

# Correzione e generazione automatica di esercizi di architettura degli elaboratori

Raffaele Tartaglione

Facoltà di Ingegneria dell'Informazione, Informatica e Statistica

Corso di laurea in Informatica



## Sommario

L'oggetto di questo tirocinio nonché dello studio della tesi è la progettazione e lo sviluppo di un'applicazione web il cui compito è quello di correggere e generare automaticamente esercizi di architettura degli elaboratori incentrati sulle cache.

Inizialmente la prima direzione intrapresa è stata quella di limitarsi a simulare il normale funzionamento di una cache ed a visualizzare tale simulazione tramite l'utilizzo di tabelle.

Si è quindi realizzata una semplice applicazione web dove una form richiedeva i dati riguardanti la cache richiesta e restituiva la cache sottoforma di tabella completata correttamente.

Tuttavia, successivamente si è deciso di plasmare questo incipit in qualcosa di più specifico e di concreta utilità, ovvero che potesse essere fruttuoso sia per uno studente che per un docente.

Oltre dunque alla generazione automatica, rappresentata dalla funzionalità precedentemente espressa con l'aggiunta dell'introduzione di determinate miglioni, si è introdotta anche l'opzione di sceglierne una ulteriore: la correzione, dove l'utente dovrà riempire una tabella sempre dopo aver completato il form fornitogli. Questa si occuperà non solo di verificare quanti errori siano stati commessi, ma anche di fornire una spiegazione dettagliata riguardo allo sbaglio fatto.

La relazione è divisa in quattro parti fondamentali: l'introduzione allo scenario di riferimento; l'analisi dei requisiti, volta a migliorare la modellazione delle funzionalità della piattaforma; la progettazione del sistema e della sua architettura; i dettagli dello sviluppo e delle fasi di implementazione.

## Storyboard

Come accennato, il primo obiettivo proposto è stato quello di rappresentare in base a dati specifici forniti dall'utente una cache sottoforma di tabella. Si è scelto come linguaggio di programmazione a tale scopo per la semplice piattaforma iniziale Python, e Flask come framework di applicazioni web.

Dopo aver visionato le potenzialità a cui l'idea iniziale poteva guidarci, si è deciso che il successivo passo sarebbe stato quello di portare il progetto verso uno prossimo stadio, prevedendo oltre alla generazione automatica di esercizi di architettura degli elaboratori inerentemente alle cache, anche la funzione di correzione di questi. Una volta visionati testi di teoria e determinate tipologie di attività, è stata completata una prima versione primitiva del programma, che consentiva all'utente di scegliere a quale delle due funzionalità affidarsi.

In seguito, si è raffinato il codice, separando la parte legata al generare e gestire le pagine web da quella che invece si occupava di costruire e correggere la cache. Per la parte incentrata su Flask, inoltre, si è deciso di convertire le funzioni in classi, più propriamente in viste (view), cosicché fosse permesso di raggruppare variabili e funzioni in maniera logica, riutilizzabile ma soprattutto ampliabile. Anche la cache viene rappresentata all'interno del programma sottoforma di classe, in quanto dovrà ospitare al suo interno sia i dati che la

rappresentano, quali ad esempio gli indirizzi o il numero di set e di vie che la caratterizzano, sia le funzioni che si occupano di edificarla e di correggerla.

I vantaggi enunciati pocanzi sono stati altrettanto ottenuti quando anche le fasi che contraddistinguono una cache, ovvero numero di blocco, indice, tag, offset (lasciato però come campo opzionale), Hit o Miss e di che tipo di Miss si è trattato, sono state trasformate in classi. Ciò però ha portato in questo caso ad un punto a favore ulteriore: ha consentito di evolvere l'algoritmo che si occupava di correggere la cache. Al tempo di questa ulteriore miglione, si è deciso di perfezionare anche il come lo sbaglio venisse comunicato all'utente, aggiungendo al già presente conteggio degli errori la presenza all'interno della pagina web corrispondente di commenti mirati alla fase di appartenenza di ciascun errore, raggruppati in base alla tipologia commessa e prendendo atto di eventuali casistiche specifiche.

Un'aggiunta che, al contrario di quest'ultima, ha influenzato invece entrambe le funzionalità finora proposte è stata quella di inserire la possibilità da parte dell'utente di scegliere se volesse che la cache che andava a generare o su cui voleva mettersi alla prova partissero o meno da uno stato iniziale. Dopodiché si è proceduti con l'aggiunta di una nuova funzionalità, ovvero l'introduzione di una vera e propria simulazione del normale funzionamento di una cache, dove con il solo premere un pulsante si può osservare come cella per cella si vada a costruire ogni tabella.

Infine, l'ultimo passo è stato di completare anche il lato estetico del progetto, evidenziando e giustificando ogni passaggio, ogni errore ed ogni aggiornamento avvenuto nelle varie tabelle. (QUEST'ULTIMO DA AMPLIARE)



# Indice

1. Introduzione
  - a. Infarinatura generale sulla memoria di un elaboratore
  - b. Che cos'è una cache
  - c. Tipologie di cache
  - d. Tipologie di miss
  - e. La necessità della piattaforma
2. Analisi dei requisiti
  - a. Identificazione degli utenti
  - b. Selezione della tipologia di funzioni
3. Progettazione
  - a. Architettura
4. Implementazione
  - a. Flask
  - b. Descrizioni classi
  - c. Le pagine realizzate
5. Conclusioni
  - a. Future migliorie possibili

Bibliografia



# Introduzione

## Infarinatura generale sulla memoria di un elaboratore

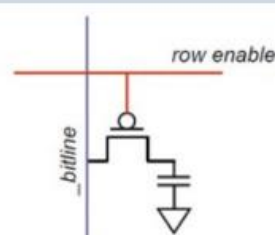
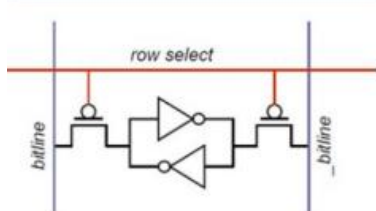
Prima di partire descrivendo la struttura e il funzionamento di una cache, è bene introdurre prima almeno le conoscenze di base riguardo la memoria all'interno di un elaboratore elettronico.

Inizio citando un passaggio da “*Discussione preliminare sul progetto logico di un dispositivo elettronico di calcolo*”, opera scritta da A.W. Burks, H.H. Goldstine e J. Von Neumann:

*“L’ideale sarebbe avere una memoria con capacità indefinitamente grande in modo che ciascuna specifica ... parola fosse immediatamente disponibile .... Siamo ... costretti a riconoscere la possibilità di costruire una gerarchia di memorie, ciascuna delle quali dotata di maggior capacità della precedente, ma accessibile meno velocemente.”*

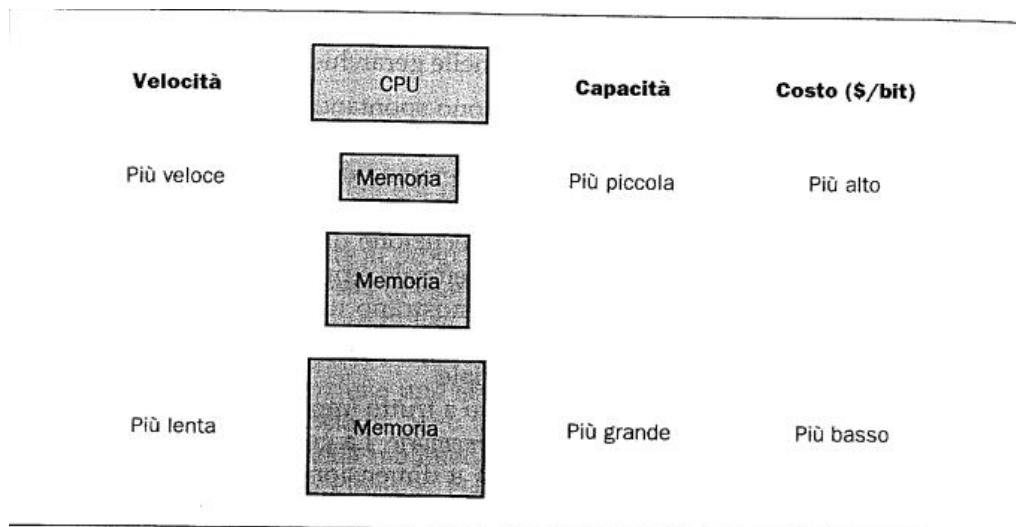
A tal proposito, è bene sottolineare che le prestazioni all'interno di un calcolatore elettronico sono limitate dai tempi di accesso ai dati. La sua memoria principale è realizzata utilizzando la tecnologia DRAM (Dynamic Random Access Memory, ovvero memoria dinamica ad accesso casuale), molto lenta in termini di tempi di accesso ma molto economica per quanto riguarda i costi materiali. Tuttavia, esistono anche tipologie di memorie più veloci: le SRAM (Static Random Access Memory), le quali al contrario sono caratterizzate da accessi più rapidi ma costi più alti.

SRAM	DRAM
Accessi più rapidi	Accesso più lenti
Costi più alti	Costi più bassi
Maggiori consumi di corrente (refresh frequenti dei dati memorizzati)	Minori consumi di correnti (i condensatori immagazzinano l'informazione per più tempo)
Circuiteria complessa	Circuiteria semplice
Bassa densità delle celle di memoria sul chip di memoria	Alta densità delle celle di memoria sul chip di memoria





Viste le differenze di costo e di tempi di accesso, risulta quindi essere estremamente vantaggioso costruire un sottosistema di memoria come una gerarchia di livelli, con la memoria più veloce vicina al processore e quella più lenta e meno costosa ai livelli inferiori, come mostrato in figura.



L'obiettivo è di presentare all'utente tanta memoria quanta ne sia disponibile nella tecnologia meno costosa con una velocità di accesso pari a quella offerta dalla tecnologia più veloce.

Dato quindi un livello di questa gerarchia, il suo contenuto è un sottoinsieme di quello di qualunque altro livello che sia più distante dal processore, mentre al livello più basso sono memorizzati tutti i dati. Inoltre, man mano che ci si allontana dal processore, i livelli richiedono un tempo sempre maggiore per l'accesso. Naturalmente, poiché tutti i programmi spendono la maggior parte del loro tempo accedendo alla memoria, il sottoinsieme di memoria è necessariamente un fattore determinante ai fini delle prestazioni.

L'obiettivo della gerarchia di memoria è dunque quello di dare al programmatore l'illusione di poter usufruire di una memoria al tempo stesso veloce (idealmente, quanto la memoria al livello più alto) e grande (quanto quella al livello più basso).

Vista la differente dimensione diventa necessario durante l'esecuzione dei programmi trasferire informazione fra memorie di livelli diversi. Anche se una gerarchia di memorie è in genere composta da più livelli, le informazioni sono di volta in volta copiate solo tra due livelli adiacenti.

