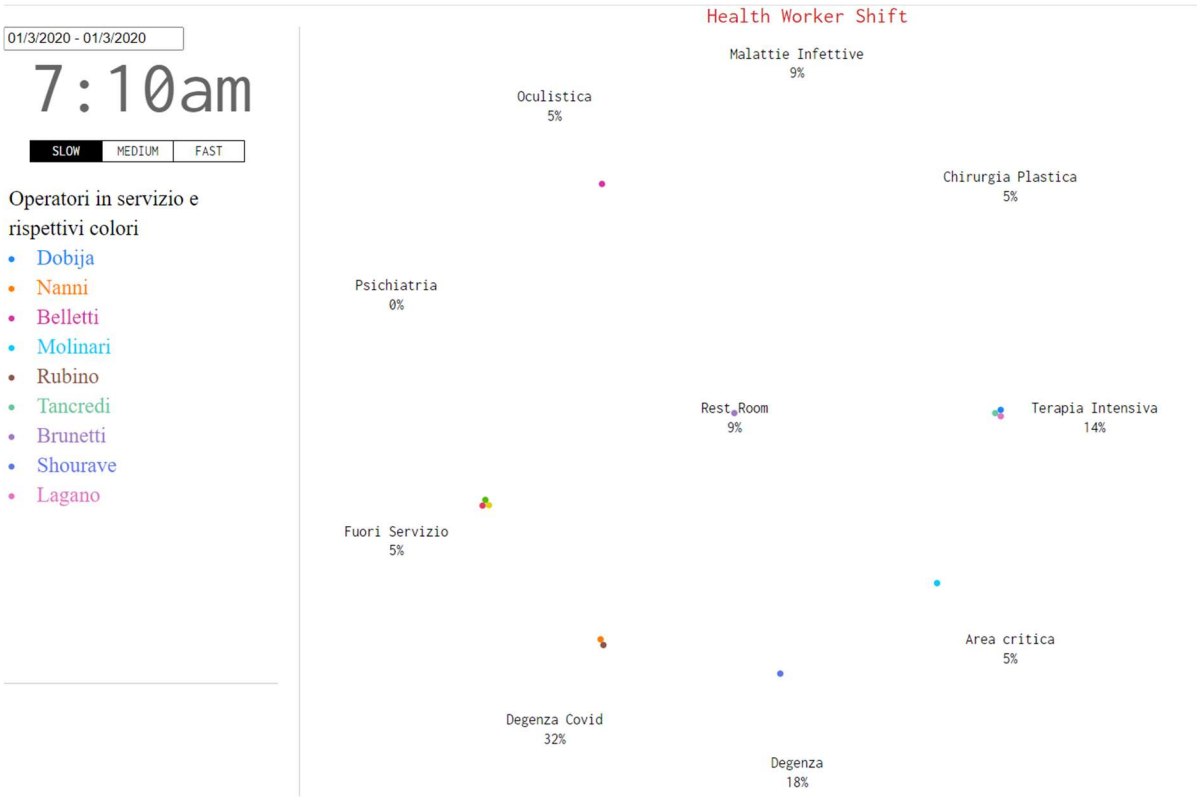

Figura 3. Visualizzazione movimenti trasportatori

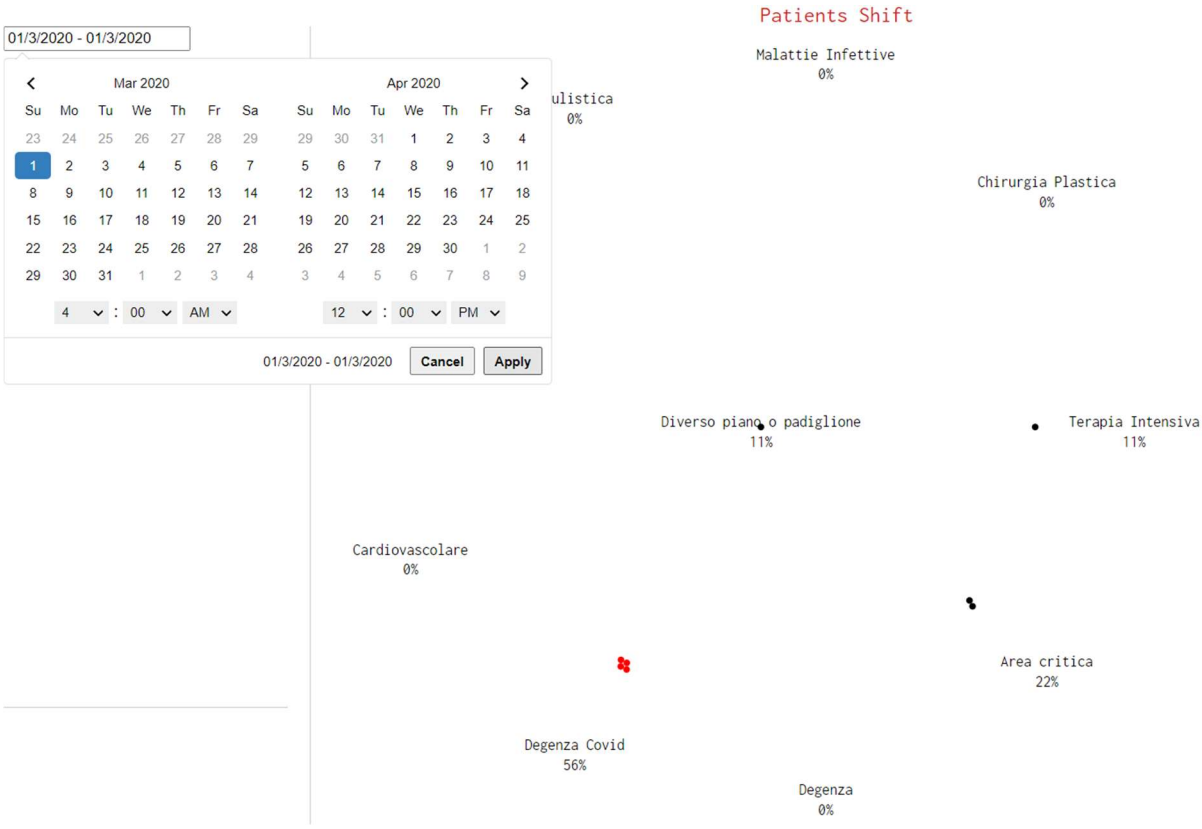

Figura 4. Visualizzazione spostamenti dei pazienti e scelta delle date di inizio e fine osservazione

4.4 Gestione dei dati

Nel programma “A Day in the life of Americans” il formato del file contente i dati, relativi agli spostamenti di mille cittadini americani nell’arco di una giornata, è un *TSV* (Tab-separeted value). Il file è un semplice insieme di stringhe, una per ogni riga, ognuna rappresentante un cittadino. Tutte le stringhe sono composte da una successione di valori numerici: quelli di posto pari rappresentano gli indici dei reparti, indicano quindi l’etichetta presso cui devono trovarsi i pallini, mentre i valori di posto dispari indicano il tempo che deve passare prima che avvenga lo spostamento verso l’indice successivo.

Come accennato nell’introduzione di questo capitolo, il mio compito non era quello di capire come gestire i dati sfruttandone tutto il potenziale, ma quello di fornirne una rappresentazione nel modo richiesto da Coopservice. Per questo motivo, e considerato quanto sia banale convertire un foglio Excel in un file *TSV*, ho deciso di trattare i dati a mia disposizione come se fossero in una forma simile a quella dei dati dell’applicazione sopra citata.
Nel caso dei dati inerenti al Policlinico S. Orsola, ogni linea fa riferimento ad un solo trasporto invece che ad un insieme di trasporti, ma contiene molte informazioni aggiuntive: per il mio prototipo ho deciso di ignorarle e di considerare solo orario di inizio, reparto di partenza e reparto di arrivo, lasciando ai colleghi che si occupano della gestione dei dati le decisioni definitive inerenti al loro formato e alla tipologia di interazione con essi.

Per il data loading ho utilizzato il metodo della libreria d3 *tsv()*, che accetta due parametri: l’URL o il pathname che identifica il file da cui si vogliono caricare i dati, e una funzione di callback.
I dati vengono passati alla funzione sotto forma di oggetti, contenuti in un vettore. Il vettore ha tanti elementi quante sono le righe del file (eccetto la prima, che contiene il nome delle colonne).
Gli attributi degli oggetti sono tanti quante le colonne, di cui portano il nome.
Esistono tre funzioni analoghe che permettono di caricare i dati da file di altrettanti formati diversi: *CSV*, *JSON* ed *XML*.
Mentre il primo è molto simile a *TSV*, gli ultimi due formati permettono di eseguire query più complesse.

