Alma Mater Studiorum · Università di Bologna

SCUOLA DI SCIENZE
Corso di Laurea in Informatica per il management

UN'APPLICAZIONE DEI WEB PLAYGROUND ALL'INSEGNAMENTO DELL'INFORMATICA

Relatore:
Chiar.mo Prof.
FABIO VITALI

Presentata da:
MATTEO BORGHI

Co-relatore:
ANGELO DI IORIO

Sessione II
Anno Accademico 2013/2014

A Paolina,
Massimo,
Maria Oliva,
Michela
e Martina.

Introduzione

Il giorno dell'esame di tecnologie web, corso che per antonomasia basa se stesso sull'uso del computer, ho constatato personalmente che la carta la fa da padrona, quasi incontrastata, negli esami di programmazione.

In verità mi ha molto colpito lo stato delle cose, in quanto è innaturale anche solo il fatto di pensare di creare una pagina web con carta e penna.

Gli studenti hanno bisogno di un ambiente che gli permetta con una certa semplicità di capire come stanno rispondendo alle richieste del compito e gli consenta quindi di testare ciò che stanno scrivendo. Come ben sappiamo il modo in cui sono concatenate, annidate e integrate tra loro le componenti che fanno parte di un elaborato di tecnologie web può non essere di facile lettura nemmeno per un grande esperto, figuriamoci per un alunno nel suo compito. Carta e penna gli consentono unicamente di effettuare qualche correzione, senza dare la certezza di aver rispettato le consegne.

Una soluzione migliore è quella di consentire allo studente di provare e riprovare il codice che sta scrivendo in modo da fornirgli un quadro ben delineato del lavoro effettuato che possa indirizzarlo alla giusta soluzione. Utilizzare tale metodologia fornirebbe un valido aiuto agli studenti, ma non si sostituirebbe alla loro preparazione, bisognerebbe comunque studiare proficuamente per ottenere buoni risultati.

Per realizzare una funzionalità di questo tipo bisogna necessariamente introdurre il calcolatore all'interno della metodologia d'esame, oggetto che fino ad ora in molti corsi viene messo in disparte e utilizzato a volte solamente nelle pratiche d'insegnamento della materia.

Se il formato dell'esame fosse machine-readable, si potrebbe pensare di fornire un servizio di valutazione automatica che dia benefici al docente sia in termini di tempo da dedicare sia in termini di oggettività della valutazione.

In questo caso essendo il computer a correggere i compiti, non ci possono essere differenze di trattamento per gli alunni, il primo compito viene processato nello stesso modo dell'ultimo.

Di norma in una prova d'esame di tecnologie web viene richiesto agli studenti di scrivere codice HTML, CSS e Javascript per realizzare una pagina web. Gli strumenti attualmente utilizzati per testare l'output prodotto in tempo reale dalla scrittura di codice con tali linguaggi direttamente nel browser sono i cosiddetti web playground, che tuttavia non prevedono di correggere in maniera automatica il lavoro svolto.

Riguardo quest'ultima funzionalità esistono in commercio specifici sistemi che permettono di valutare una pagina web definita in una certa url in maniera statica e dinamica, ma non forniscono un ambiente integrato dove poterla creare.

Confrontandomi con il mio docente di tecnologie web ho cercato di capire se fosse pensabile sviluppare uno strumento che rispondesse alle necessità sia degli studenti, fornendogli un ambiente di lavoro confortevole, sia dei docenti, lasciando alla macchina il peso della correzione e fosse effettivamente utilizzabile in ambito accademico. Dopo diverse discussioni e l'analisi del contesto scientifico si è deciso di intraprendere l'avventura ExaM Bin.

L'obiettivo di questo progetto è quello di dimostrare che è possibile in maniera proficua fare esami di tecnologie web all'università senza carta e penna.

Per iniziare lo sviluppo si è pensato di rispondere alle necessità relative agli studenti e, vista la comodità dei web playground per la creazione e il testing di codice, si è deciso di creare il progetto in questione come adattamento di una piattaforma di questo tipo alle esigenze di sicurezza e verificabilità tipiche di una prova d'esame. ExaM Bin nello specifico nasce come estensione dall'applicativo di web playground, denominato JSBin poiché è l'unico della

sua categoria reinstallabile su macchina dell'utente e open source.

Le funzionalità offerte da tale strumento sono state utilizzate per la pagina dedicata alla realizzazione del compito degli studenti. A questa base per l'alunno sono state aggiunte le funzionalità di autenticarsi al sistema e di consegnare il compito redatto. Invece al docente, relativamente alla prova d'esame, viene data la possibilità di fornirne il codice di partenza a livello di HTML e/o CSS e/o Javascript, di gestirne la durata e di correggerne in maniera automatica il contenuto.

Un particolare riferimento va a quest'ultimo servizio dato che è stato sviluppato, diversamente dagli strumenti in commercio, servendosi di framework di unit test utilizzabili sia client side che server side. Questo per permettere al professore di controllare con mano, senza interagire con il sistema, le regole a cui sottoporre ogni singolo compito redatto dagli studenti.

Ultimo, ma non per questo meno importante, l'aspetto economico, infatti uno degli obiettivi era quello di mantenere al minimo possibile il livello dei costi legati all'acquisizione di licenze d'uso di software di base e dei successivi costi relativi alla manutenzione e modifica delle funzionalità della piattaforma. Un ruolo importante in questa sezione è stato quello dei software open source gratuiti che hanno permesso di realizzare l'applicativo senza alcun esborso di denaro, ma soltanto con l'applicazione delle opportune conoscenze nella programmazione.

Dopo pochi mesi dalla sua nascita, il progetto è stato sottoposto ad un numero limitato di test, ma per ora sembra funzionare correttamente garantendo l'integrità di tutte le funzionalità implementate al suo interno.

L'obiettivo in un futuro prossimo è quello che ExaM Bin possa essere utilizzato per fare esami di tecnologie web in qualsiasi ambito a diversi livelli di difficoltà, passando dal percorso prettamente umanistico della scuola di lettere a quello più tecnico del dipartimento di Informatica.

Proseguendo nella trattazione, nel capitolo 1 verrà sviluppato il contesto scientifico, nel capitolo 2 si parlerà degli strumenti di code playground, parte fondamentale del progetto, nel capitolo 3 si mostrerà ExaM Bin a livello di

idea e funzionalità. Infine nel capitolo 4 verranno resi noti tutti i dettagli implementativi della soluzione proposta al problema con eventuali sviluppi futuri presentati nelle Conclusioni.

Indice

Elenco delle figure

Elenco listati di codice

Capitolo 1

Esami all'Università: computer-based?

La valutazione, intesa come valutare e farsi valutare, è una componente importante del processo educativo a livello universitario.

Fornisce un mezzo per misurare l'apprendimento del materiale didattico da parte degli studenti e stabilire se essi hanno acquisito le necessarie conoscenze e competenze per accedere al successivo livello accademico.
Selezionare un appropriato metodo di valutazione può essere critico per la realizzazione degli obiettivi a livello educativo e può avere effetti sulle attitudini e l'approccio all'apprendimento degli studenti dinanzi al corso di studio.

Per di più, un metodo di valutazione può essere uno strumento per incoraggiare e motivare gli studenti a raggiungere una completa e profonda conoscenza della materia [DBP04].

Convinto dell'importanza del processo di valutazione a livello universitario ho analizzato la problematica per predisporre la creazione di uno strumento informatico per affrontarla.

1.1 Esami di informatica all'Università

Selezionare il tipo di esame più appropriato è diventato, ed è tutt'ora, un problema nei corsi di programmazione.

Nella maggior parte dei corsi di programmazione a livello universitario la prova finale viene effettuata in formato cartaceo, ed agli studenti viene chiesto di tipicamente di correggere o scrivere interi pezzi di codice in un ambiente diverso da quello in cui si sono esercitati per imparare la materia.

In molti casi questo tipo di approccio si è mostrato insoddisfacente, poiché presenta svantaggi sia per il rendimento degli studenti, dato che non fornisce un ambiente reale di programmazione dove si hanno a disposizione tutti gli strumenti per verificare e testare ciò che è stato redatto; sia per i professori, in quanto in questa maniera non riescono a capire le vere abilità e conoscenze apprese dagli studenti.

Tutto questo unito all'aumento del numero di alunni nelle classi, ha portato i professori alla ricerca di nuove metodologie per fare esami.

Per prima cosa, è stata scartata la possibilità di fare esami orali, sia per le tempistiche, sia perché presenta le stesse limitazioni degli esami cartacei, visto che ad un pezzo di carta viene sostituita una lavagna.

Ben-Ari [ABT97] nota che l'apprendimento degli studenti nei corsi di programmazione dovrebbe essere attivo e non passivo: l'obiettivo di questo tipo di insegnamenti è si quello di riuscire a far costruire i corretti modelli mentali riguardo i concetti astratti di come il computer lavora e la natura delle variabili di programmazione, ma essendo l'informatica una materia prettamente pratica, deve fornire agli studenti l'opportunità di sviluppare la conoscenza dei principi della materia attraverso il lavoro pratico, a condizione che questo sia svolto di concerto con l'intervento dell'uomo per superare idee sbagliate e perfezionare i modelli mentali.

Quindi con il desiderio di trovare un metodo di valutazione complessiva effettivo molti insegnanti hanno pensato di utilizzare un approccio computer-based.

1.1.1 Esami Computer-based

La metodologia computer-based per fare esami all'università si serve del computer come strumento intermediario nel processo d'esame, in cui vengono memorizzate tutte le informazioni relative al test effettuato da ogni singolo alunno e con il quale viene permesso di realizzare e consegnare l'intero elaborato richiesto.

Quando si parla di esami computer-based nell'ambito informatico, ci si riferisce nella maggior parte dei casi ad esami di laboratorio, dove gli studenti attraverso l'utilizzo di un elaboratore, devono rispondere alle consegne del docente, ad esempio scrivendo un intero programma, o piccoli frammenti di codice.

Molti esperimenti hanno dimostrato negli anni che questo tipo di metodologia di valutazione fornisce un ambiente più appropriato nel quale testare le abilità di programmazione degli studenti rispetto agli esami cartacei in cui sono limitati all'uso di carta e penna.

Chamillard [CJ01] riscontra ottimi risultati utilizzando due sessioni di "pratiche di laboratorio" per circa 500 studenti nel corso introduttivo di Informatica dell'U.S. Airforce Academy. Stessa cosa si può affermare per Califf [CG02] che un anno più tardi decide di istituire un test di laboratorio online per gli alunni del corso di Informatica di base per due anni, sottoponendo la piattaforma a 200 e poi 280 studenti per anno. Negli studi di Chamillard e Califf, i test computer-based vengono somministrati in aggiunta alle consuente prove cartacee di fine corso, e come propongono Haghighi e Sheard [HS05] in un esperimento simile, dopo aver somministrato dei questionari ai partecipanti, questa metodologia viene apprezzata anche dalla maggior parte degli studenti.

English [ENG02] invece descrive di aver ottenuto un sostanziale miglioramento della valutazione degli alunni utilizzando un esame di fine corso interamente online per 64 studenti del primo anno del corso di programmazione. Come propone l'autore, l'uso di una metodologia computer-based negli esami può fornire gli strumenti necessari agli studenti per esprimere al meglio

le proprie capacità di programmazione; poiché quando questi rispondono ad una domanda di programmazione, ad esempio fornendo un frammento di codice, gli si può dare la possibilità di testare ciò che hanno scritto, osservarne i risultati correggendo eventuali errori per poi consegnare il programma finito. Questo tipo di testing permette agli studenti di mostrare ai docenti una più profonda conoscenza della materia che non gli sarebbe possibile dimostrare attraverso un esame cartaceo.

Ulteriore benefit portato dall'utilizzo di questa metodologia si ha nel fatto che il formato di consegna di ogni singolo esame è machine-readable quindi si può prestare ad ulteriori elaborazioni da parte del computer stesso.

Riguardo quest'ultimo aspetto, English parla della possibilità di far effettuare agli elaboratori validazioni automatiche, che permettano di creare un sistema di valutazione automatizzato poi moderabile dall'intervento dell'uomo.

La creazione di un sistema di questo tipo è il passo successivo nell'utilizzo del computer nella metodologia d'esame in quanto permette al docente di sfruttare al massimo le potenzialità dell'elaboratore, non solo per fornire agli alunni un ambiente più favorevole in cui effettuare l'esame, ma anche per diminuire il proprio carico di lavoro, abbreviando il tempo necessario per correggere ogni singola prova.

1.2 Valutazione automatica di esercizi di programmazione

Una metodologia di valutazione automatica si ha quando si utilizza il computer non solo per lo storage delle risorse e come ambiente di realizzazione e consegna del compito degli studenti, ma come componente attivo nel processo di valutazione; sostanzialmente ci si serve del computer per la correzione di un compito di programmazione [CEA03].