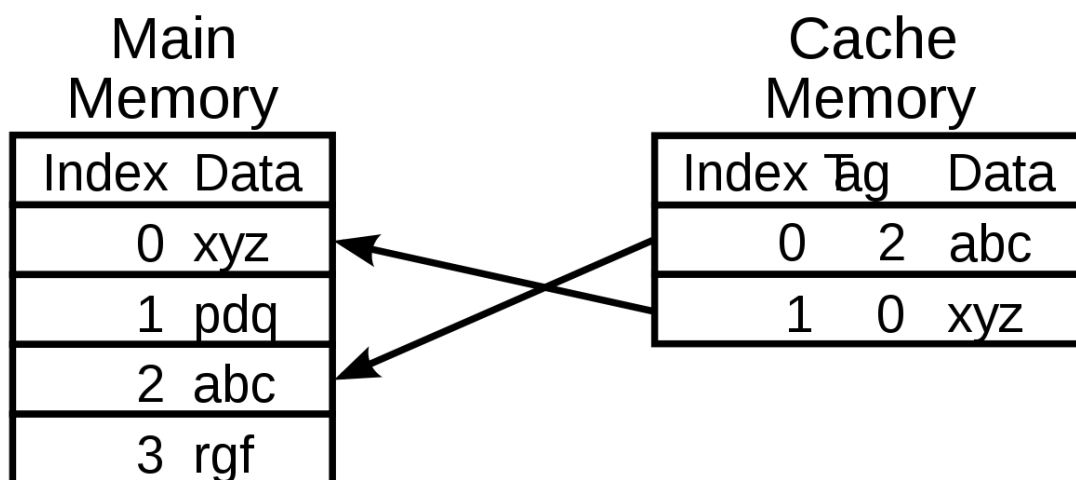
## Che cos'è una cache

Con il termine cache, in informatica, si indica un dispositivo hardware, o più precisamente un'area di memoria estremamente veloce ma solitamente di un basso ordine di grandezza di capacità, ideato per velocizzare le operazioni di accesso alla memoria (sia in lettura sia in scrittura) da parte del processore e di conseguenza velocizzare l'esecuzione dei programmi.

La *cache*, infatti, è un componente hardware che, affiancato alla memoria principale del sistema (ovvero il disco rigido), serve a immagazzinare dati utili a eseguire con maggior velocità le future richieste di accesso alla memoria effettuate dal processore. All'interno della memoria *cache* possono essere conservati dati e valori appena “trattati” dal processore o duplicati di dati conservati su altri supporti di memoria presenti nella macchina (come, per l'appunto, il disco rigido) e che, per un motivo o per l'altro, potranno essere richieste a breve dall'unità centrale di calcolo.

Quando è necessario l'accesso ad un dato, una copia di questo viene prima cercata nella *cache*: se è presente e valida, viene utilizzata tale copia; altrimenti il dato viene recuperato dalla memoria principale, e memorizzato nella *cache*, nel caso possa servire successivamente.

Dal punto di vista logico la *cache* è costituita da un *pool* di “entrate” nelle quali archiviare il *datum* (i pacchetti dati) e un *tag* identificativo che specifichi la corrispondenza tra il *datum* e i pacchetti dati “originali” conservati sulla memoria principale del sistema. Quando un *client cache* (che può essere il processore, il disco rigido o il browser web) ha necessità di accedere a dei dati, effettua prima di tutto una scansione della *cache*. Nel caso in cui nella memoria “di appoggio” sia presente il *tag* dei dati cercati, questi saranno utilizzati nell'esecuzione dell'operazione in corso (*cache hit*).



Una semplice analogia può illustrare più efficacemente i principi fondamentali finora descritti ed i meccanismi su cui si basa la realizzazione dei sistemi di memoria. Si prenda in considerazione uno studente che deve fare una ricerca e si reca in biblioteca. Lo studente siede ad una scrivania con alcuni libri che ha preso dagli scaffali e che sta esaminando. Tuttavia, nota di non avere libri che trattino un determinato argomento; lo studente quindi li andrà a prendere. Una volta che lo studente avrà davanti a sé sulla scrivania una buona selezione di libri è probabile che molti degli argomenti di cui avrà bisogno saranno in essi contenuti, e potrà spendere la maggior parte del tempo consultando i testi sulla scrivania senza dover ritornare sugli scaffali. La possibilità di avere parecchi libri davanti a sé fa risparmiare tempo rispetto a poterne avere uno solo, in quanto si dovrebbe continuamente tornare agli scaffali per riportare il libro e prenderne un altro.

Lo stesso principio permette proprio di creare l'illusione di una memoria di grandi dimensioni a cui si può accedere con la stessa velocità di una piccola memoria, come descritto più nel dettaglio nel capitolo precedente: come non era necessario accedere a tutti i testi della biblioteca con la stessa probabilità, così un programma non ha la necessità di accedere a tutto il suo codice ed a tutti i suoi dati con egual probabilità. In caso contrario non sarebbe possibile rendere veloci la maggior parte degli accessi in memoria e contemporaneamente avere grandi quantità di memoria nei calcolatori, come sarebbe impossibile tenere tutti i libri della biblioteca sulla scrivania ed avere qualche possibilità di trovare velocemente l'argomento cercato.

Questo principio è detto di località e dice che in un dato istante di tempo i programmi fanno accesso ad una porzione relativamente piccola del loro spazio di indirizzamento (verificato analizzando l'esecuzione di un gran numero di programmi), così come lo studente faceva accesso ad una piccola porzione dei libri della biblioteca.

Vi sono due tipi diversi di località:

- Località temporale: quando si fa riferimento a un elemento di memoria, c'è la tendenza a far riferimento allo stesso elemento entro breve (es.: il riutilizzo di istruzioni e dati contenuti nei cicli)
- Località spaziale: quando si fa riferimento a un elemento di memoria, c'è la tendenza a far riferimento entro breve tempo ad altri elementi che hanno indirizzo vicino a quello dell'elemento corrente (es.: eseguita un'istruzione si tende ad elaborare quella immediatamente successiva; quando si accede a dati organizzati in vettori o matrici, l'accesso a un dato è seguito dall'accesso al dato immediatamente successivo, etc.)

È proprio il giocare sul principio di località che caratterizza la memoria di un calcolatore ad essere realizzata come una gerarchia di memoria.

## Tipologie di cache

La memoria cache è vista come organizzata in linee.

Quando la CPU richiede un indirizzo:

- Se il dato richiesto dalla CPU fa parte di uno dei blocchi presenti nella cache di livello superiore, si dice che la richiesta ha avuto successo (HIT)
- Se invece il dato non si trova nel livello superiore, si dice che la richiesta fallisce (MISS). In questo caso per trovare il blocco che contiene i dati richiesti, bisogna accedere al livello inferiore della gerarchia e trasferire il blocco corrispondente al livello superiore

Occorre quindi definire le possibili modalità per consentire ad ogni parola della memoria indirizzabile di trovare (dove necessario) una posizione della cache in cui possa essere trasferita – in altre parole, occorre definire una corrispondenza tra l'indirizzo in memoria della parola e la locazione nella cache.

A questo scopo, sono state definite tre soluzioni diverse:

1. Cache a indirizzamento diretto (direct mapped cache);

2. Cache set-associativa a n-vie (n-way set associative cache);
3. Cache completamente associativa (fully associative cache).

#### Cache direct mapped:

Ogni blocco viene messo in una linea fissata (che dipende dal numero del blocco).

PRO: hardware più semplice e meno costoso; è facile determinare in quale linea cercare il dato.

CONTRO: blocchi diversi mappano nella stessa linea, se gli accessi a questi blocchi si alternano si ottengono molti MISS e si crea il fenomeno del trashing.

#### Cache completamente associativa:

Un blocco può essere messo in una linea qualsiasi.

PRO: massima flessibilità

CONTRO: hardware più complesso e più costoso

#### Cache set-associativa a n-vie:

Le linee sono suddivise in S gruppi (set) formati ciascuno da n elementi, o vie, e ogni blocco è disposto in una qualsiasi delle linee del set fissato (che dipende dal numero del blocco).

### **Tipologie di miss**

Un accesso alla cache può dare MISS per tre motivi (in una cache con W ways e S sets):

- **Cold start:** è la prima volta che quel blocco viene richiesto. Va dunque necessariamente caricato per forza all'interno della cache;
- **Conflict:** il blocco è stato sovrascritto per via del grado di associatività della cache e sarebbe stata una HIT se la cache fosse stata fully-associative. Quindi se il blocco è stato richiesto meno di  $W \cdot S$  volte prima in una fully-associative, allora avremmo avuto una HIT;
- **Capacity:** il blocco è stato sostituito ma non sarebbe stato possibile avere una HIT nemmeno se la cache fosse stata fully-associative. Quindi se il blocco è stato richiesto più di  $W \cdot S$  volte prima in una fully-associative, allora avremmo avuto una MISS

Le proprietà delle miss della cache per lo stesso set di dati e vari tipi di cache sono le seguenti:

- Le miss di cold start avvengono con la stessa frequenza in tutte le cache di tipo direct mapped, set associative e fully associative;

- Le miss di conflitto avvengono più spesso nelle cache di tipo direct mapped, con una frequenza media nelle cache di tipo set associative e non avvengono mai nelle cache di tipo fully associative;
- Le miss di capacità avvengono di rado nelle cache di tipo direct mapped, con una frequenza media nelle cache di tipo set associativo e più spesso nelle cache di tipo fully associative.

## **La necessità della piattaforma**

Lo sviluppo di un'applicazione web capace di correggere e generare automaticamente esercizi di Architettura degli elaboratori sulle cache è stato ritenuto necessario tanto per gli studenti quanto per i docenti.

Nel primo bacino di utenza, l'utilità che la piattaforma acquisisce risiede nel consentire allo studente di poter ottenere un feedback immediato riguardo alla risoluzione di quesiti costruiti con i dati forniti in input. Verrà infatti visualizzato, oltre alla quantità di errori, anche l'andamento complessivo che terrà conto di una eventuale diffusione di un dato sbagliato, in quanto sono presenti calcoli e formule che si approvvigionano nei dati precedentemente calcolati, sottoforma di commenti mirati agli errori commessi, esplorando anche casistiche specifiche. Così facendo, si cerca il più possibile di mettere davanti allo studente l'attuale condizione della sua preparazione, attenuando dove possibile, per fornire un'esperienza totalmente personalizzata e mirata.

Ancor più utile potrebbe risultare per loro usufruire della funzione di simulare il funzionamento di una cache, dove tutti i passaggi vengono presentati man mano che si costruisce la tabella cella dopo cella.

Per quanto riguarda invece l'utenza dei docenti, questa troverà estremamente utile la generazione automatica degli esercizi. Vista infatti la natura complessa e la prolissità che risiedono nella creazione ex novo di un esercizio destinato a far parte di un esame, si è soliti riproporre tracce simili tra loro. Si rischia di conseguenza di portare lo studente a imparare a memoria determinati passaggi anziché sfruttare solamente il proprio ragionamento e ciò che si è studiato per risolvere un esercizio. Grazie dunque a tale funzionalità, si avrà a disposizione una quantità illimitata di items da proporre, generati unicamente mediante i semplici input forniti.