CAPITOLO 6 - Conclusioni e lavoro futuro

In quest'ultimo capitolo presenterò alcune considerazioni sul lavoro svolto.
Verrà fornita una breve panoramica su quello che è stato fatto, per poi parlare di alcuni degli aspetti da tenere in considerazione per concludere il progetto.

6.1 Conclusioni

L'elaborato che ho presentato, come già detto nel corso dei precedenti capitoli, si occupa di una parte del progetto: la rappresentazione di dati presenti in un file TSV, nelle modalità richieste da Coopservice.
Dopo un'ampia fase di studio delle tecnologie, per me nuove, che ho descritto nei precedenti capitoli, sono stati realizzati prototipi sempre più vicini alle richieste di Coopservice. A partire da una semplice applicazione "punti mobili", in cui dei pallini si spostano in un dato momento dalle coordinate iniziali a quelle finali, fino all'ultimo prototipo in cui ogni reparto è associato ad una coordinata sullo schermo, e vengono rappresentate informazioni diverse a seconda che l'utente scelga di visualizzare lo spostamento dei pazienti o dei trasportatori.

Si può concludere che:

- Una volta appresi i concetti base inerenti alla struttura delle pagine HTML e la semantica di JavaScript, è facile capire come utilizzare D3, e come sfruttare la maggior parte delle funzionalità che mette a disposizione.
- La libreria D3 si è rivelata essere idonea a lavorare con la quantità di dati utilizzata, rendendo il programma responsive e prestante.
- Il programma realizzato non è in grado di rispondere a tutte le esigenze avanzate da Coopservice, e non può pertanto rappresentare il progetto concluso.

6.2 Lavoro futuro

Allo stato attuale, gli aspetti da valutare per concludere il progetto sono per lo più quelli inerenti all'interazione coi dati ed alla loro interpretazione.
Più in particolare, sarà necessario valutare quale modalità tra quelle esistenti è la migliore per la lettura dei dati forniti da Coopservice.
Le opzioni, ad esempio, possono essere quella di trasformare le tabelle Excel in file TSV ed applicare dei filtri in fase di *data loading,* oppure quella di utilizzare librerie come jQuery, o tecnologie come AJAX. O ancora, un'opzione potrebbe essere quella di sfruttare il formato *Json* o XML.

Per prendere questa decisione bisognerà valutare anche in che momento popolare il vettore con gli spostamenti, ovvero quando eseguire le *query* verso il file contenente i dati.

Un'idea potrebbe essere quella di inserire i valori nell'array quando l'utente cambia l'intervallo di date di interesse tramite il *data picker*, oppure quando l'utente cambia la sezione dell'ospedale che desidera visionare.
Un'altra possibilità è quella di popolare l'array ogni volta che l'utente interagisce con il programma in una delle due modalità di cui sopra.
Questi aspetti impattano sulla scelta della modalità di interazione perché variano la frequenza con cui si interroga la base di dati.

Una volta fatta questa scelta, bisognerà realizzare il codice che permette di "navigare" tra i diversi piani e padiglioni del Policlinico S. Orsola-Malpighi, mostrando le etichette dei reparti nel piano che l'utente ha scelto di visionare.

Da ultimo va considerato che allo stato attuale, i dati all'interno dei file Excel, non sono ancora sfruttati a pieno. Molte informazioni, ad esempio il tempo effettivo di lavoro giornaliero medio dei trasportatori, oppure il piano maggiormente coinvolto negli spostamenti, sono deducibili dalle tabelle, e potrebbero essere molto interessanti per Coopservice.

SITOGRAFIA

1. Coopservice. www.coopservice.it/
2. Servizi di Coopservice www.coopservice.it/en/hospital-logistics
3. Html, JavaScript, Css. www.html.it/ developer.mozilla.org/
4. Visual Studio Code code.visualstudio.com/
5. D3. d3js.org riptutorial.com/ebook/d3-js
6. Flowing Data flowingdata.com/

RINGRAZIAMENTI