Il computer può essere utilizzato in varie maniere per supportare l'attività di apprendimento e i processi in cui si articola un corso di studio.

Questi possono dare un voto numerico o un feedback in formato sia testuale che visuale.

Nei corsi di e-learning e web-based questo tipo di valutazione è estensione naturale del corso.
Computer Science, come area di interesse, è ben posizionata per beneficiare di valutazioni automatizzate; chi insegna la materia può spesso usare la propria esperienza per sviluppare sistemi che lo aiutino a ridurre il proprio carico di lavoro.

Computer utilizzato come assistente nella correzione degli esami, passando con il tempo da una mera valutazione statica del programma, ad esempio fornendo una valutazione di "corretto" o "non corretto", fino ad una valutazione effettuata tramite un'analisi dinamica. Quest'ultima permette al professore di testare tutti i casi d'uso di cui ha bisogno e, come riporta Ala-Mutka [ALA05], viene spesso utilizzata in ambito accademico al fine di fornire una valutazione più completa e oggettiva possibile, verificando al meglio le funzionalità e l'efficienza di ogni singolo elaborato.

1.2.1 Breve storia del campo

Sono state compiute molte indagini riguardo la validazione automatica, sia a livello di utilizzo che di creazione di strumenti in grado di applicarla. Queste hanno dunque consentito di ottenere un quadro di riferimento ben delineato del campo in questione.

Le ricerche hanno fornito inoltre una disamina su come viene percepita la valutazione automatica dagli educatori; uno dei risultati più interessanti è che gli insegnanti privi di familiarità con tale tipo di giudizio ne considerano le potenzialità più limitate rispetto a coloro che possiedono una maggiore esperienza riguardo l'utilizzo di questi strumenti [CEA03].

Nella sua recensione Douce [DLO05] esamina da vicino il fenomeno della validazione automatica, che sembra esistere fin da quando gli educatori hanno chiesto agli studenti di creare i loro primi progetti software, dividendo la storia degli strumenti creati in tre generazioni:

- prima generazione - sistemi di valutazione primitivi:
  strumenti creati ad hoc utilizzando compilatori ed editor di testo, difficili da utilizzare e sviluppare. Il primo esempio di automatizzazione del processo di valutazione può essere riscontrato nel lavoro di Hollingsworth [HOL60].

- seconda generazione - sistemi tool-oriented:
  strumenti creati con l'ausilio di tools e set di utility già esistenti, che molto spesso possono essere utilizzati ed attivati attraverso command-line o GUI dedicate [REE89].

- terza generazione - sistemi Web-Oriented:
  strumenti di valutazione che sfruttano gli sviluppi nell'ambito delle tecnologie web e adottano un sistema di testing sempre più sofisticato.

In quest'ultima generazione troviamo i tool sviluppati al giorno d'oggi, che danno maggior enfasi a specifici campi, tra cui emerge il Web-Programming; ambito in cui si pensa verrà continuata la ricerca e lo sviluppo anche nei prossimi anni in maniera particolarmente marcata [IAK10].

1.2.2 Web programming

Con sistemi di valutazione automatica nel campo del Web-programming tipicamente si testa la parte client-side di un sito web creato da uno studente, costituito dalla realizzazione di codice HTML e CSS per quanto riguarda la parte statica e stilistica, e Javascript per la parte dinamica, ignorando l'implementazione server-side. Esistono anche sistemi, relativamente a questo campo, nel mondo accademico che si occupano di valutare l'apprendimento degli studenti ad esempio nei corsi si sicurezza informatica [GUT06] o più nello specifico della sicurezza applicata al corretto funzionamento degli strumenti di E-commerce, fattore molto importante nella loro realizzazione [CW10].

Questo tipo particolare di strumenti nasce come evoluzione dei sistemi di valutazione di interfacce grafiche (si tratta infatti di valutare un'interfaccia

creata con l'ausilio delle tecnologie web) che vengono sviluppati soprattutto per valutare lavori creati con una determinata piattaforma di riferimento. Riguardo il campo delle applicazione desktop Java, Sun e Jones [SJ04] presentano uno scenario basato sulla valutazione di interfacce grafiche costruite con tale linguaggio dove però l'insegnante deve definirne in maniera accurata e minuziosa i componenti che gli studenti andranno a creare in modo tale che questi possano combaciare con oggetti Java definiti specificatamente per effettuare la correzione. Per sopperire a tale limitazione, Feng e McAllister [FM06] introducono un sistema di valutazione che permette di controllare qualsiasi tipo di interfaccia, rimanendo sempre però legato alla piattaforma nativa, anche in questo caso relativa al linguaggio Java. Ultimo esempio interessante è quello di Web-Cat [EP08] sistema estendibile e personalizzabile che permette agli studenti di presentare sia il codice redatto come soluzione all'incarico fornito, sia il file di test su cui effettuare le verifiche dei propri applicativi. Gli alunni sono valutati su come testano il proprio codice, tipicamente redatto utilizzando i linguaggi Java o C++.

Dallo studio di questi lavori, sono venuti alla luce i sistemi di valutazione automatica web-based più recenti relativi, nello specifico, alla verifica di esercizi di programmazione web. Questi sono sviluppati permettendo ai professori di effettuare una validazione sia statica che dinamica, consentendo inoltre agli studenti di consegnare il proprio elaborato ponendo l'applicativo web da loro realizzato in un apposito indirizzo conosciuto dal sistema.

Un esempio è il tool AWAT [SQF08] che fornisce un ambiente per l'assegnazione di incarichi di programmazione web dove agli studenti è concesso di presentare l'url del sito che loro hanno sviluppato. Il professore definisce i componenti che dovrebbero esistere nella pagina web in un foglio Excel e la piattaforma esegue il test usando la libreria Watir di Ruby che interagisce direttamente con il web browser. Anche APOGEE [FPQ08], altro sistema di valutazione automatica a livello web, utilizza questa libreria per effettuare la verifica di una pagina web ma, a differenza del precedente, fornisce come risultato una serie di screenshot che possono essere utilizzati come feedback

allo studente per spiegargli perché il test è fallito. Entrambe le piattaforme citate sono open source poiché vogliono aprirsi a nuove idee e modifiche della comunità, diffondendosi in maniera più capillare. Esistono anche piattaforme serverless, un esempio è il lavoro di Karavirta e Ihantola [KI10], dove viene data la possibilità ad uno studente di rispondere ad una domanda nel linguaggio Javascript avendo a disposizione nella pagina stessa una sezione dedicata alla visualizzazione degli errori scaturiti dall'esecuzione del codice redatto.

Tale tipo di strumenti fornisce un servizio di validazione automatica che può generare un voto e/o dei feedback per il lavoro degli studenti, quindi dare una grande mano al professore nella fase di correzione, ma non è dotato di un ambiente di sviluppo in grado di aiutare gli alunni a svolgere il proprio compito di programmazione web e presentarne il contenuto al sistema per la correzione.

Questa rappresenta una grossa limitazione per un corso di studi che ha bisogno di testare applicativi web, perché oltre al sistema per la valutazione automatica deve a parer mio possedere un ambiente grafico per l'implementazione del compito, non solo un form dove presentare l'indirizzo della pagina web creata.

Un'altra limitazione può essere osservata a livello di scrittura dei test, in quanto viene richiesto al docente di redigere un file contenente le regole da verificare, che nella maggior parte dei casi per essere creato necessita della conoscenza di specifici linguaggi di programmazione.

Il professore inoltre non può provare il suo test su di un compito d'esame creato ad hoc come soluzione con l'intenzione di verificare, senza interagire con la piattaforma, se ciò che ha scritto è effettivamente corretto e controlla il maniera puntuale ciò di cui ha bisogno.

Relativamente alla limitazione rappresentata dall'assenza di un ambiente di sviluppo, vista la specificità degli strumenti nel campo della sola valutazione automatica, le piattaforme ad oggi più comode che offrono una soluzione a questo problema sono i cosiddetti code playground, identificati con il nome

di web playground, quando si fa riferimento alla programmazione in ambito web.

Una descrizione più approfondita ed esaustiva riguardo questo tipo di applicativi verrà fornita nel capitolo successivo poiché questi rappresentano un ambiente molto adatto a svolgere nel migliore dei modi esercizi di programmazione, che potrebbe essere utilizzato anche dagli studenti per fare esami di tecnologie web all'università.

Capitolo 2

Code Playground

I code playground sono ambienti messi a disposizione dei programmatori o qualsiasi altro tipo di persona che vuole anche solamente avere un timido approccio con l'informatica, per redigere codice con semplicità.

Il termine nasce dall'unione di due parole: code, che rappresenta l'atto di codifica o il codice stesso e playground, traducibile con "campo da gioco" o "parco giochi". Quindi letteralmente possiamo identificare un code playground come un parco giochi dove poter divertirsi a scrivere codice.

Questo tipo di strumenti offre la possibilità di redigere codice ma anche in alcuni casi di testarlo, sia in maniera statica, controllando eventuali errori di sintassi, che dinamica, provando l'effettiva bontà del programma andandolo ad eseguire. In quest'ultimo livello di analisi non consente di testare in maniera automatica qualsiasi caso d'uso, ma semplicemente fornisce un'indicazione sul fatto che il programma funzioni o meno.

Oggigiorno in commercio troviamo molti strumenti che possiedono queste caratteristiche, i quali si differenziano in base al linguaggio di programmazione che permettono di utilizzare. Un particolare tipo di code playground ad oggi molto in voga, tra esperti o meno del settore, è quello relativo alla programmazione in ambito web, parliamo dei cosiddetti web playground.

2.1 Web Playground

Con il termine web playground[1] (o online Javascript IDE, browser based Javascript IDE, Javascript live coding enviroment, Javascript sandbox) si indica un ambiente di sviluppo (IDE, Integrated development Enviroment) che è ospitato in un browser, con l'obiettivo di facilitare la scrittura di codice HTML, CSS e Javascript per lo sviluppo di applicativi web. Molti di questi ambienti permettono di salvare e condividere il proprio lavoro attraverso un link, per poterlo diffondere ad esempio in quei siti come StackOverFlow[2] dove per rispondere ad una domanda si può allegare un pezzo di codice con la propria soluzione.

Sono anche utilizzati come strumento pedagocico da alcune istituzioni come Khan Academy[3] per permettere agli studenti di fare esperienza nello scrivere codice in un ambiente dove possano vedere il risultato dei loro programmi senza il bisogno di alcuna installazione al di là di un semplice web-browser.

2.1.1 Tool in commercio

Attualmente in commercio troviamo molti sistemi web playground e i tre più popolari sono[SHA14]:

- JSBin[4](di Remy Sharp): è stato ed è tutt'ora uno strumento molto apprezzato dal pubblico dato che si focalizza sulle necessità basilari degli utenti e le gestisce estremamente bene. Consente a differenza di tutti gli altri tool in commercio di poter interagire con una vera e propria console dei comandi, per poter ad esempio testare direttamente nell'ambiente di lavoro il comportamento dinamico fornito al codice che si sta sviluppando.

[1] http://en.wikipedia.org/wiki/Online_JavaScript_IDE
[2] http://stackoverflow.com/
[3] http://en.wikipedia.org/wiki/Khan_Academy
[4] http://jsbin.com/

FEATURE COMPARISON TABLE

	JSbin	JSfiddle	Codepen
Live Output	Yes	No (In Pipeline)	Yes
HTML pre-processor	Markdown, Jade	No	Markdown, Jade, Haml, Slim
CSS pre-processor	LESS	SCSS	SCSS, SASS, LESS, Stylus
JS pre-processor	CoffeeScript, TypeScript, Traceur, JSX	CoffeeScript	CoffeeScript, LiveScript
CSS libraries	No	Normalize	Normalize, Reset
JS libraries	40+	30+	8
External file add	Manual	Easy	Very Easy
Keyboard Shortcuts	Yes	Yes	Yes
Account	Free	Free	Free + Paid
Like, Comment, Follow	No	No	Yes
Fork	No	Yes	Yes
Private works	Paid	Free	Paid
Tags	No	No	Yes
Panel Hide	Yes	No	Yes (minimizable)
Error & Warnings	Only JS (Real time)	Only JS (Not real time)	HTML, CSS, JS (Not real time)
Collaboration	No	Very good and free	Good but paid
Theme	Yes	No	Yes
Dummy Ajax	No	Yes	Yes
Bracket Highlight	Yes (Customizable)	Yes	Yes
Auto end bracket	Yes	No	No
Line Numbers	Yes (Customizable)	Yes	Yes
Speed	Very fast	Slow	Fast
Embedding	Yes	Yes	Yes
SEO friendly	Yes	Yes	Yes
Download	Yes	No	Yes
Locally Installable	Yes	No	No

Figura 2.1: Confronto funzionalità Tool di web playground [SHA14]

- JSfiddle[5](di Oskar Krawczyk): è stato uno dei primi web playground ed ha avuto una grande influenza per tutti gli strumenti che sviluppati successivamente nel campo. Può essere utilizzato per qualsiasi combinazione di HTML, CSS e Javascript. Al giorno d'oggi offre nella maggior parte funzionalità semplici, ma anche più complesse, come la simulazione di chiamate AJAX.