Ma forse il giovamento più grande che è possibile trarre è da ritrovarsi nella possibilità tra gli input di poter scegliere di influenzare la tabella finale, che andrà a rappresentare la cache su cui si baserà l'esercizio, imponendo che sia presente una determinata condizione. Questa, ad esempio, può essere la necessità che ci sia tra i tipi di miss una miss di capacità o di conflitto, qualora il docente voglia testare le conoscenze dei suoi allievi in maniera mirata.

# Analisi dei requisiti

## Identificare gli utenti

Inizialmente sono stati raccolti ed analizzati i requisiti principali dell'applicazione, così da poterne definire ulteriori funzionalità e guidare le successive fasi dello sviluppo.

Coloro che interagiranno con il sistema vengono suddivisi in:

- Utente studente
- Utente docente

Gli utenti docenti potranno accedere alla funzione di generazione automatica degli esercizi sulle cache, immettendo nei campi dei moduli HTML le caratteristiche volute e ricevendo nella pagina web seguente la tabella rappresentante la cache avente tutti i campi riempiti correttamente. I docenti potranno anche imporre determinate condizioni e situazioni alla cache che desidereranno ottenere, in particolare se questa debba presentare necessariamente almeno una miss di conflitto, una miss di capacità, oppure si potrà scegliere di non richiedere nulla di specifico; in tal caso la presenza o meno di tali tipologie di miss verrà lasciato interamente al caso.

Gli utenti studenti invece potranno accedere alla funzione di correzione delle cache, dove verranno presentati con la descrizione dell'esercizio a loro proposto e dovranno completare la tabella, resa editabile, con i dati corretti. Qui gli studenti potranno scegliere se risolvere l'esercizio come se fosse un vero e proprio esame, oppure risolverlo prendendosi il proprio tempo. Nel primo caso la cache che gli verrà presentata davanti dovrà essere completamente riempita e, una volta sottomessa, verrà comunicato all'utente la quantità di errori commessi insieme anche alla sicuramente più importante descrizione di che tipo di errore si è commesso più frequentemente. Inoltre, per migliorare ancor di più l'esperienza dell'utente, gli sarà riproposta la medesima tabella dove sono stati individuati gli errori commessi, evidenziandone in rosso le celle dove essi risiedono. Presa dunque coscienza delle proprie imprecisioni, lo studente potrà ritentare l'esercizio una seconda volta, evitando quindi di inserire nuovamente tutti i dati dell'esercizio per poter riprovarci finché non si sentirà soddisfatto. Nel secondo caso, invece, lo studente avrà modo di verificare la correttezza del completamento della tabella cella per cella, quindi non dovrà aspettare di completare interamente la tabella ma potrà verificare immediatamente che ha commesso un errore e magari impedirgli di commetterne altri dello stesso tipo nelle fasi successive.

Entrambe le tipologie di utenti avranno la possibilità di scegliere qualora volessero costruire una cache partendo da una iniziale. In tal caso compariranno altre due tabelle oltre a quella della cache, le quali rappresenteranno la cache iniziale stessa e la cache finale. Quest'ultima sarà già riempita nel caso ci si trovi nella funzione di generazione automatica degli esercizi, mentre dovrà essere completata correttamente qualora ci trovassimo invece nel caso della correzione della cache, in quanto rientrerà nell'insieme dei dati che subiranno il controllo per la verifica e la valutazione, e influirà di conseguenza sul giudizio finale.

Ad affiancare la generazione automatica e la correzione di esercizi di Architettura degli elaboratori vi è anche la funzionalità di simulare il comportamento di una cache cella per cella. Tramite l'ausilio di un pulsante, lo studente potrà quindi ricostruire la tabella a poco a

poco, vedendo ogni passaggio preso singolarmente ed evidenziato. In tal modo la presenza di un determinato dato viene giustificata agli occhi dell'utente, in base alla fase in cui esso si trova. Ad esempio, qualora si dovesse trovare nella fase del tipo di miss e nella cella evidenziata ci sia una miss di capacità o di conflitto, il programma farà comparire una tabella rappresentante la cache come fully associative in maniera tale da mettere in evidenza la distanza tra i due indirizzi che ha portato alla manifestazione di quel particolare tipo di miss. La finalità, dunque, del motivo di questa costruzione è quella di fornire allo studente un'edificazione di una tecnica di risoluzione degli esercizi sì metodica, ma che percepisca come utile per le proprie basi riguardo alla tipologia del quesito richiesto.

Un'ulteriore efficacia è da ritrovarsi infine proprio nella risoluzione degli esercizi stessa, in quanto questa funzionalità si è dimostrata perfetta anche per quanto riguarda la composizione dell'algoritmo per la correzione, essendo sia all'altezza tanto dell'utente quanto del programmatore. Infatti, il costruire la tabella cella dopo cella ha permesso di poter constatare l'effettiva correttezza dei dati dell'algoritmo designato al compito.

Le celle prese singolarmente hanno consentito l'eliminazione di una visione d'insieme che avrebbe potuto lasciar passare inosservate imprecisioni all'occhio umano che sarebbero potute rimanere presenti eventualmente anche nel prodotto ultimato.

Tuttavia, il beneficio più grande è stato tratto dalla programmazione della parte inerente alla generazione automatica di esercizi. Infatti, fintanto che, per testare l'algoritmo, si sono utilizzati esempi di esercizi già completati e corretti, non sono nate problematiche. Quando si è trattato di evolversi e generarne di randomici invece, proprio la casualità e la mancanza di riferimenti esterni per verificarne la correttezza hanno aumentato i tempi di revisione. Grazie quindi al simulatore, si è ottenuta l'accuratezza degli esercizi generati impiegando meno tempo.

## **Selezione della tipologia di funzione**

Le funzioni presentate nelle pagine web saranno quelle di generazione automatica e di correzione di esercizi. In base a quale funzione si sceglierà, si avrà davanti una pagina web richiedente i dati necessari per proseguire con la funzionalità richiesta, come ad esempio nel caso della generazione automatica si potrà specificare la tipologia di cache (direct mapped cache, n-way set associative cache e fully associative cache), il tipo di dato (byte, word, half word) e così via. Si giungerà poi alla pagina web dedicata interamente alla tabella, la quale rappresenterà la cache richiesta correttamente completata o da completare a seconda del tipo di utente.

Tuttavia, non sempre i dati richiesti in input dalle due funzionalità saranno gli stessi; ciò deriva dalla natura stessa di queste ultime. Infatti, nel caso della generazione automatica, il fulcro della funzionalità risiede nel creare ex novo una lista di indirizzi tali da adempiere sia alle informazioni fornite dall'utente, sia soprattutto alla condizione imposta per poter rispecchiare situazioni ben definite. Saranno quindi richiesti necessariamente dati quali la quantità di indirizzi voluta fino a giungere alla possibilità di scegliere qualora la cache debba presentare un particolare tipo di miss oppure no.

Nel caso invece della correzione, il focus risiede per la maggior parte nella pagina web finale, dove l'utente potrà mettere al vaglio le proprie conoscenze e i propri ragionamenti e

potrà vederli concretizzati grazie ad una valutazione ad hoc del proprio percorso. Il programma infatti sarà in grado di ricevere l'esercizio svolto dall'utente mediante una o più tabelle da completare, sempre a seconda delle scelte e delle informazioni fornite in input, di correggerle in maniera puramente dinamica onde poter raggiungere un grado di efficienza valido, e di restituire non solo la quantità di errori commessi e dei commenti riguardo le lacune mostrate, ma persino di valutarli sotto un modello di gradazione diverso da quello comunemente usato. Sarà di fatto richiesto di ridurre la propagazione di un eventuale errore commesso per riuscire ad esaminare nella maniera più appropriata alle doti autentiche dell'utente il suo andamento. Questo traguardo è stato raggiunto nel modo precedentemente nominato: grazie sempre ad un approccio dinamico di costruzione della cache, si è potuto intervenire direttamente sulla cella sbagliata e ad agire in modo consono alla situazione verificatasi con azioni mirate.

Inoltre, un'ulteriore richiesta soddisfatta dal programma è quella di non limitarsi banalmente a comunicare allo studente, ad esempio, di aver commesso un errore per quanto riguarda il calcolo dell'index, ma piuttosto di riscontrare se l'errore avvenuto sia individuabile tra vari campioni di errori più comunemente commessi durante gli esami. Infatti, a seguito di uno studio accurato, si è deciso di "venire incontro" ancor di più nei confronti dello studente e della sua esperienza educativa, fornendogli quindi un aiuto quanto più concreto possibile. Ad esempio, si è visto che tra gli errori più frequenti è presente l'inversione, durante i calcoli, dell'utilizzo del valore corrispondente all'index con quello invece del tag, e che quindi varrà la pena prenderne atto e assicurarsi di farlo presente qualora se ne trovasse riscontro.

Infine, si è deciso anche di optare per una versione più user-friendly della generazione automatica di esercizi, concentrandosi sulle modalità con cui la tabella rappresentante la cache venisse mostrata e fornita all'utente, che ha portato dunque alla realizzazione di un vero e proprio simulatore di cache. Si è infatti scelto, simulando il funzionamento di una cache, di rappresentarla cella per cella, giustificando ogni passaggio compiuto, facendo effettivamente vedere all'utente come la cache si costruisse e come memorizzasse e sostituisse i valori che si trovano al suo interno, mentre una tabella tiene conto di quanto avviene e dei dati necessari a tutto ciò. Tutto ciò ha il compito di facilitare l'edificazione di una cache e quindi di mostrare un metodo sì metodico ma completo e soddisfacente della risoluzione di un esercizio.

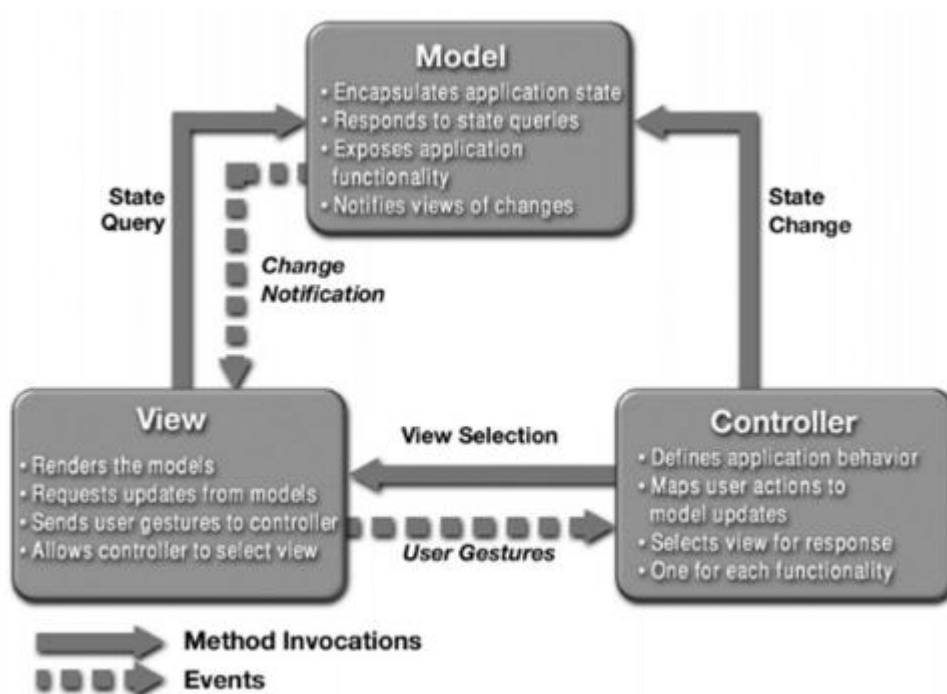


# Progettazione

## Architettura

Il progetto si basa sul pattern architetturale **Model-view-controller (MVC)**, o in italiano **modello-vista-controllo**). Il pattern è basato sulla separazione dei compiti fra i componenti software che interpretano tre ruoli principali:

- il *model* fornisce i metodi per accedere ai dati utili all'applicazione;
- la *view* visualizza i dati contenuti nel model e si occupa dell'interazione con utenti e agenti. In questo caso avremo una view per l'utente studente ed una view per l'utente docente.
- il *controller* riceve i comandi dell'utente (in genere attraverso la view) e li attua modificando lo stato degli altri due componenti.



I vantaggi apportati dal **pattern MVC** sono molteplici e riguardano soprattutto la riusabilità del codice.

La parte del **model** racchiude le risorse dell'applicazione web, e protegge il resto dell'applicazione da eventuali cambiamenti.

La **view** include solo la parte della presentazione del sito permettendo di cambiare il look e il feel del sito agendo solo su di essa.

La parte del **controller** racchiude invece la business-logic e quindi regole legate alla visualizzazione delle informazioni e alla memorizzazione dei dati.

È grazie a questa separazione di compiti proposta dall'approccio MVC che si favorisce il riuso del codice grazie a soluzioni già pronte ai problemi comuni, un aumento della produttività, oltre a facilitare la manutenzione del software, la sua scalabilità, uniformità e standard.

Nello specifico i **vantaggi** sono i seguenti:

- Esiste la possibilità di scrivere viste e controllori diversi, utilizzando lo stesso modello di accesso ai dati e quindi, come già accennato, riutilizzare parte del codice già steso in precedenza. Infatti, la separazione di model e view permette a molte view di usare gli stessi oggetti del modello.
- L'utilizzo di un modello rigido e di regole standard nella stesura del progetto facilita un eventuale lavoro di manutenzione e si agevola la comprensione anche da parte di altri programmatori.
- La possibilità di avere un controllore separato dal resto dell'applicazione rende la progettazione più semplice e permette di concentrare gli sforzi sulla logica del funzionamento.

# Implementazione

## Flask

Per la realizzazione di questa applicazione web si è scelto di utilizzare **Flask**.

Flask è un framework di applicazioni web scritto in Python, scelto come linguaggio di programmazione, basato sul toolkit Werkzeug WSGI e sul motore di template Jinja2. Il Web Server Gateway Interface (WSGI) è stato adottato come standard per lo sviluppo di applicazioni Web Python. WSGI è una specifica per un'interfaccia universale tra il server web e le applicazioni web. Jinja invece è un popolare motore di creazione di template veloce, espressivo ed estensibile per Python. Un sistema di modelli Web combina un modello con una determinata origine dati per eseguire il rendering di pagine Web dinamiche.

Flask è spesso indicato come un micro-framework, in quanto mira a mantenere il nucleo di un'applicazione semplice ma estensibile. Flask non ha un livello di astrazione integrato per la gestione del database, né ha un supporto per la convalida, ma supporta le estensioni per aggiungere tale funzionalità all'applicazione.

Per osservarne e descriverne sia le sue funzionalità che il suo essere intuitivo, trovo che l'approccio migliore sia quello di mostrare le sue caratteristiche tramite un semplice ma esaustivo esempio.

Avvalendomi infatti delle parole di Lucio Anneo Seneca:

*“La via per imparare è lunga se si procede per regole, breve ed efficace se si procede per esempi”.*

Per iniziare, si prenda il seguente codice introduttivo:

```
from flask import Flask
app = Flask(__name__)

@app.route('/')
def hello_world():
    return 'Hello World'

if __name__ == '__main__':
    app.run()
```

Da queste poche righe è già possibile trarre sia dei fattori essenziali sia delle importanti riflessioni:

- La presenza dell'importazione del modulo flask nel progetto è obbligatoria.
- Un oggetto della classe Flask sarà la nostra applicazione **WSGI**.
- Il costruttore Flask prende il nome del **modulo corrente** (`__name__`) come argomento.
- La funzione **route()** della classe Flask è un decoratore, che dice all'applicazione quale URL deve chiamare la funzione associata.

- L'URL **'/'** è associato alla funzione **hello\_world()** . Quindi, quando la home page del server web viene aperta nel browser, verrà visualizzato l'output di questa funzione.
- Infine il metodo **run()** della classe Flask esegue l'applicazione sul server di sviluppo locale.

I moderni framework Web utilizzano la tecnica di routing per aiutare un utente a ricordare gli URL dell'applicazione. È utile accedere direttamente alla pagina desiderata senza dover navigare dalla home page.

Il decoratore **route()** in Flask viene utilizzato per associare l'URL a una funzione. Ad esempio:

```
@app.route('/hello')
def hello_world():
    return 'hello world'
```

Qui, la regola dell'URL **'/hello'** è legata alla funzione **hello\_world()** . Di conseguenza, se un utente visita **http://localhost:5000/hello** URL, l'output della funzione **hello\_world()** verrà visualizzato nel browser.

La funzione **add\_url\_rule()** di un oggetto dell'applicazione è disponibile anche per associare un URL con una funzione, come nell'esempio precedente, viene utilizzata **route()** .

È possibile restituire l'output di una funzione legata a un determinato URL sotto forma di HTML. Ad esempio, nello script seguente, la funzione **hello()** renderà **'Hello World'** con il tag **<h1>** allegato.

```
from flask import Flask
app = Flask(__name__)

@app.route('/')
def index():
    return '<html><body><h1>Hello World</h1></body></html>'

if __name__ == '__main__':
    app.run(debug = True)
```

Tuttavia, la generazione di contenuto HTML dal codice Python è ingombrante, specialmente quando è necessario inserire dati variabili ed elementi del linguaggio Python come condizionali o loop. Ciò richiederebbe frequenti “escape” dall'HTML.

È qui che si può sfruttare il motore di template **Jinja2**, su cui si basa Flask. Invece di restituire raw HTML dalla funzione, un file HTML può essere visualizzato grazie alla funzione **render\_template()**.

```
from flask import Flask
app = Flask(__name__)

@app.route('/')
def index():
    return render_template('hello.html')
```

```
if __name__ == '__main__':  
    app.run(debug = True)
```

Flask proverà a trovare il file HTML nella cartella dei modelli, nella stessa cartella in cui è presente questo script.

- Cartella dell'applicazione
  - Hello.py
  - templates
    - hello.html

Il termine **'web templating system'** si riferisce alla progettazione di uno script HTML in cui i dati variabili possono essere inseriti dinamicamente. Un sistema di modelli Web comprende un motore di modelli, una sorta di origine dati ed un elaboratore di modelli.

Flask utilizza il motore di template **jinja2**. Un modello Web contiene segnaposto intervallati dalla sintassi HTML per variabili ed espressioni (in questi casi espressioni Python) che vengono sostituiti valori quando viene eseguito il rendering del modello.

Il codice seguente viene salvato come **hello.html** nella cartella dei modelli.

```
<!doctype html>  
<html>  
  <body>  
  
    <h1>Hello {{ name }}!</h1>  
  
  </body>  
</html>
```

Quindi, esegui il seguente script dalla shell Python.

```
from flask import Flask, render_template  
app = Flask(__name__)  
  
@app.route('/hello/<user>')  
def hello_name(user):  
    return render_template('hello.html', name = user)  
  
if __name__ == '__main__':  
    app.run(debug = True)
```

La parte **variabile** dell'URL viene inserita nel **segnaposto {{ name }}**

Il motore del modello **jinja2** utilizza i seguenti delimitatori per l'escape dall'HTML.

- {% ... %} per gli estratti conto
- {{ ... }} per le espressioni da stampare sull'output del modello
- {# ... #} per i commenti non inclusi nell'output del modello
- # ... ## per le istruzioni di riga

Nell'esempio seguente viene illustrato l'uso dell'istruzione condizionale (if) nel modello. La regola URL per la funzione **hello()** accetta il parametro intero. Viene passato al modello **hello.html**. Al suo interno viene confrontato il valore del numero ricevuto (segni) (maggiore o minore di 50) e di conseguenza l'HTML viene renderizzato condizionalmente.

Lo script Python è il seguente:

```
from flask import Flask, render_template
app = Flask(__name__)

@app.route('/hello/<int:score>')
def hello_name(score):
    return render_template('hello.html', marks = score)

if __name__ == '__main__':
    app.run(debug = True)
```

Lo script del modello HTML di **hello.html** è il seguente:

```
<!doctype html>
<html>
  <body>
    {% if marks>50 %}
      <h1> Your result is pass!</h1>
    {% else %}
      <h1>Your result is fail</h1>
    {% endif %}
  </body>
</html>
```

Si noti che le istruzioni condizionali **if-else** e **endif** sono racchiuse nel delimitatore **{%..%}**.

È proprio grazie a tali delimitatori che sarà possibile, sempre mediante Jinja, rappresentare le tabelle, tra cui quelle che andranno a raffigurare le cache, in maniera dinamica.

I dati dalla pagina Web di un client vengono inviati al server come oggetto di richiesta globale. Per elaborare i dati della richiesta, è necessario importarli dal modulo Flask.

Gli attributi importanti dell'oggetto richiesta sono elencati di seguito –

- **Form** - È un oggetto dizionario contenente coppie chiave e valore di parametri del modulo e relativi valori.
- **args** - contenuto analizzato della stringa di query che fa parte dell'URL dopo il punto interrogativo (?).
- **Cookie** - oggetto dizionario contenente nomi e valori dei Cookie.
- **files** – dati relativi al file caricato.
- **method** – metodo di richiesta corrente.

Per quanto riguarda invece le pagine web HTML vere e proprie, a sostegno delle funzionalità di queste è possibile trovare script sia in Javascript che in JQuery, senza i quali non sarebbe stato possibile applicare e consentire operazioni ben definite, come ad esempio la presenza

o meno di un'area di testo qualora una determinata opzione venga selezionata, o il definire funzioni che devono essere invocate quando un pulsante viene premuto.

## Descrizioni classi viste

Flask introduce viste collegabili che si basano su classi anziché su funzioni. L'intenzione principale è che si possa sostituire parti delle implementazioni e in questo modo avere viste collegabili personalizzabili.