- Codepen[6](di Chris Coyier, Tim Sabat and Alex Vasquez): il miglior strumento sotto il punto di vista dell'aspetto dell'interfaccia grafica proposta all'utente. Il servizio mette in evidenza nella pagina principale i lavori più popolari e consente acquistando l'abbonamento pro di servirsi della modalità Collaborativa che consente di effettuare una vera e propria sessione di pair programmaing e della modalità Professore, per consentire ad un gruppo di studenti di seguire le dimostrazioni di codice del docente e chattare con gli altri.

Come si può notare in Figura 2.1 questi tre tool offrono diverse funzionalità a livello di ambiente di sviluppo integrato e condivisione dei propri lavori; c'è chi offre una maggiore velocità, chi un'iterazione con i social network, chi un maggior numero di librerie da poter aggiungere. La voce che mi sta più a cuore però è l'ultima presente nella tabella di comparazione, denominata "locally installable", che significa strumento reinstallabile sulla macchina dell'utente.

Solo JSBin tra i tool analizzati presenta la voce "Yes" in questa casella ed è per questo motivo, unito alla sua natura open source , atta a renderlo modificabile liberamente dalla comunità, che è possibile crearne una installazione personalizzata direttamente sulla propria macchina.

Questo consente di utilizzare JSBin anche per assolvere funzionalità diverse da quelle per cui è stato progettato, consentendo all'utente di modificarne localmente in codice, ad esempio adattando tale piattaforma per rispondere a nuove esigenze.

[5] `http://jsfiddle.net/`
[6] `http://codepen.io/`

2.2 JSBin

JSBin è una piattaforma collaborativa per lo sviluppo e la messa a punto di applicazioni web creata nel 2008 da Remy Sharp, è completamente open source e suo codice è disponibile online [7].

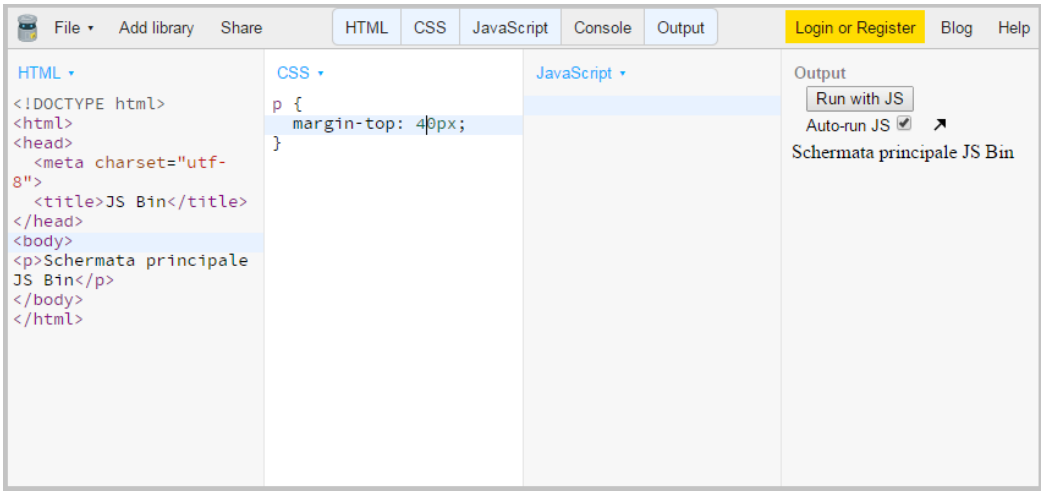

Figura 2.2: Schermata principale JSBin

La piattaforma permette di[8]:

- Scrivere codice, salvandolo e potendolo vedere renderizzato in una preview in real-time;
- Condividere il proprio lavoro tramite un url con altre persone, permettendogli di modificarlo a loro volta;
- CodeCast (termine definito direttamente dal suo fondatore[9]): condivisione di ciò che si sta scrivendo in JSBin in tempo reale;
- Remote Rendering: visualizzare l'output di un applicativo web realizzato con JSBin in ogni dispositivo su qualunque piattaforma, aggiornato in tempo reale;

[7] https://github.com/jsbin/jsbin
[8] http://jsbin.com/about
[9] https://remysharp.com/2013/11/14/what-is-codecasting

- Debug di chiamate Ajax remote;

- Installare localmente l'applicativo stesso, per poter utilizzarlo anche offline e personalizzarlo per le proprie esigenze.

Ci sono moltissime funzionalità in JSBin oltre quelle sopra citate, ma l'obiettivo dell'applicativo è di fornire uno strumento per aiutare a risolvere i problemi, esplorare la tecnologia e insegnare agli altri.

Per dare un'idea più precisa e puntuale riguardo a cosa rappresenta JSBin, il suo ideatore utilizza un'analogia:

"If you wanted to build the coolest, fastest, power hungry car in the world, or perhaps teach a group of students how an engine works or show how aerodynamics are applied to car design, then JS Bin is the garage full of all the tools you need to do your job." [10]

Ci sono alcuni casi di utilizzo di JSBin proposti dal suo fondatore, Remy Sharp. Il primo è l'insegnamento. Gli insegnanti possono utilizzare JSBin come piattaforma per mostrare come funzionano i linguaggi web (HTML, CSS, Javascript, ma anche linguaggi che li generano, come CoffeeScript, LESS e Markdown).

Gli studenti hanno solo bisogno di utilizzare un browser per avviare la creazione di una pagina web e di vedere l'effetto diretto del loro codice nell'output.

I docenti possono anche utilizzare la funzionalità di CodeCasting di JSBin condividendo con gli studenti l'url della pagina web dove stanno scrivendo il loro codice in tempo reale, in modo tale che una volta connessi a suddetta url gli alunni possano osservare le modifiche effettuate in real-time dal professore, grazie all'aggiornamento istantaneo fornito da JSBin. In questa maniera viene permesso l'insegnamento o i tutorial da remoto.

[10] https://github.com/jsbin/learn/blob/master/public/help/what-is-jsbin.md

La mia idea invece riguardo all'utilizzo di JSBin in ambito educativo è quella di usare tale strumento per fare esami di tecnologie web all'università sfruttando parte delle sue caratteristiche già esistenti come web playground per consentire agli studenti di lavorare in un ambiente confortevole, estendendole per permettere al professore di gestire un'intera prova d'esame dall'inizio alla fine.

Capitolo 3

ExaM Bin: nuovo approccio per fare esami di tecnologie web all'università

3.1 Esami di tecnologie web all'università: evoluzione ideale

Per gli insegnanti di computer science è diventato con il tempo un problema capire quale metodologia utilizzare per valutare le abilità effettivamente acquisite da uno studente durante l'apprendimento della materia.

Parlando però di corsi universitari a livello di programmazione web la naturale forma di valutazione dovrebbe essere computer-based, poiché questo tipo di approccio come si evince dal primo capitolo, consente di avere una fotografia più nitida e definita delle conoscenze possedute dagli alunni a livello di abilità pratiche.

La materia si presta a trarre notevole beneficio da questo tipo di metodologia in quanto il professore può così fornire agli studenti un ambiente dove creare una pagina web, capacità che si è soliti richiedere come verifica delle abilità acquisite in un corso di tecnologie web. In questo modo l'alunno è

in grado di testare il rendering del codice scritto ed ha a disposizione una console dei comandi per, ad esempio, la verifica dei comportamenti dinamici di cui la pagina è dotata (esecuzione codice Javascript).

Inoltre essendo il web-programming uno dei campi di notevole interesse riguardo la valutazione effettuata in maniera automatica, il docente potrebbe voler far diminuire il proprio carico di lavoro facendo correggere i compiti direttamente alla macchina.

Quindi la piattaforma ideale che il professore dovrebbe utilizzare per fare ad esempio un esame finale computer-based, nel quale testare la abilità pratiche di programmazione web, dovrebbe fornire:

- un ambiente di sviluppo integrato
- un sistema server-side di valutazione automatica
- un sistema di gestione della temporizzazione

Infine dovrebbe essere open source per permetterne una migliore diffusione nella comunità, dando la possibilità ai futuri utilizzatori di modificare la piattaforma a loro piacimento, in modo tale che possa con semplicità rispondere alle loro particolari esigenze.

3.2 ExaM Bin: un approccio nuovo

ExaM Bin è un'applicazione open source per fare esami di tecnologie web all'università, basata sul tool open source per debugging collaborativo a livello di sviluppo web denominato JSBin. I suoi obiettivi sono quelli di permettere ad un professore di gestire nella sua interezza una prova d'esame con le esigenze ad essa annesse di sicurezza e verificabilità e ad uno studente di poter interagire con l'ambiente più consono per provare le proprie abilità di programmazione.

L'idea alla base della progettazione e dello sviluppo della piattaforma è nata dall'intenzione di utilizzare le recenti risorse nel campo delle tecnologie

web per costruire un'applicazione utile ed affidabile nell'ambito accademico, che migliorasse l'esperienza d'esame sia di professori che di studenti.

Il concetto si fonda anche su una mia esperienza personale, dato che quest'anno ho dovuto affrontare un esame di tecnologie web in formato cartaceo, che mi ha permesso di rilevare in prima persona tutte le difficoltà e limitazioni di tale metodologia applicata alla materia.

Dover renderizzare nella propria testa una pagina web non è cosa semplice, per non parlare dell'esecuzione di codice Javascript; si barcolla nel buio, si fanno correzioni su correzioni fino ad arrivare ad una versione del codice da presentare al docente che non sapremo mai se funzionerà.

Quindi la prima esigenza per lo sviluppo dell'applicativo, derivante dalla mia esperienza personale, è stata quella di fornire un ambiente agli studenti dove potessero visualizzare in tempo reale il rendering della pagina web che stavano creando e avere inoltre la possibilità di testare il comportamento dinamico fornito a suddetta pagina, potendo eseguire codice Javascript.

Altra necessità era quella di fornire al professore un applicativo che rispondesse alle esigenze di sicurezza e verificabilità tipiche di una prova d'esame e che gli permettesse di usufruire di un servizio di validazione automatica.

Ultimo, ma non meno importante, l'aspetto open source, poiché si è deciso di utilizzare tecnologie open source presenti sul mercato per lo sviluppo del tool e per dare la possibilità di applicare modifiche e studiare liberamente il codice del progetto, in modo tale che chiunque possa personalizzarlo per adeguarlo alle proprie esigenze.

Visto il notevole benefit portato dagli ambienti di web playground nel rispondere all'esigenza degli alunni riguardo al fatto di poter lavorare in un contesto confortevole, ExaM Bin nasce come adattamento di una piattaforma open source di questo tipo alle peculiarità di una prova d'esame. Nello specifico si è deciso di iniziare lo sviluppo partendo dall'applicativo JSBin, unico nel suo genere a consentire una sua installazione locale nella macchina dell'utente, personalizzabile a piacimento.

A questa base sono stati integrati, un sistema di gestione della tempo-

rizzazione e un modulo per eseguire la correzione automatica dei compiti per poter rispondere attraverso l'uso del calcolatore anche alle esigenze dei docenti.

3.3 Quadro generale delle funzionalità

ExaM Bin permette di gestire nella sua completezza una prova d'esame. Essendo i reali utilizzatori studenti e professori, l'applicativo fornisce funzionalità diverse e personalizzate in base a queste due categorie di utenti.

Queste sono a grandi linee le funzionalità che il sistema offre:

- Permette al docente di realizzare una domanda di HTML e/o CSS e/o Javascript e/o vari framework, di cui può fornire codice mezzo completo o con errori.
- In un laboratorio chiuso ad Internet, ma collegato ad un server su cui gira una versione di ExaM Bin, consente agli studenti dopo una procedura di login di avere a disposizione un ambiente di lavoro JSBin like semplificato per poter rispondere alla domanda posta dal docente, avendo sempre a disposizione console e output del compito che stanno redigendo.
- Permette di salvare il compito di ogni studente in una copia privata aggiungendo nome, cognome e matricola dell'alunno in questione; il meccanismo di salvataggio viene attivato ad ogni modifica apportata dallo studente nell'interfaccia dove realizza il compito.
- Dopo aver effettuato le modifiche necessarie al testo per risolvere l'esercizio, lo studente ha la possibilità di dichiarare di aver concluso il compito e consegnarlo al docente per la correzione.
- Infine, a rete riattivata, permette al professore di correggere/validare i compiti di una certa prova d'esame in maniera automatizzata, sia stan-

dard che personalizzata, eseguendo un test su ogni compito e generando un rapporto completo.