Una vista basata su classi è una classe che funge da funzione di vista. Poiché si tratta di una classe, è possibile creare diverse istanze della stessa con argomenti diversi, per modificare il comportamento della vista. Questo è anche noto come visualizzazioni generiche, riutilizzabili o collegabili.

Quando si dispone di una visualizzazione basata sulla classe, sorge la domanda a cosa “self” punti. Il modo in cui funziona consiste nel fatto che quando la richiesta viene inviata, viene creata una nuova istanza della classe e il metodo **dispatch\_request()** viene chiamato con i parametri della regola URL. La classe stessa viene istanziata con i parametri passati alla funzione **as\_view()**.

```
from flask.views import View

class ListView(View):
    def get_template_name(self):
        raise NotImplementedError()

    def dispatch_request(self):
        return render_template(self.get_template_name())
```

Per ciascuna sottoclasse di ListView, si è proceduto sovrascrivendo la funzione **get\_template\_name()**, inserendo il nome della pagina Jinja a cui si fa riferimento e, ad esempio nel caso della sottoclasse TableGenerateView, si è dovuta sovrascrivere anche la funzione **dispatch\_request()**, che invece permette di prendere i valori dalle pagine web, trattarli in base alla funzionalità e infine fornirli alla pagina web successiva.

### CODICE DA COMPLETARE E SISTEMARE

```
class TableGenerateView(ListView):
    def get_template_name(self):
        return 'tableg.jinja'

    def dispatch_request(self):
        args = request.args
```

```

if request.method == 'POST':
    args = request.form

indirizzi = args.get('indirizzi')
addresses = indirizzi.split()

cache_type = args.get('cache_type')
data_type = args.get('data_type')
lvl = int(args.get('lvl',1))

#AGGIUNGERE ALTRI DATI INPUT DI MAIN

global cache_table
cache_table = cache.Cache(addresses, cache_type, data_type, lvl)
cache_table.riempicache()
dati = cache_table.data

for l in range(len(dati)):
    dati[l][0].insert(0, "Block#")
    dati[l][1].insert(0, "Index")
    dati[l][2].insert(0, "Tag")
    dati[l][3].insert(0, "Hit/Miss")
    dati[l][4].insert(0, "Tipo miss")

addresses.insert(0, "Address")

tables = []
for i in range(lvl):
    tables.append((i+1,addresses, dati))
return render_template(self.get_template_name(), tables=tables)

```

Inoltre, tutto ciò si è potuto implementare mediante l'ausilio del metodo di Flask `add_url_rule()`, il quale consente di collegare la classe alla pagina web corrispondente.

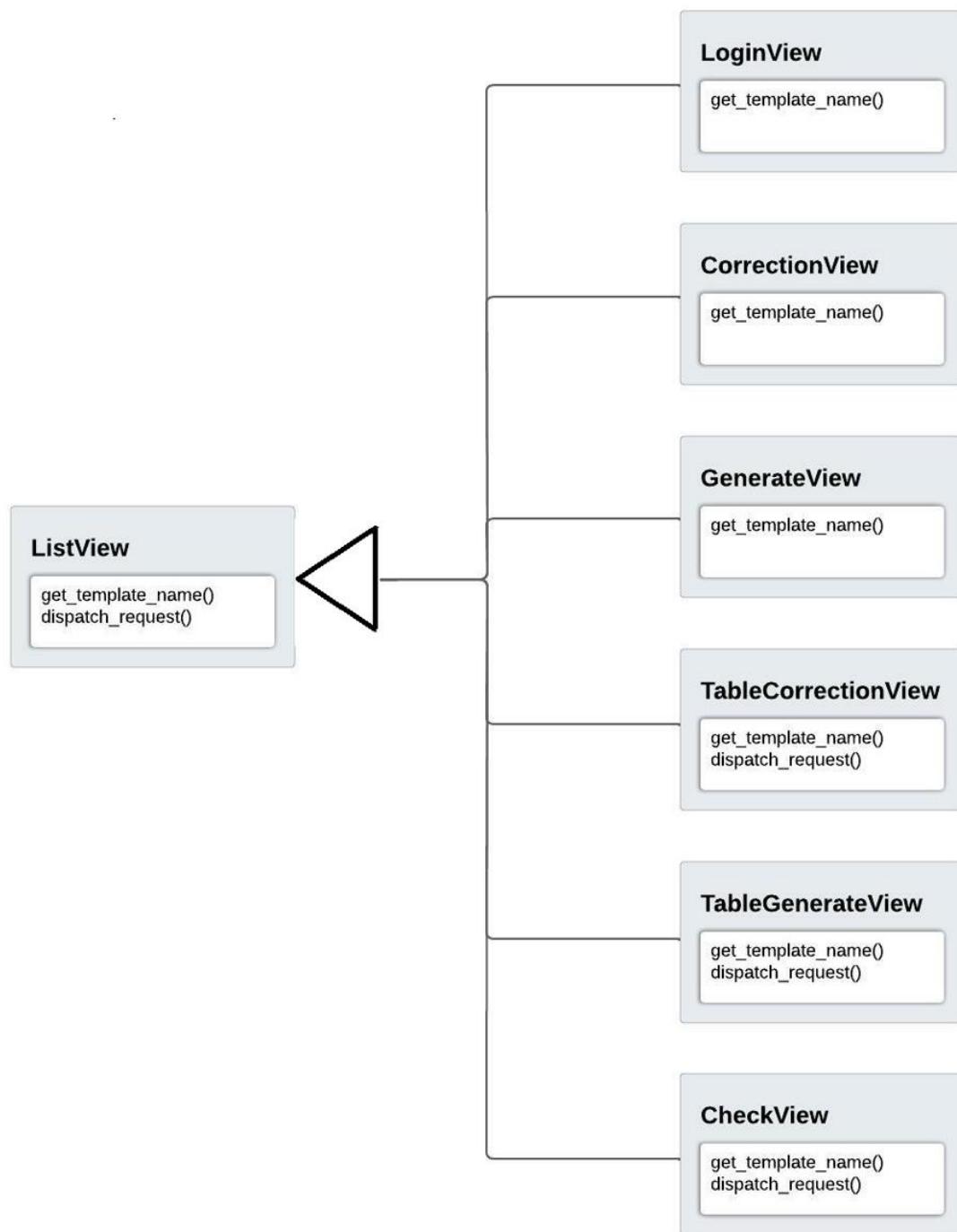
```

app.add_url_rule(
    "/cacheg",
    view_func=TableGenerateView.as_view("tableg-view"),
    methods=["GET", "POST"],
)

```

Infine è stata presa la decisione di utilizzare come pattern comune quello di implementare la vista con `methods=["GET", "POST"]`, in quanto ulteriori altre sottoclassi possono ereditare o persino cambiare del tutto i metodi.

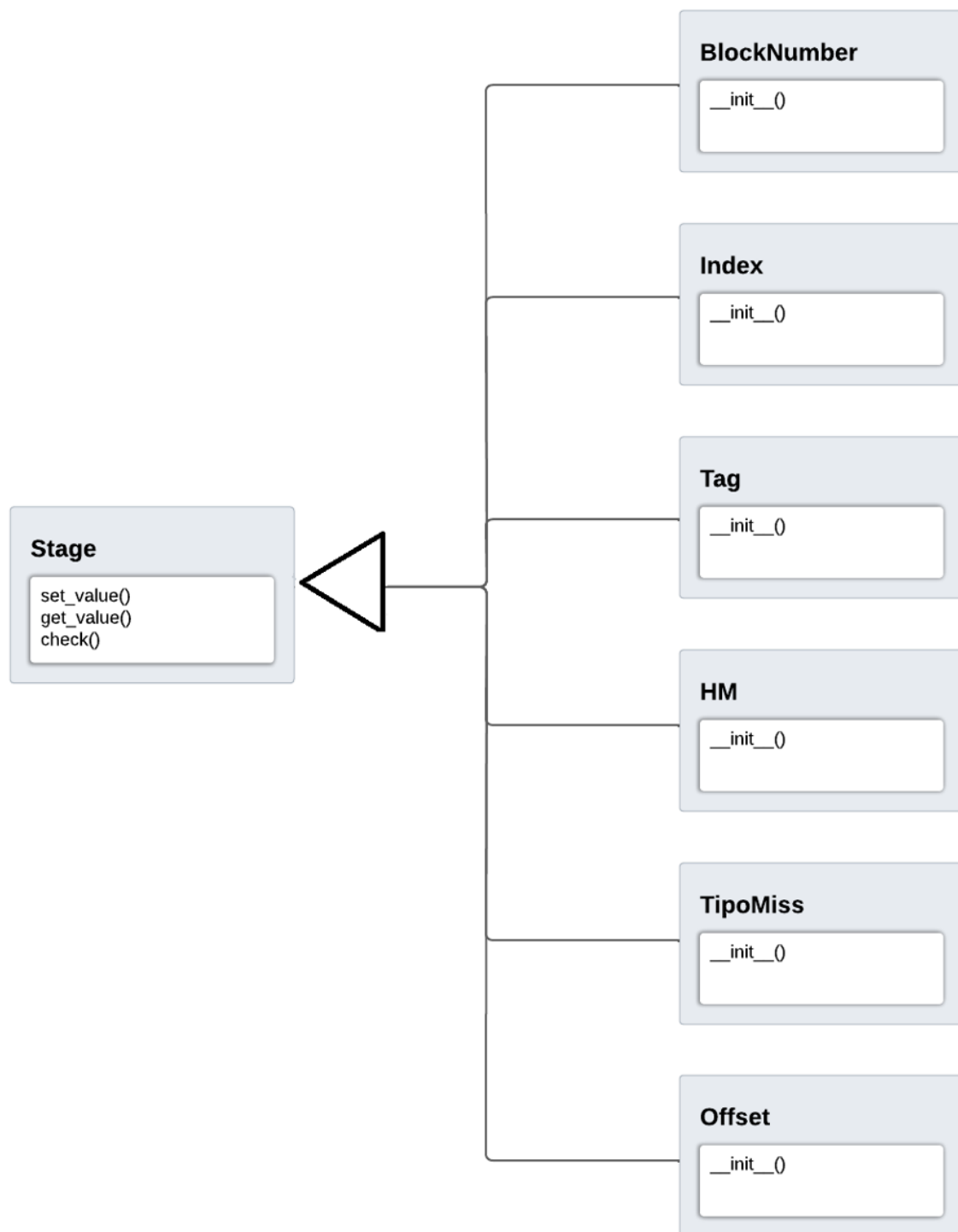




## Descrizioni classi fasi

Per quanto riguarda invece l'implementazione della cache, anche qui si è ritenuto opportuno utilizzare le classi.

Si è scelto di assegnare ad ogni cella della tabella la rispettiva classe, in base alla fase in cui ci si trova: "BlockNumber" per il numero di blocco, "Index" per l'indice, "Tag" per il tag, "HM" per individuare se si è trattato di una Hit o di una Miss, "TipoMiss" per identificare che tipo di Miss ci troviamo davanti, e infine "Offset" per l'offset qualora richiesto, in quanto implementato come dato facoltativo poiché la sua presenza o meno è una scelta la cui decisione è lasciata all'utente.



```
class Stage():  
  
    def set_value(self, v):
```

```

        self.value = v

    def get_value(self):
        return self.value

    def check(self, v):
        if self.value != v:
            self.value = v
            return self.errore

class BlockNumber(Stage):
    value = " "
    errore = "Numero di blocco errato"

    def __init__(self, indirizzo, dim_blocco):
        self.value = int(indirizzo/dim_blocco)

# ...

```

Ogni sottoclasse potrà usufruire dei metodi della classe Stage(): `set_value(self)`, la quale aggiornerà il valore assegnato per quella fase, `get_value(self)` che invece si occuperà soltanto di restituirlo, e infine avremo `check(self,v)`, che sarà di fondamentale importanza nell'ambito della correzione degli esercizi poiché si occuperà non solo di aggiornare il valore a seguito di un errore commesso dallo studente, ma anche di fornire il messaggio di errore specifico della fase in cui ci troviamo. Si ricorda che il valore viene aggiornato con il dato errato poiché si vuole non penalizzare lo studente nel caso in cui l'errore possa portare ad una propagazione dello stesso che quindi ne genererebbe inevitabilmente altri. Ovviamente, però, ciascuna sottoclasse verrà inizializzata in maniera diversa rispetto alle altre in quanto rispecchierà i calcoli riguardanti la fase che rappresenterà.

## Particolari casistiche di errori

Si è ritenuto essenziale ricavare un paragrafo interamente dedicato alle tipologie di errore che è possibile riscontrare a seguito della correzione di un esercizio. Per come erano state ideate inizialmente le fasi sottoforma di classi, lo spazio inerente al fornire la motivazione dell'errore segnalato era occupato unicamente dall'esigua stringa che si limitava ad avvertire che "il numero di blocco è errato" oppure che "c'è un errore nella costruzione".

Tuttavia, ben presto trasparì la necessità di ampliare il grado di utilità che questa funzionalità consentiva di ottenere, valutando anche di fornire all'utente osservazioni ben più mirate rispetto a quelle precedentemente fornite. È stata dunque introdotta la possibilità di verificare che lo sbaglio commesso dallo studente possa derivare da fattori che valga la pena di valutare e di presentare all'utente specificatamente. Ad esempio, si è introdotto un controllo per verificare che, riscontrato un valore discordante sia in un index che in un tag, questi non siano stati semplicemente invertiti e quindi si invita lo studente a prestare

maggiore attenzione o, per evidenziare invece una situazione in cui è avvenuto un errore di calcolo, si controlla che a seguito di una divisione, sia stata presa la parte intera inferiore come nel caso del calcolo del numero di blocco.

Inoltre, si è deciso di esplorare situazioni che sono esenti dalle operazioni matematiche o da distrazioni, ma che riguardano più nel dettaglio, ad esempio, i disturbi specifici dell'apprendimento quali la discalculia, che può presentarsi associata a dislessia, ma anche ad altri disturbi. La discalculia è un disturbo specifico dell'apprendimento nella comprensione dell'aritmetica, come la difficoltà nell'apprendimento di come manipolare i numeri. La discalculia, quindi, rende difficoltose le procedure esecutive per lo più implicate nel calcolo scritto: la lettura e scrittura dei numeri, l'incolonnamento, il recupero dei fatti numerici e gli algoritmi del calcolo scritto vero e proprio.

È stato dunque introdotto un ulteriore controllo all'interno di ciascuna fase che compia operazioni matematiche per cui verrà segnalato anche il caso in cui l'ordine delle cifre è stato invertito, in quanto questa difficoltà presenzia tra gli ambiti di quelle che possono essere manifestate da un ragazzo discalculico.

## Descrizioni classe “Cache”

All'interno della classe “Cache” infine troveremo le funzioni che ci permetteranno di costruirla, svolgendo tutti i calcoli necessari per individuare, per ogni indirizzo proposto, il relativo numero di blocco, indice, tag, se si tratta di una “Hit” o di una “Miss” e infine di che tipo di miss si tratta, in base ai dati forniti in input.

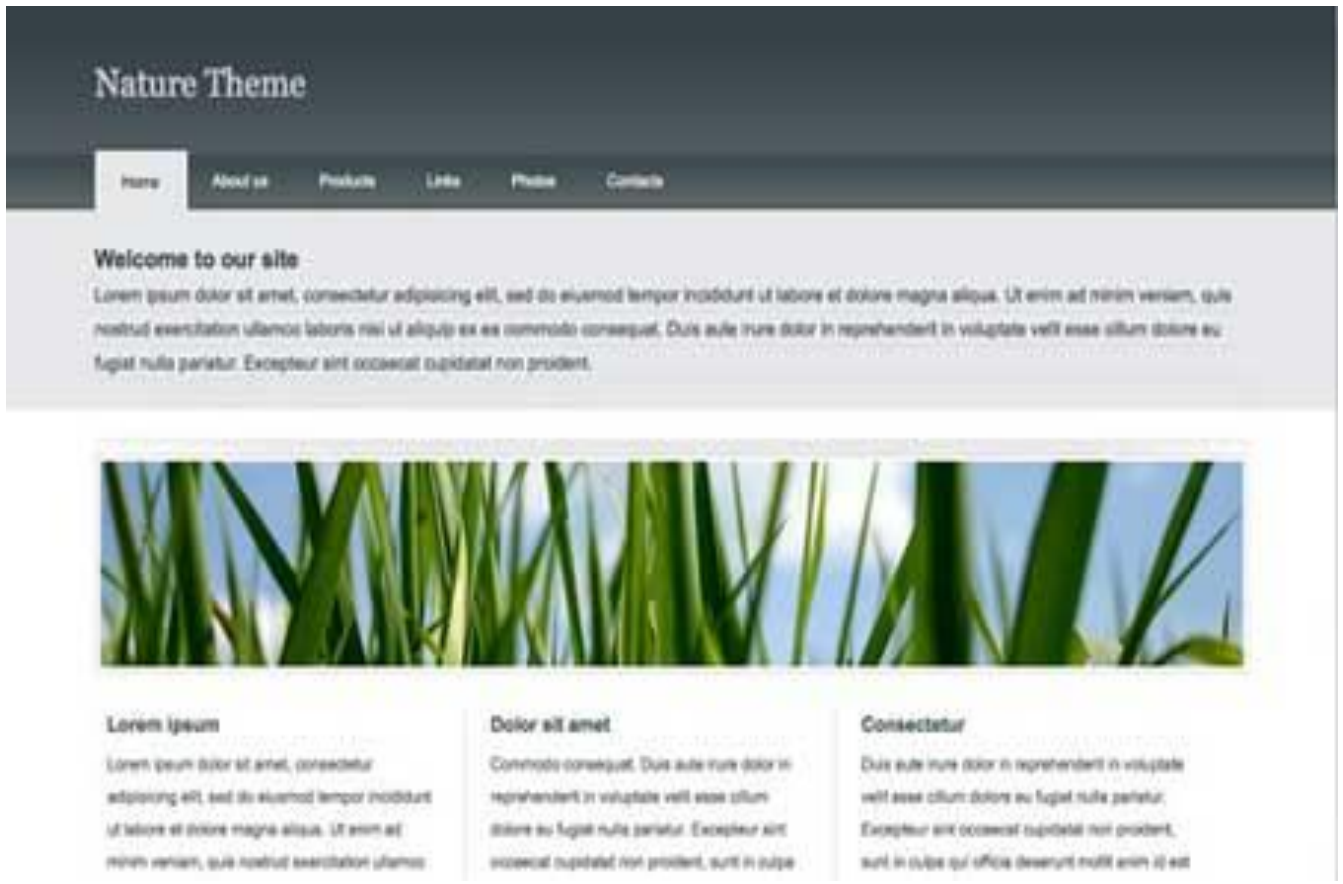
Sarà anche la sede della funzione `correggi(...)`, la quale avrà il compito invece di correggere, per l'appunto, l'esercizio anziché limitarsi al solo costruirlo.

A tale proposito, si è formata la funzione in maniera tale che la tabella venisse costruita e corretta in maniera dinamica. Man mano che la si scorre, si calcola il valore della nuova cella e lo si confronta con quello fornito dall'utente. Se il valore è il medesimo, la funzione proseguirà con la correzione del resto della cache, mentre se il valore dovesse essere discordante, allora la funzione terrà conto del fatto che si è verificato un errore e di che tipo si tratta, sostituirà il dato con quello errato nella cella in questione, e si proseguirà nuovamente. In tal modo si eviterà, ogni volta che verrà individuato un errore, di ricalcolare interamente la sottomatrice della cache che ne sarà affetta, di conseguenza, proprio grazie alla scelta del suddetto approccio dinamico.

```
def correggi(...):  
  
    #... calcolo del valore corretto della cella in cui ci troviamo ...  
  
    res = data[1][f][i].check(int(data_utente[1][f][i]))  
  
    if res is not None: # c'è stato un errore  
        errori += 1  
        commenti.append(res)
```



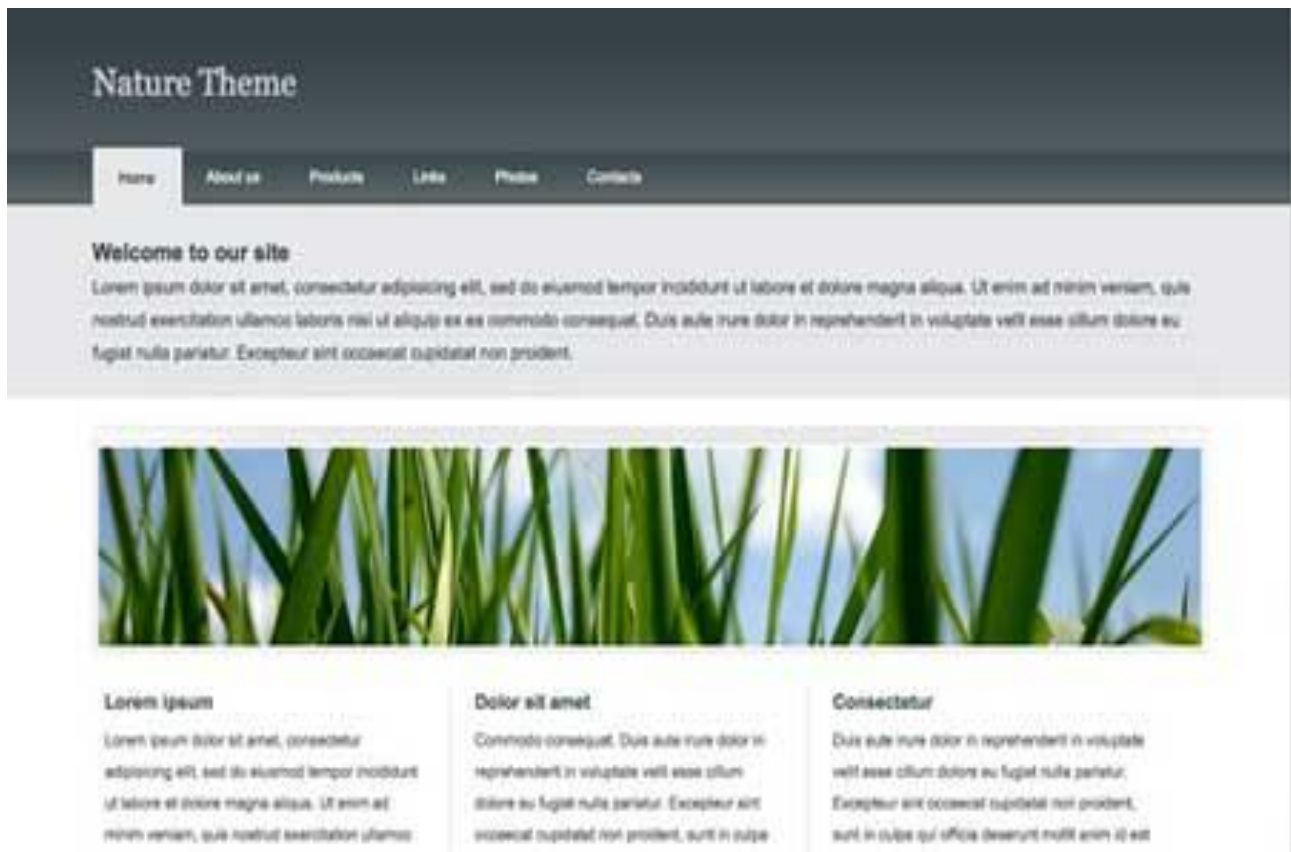
## Le pagine realizzate (immagini da inserire)