L'applicativo è pensato per essere installato su di un server a cui gli studenti si possono collegare da un ambiente asettico, come ad esempio un laboratorio chiuso ad internet, dove la rete viene riattivata una volta che la prova d'esame è conclusa per consentire al docente di utilizzare il servizio di validazione automatica.

Le linee guida che hanno portato alla realizzazione delle interfacce atte alla fruizione dei servizi sopra citata, si riscontrano nel fatto di garantire una navigazione il più semplice e intuitiva all'utente.

Andiamo ora ad analizzare nel dettaglio le funzionalità suddividendo l'analisi per utente utilizzatore.

3.4 Funzionalità Professore

Il progetto trattato offre al professore tutte le funzionalità ad esso dedicate tramite un'interfaccia web, a cui ci si può collegare utilizzando un qualsiasi web browser installato nella macchina dedicata all'insegnante.

Il docente infatti non viene sottoposto a nessun controllo a livello di identificazione, per esempio attraverso una schermata di login, poiché l'applicativo è stato progettato in maniera tale da consentire l'accesso ai servizi dedicati al professore solamente da una certa postazione configurata nelle impostazioni di partenza.
Le funzionalità legate al professore sono suddivise in due macroaree:

- gestione prova d'esame
- gestione correzione prova d'esame

Per ognuna della due sezioni è stata progettata un'apposita interfaccia web per dare una suddivisione netta a questi due tipi di servizi che vengono

effettuati in momenti diversi. Lo spazio legato alla gestione della prova d'esame è disponibile all'indirizzo "/professor", il secondo invece è disponibile all'indirizzo "/professor/correction".

3.4.1 Gestione prova d'esame

La gestione della prova d'esame può essere effettuata solamente dal professore, da una macchina configurata appositamente per lui. Si può entrare nell'interfaccia grafica di tale servizio accedendo attraverso il browser web all'indirizzo "/professor" del server in cui è installato l'applicativo ExaM Bin.

Figura 3.1: Pannello di controllo del professore

Come si può notare nella Figura 3.1 la navigazione nell'interfaccia del pannello di controllo del professore è governata da tre pannelli nella parte superiore della pagina che individuano le varie fasi di cui si compone l'intera gestione della prova d'esame:

1. **Setup**: dove viene data la possibilità al professore di inserire le informazioni per iniziare una nuova prova d'esame;

2. **Start**: dove il docente ha la possibilità di far iniziare ufficialmente la prova d'esame;

3. **Exam Info**: dove l'insegnante ha a disposizione le informazione relative all'andamento dell'attuale prova d'esame e può ufficializzare il termine della stessa.

Inizialmente solo uno dei pannelli è attivo, in base alla fase dell'esame in cui ci troviamo a livello di temporizzazione.

I tre pannelli sono legati in maniera consequenziale tra loro, quindi ad esempio non si può avere accesso ai pannelli di Start e Exam Info se non si è concluso la fase di Setup. Il termine delle operazioni relative ad una certa sezione avviene quando il professore fa scattare un certo tipo di evento nel sistema, nella maggior parte dei casi premendo un pulsante che gli permette di essere indirizzato nella sezione successiva rispetto a quella in cui si trova. Infatti se il docente termina la fase di setup delle informazioni utili a far iniziare il compito e preme il pulsante "Permetti agli studenti di loggarsi", comunicherà in questo modo al sistema che è pronto a far iniziare la prova d'esame e quindi nel panello di controllo verrà reindirizzato alla sezione di Start.

La consequenzialità ruota attorno al ciclo di vita dell'esame stesso, terminata la fase di Setup si passa alla fase di Start, terminata la fase di Start si passa a quella di Exam Info; il cammino è ciclico. Arrivati alla fase di Exam Info il professore avrà a disposizione un comando per terminare l'attuale prova d'esame, alla cui attivazione, verrà reindirizzato nella sezione relativa al setup per poter iniziarne una nuova.

Parleremo di ogni transizione di stato più accuratamente nelle sottosezioni seguenti atte a definire maggiormente nel dettaglio ogni funzionalità delle singole sezioni presenti nel pannello di controllo del professore.

3.4.1.1 Setup

Nel pannello di setup vengono messi a disposizione del professore gli strumenti che consentono di caricare i dati per poter cominciare in modo corretto una nuova prova d'esame. Nell'interfaccia grafica come si può notare in Figura 3.1 sono presenti due form distinti e affiancati tra loro; uno sulla

sinistra relativo alla gestione dei file del compito e uno sulla destra relativo alla gestione della temporizzazione della prova d'esame.

Nel primo form relativo alla gestione dei file del compito, visibile in Figura 3.2, viene data la possibilità al docente di precaricare un file HTML e/o CSS e/o Javascript (le tecnologie web che vogliono essere valutate con la piattaforma ExaM Bin), che premendo il pulsante "Aggiorna" verranno utilizzati come base per lo svolgimento del compito. Il contenuto di ogni file viene riversato nell'area dedicata alla scrittura dello specifico codice nella schermata dedicata alla realizzazione pratica dell'esame.

Il pulsante è chiamato volutamente aggiorna e non ad esempio carica file poiché il sistema al suo avvio presenta già un file predisposto per contenere il codice iniziale. Quando si vanno ad inserire i file del compito in realtà si aggiornano delle informazioni già presenti nel sistema. Si è deciso di adottare questo tipo di approccio poiché consente al professore di utilizzare in una nuova prova d'esame gli stessi file che ha caricato nell'ultima, senza doverli necessariamente ricaricare nella piattaforma.

Figura 3.2: Form gestione file compito

Tale meccanismo permette al professore di fornire codice incompleto o errato e verificare le abilità pratiche di programmazione degli alunni chiedendogli di completarlo o di correggere gli eventuali errori ivi presenti. Inoltre in suddetto form l'insegnante premendo sul pulsante denominato "Attuale" può accedere e visualizzare la schermata d'esame a cui gli studenti verranno indirizzati a prova iniziata.

Infine con un semplice click sul pulsante "Pulisci" il docente potrà eliminare l'intero contenuto della schermata d'esame nelle sezioni HTML, CSS e Javascript, in modo tale da fornire un ambiente pulito agli studenti sen-

za nessun riferimento iniziale. In questo caso gli alunni quando dovranno iniziare il compito si troveranno davanti tre sezione vuote a livello di codice HTML, CSS e Javascript, dove potranno implementare da zero la propria soluzione, seguendo le consegne del professore. Della schermata relativa all'ambiente d'esame parleremo più approfonditamente nella sezione dedicata alle funzionalità per lo studente.

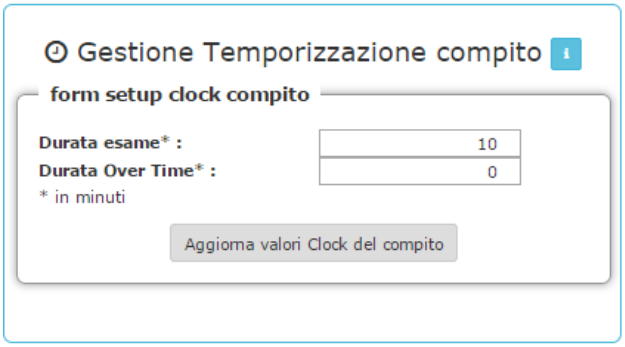

Figura 3.3: Form gestione temporizzazione

Nel secondo form, visibile in Figura 3.3, relativo alla gestione della temporizzazione, il professore può visualizzare i valori di default a livello di durata esame e durata over time presenti come placeholder negli appositi elementi hmtl del form. Può decidere di cambiarne il valore in qualsiasi momento digitandolo nell'apposito spazio e premendo il pulsante "Aggiorna valori Clock del compito". Con Over time ci si riferisce a quel lasso di tempo che scatta al termine legale della prova d'esame, con il quale il professore può consentire agli studenti di ultimare il proprio lavoro per poi consegnarlo al sistema senza che quest'ultimo termini in maniera automatica l'attuale prova d'esame. La gestione della temporizzazione lato server verrà trattata in maniera più approfondita in un'apposita sezione nel capitolo successivo.

Come possiamo notare in ciascuno dei due form vicino al titolo è presente un'icona su sfondo blu che, se premuta, fornisce attraverso l'apertura di una finestra modale le informazione necessarie alla corretta e consapevole compilazione del form da parte dell'utente stesso.

Infine nell'area inferiore della sezione di setup come visibile in Figura 3.1, troviamo il pulsante "Permetti agli studenti di loggarsi" con la pressione del quale il professore dichiara di aver completato la fase di setup. Tutte le in-

formazioni necessarie alla partenza di un nuovo esame sono quelle corrette e la piattaforma è pronta per permettere al docente di dare il via in maniera ufficiale al compito. Inoltre con la pressione di quest'ultimo pulsante, l'insegnante consentirà agli alunni di accedere al sistema facendo login e passerà nella sezione successiva del pannello di controllo relativa alla fase di start.

Non è detto che il professore abbia deciso di inserire nuovi file o nuovi orari per l'esame, il sistema in qualsiasi caso darà la possibilità al professore di passare alla fase successiva con i valori di default presenti in esso in quello stesso istante.

3.4.1.2 Start

L'interfaccia relativa alla sezione di start come si può notare in Figura 3.4 è molto semplificata e il suo unico obiettivo è quello di dare la possibilità al professore di far partire a tutti gli effetti la prova d'esame.

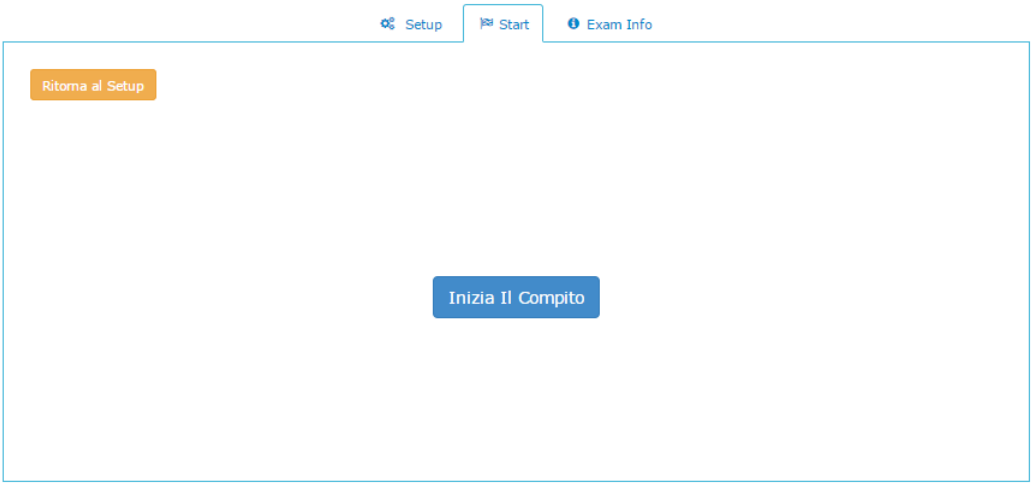

Figura 3.4: Sezione Start pannello di controllo del professore

Il caso d'uso relativo a questa sezione è focalizzato ovviamente sul docente che tuttavia oltre ad interagire con il pannello di controllo deve verificare ciò che stanno facendo gli alunni sulle proprie macchine. Infatti il professore, dopo aver verificato visivamente che tutti gli studenti partecipanti alla prova

d'esame abbiano fatto login (quest'ultimi verranno indirizzati in un'apposita pagina dopo essersi identificati al sistema), ha la facoltà di premere sul pulsante "Inizia il compito" che permetterà alla piattaforma di far iniziare l'esame. Questa operazione consentirà il reindirizzamento automatico degli alunni nella schermata per la realizzazione dell'esame e porterà il professore nella sezione finale del pannello di controllo denominata Exam info.
Il compito è ufficialmente iniziato.

In caso di errori nella fase di setup il professore potrà decidere di ritornarvi premendo il pulsante "Ritorna al Setup" (in alto a sinistra su sfondo arancio), che gli darà la possibilità di rivedere le proprie scelte in fatto di informazioni iniziali.

3.4.1.3 Exam Info

Infine giungiamo all'ultima sezione del pannello di controllo, Exam Info. L'esame è iniziato correttamente e gli studenti stanno svolgendo il loro compito.