### Autenticazione

La schermata iniziale presenta semplicemente l'utente con lo scegliere la funzionalità a cui gradisce accedere: Correzione, Generazione o Simulazione.

Se correzione -> pagina con esercizio da completare -> pagina con correzioni e "valutazione"



## Correzione

Qualora si scegliesse di utilizzare la “correzione”, l'utente verrà accolto da un form compilabile che richiede varie informazioni riguardanti la cache su cui si desidera mettersi alla prova. Sarà possibile dunque scegliere la lista di indirizzi interessata, di quanti livelli sarà composta la cache, di che tipo saranno i livelli, il tipo e la dimensione del dato trattato, il numero di vie e dei set, su cui viene posto un controllo in quanto dipendono anche dal tipo di cache selezionato per ciascun livello. Inoltre, sarà possibile anche decidere se si desidera partire da una cache già riempita oppure no, e se sì a quale livello questa si riferisce. Dopodiché l'ultima opzione rimasta riguarda lo scegliere se si desidera avere una correzione come se ci trovassimo in una situazione simile ad un esame, ovvero dove la comunicazione degli errori viene effettuata unicamente quando l'utente sottomette l'esercizio completato, oppure una correzione più ad agio, dove l'utente potrà invece confrontare il valore introdotto con quello corretto cella per cella.

# Nature Theme

[Home](#)[About us](#)[Products](#)[Links](#)[Photos](#)[Contact](#)

## Welcome to our site

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident.



### Lorem ipsum

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco

### Dolor sit amet

Conmodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa

### Consectetur

Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est

## Correzione caso esame

# Nature Theme

[Home](#)[About us](#)[Products](#)[Links](#)[Photos](#)[Contact](#)

## Welcome to our site

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident.



### Lorem ipsum

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco

### Dolor sit amet

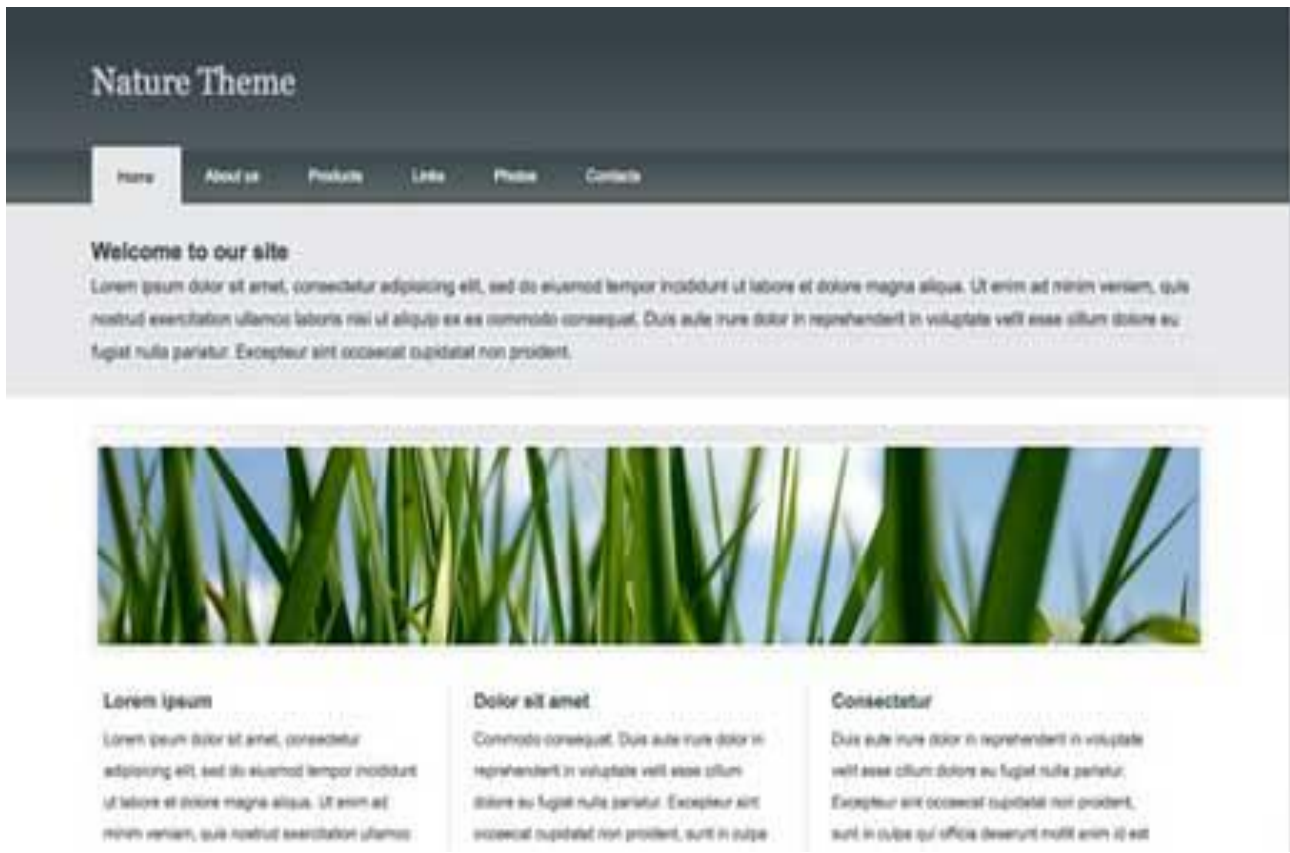
Conmodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa

### Consectetur

Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est

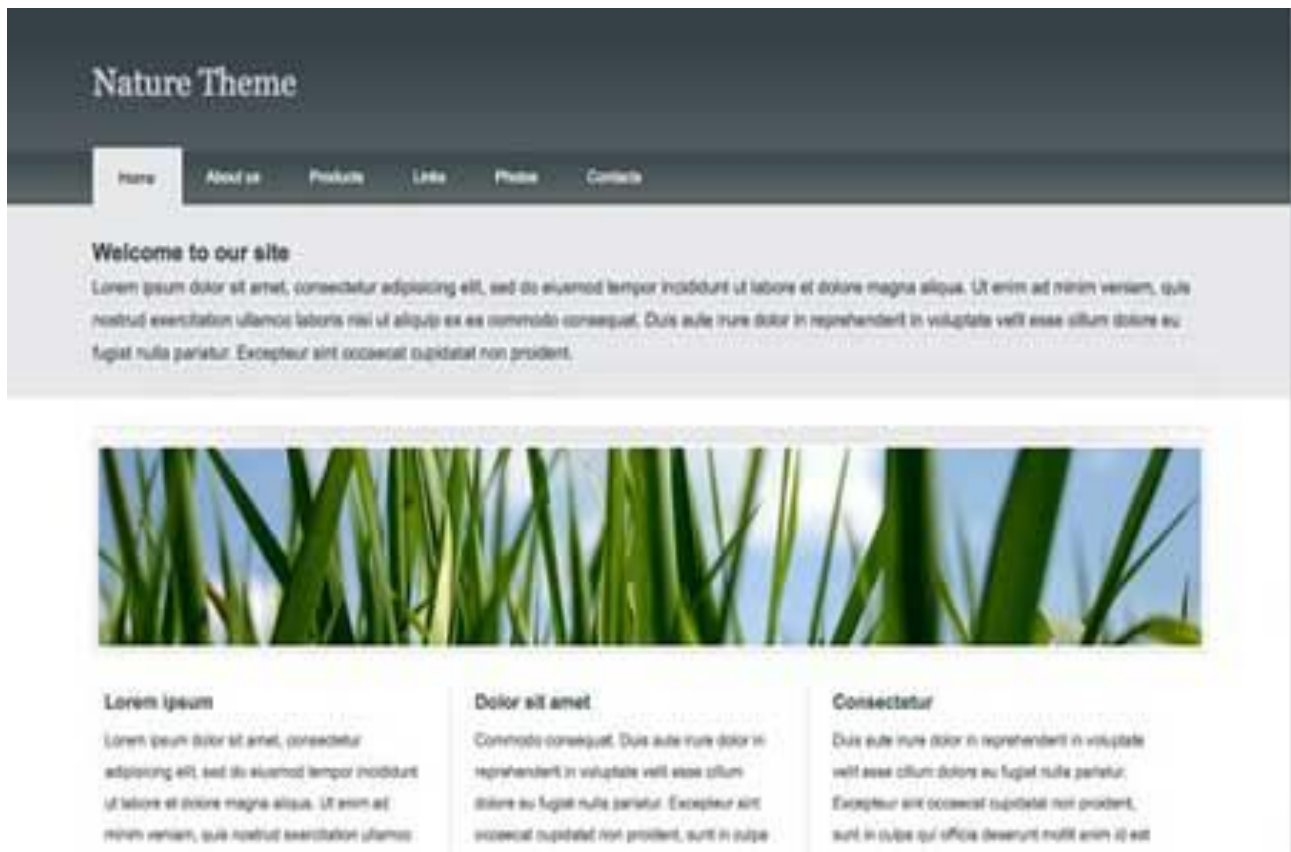


## Correzione valutazione caso esame



## Correzione caso cella per cella

Se generazione -> pagina con tabella completata e con descrizione tipo cache impiegato



## Generazione

Se invece l'utente selezionerà la funzionalità generazione, si troverà davanti una form da compilare molto simile a quella presentata nella correzione, ma non sarà più necessario inserire la lista di indirizzi desiderata, bensì la quantità di indirizzi e la dimensione massima che questi debbano avere. Questa scelta è ovviamente giustificata dalla natura stessa dell'opzione selezionata, dove si vuole creare un esercizio valido, ma pone le basi per un'ulteriore scelta proposta. Infatti, l'utente avrà anche la possibilità di scegliere se nella sua cache desidera la presenza di almeno un particolare tipo di miss o meno, nello specifico miss di capacità o miss di conflitto, e perfino a che livello vuole che esso compaia. Così facendo il programma utilizzerà questi dati per generare l'esercizio più consono alle necessità impartite.

# Nature Theme

[Home](#)[About us](#)[Products](#)[Links](#)[Photos](#)[Contact](#)

## Welcome to our site

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident.



### Lorem ipsum

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat.

### Dolor sit amet

Conmodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est

### Consectetur

Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est

## Generazione caso

Se simulazione -> pagina con 2 tabelle la cui seconda completata in parte



## Simulazione

Infine, possiamo trovare l'opzione di visualizzare una vera e propria simulazione del corretto funzionamento di una cache. Completando con la medesima form riscontrabile nella generazione automatica degli esercizi, l'utente, una volta deciso di proseguire, non si vedrà tuttavia presentare una cache completa e edificata. Bensì, con questa funzionalità l'utente erigerà la tabella cella per cella, colonna per colonna, livello per livello, in maniera tale che possa essere conscio del perché un determinato dato si trovi in quella specifica posizione. Inoltre, potrà anche contare sulla presenza di una cache vera e propria, che si aggiornerà in base al protocollo LRU (Least Recently Used) una volta arrivati nella riga corrispondente al verificare se si è trattato di una hit (H) o di una miss (M), e di una tabella che mostrerà il livello a cui apparterrà una miss di capacità (Cap.) o di conflitto (Conf.) sottoforma di cache di tipo fully associative, così che possa risaltare all'utente la distanza che vige tra i due valori che hanno portato alla presenza di quella tipologia di miss.

# Nature Theme

[Home](#)[About us](#)[Products](#)[Links](#)[Photos](#)[Contact](#)

## Welcome to our site

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident.



### Lorem ipsum

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco

### Dolor sit amet

Conmodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa

### Consectetur

Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est

## Simulazione caso

## Conclusioni

In questa relazione è stato descritto il lavoro svolto durante l'attività di tirocinio, inizialmente presentando la necessità di un'applicazione web che permette di generare automaticamente e correggere esercizi di Architettura degli elaboratori inerenti alle cache sfruttando un unico canale, per poi ripercorrere le principali fasi e scelte affrontate.

L'aver realizzato la piattaforma con un approccio modulare ed estensibile, principalmente grazie all'utilizzo della programmazione orientata agli oggetti mediante l'utilizzo di classi, è da considerare come un vantaggio tecnico fondamentale. In tal modo sarà reso facile e possibile attuare future modifiche, aggiunte e nuove implementazioni, che andranno dunque ad ampliare le funzionalità e ad evolvere la struttura dell'applicazione.

Il lavoro svolto fino ad ora ha prodotto un servizio pronto ad interfacciarsi con le richieste iniziali descritte nei paragrafi precedenti; tuttavia, mancano ancora delle possibili "estensioni" che forniscano alle varie tipologie di utenti altre funzionalità o che arrivino persino ad ampliare e migliorare la piattaforma stessa. Nel prossimo paragrafo, quindi, vengono espressi alcuni percorsi o semplici spunti da seguire per continuare ad elevare l'utilità di questo sistema.

## Difficoltà incontrate

Non sono state incontrate difficoltà particolarmente impegnative nel corso dello svolgimento della tesi. Spesso, infatti, queste riguardavano mancanze inerentemente ai linguaggi di programmazione, framework e motori di template utilizzati specificatamente per la realizzazione dell'applicazione web, colmate prevalentemente da ricerche di compendi e soluzioni a problematiche specifiche.

Tuttavia, ci sono stati alcuni ostacoli incontrati degni di nota, come ad esempio la necessità di rimanere nella stessa pagina web, come nel caso della simulazione. Infatti, nel caso del primo accesso alla pagina rappresentante la cache sottoforma di tabella, i dati per costruirla vengono presi dalla form della pagina precedente. Inizialmente però quando si preme per la prima volta il pulsante "Avanti", anziché far comparire la cella successiva, appariva invece la pagina di errore catturato dal debugger. Infatti, la pagina non aveva più i mezzi per costruire la tabella, perché la pagina web precedente non era quella contenente la form. E' stato necessario quindi modificare il codice, controllando che qualora fosse stata la prima visita alla pagina web, si sarebbe dovuto prendere normalmente le informazioni dai campi della form, per poi essere salvati questi mediante la funzione *json.loads()*; mentre qualora si fosse trattato di una visita successiva, si sarebbero ripresi dalla funzione *json.dumps()*.

```
args = request.form

if not args.get('params'):

    # ... si prendono i dati dalla form
    # grazie alla funzione args.get(...)
```

```

else:
    lista_dati = json.loads(args.get('params'))

# ...

params = json.dumps([lista_dati])

```

Un'altra difficoltà che mi ha messo a dura prova risiede nel gestire una cache che possiede uno stato iniziale. Infatti, l'introduzione a questa nuova casistica ha generato la necessità di variare innanzitutto il form riguardo la richiesta di riempimento dati da parte dell'utente. Si è dovuto quindi gestire una tabella editabile in cui si sarebbe dovuta aggiungere riga su riga tutte le informazioni necessarie. Dopodiché è stato il turno di modificare l'algoritmo che si utilizzava per costruire la cache, introducendo la possibilità che esistesse uno stato iniziale su cui bisognava prestare particolare attenzione, e di dover introdurre anche la costruzione di quello che poi sarebbe stata la cache finale, presenze queste che hanno inevitabilmente inciso anche sulla costruzione dell'algoritmo che si occupa invece di correggere la cache, sebbene in questo caso la loro influenza si è dimostrata essere meno incisiva, non avendo variato in modo determinante la logica e l'ordine al suo interno. Tuttavia, tramite controlli mirati e l'efficiente gestione delle tabelle rappresentanti sia la cache iniziale che la cache finale si è potuto superare anche questo ostacolo.

```

def riempicache(cache_table):
    # ...

    elif init != {} and cache_table.get_lvlnit() - 1 == 1:
        """
        - se init[index] != tag: L
        - se init[index] != tag MA PRIMA init[index] ==
tag: cap o conf
        - altrimenti: H
        """
        if stor_init[data[1][1][i].get_value()][-1] !=
str(data[1][2][i].get_value()) and str(data[1][2][i].get_value()) not in
stor_init[data[1][1][i].get_value()]:
            stor_init[data[1][1][i].get_value()].append(str
(data[1][2][i].get_value()))

        #devo controllare storico accessi
        data[1][4][i] = fasi.TipoMiss("L")

```



```

elif stor_init[data[1][1][i].get_value()][-1] !=
str(data[1][2][i].get_value()) and str(data[1][2][i].get_value()) in
stor_init[data[1][1][i].get_value()]:
    stor_init[data[1][1][i].get_value()].append(str
(data[1][2][i].get_value()))

n_way = cache_table.get_ways()[1]

# ...

```

## Future migliorie possibili

### Pagina di login

Una prima possibile miglioria che senz'altro è da ritenersi valida consta nel creare una pagina di login vera e propria, aggiunta che dovrà necessariamente includere anche la gestione e la manutenzione di un database. Quest'ultimo avrà il compito di conservare al suo interno le credenziali dei vari utenti, da intendersi come ad esempio l'email istituzionale e la password, seguite dal tipo di autorizzazione che avrà tale profilo, tenendo conto, nel caso, dell'email stessa. Queste autorizzazioni incideranno sull'abilitazione che avrà l'utente sulle varie funzionalità rese disponibili; nello specifico se l'email istituzionale apparterrà ad un docente, questo potrà accedere alla funzionalità che avrà il compito di generare automaticamente esercizi, mentre se l'email istituzionale apparterrà allo studente, questo potrà usufruire della correzione degli esercizi e della simulazione della cache. Un caso specifico inoltre verrà ricoperto nel caso in cui l'utente registrato dovrà ricoprire la carica di amministratore. Tale scenario comporterà l'assegnazione del grado di autorizzazione più alto e quindi l'accesso di tutte le funzionalità ed eventualmente anche l'accesso alla lista completa degli utenti.

Per creare ed implementare un sistema di registrazione e login, si consiglia di scrivere le varie componenti in PHP utilizzando MySQL. **(AMPLIARE)**

Inoltre, si è pensato anche alla possibilità in futuro di includere sempre all'interno del database di informazioni inerenti alla classe di appartenenza di un utente studente e ad un dato che sottolineerà l'andamento nel risolvere esercizi sulle cache. Si potrà quindi ad esempio mostrare la media dei suoi punteggi complessivi, rapportabile quindi alla media complessiva della classe di appartenenza.

È tuttavia necessario prima di tutto inserire tra le pagine web costruite all'interno del progetto crearne anche una che permetta la registrazione dell'utente, ovvero una pagina in cui verranno richiesti dati all'utente coerenti con un background universitario, per poi inserirle all'interno del database dopo aver constatato la validità di tali informazioni, con annesso il grado di autorizzazione scaturito.



## **Ampliare casistiche errori**

Quando si è discusso riguardo alla correzione degli esercizi, è stata presentata la possibilità da parte dell'utente di ricevere commenti mirati riguardo agli errori commessi. Questa tipologia di feedback, tuttavia, è stata introdotta mediante alcuni esempi di errori tra i più comuni riscontrati, come ad esempio l'inversione tra l'index ed il tag durante i calcoli. Il focus principale, infatti, era il porre le basi per questo tipo di esperienza di correzione personalizzata, per poi sfociare in un possibile studio più accurato e completo riguardo le casistiche di errori più comuni o che comunque valga la pena evidenziare. La volontà, quindi, sarebbe quella di far emergere scenari ben più dettagliati che poggino proprio su queste basi e che le rendano più salde e che portino ad un deciso miglioramento di questo ambito.



## Bibliografia (da sistemare)

1. <https://flask.palletsprojects.com/en/2.2.x/>
2. [https://www.tutorialspoint.com/flask/flask\\_application.html](https://www.tutorialspoint.com/flask/flask_application.html)
3. A.W. Burks, H.H. Goldstine e J. Von Neumann, *Discussione preliminare sul progetto logico di un dispositivo elettronico di calcolo*, 1946
4. David A. Patterson, John L. Hennessy, "Computer Organization and Design, The hardware/software interface 5<sup>th</sup> Edition, Morgan Kaufmann, 2014
5. David A. Patterson, John L. Hennessy, "Struttura, organizzazione e progetto dei calcolatori. Interdipendenza tra hardware e software, Jackson Libri, 2000