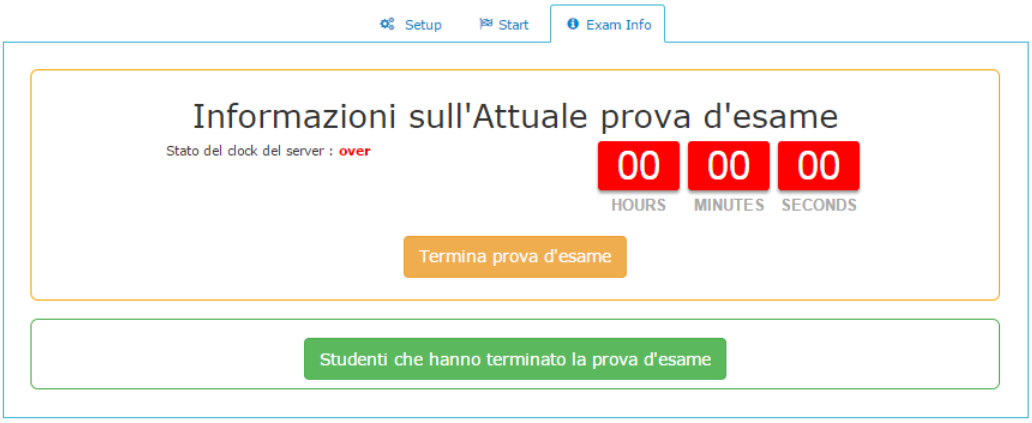

Figura 3.5: Sezione Exam Info pannello di controllo del professore

In questo pannello come esplicitato dal titolo, vengono fornite le informazione relative alla prova d'esame in corso. Nella parte superiore dell'interfaccia, visibile in Figura 3.5, viene mostrato lo stato attuale del clock di sistema e al suo fianco attraverso un countdown viene visualizzato il tempo mancante

al termine dell'esame. Nell'area in basso invece, contraddistinta da un bordo verde, viene consentito al professore, prendendo il pulsate "Studenti che hanno terminato l'esame", di visualizzare nome, cognome e matricola degli studenti che hanno dichiarato al sistema di aver terminato il proprio compito. Questo meccanismo è stato realizzato per dare un feedback immediato al docente.

Al termine del tempo effettivo della prova d'esame, quando tutti gli studenti hanno consegnato volontariamente o meno il proprio compito, nell'area relativa alle informazioni attuali, il countdown cambierà colore di sfondo, diventando rosso, e verrà visualizzato un nuovo pulsante denominato "Termina prova d'esame". Tale oggetto consente al professore, se premuto, di dichiarare al sistema che l'attuale prova d'esame è conclusa. In questa maniera il docente ritornerà nella sezione di setup del pannello di controllo per poter impostare le informazioni relative all'inizio di un nuovo compito.

Riassumendo il tutto relativamente alla gestione della prova d'esame, il professore, dopo aver settato i valori iniziali del compito, può dare la possibilità agli studenti di fare login nel sistema. Una volta verificato visivamente che tutti i presenti si sono autenticati al sistema può far iniziare effettivamente l'esame avendo in ogni momento a disposizione le informazioni relative alla temporizzazione e la possibilità di verificare quali sono gli studenti che hanno terminato il compito. Infine può concludere la prova d'esame predisponendo il sistema per una nuova.

3.4.2 Gestione correzione prova d'esame

La seconda macroarea di interesse per il docente è quella relativa alla gestione della correzione dei compiti di una singola prova d'esame. Il servizio è disponibile all'indirizzo "professor/correction" del server in cui è stato installato l'applicativo ExaM Bin e vi si può accedere direttamente o attraverso il pannello di controllo del professore premendo il pulsante in altro a destra, denominato "pagina di correzione", presente in Figura 3.1.

Lo scopo principale di questa sezione è quello di consentire al docente di effettuare la validazione automatica degli esami redatti in una certa prova. Ovviamente anche questa funzionalità è accessibile solamente attraverso la postazione configurata per il docente. In particolare per una corretta fruizione del servizio il server in cui è installato ExaM Bin deve poter accedere alla rete internet.

Figura 3.6: Pannello di correzione compiti del Professore

L'interfaccia grafica proposta, come si può notare in Figura 3.6, presenta tre sezioni principali; due delle quali dedicate al setup delle informazioni per procedere alla correzione dell'esame e una più in basso, con il contorno verde, che permette di effettuare realmente la verifica mostrando in una tabella il report di tutti i risultati.

La correzione degli elaborati di una certa prova d'esame viene fatta applicando sia un'analisi statica che controlla imperfezioni stilistiche ed errori a livello di compilazione del codice stesso, sia un'analisi dinamica che permette al professore di verificare qualunque caso d'uso lui voglia eseguendo il codice stesso. Per l'analisi statica non viene chiesto nessuno sforzo al docente, invece parlando di analisi dinamica il professore deve creare un file con una suite

di test a cui sottoporre ogni compito redatto dagli studenti. Il risultato, a livello di errori riscontrati, viene pesato e il sistema propone una valutazione al docente.

3.4.2.1 Setup informazioni validazione

Anche la fase di correzione ha bisogno di alcune informazioni per un corretto setup, che sono presenti di default nel sistema, ma che possono essere aggiornate in qualsiasi momento dal professore. Queste riguardano la validazione personalizzata e i pesi utilizzati dal sistema per fornire al docente la proposta di voto.

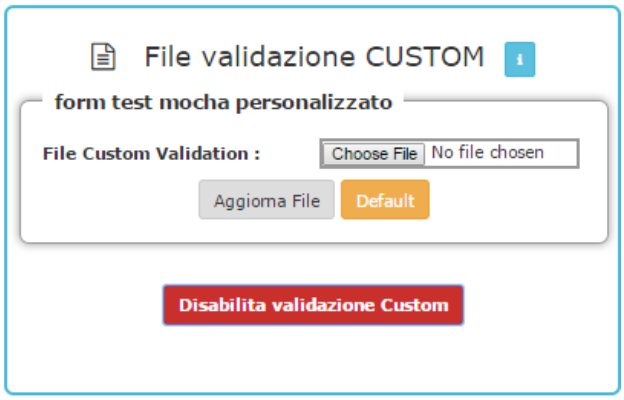

Figura 3.7: Form setup file validazione custom

Dovendo l'insegnante fornire al sistema il file contenente le direttive sulle quali eseguire i test personalizzati a livello di singoli casi d'uso, in un primo form (visibile in Figura 3.7) si dà la possibilità al professore di fare l'upload di tale file. Nel caso in cui il docente volesse eseguire questo tipo di validazione con il file presente nel sistema può visualizzarlo in qualsiasi momento premendo il pulsante "Defaul". Quindi se il professore carica un nuovo file per la validazione personalizzata e e lo invia al sistema tramite il pulsante "Aggiorna file", quando andrà a premere "Default" vedrà come file impostato per la validazione custom nell'applicativo il file che ha appena uplodato.

Dato che la validazione custom può richiedere tempo al professore per redigere il file con cui testare il compito degli alunni, il sistema fornisce la possibilità premendo il pulsante "Disabilita validazione Custom" di disattivare questo particolare meccanismo in fase di correzione. L'applicativo in

questo caso effettuerà una mera analisi statica del codice e il form in questione conterrà un solo pulsante su sfondo verde denominato "Abilita validazione custom" come visibile in Figura 3.6, che se premuto ripristinerà la questo tipo di validazione nella correzione del compito.

Inoltre cliccando sul pulsante con l'icona informativa vicino all'intestazione del form viene mostrata all'utente una finestra modale che fornisce tutte le informazioni necessarie per generare in maniera corretta un file per la validazione personalizzata, fornendone anche un esempio.

Si è fatto in modo che la scrittura di un file di validazione dinamica fosse la più semplice possibile, non richiedesse al docente particolari conoscenze ed inoltre fosse testabile dal professore al di fuori del sistema. Quest'ultimo punto è particolarmente interessante poiché: come fa il professore a sapere se il file da lui redatto effettua i test in maniera corretta senza averlo mai provato?

Per rispondere a questa domanda il sistema richiede un file di validazione personalizzata che può essere facilmente integrato in una pagina web. Questo poiché non si è deciso di creare un modulo di correzione automatica ad hoc per la piattaforma, ma ci si è serviti di strumenti di unit test che potessero avere una semplice integrazione sia client che server side. Nello specifico parliamo di Mocha.js e Chai.js, di cui parleremo approfonditamente in una specifica sezione nel prossimo capitolo.

Quindi il professore prima di correggere il compito può crearsi una sua soluzione della prova d'esame e poi aggiungerci il file che ha creato per la validazione personalizzata per testare con il suo web browser in prima persona se le regole che ha definito sono corrette ed esaustive.

Nella generazione del voto proposto dal sistema relativamente ad ogni prova d'esame un fattore importante è rappresentato dai pesi attribuiti ad ogni singolo errore identificato a livello di validazione del compito.

Queste informazioni sono presenti nel secondo form, affiancato sulla destra a quello descritto precedentemente, visibile in Figura 3.8. Come placeholder nelle apposite textbox sono presenti i valori attualmente attribuiti

dal sistema sia in ambito statico (HTML, CSS e Javascript standard) che dinamico (validazione personalizzata).
Inoltre come ultimo campo del form troviamo il voto di partenza.

Per cambiare uno qualsiasi di questi pesi al professore basta digitare il nuovo valore nella textbox relativa a quel determinato campo e premere il pulsante "Aggiorna pesi per validazione" presente nella parte inferiore del form. Ogni errore riscontrato verrà moltiplicato per il proprio peso e il valore ottenuto verrà sottratto al voto di partenza, in modo tale da ottenere la proposta di voto che sarà presentata al docente a correzione avvenuta nel report complessivo. La Figura 3.9 mostra un esempio di generazione del voto finale relativamente ad un singolo compito d'esame da parte de sistema.

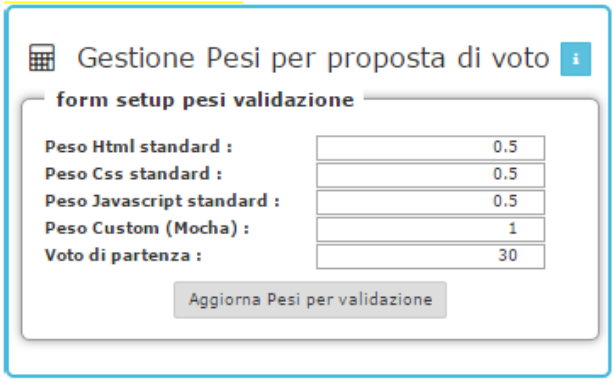

Figura 3.8: Form setup pesi proposta di voto

```
Esempio:
Peso validazione HTML standard = 0.5      errori HTML standard = 1
Peso validazione CSS standard = 0.5       errori CSS standard = 2
Peso validazione JS standard = 1          errori JS standard = 0
Peso validazione CUSTOM standard = 2      errori CUSTOM standard = 1
Voto di partenza = 30

Voto finale = 30 - (0.5 * 1 + 0.5 * 2 + 1 * 0 + 2 * 1) = 26.5
```

Figura 3.9: Esempio generazione voto

3.4.2.2 Correzione

Dopo aver modificato o meno le impostazioni del sistema a livello di correzione dei compiti il professore ed ovviamente anche il sistema sono pronti

per la validazione della prova d'esame.

Per far svolgere questa operazione in maniera automatica alla piattaforma al docente è richiesta l'iterazione con la form di avvio del processo di correzione visibile in figura Figura 3.10 e presente nella parte inferiore dell'interfaccia complessiva di correzione (Figura 3.6).

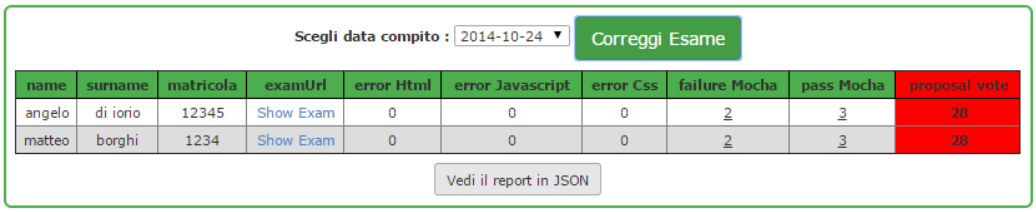

Figura 3.10: Sezione Correzione Esami

Per prima cosa il docente deve selezionare la data dell'esame che vuole correggere e poi fare click sul pulsante "Correggi Esame". Questo evento avvierà il processo di correzione lato server che dopo aver elaborato uno per uno gli esami sostenuti in quella giornata invierà un rapporto completo e dettagliato.

Questo rapporto verrà visualizzato nella form di correzione in un'apposita tabella, dove in ogni riga saranno caricati i dati relativi ad un singolo compito riguardo l'identificazione dell'alunno, gli errori commessi e il voto finale proposto. Inoltre in ogni record è presente un link al compito consegnato dallo studente, in modo tale che il professore possa verificare con mano il lavoro effettivamente svolto.

Se il sistema ha riscontrato degli errori nella correzione del compito il professore con un semplice click del mouse sul numero presente nella relativa casella del report potrà visualizzarli in dettaglio tramite una finestra modale dedicata. Per esempio se un compito non ha passato due test definiti dal professore nella validazione personalizzata, nel report relativo a quel compito sarà presente il numero due nella cella relativa agli errori a livello di quel tipo di validazione, e cliccandovici sopra verrà sottoposta al docente una modale contenente i titoli di quei due test che non sono andati a buon fine.

Inoltre l'insegnante potrà visualizzare il file json contenente il report completo dei risultati della prova d'esame cliccando sul pulsante che comparirà sotto la tabella, denominato "Vedi il report in JSON". In questa maniera potrà avere a disposizione anche offline tutti i dati relativi alla correzione appena effettuata per poter verificarli in qualsiasi momento senza interagire con il sistema stesso.

Riassumendo il tutto relativamente alla gestione della correzione dei compiti di una prova d'esame, il professore a livello di setup può definire una suite di test per la validazione cosiddetta "custom" e i pesi di ogni singolo errore utilizzati dal sistema per la generazione del voto proposto. Successivamente può attivare il servizio di correzione automatica server-side abilitando o meno la validazione personalizzata e visualizzare i risultati in un report complessivo in formato tabellare con cui può avere maggiori dettagli riguardo ogni tipo di errore riscontrato e accedere inoltre al compito di ogni singolo alunno che ha preso parte alla prova d'esame.

3.5 Funzionalità Studente

Passiamo ora alle funzionalità relative allo studente che personalmente mi stanno più a cuore. In poche parole ad ogni alunno è concesso di fare un compito di tecnologie web nell'ambiente più consono possibile.

Come prima cosa viene permesso allo studente di identificarsi al sistema e successivamente, dopo che il professore ha dichiarato l'effettivo inizio dell'esame, gli viene consentito l'accesso alla pagina contenente l'ambiente in cui avrà modo di sviluppare il compito consegnandolo una volta terminato.

3.5.1 Login

La prima interfaccia con cui gli studenti hanno a che fare è quella di login. Il servizio è disponibile all'indirizzo "/student", che viene indicato direttamente dai docenti agli studenti stessi, su di un pezzo di carta, sulla lavagna o in altra maniera.

Figura 3.11: Pagina di login dello studente

L'interfaccia in questione come si può notare in Figura 3.11 richiede agli alunni la compilazione di tre campi per la propria identificazione con il sistema: nome, cognome e matricola, che verranno utilizzati come metadati del compito nel suo effettivo storage nel sistema. Questi valori sono necessari per creare una prova d'esame univoca. Non viene verificato se effettivamente esiste uno studente con quei dati all'università di Bologna poiché l'aggiornamento delle informazioni di cui ci si dovrebbe servire non avviene in tempo reale, quindi un alunno potrebbe essere iscritto regolarmente all'università, ma non ancora presente nelle strutture informatiche.

Il pulsante di login viene abilitato nella schermata solamente quando il professore ha terminato la fase di setup delle informazioni relative al compito, quindi quando ha premuto sul pulsante "Permetti agli studenti di loggarsi". Una volta effettuato il login gli studenti vengono reindirizzati nella pagina di setup avvenuto, dove è presente a caratteri cubitali la scritta "Studente Loggato correttamente e pronto al compito". L'interfaccia così evidente è stata progettata per facilitare la verifica visiva da parte del professore che tutti gli utenti sono pronti a sostenere l'esame.

Accertato che tutti i partecipanti siano entrati nel sistema il docente da inizio al compito e gli studenti vengono automaticamente reindirizzati alla

pagina di realizzazione compito.

3.5.2 Realizzazione compito

Una volta iniziato effettivamente l'esame gli studenti sono reindirizzati in questa pagina dove possono provare al professore le loro abilità in fatto di tecnologie web rispondendo alla domanda fornitagli dal docente nel testo d'esame.

L'ambiente proposto, come si può notare in Figura 3.12, è simile a quello dell'applicativo JSBin[1] su cui, come noto, è basata l'implementazione del progetto ExaM Bin (da notare l'assonanza nei nomi).

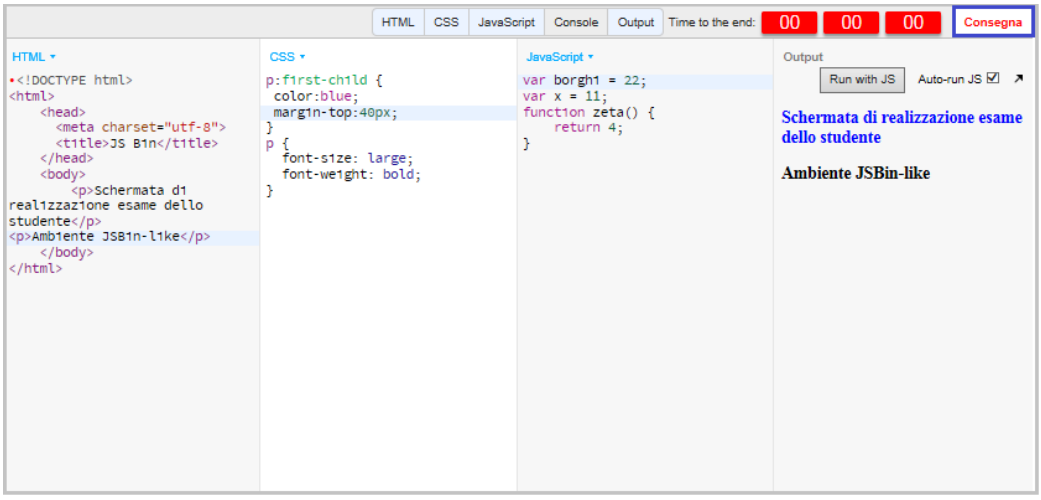

Figura 3.12: Pagina di realizzazione compito

Analizzando l'interfaccia possiamo notare cinque tab nella parte superiore in posizione centrale:

- HTML
- CSS
- Javascript
- Console
- Output

[1] http://jsbin.com/

Le prime tre definiscono le aree in cui lo studente può redigere il codice relativo al linguaggio di programmazione web indicato nell'etichetta. L'area Console invece mette a disposizione una vera console dei comandi al servizio degli studenti per testare ad esempio il comportamento dinamico della pagina che stanno sviluppando. Infine l'ultima, ma non meno importante, l'area relativa all'Output permette agli studenti di visualizzare in real time il rendering fornito dal proprio codice eseguendolo con l'ausilio o meno di codice Javascript. Ovviamente l'alunno può decidere in qualsiasi momento quali tab tenere aperte sotto il suo diretto controllo e quali chiudere.

Lo studente ha tutti gli strumenti necessari per testare ciò che sta facendo, può commettere errori, tentare di correggerli e provare e riprovare il comportamento dinamico del suo elaborato, qualcosa di impensabile in un'esame cartaceo dove si è limitati a carta e penna e alla propria immaginazione.

La prima volta che lo studente interagisce con una delle tre aree di codifica presenti nell'interfaccia il sistema crea uno spazio apposito al suo interno dove verrà mantenuto il compito in questione ed inoltre reindirizza lo studente implicitamente ad un url univoco generato con le credenziali che esso ha fornito in precedenza in fase di login. Ovviamente solo il ragazzo in questione può avere accesso a tale area. In questa maniera non si consente agli studenti di visualizzare altri compiti in fase di elaborazione tranne il proprio, evitando così eventuali tentativi di copiatura.

Il compito continuerà ad essere salvato automaticamente dal sistema senza richiedere nessuno sforzo all'utente, ogniqualvolta che quest'ultimo effettui una qualsiasi modifica al proprio lavoro.

Un'etichetta con la scritta "Saved", come si può notare in Figura 3.13, comparirà vicino al nome della sezione per identificare che tale meccanismo è andato a buon fine.

Figura 3.13

3.5.3 Consegna del compito: l'ultimo click

Per concludere descriviamo l'ultima funzionalità relativa allo studente, la consegna del compito. Nell'angolo in alto a destra in Figura 3.12 oltre al tempo rimanente per l'esecuzione della prova, visibile attraverso un oggetto countdown, si trova il pulsante che permette allo studente di dichiarare di aver terminato l'esame.

Una volta effettuato quest'ultimo click il sistema dovendosi accertare della volontà dello studente gli chiederà se è effettivamente pronto a consegnare il codice che a prodotto fino a quel momento come sua ultima versione.

In caso affermativo il sistema etichetta come concluso l'esame dello studente e lo reindirizza nella pagina di fine compito rendendo di fatto inaccessibili tutti i servizi del sistema all'alunno, che ora può lasciare la postazione del laboratorio dopo aver passato la più bella esperienza della propria vita.

Questo accade se lo studente decide di consegnare il compito prima dello scadere della durata dell'esame, ma se lo studente si attarda, perde di vista il tempo o è semplicemente tra le nuvole, cosa succede?
Ci pensa il sistema a consegnare per lui.

Per prima cosa quand'è finito il tempo della prova d'esame l'alunno viene avvertito visivamente dalla comparsa di un alert e dal fatto che cambia il colore di sfondo dell'oggetto countdown, che per l'occasione diventa rosso. Dopo queste prime avvisaglie il sistema consegnerà il compito automaticamente solo allo scadere del tempo di over time, intervallo di tempo che, come abbiamo visto precedentemente, può essere settato dal professore nell'incipit della creazione della prova e come vedremo successivamente, indica quel lasso di tempo in cui viene data ancora la possibilità agli studenti di procedere con lo sviluppo del proprio elaborato e di consegnarlo quando hanno terminato. Insomma parliamo di quei cinque minuti in più che il professore può decidere di concederci o meno dopo la fine puntuale del compito per ultimare ciò che abbiamo fatto.

Quindi il compito in maniera volontaria o meno viene sempre consegnato al sistema. Appena lo studente dichiara alla piattaforma di aver concluso,

il suo nominativo viene messo a disposizione del professore tra coloro che hanno terminato l'esame.

Riassumendo il tutto, gli studenti possono identificarsi al sistema tramite una pagina di login e successivamente dopo che il professore ha dato inizio al compito gli viene permesso di lavorare alla loro soluzione nella pagina di realizzazione dell'esame (ambiente JSBin like) che inoltre gli consente di dichiarare di aver concluso il compito, sottoponendo ciò che hanno elaborato fino ad ora alla correzione del docente.

3.6 Funzionalità accessorie

Le funzionalità accessorie del sistema riguardano per lo più il fatto di cautelarsi da possibili errori dovuti sia a malfunzionamenti degli strumenti utilizzati per accedere all'applicativo, ad esempio il web browser, sia ad un non corretto uso da parte dell'utente. Non trattano nello specifico errori interni al server.

Questo tipo di funzionalità riguardano sia lo studente che il docente e ora le andremo ad analizzare nel dettaglio.

3.6.1 Reset configurazione Esame

IL reset dell'attuale configurazione per l'inizio della prova d'esame è una funzionalità messa a disposizione al professore nel suo pannello di controllo.

Il docente dopo aver premuto il pulsante "Permetti agli studenti di loggarsi" dichiarerà al sistema che le attuali informazioni relative all'inizio di una nuova prova d'esame sono quelle corrette e avrà accesso al pannello di start. In suddetta area l'insegnante potrà ovviamente far iniziare ufficialmente l'esame, ma dato che potrebbe essersi accorto di aver configurato in maniera errata la prova d'esame gli verrà data la possibilità tramite il pulsante di "Ritorna al Setup" di tornare alla schermata di setup (altrimenti inaccessibile).

In questa maniera si consente al professore di modificare i parametri di partenza in qualsiasi momento prima di dare inizio ufficialmente alla prova d'esame.

3.6.2 Recupero compito studente

Questa funzionalità si riferisce allo studente e nello specifico al crash della sua attuale pagina di realizzazione del compito.

Questo può accadere per esempio se il web browser in cui lo studente ha instaurato una connessione con ExaM Bin ha un malfunzionamento o se lui stesso ha accidentalmente chiuso la propria schermata d'esame.

Per ovviare alla perdita di tutti i dati del compito in fase di svolgimento si è pensato di adottare un meccanismo di recupero che riportasse l'alunno alla sua ultima versione salvata sul server del compito.

Lo studente dopo aver avuto un problema tecnico tra quelli descritti precedentemente dovrà essere invitato dal docente a ritornare alla sua pagina principale, presente all'indirizzo "/student" del server in cui è installato l'applicativo ExaM Bin. Qui troverà sempre il form di login, in questo caso disabilitato, ma in più potrà notare il pulsante "torna al tuo compito" poiché il sistema ha identificato il fatto che esiste già un compito redatto dallo studente loggato attualmente nella macchina relativo all'attuale prova d'esame.

Premendo tale pulsante lo studente verrà reindirizzato nella pagina di realizzazione che sarà caricata con il codice HTML, CSS e Javascript da lui redatto fino ad un'istante prima del malfunzionamento. Il sistema però alla prima modifica che l'alunno tenterà di apportare al codice gli farà notare tramite un alert esplicativo, simile a quello in Figura 3.14, che questa sessione di realizzazione non è quella originale poiché la schermata dove inizialmente il ragazzo ha iniziato a redigere il proprio esame è stata persa causa malfunzionamenti o errato utilizzo dell'applicativo.

Quindi verrà data la possibilità allo studente di creare una sua nuova copia dell'esame premendo nell'apposito link presente nell'alert. In questa maniera

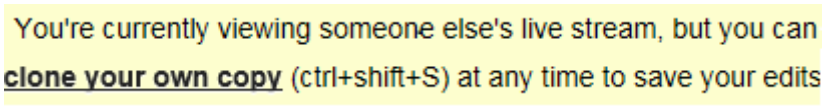

Figura 3.14: Alert creazione nuova copia

viene consentito all'alunno di ritornare correttamente a svolgere il proprio esame. Se lo studente non preme nel link, le modifiche da lui effettuate nella schermata di realizzazione non verranno salvate nel sistema.

La realizzazione di questo meccanismo si base in gran parte sulle funzionalità già messe a disposizione da JSBin.

Questi sono i casi d'uso per cui si è pensato di inserire tali procedure accessorie che permettono agli utenti di cautelarsi da una serie di piccoli malfunzionamenti, di cui ad oggi non se ne conosce particolarmente la natura vista la scarsa dose di test effettuati. Quindi si presume che il numero di questo tipo di funzionalità crescerà con il tempo mano a mano che verranno eseguiti test sempre più articolati e complessi sull'applicativo stesso per capire effettivamente le reali mancanze.

Capitolo 4

ExaM Bin: dettagli implementativi

Dopo aver presentato ad alto livello le funzionalità dell'applicativo, ci si addentra ora nei dettagli implementativi. Verranno mostrati e commentati i frammenti di codice più significativi, elencando inoltre i problemi riscontrati in fase di sviluppo. Sarà data particolare importanza ai moduli software che sono stati creati ad hoc facendo interoperare tra loro diverse tecnologie, presentando i punti di contatto e differenza con altri applicativi che presentano funzionalità simili a quelle offerte da ExaM Bin.

Gli argomenti trattati in questo capitolo prevedono la conoscenza delle seguenti tecnologie di sviluppo:

- Linguaggio di scripting client side Javascript
- Linguaggio di markup HTML
- Linguaggio CSS
- Piattaforma Node.js
- Motore di template Handlebar

4.1 Primi passi: da JSBin a ExaM Bin

Per lo sviluppo dell'applicazione si è deciso innanzitutto di partire prendendo come base un tool open source di web playground, dato che l'idea alla base era quella di utilizzare le risorse più recenti nel campo delle tecnologie web e la prima necessita era quella di fornire un ambiente di lavoro il più confortevole possibile agli studenti per la realizzazione del loro esame.

Lo scopo quindi è stato quello di adattare una piattaforma di web playground alle esigenze di sicurezza e verificabilità tipiche di una prova d'esame.

I primi studi sono stati fatti a livello degli strumenti che attualmente in commercio consentono di avere un ambiente di sviluppo integrato per le tecnologie web, i cosiddetti "web playgorund" che permettono di sviluppare e testare codice senza il bisogno di altre installazioni da parte dell'utente nella macchina, a parte un qualsiasi browser.

Come descritto nel capitolo 2 nella sezione dedicata a questo tipo di piattaforme, si sono volute analizzare le caratteristiche e funzionalità di quelle in commercio, per trovare quella più adatta per lo sviluppo del progetto.

Si è deciso quindi di partire dalla piattaforma open source di debugging collaborativo online denominata JSBin, soprattutto poiché oltre alle indiscusse qualità tecniche era l'unica che permetteva un'installazione locale ed il suo codice era presente online. Ovviamente la schermata principale di JSBin, che rappresenta un online javascript IDE sarebbe poi diventata la schermata di realizzazione compito di ExaM Bin.

Il primo passo a livello realizzativo non è stato implementativo ma bensì conoscitivo, poiché la necessità era quella di capire come il progetto JSBin era stato implementato al suo interno e che tipo di tecnologie utilizzasse nello sviluppo. Cosi ho dovuto studiare la piattaforma Node.js[1], su cui è attualmente progettato JSBin, che consente di costruire facilmente applicazioni veloci e scalabili a livello di rete. Oltre alla piattaforma in sé, ho dovuto ampliare la

[1] http://nodejs.org/

mia conoscenza riguardo i moduli software utilizzati internamente da JSBin per avere una miglior visione d'insieme dell'intero strumento.

Dopo aver compreso i meccanismi di base come prima operazione a livello implementativo ho effettuato un "fork" della directory github contenente il codice sorgente di JSBin[2], questa operazione mi ha consentito di creare una mia copia personale del progetto in questione[3] da poter modificare a piacimento per poter sviluppare ExaM Bin.

Insomma la base del mio lavoro non è stata nulla, il mio progetto non partiva da zero, ma da JSBin.

Terminata la prima fase di studio e la creazione della directory dove avrei sviluppato e, dove tutt'ora sviluppo il mio applicativo, ho iniziato ad effettuare dei test pratici per trovare il modo migliore di inserire sia delle modifiche ai moduli originari di JSBin, estendendoli sia dei veri e propri nuovi moduli software creati ad hoc per il mio progetto.

Le modifiche principali da inserire nel progetto originale erano relative a:

- creazione di un pannello di controllo e correzione per il docente;
- creazione di una interfaccia per l'autenticazione degli studenti (login);
- creazione di un servizio di Clock d'aula;
- creazione di un servizio di valutazione automatica;
- modifica dei meccanismi di salvataggio dei sorgenti;
- modifica dei meccanismi di sicurezza.

Tutte le librerie e file creati personalmente per lo sviluppo del progetto sono contenuti nelle directory denominate "myjsbin" che si trovano in diversi livelli di profondità nella gerarchia dello scheletro del progetto JSBin in base al tipo di funzionalità implementata. Se parliamo di un modulo software questo si troverà nella directory "myjsbin" presente nella cartella "lib", se

[2]https://github.com/jsbin/jsbin
[3]https://github.com/mborghi/jsbin

invece ci riferiamo ad un file css o javascript creato per essere associato ad una pagina web lo troveremo nella directory "myjsbin" all'interno della sottocartella relativa al linguaggio di programmazione (css o javascript) presente nella directory public del progetto.

Gli indirizzi dei servizi offerti dall'architettura RESTful di JSBin sono presenti nel file routes.js ed è qui che ho inserito anche i percorsi relativi ad ogni servizio che ho realizzato, delegando nella maggior parte dei casi la gestione della risposta ad un apposito modulo.

Dato che la piattaforma di partenza utilizza il motore di template Handlebar per il rendering delle pagine web fornite in risposta dai suoi servizi, ho dovuto creare le interfacce, di cui si è ampiamente discusso nel capitolo precedente, utilizzando le linee guida a livello di codifica dettate da tale libreria. Questi oggetti sono stati creati con l'ausilio del framework Bootstrap e della libreria jQuery.

Andiamo ora ad analizzare nel dettaglio gli step implementativi più interessanti che hanno caratterizzato lo sviluppo di ExaM Bin.

4.2 Setup file compito

Il primo scoglio a livello di realizzazione del progetto è stato relativo al setup dei file del compito che il professore voleva far svolgere ai propri alunni; il problema era capire come i file(anche vuoti), uplodati dal professore come base del lavoro degli studenti dal suo pannello di controllo, potessero essere visualizzati nella schermata di realizzazione del compito degli studenti.

Dato che JSBin nella propria schermata principale, carica di default dei frammenti di codice, ho pensato di analizzare a fondo il meccanismo alla base di questa funzionalità ed ho notato che questo codice veniva preso per ogni sezione di programmazione da un file denominato default presente nella directory views del progetto ed aveva estensione diversa in base al linguaggio di riferimento.

Nello specifico JSBin mette a disposizione per la sua schermata iniziale:

- **dafault.html** : contenente il codice HTML da inserire nella sezione dedicata allo sviluppo di codice HTML;
- **dafault.css** : contenente il codice CSS da inserire nella sezione dedicata allo sviluppo di codice CSS;
- **dafault.js** : contenente il codice Javascript da inserire nella sezione dedicata allo sviluppo di codice Javascript.

Quindi ho pensato di prendere i tre file uplodati dal professore e sovrascriverne il contenuto nei tre di default. Ho effettuato tale operazione servendomi del modulo node "fs" che permette di gestire in maniera semplice e puntuale l'accesso al file system di una macchina e definito il servizio all'indirizzo "/uploadFileCustom".

Con l'utilizzo di questo meccanismo la pagina di realizzazione dell'esame è fornita agli studenti all'indirizzo "/" del server su cui è presente l'applicativo, viene quindi restituita dal servizio radice dell'applicativo.

Ho avuto dei problemi nella realizzazione di questo meccanismo poiché JSBin lato server non accettava i dati dei form con enctype pari a "multipart / form-data" e inoltre non consentiva di effettuare richieste di tipo POST diverse da quelle configurate al suo interno. Così ho dovuto inserire l'indirizzo della funzionalità nell'array "ignore", contenente i servizi da ignorare per il controllo di questo tipo di richieste, passato come parametro alla libreria csfr del modulo middleware nel file app.js. Questo poiché le mie conoscenze in fatto di sistemi middleware sono pressoché nulle e quindi non avrei saputo dove mettere le mani per gestire al meglio questo servizio.

Inoltre ho aggiunto un nuovo modulo, preso tra quelli già esistenti in commercio, parliamo in questo caso della libreria denominata multipart del pacchetto express, oggetto alla base della creazione di un applicativo web node, che mi ha permesso di far elaborare al sistema i dati di un form con quella particolare codifica.

4.3 Temporizzazione

La temporizzazione della prova d'esame è stata gestita creando un modulo apposito all'interno del progetto, denominato timer. Questo permette di gestire il clock d'aula, un servizio web che cicla attraverso sette stati, con alcuni passaggi manuali e alcuni automatici e, che a interrogazione risponde con informazioni rilevanti per l'andamento della prova d'esame. Ogni computer d'aula lo interroga con frequenza stabilita da un intervallo del tipo 3/5 secondi, che va interpretato come segue: 3000 + Math.random(2000) millisecondi, ovvero un numero random di millisecondi tra 3000 e 5000, per evitare che un'iniziale sovrapposizione di richieste provochi successive e regolari sovrapposizioni di tempi.

Gli stati in cui si articola la temporizzazione sono i seguenti:

1. **NoTest**: non c'è nessun test in corso. Il servizio web restituisce un JSON semplice del tipo: { status: "notest" }

2. **Setup**: il test sta per iniziare ed è possibile da parte degli studenti registrarsi. Il passaggio dallo stato di NoTest allo stato di Setup è ottenuto manualmente, e si verifica quando il docente nella schermata appunto di Setup del suao pannello di controllo clicca sul pulsante "Consenti agli studenti di fare login presente in fondo alla pagina. In questa fase è consentito agli studenti di identificarsi al sistema, recandosi nella pagina web dove è disponibile tale servizio, che come abbiamo visto in precedenza verrà indicata dal docente. I client monitorano il servizio di clock con frequenza 3/5 secondi per aspettare l'inizio del test. Ad ogni richiesta il server risponde con un JSON : { status: "setup" }.

3. **Ready**: quando tutti gli utenti sono regolarmente loggati, il docente passa allo stadio successivo, che ha il solo scopo di rendere più frequente il check di inizio, da parte della pagina di Setup in cui sono stati reindirizzati gli alunni dopo il loro login, in modo tale che tutti inizino

a meno di un secondo gli uni dagli altri. In questa fase viene anche caricato l'indirizzo della pagina d'esame. Ricevendo questo messaggio i client iniziano a controllare lo stato del clock interno al server ogni 0.5/1 secondo.

Ad ogni richiesta il server risponde con un JSON tipo:

```
{
     status: "ready",
     url: "http://abigaille.cs.unibo.it:7002/paginaEsame"
}
```

4. **Start**: dopo 5 secondi dall'inizio del ready, il server passa automaticamente in stato di Start. Comunica al client in questo modo che è ora di iniziare, e inizia al suo interno il conto alla rovescia fino alla fine del test. Il client in questa fase interroga il server con una frequenza di 5/10 secondi, ricevendo il numero ufficiale di millisecondi mancanti (che alla prima ricezione verrà impostato come valore di partenza del conto alla rovescia presente nella pagina di realizzazione del compito). Ad ogni richiesta il server risponde con un JSON tipo:

```
{
     status: "start",
     url: "http://abigaille.cs.unibo.it:7002/paginaEsame",
     timeout: "1234567"
}
```

5. **Overtime**: quando mancano 15 secondi alla fine ufficiale del test, il server passa in questa modalità, che ha il solo scopo di rendere più frequente il check di fine compito. Questa fase può durare anche diversi minuti (Overtime), su indicazione del docente. I client controllano ogni 4/5 secondi. Ad ogni richiesta il server risponde con un JSON :
{ status: "overtime" }.

6. **Almost Over**: quando mancano 5 secondi alla fine dell'overtime deciso dal docente (che può essere anche zero), si passa allo stato di almost over, che ha il solo scopo di rendere ancora più frequente il check di fine compito. I client infatti controllano ogni 0.5/1 secondi. Ad ogni richiesta il server risponde con un JSON : { status: "almostover" }.

7. **Over**: il passaggio allo stato di over avviene in automatico al termine del periodo di over time. Se ci sono ancora sessioni di realizzazione attive queste vengono consegnate in automatico al sistema per la successiva correzione. Il compito è terminato. Per ritornare allo stato di "notest" il docente deve premere sull'apposito pulsante nel pannello di Exam Info che consentirà di creare una nuova prova d'esame. Ad ogni richiesta il server risponde con un JSON : { status: "over" }.

Il servizio web di clock d'aula viene richiesto attraverso chiamate AJAX all'indirizzo "/getClockAula" sia dalle pagine dedicate al docente che allo studente per identificare il periodo temporale in cui si trova il sistema in modo tale da poter effettuare le corrette elaborazioni in ogni momento. Il pannello di controllo del professore per modificare i valori del clock di sistema in maniera manuale come capita nelle fasi ad esempio, di setup e start, contatta il servizio all'indirizzo "/setClockAula" che gli consente di impostarne il nuovo valore.

4.4 Storage compito

Passiamo ora alla parte storage relativa al salvataggio del compito redatto da uno studente. Per lo sviluppo di questa sezione ho utilizzato gran parte delle funzionalità già messe a disposizione da JSBin.

JSBin per memorizzare tutti i dati relativi all'applicativo si serve di un database composto da varie tabelle e mette a disposizione una serie di librerie software che ne permettono la gestione attraverso due tra i maggiori DBMS (DataBase Management System) open source presenti online, Sqlite e MySql.

Data la mia maggior esperienza con l'utilizzo di basi di dati gestite con MySql ho deciso di utilizzare questo sistema per la gestione dei dati.

Delle varie tabelle presenti nel file full-db-v3.mysql.sql nella directory "build" di JSBin (utile per creare il proprio database in locale) per lo sviluppo del progetto ne ho utilizzata solamente una, la tabella Sandbox, dove vengono originariamente salvati i lavori redatti online dagli utenti del sistema.
In Figura 4.1 possiamo vedere l'istruzione di create table utile alla generazione della tabella, con sottolineati in colore rosso i campi principali.

```
CREATE TABLE `sandbox` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `javascript` mediumtext COLLATE utf8mb4_unicode_ci,
  `html` mediumtext COLLATE utf8mb4_unicode_ci,
  `created` datetime DEFAULT NULL,
  `last_viewed` datetime DEFAULT NULL,
  `url` char(255) COLLATE utf8mb4_unicode_ci DEFAULT NULL,
  `active` char(1) COLLATE utf8mb4_unicode_ci NOT NULL DEFAULT 'y',
  `reported` datetime DEFAULT NULL,
  `streaming` char(1) COLLATE utf8mb4_unicode_ci DEFAULT 'n',
  `streaming_key` char(32) COLLATE utf8mb4_unicode_ci NOT NULL,
  `streaming_read_key` char(32) COLLATE utf8mb4_unicode_ci NOT NULL,
  `active_tab` varchar(10) COLLATE utf8mb4_unicode_ci NOT NULL,
  `active_cursor` int(11) NOT NULL,
  `revision` int(11) DEFAULT '1',
  `css` mediumtext COLLATE utf8mb4_unicode_ci,
  `settings` mediumtext COLLATE utf8mb4_unicode_ci,
  PRIMARY KEY (`id`),
  KEY `viewed` (`last_viewed`),
  KEY `url` (`url`(191)),
  KEY `streaming_key` (`streaming_key`),
  KEY `spam` (`created`,`last_viewed`),
  KEY `revision` (`url`(191),`revision`)
) ENGINE=InnoDB CHARSET=utf8mb4 COLLATE=utf8mb4_unicode_ci;
```

Figura 4.1: Campi tabella SandBox

Nella tabella in questione ExaM Bin salva tutti i compiti degli utenti in maniera personalizzata, memorizzando nello specifico il codice redatto in

ogni area nella schermata di realizzazione in maniera separata, servendosi del campi “html”, “css” e “javascript”.

Quando lo studente effettua la prima modifica alla schermata del suo compito viene creato un nuovo record nella tabella sopra citata attraverso la chiamata ad un servizio già presente in JSBin e, ad ogni altra modifica dell’elaborato, sempre sfruttando un meccanismo interno già implementato (sia lato client che server) il record precedentemente creato viene aggiornato con le ultime modifiche apportate dall’utente.

Per fare in modo che il record generato facesse riferimento in maniera univoca ad un certo studente ho personalizzato la creazione del campo “url” della tabella Sandbox. Questa operazione l’ho effettuata modificando il metodo shortCode del file utils.js nella directory “lib”, funzione con cui JSBin genera il campo in questione.

Qui servendomi delle informazioni con cui lo studente si autentica al sistema: nome, cognome e matricola e la data odierna in formato americano ottenuta attraverso l’uso del modulo node “moment”, ho creato il campo url nel modo che si può osservare nel Listato 4.1:

```
1 var url = moment().format('YYYY-MM-DD') + "_" + studentData.
    name + "_"+ studentData.surname + "_" + studentData.
    matricola + "_" + index_autoincrement;
```

Listato 4.1: Creazione campo ’url’ tabella Sandbox

Ho utilizzato anche un id autoincrementante per non avere problemi con la funzionalità accessoria relativa al recupero del compito di un certo studente, poiché come discusso precedentemente, quando lo studente perde per qualsiasi motivo la sua attuale sessione di lavoro deve chiedere al sistema di crearne una nuova ripartendo da dove era arrivato. Questo tipo di comportamento richiede al sistema di creare un nuovo record nella tabella Sandbox, che nel caso in cui non ci fosse l’id autoincrementante nella creazione dell’url, avrebbe un campo “url” identico a quello del record creato per la sessione appena persa, poiché le informazioni relative allo studente e alla data odierna

sono le stesse. Questo fatto porterebbe alla violazione del vincolo di unicità a cui il campo in questione è sottoposto.

Infatti il campo "url" è una chiave primaria, non dichiarata esplicitamente nel database, ma ottenuta implicitamente nello sviluppo del sistema, poiché per accedere ad un qualsiasi lavoro svolto sulla piattaforma JSBin, l'indirizzo da digitare è del tipo :

http://jsbin.com/campoUrlTabellaSandbox/campoVersionTabellaSandbox .

Il campo "version" si riferisce alla versione del software redatto, poiché JSBin fornisce la funzionalità di creare milestone del lavoro che si sta portando avanti, modificando solamente il campo versione nella creazione di un nuovo record e mantenendo inalterato il campo "url".

Ad ogni studente sfruttando questo meccanismo viene messo a disposizione un indirizzo dedicato per lo svolgimento del compito, dove inoltre, come vedremo nella sezione relativa alla sicurezza, solo a lui è permesso l'accesso.

Ovviamente è stato inserito un separatore tra i campi in fase di creazione della variabile url, definito dal carattere '_'(underscore), per permettere di effettuare il parsing di suddetto valore una volta recuperati i dati relativi allo studente e alla prova d'esame di un certo record dalla tabella Sandbox. Inoltre vista la conformazione dell'indirizzo dedicato alla stesura dell'elaborato di ogni singolo alunno, è stato necessario effettuare l'escape degli oggetti stringa contenenti i valori riferiti a nome e cognome per fare in modo che tali oggetti potessero essere contenuti senza problemi all'interno della definizione di un indirizzo web.

E' stata inserita anche la data odierna per identificare il fatto che lo studente ha partecipato alla prova d'esame effettuata in quello stesso giorno. Infatti una prova d'esame è identificata per ExaM Bin come l'insieme dei compiti redatti in una certa data. Per ora il sistema non gestisce la possibilità di identificare in maniera puntuale due sessioni d'esame eseguite lo stesso giorno, ma per uno sviluppo futuro è pensabile riorganizzare al meglio la ge-

stione del database, creando anche tabelle proprie per identificare in maniera particolare ogni singola prova d'esame e ogni studente che accede al sistema. In modo tale da permettere di eseguire anche più di una prova d'esame al giorno, ed eliminando la complessità nella creazione del campo "url" che potrebbe essere creato in maniera randomica (come fa attualmente JSBin), poiché si assocerebbero ad ogni record della tabella Sandbox dei riferimenti, come chiavi esterne, a campi delle tabelle esami ed utenti per identificare in maniera univoca il lavoro di uno studente in una certa prova d'esame.

Passiamo ora alla consegna del compito. Per identificare tale operazione il il sistema aggiunge la stringa "_finish" al termine del campo url relativo al record in cui è presente l'esame. Infatti quando uno studente dichiara di voler consegnare il proprio elaborato viene effettuata una chiamata AJAX al servizio presente all'indirizzo "/deliveryExam dedicato a gestire tale operazione, che dopo aver recuperato il campo url della tabella Sandbox presente nell'url del chiamante esegue l'upload nel database del record a cui tale valore si riferisce inserendovi la stringa "_finish" in coda. Gli elementi della collezione di dati che contengono nella colonna "url", dell'unica tabella presente, una determinata data all'inizio e la parola "_finish" in coda, rappresentano per la piattaforma la totalità dei compiti consegnati in una data prova d'esame.

Infine trattiamo ora brevemente il meccanismo per la creazione delle query necessarie all'interrogazione del database. Per eseguire tali operazioni ci si è serviti del procedimento utilizzato da JSBin. Quindi si è aggiunto il template dell'interrogazione, fornendogli un nome identificativo, al file "sql_templates.json" presente nella sottocartella "db" della directory "lib" e si è generato il metodo che lo eseguisse nel modulo "mysql.js" dato che si è deciso di utilizzare MySql come DBMS di riferimento. Inoltre, visto che JSBin utilizza un oggetto di tipo Store (implementato nel file "store.js") per contenere le funzionalità che permettono di interfacciarsi alla metodologia di salvataggio, si è inserito il nome della query nell'array methods di tale elemento in modo tale da mettere a disposizione del sistema l'interrogazione precedentemente definita.

4.5 Valutazione automatica

Affrontiamo ora la parte implementativa relativa alla validazione dei compiti realizzati dagli studenti, da me interamente sviluppata.

L'obiettivo era quello di realizzare un servizio server-side che, a compito finito e con la rete riattivata, eseguisse una serie di test sui compiti di una certa prova d'esame e generasse un documento riepilogativo con valutazioni e voti. Il servizio è stato posto interamente all'indirizzo "/correction/validateExams" del server in cui è installato ExaM Bin.
Si è pensato in fase di sviluppo, di suddividere la validazione in due macro aree:

- Statica: dove attraverso l'uso di regole di validazione assolute, senza l'utilizzo di alcun parametro si andava a testare la correttezza sintattica del codice senza procedere all'effettiva esecuzione del programma.

- Dinamica: dove con l'utilizzo di testing unit realizzate dal docente viene permesso di verificare ogni tipo di regola, sia a livello stilistico che procedurale, eseguendo effettivamente il codice redatto dallo studente in maniera tale da poter controllare qualsiasi caso d'uso.

Si è voluto inserire anche una validazione di tipo dinamico per dare al professore maggior libertà nella scelta delle regole a cui la macchina dovesse sottoporre ogni compito. Questo tipo di validazione è considerato anche "custom" (termine a cui si fa riferimento più volte in sezioni precedenti), poiché a differenza della validazione di tipo statico, in questo ambito è il docente che decide effettivamente come e cosa andare a verificare.

Ora analizzeremo nel dettaglio come è stato possibile implementare questo tipo di funzionalità in maniera distinta.

4.5.1 Validazione statica

Parlando di validazione statica, ci riferiamo alla validazione effettuata attraverso l'uso di regole assolute e standard, senza l'utilizzo di alcun pa-

rametro. L'obiettivo è quello di testare la correttezza sintattica del codice, individuando possibili sezioni insicure, senza mai procedere all'effettiva esecuzione del programma.

Per l'effettiva realizzazione del servizio, sono stati utilizzati tre validatori assoluti:

- W3C Markup Validation Service[4]: per testare il codice HTML;
- W3C CSS Validation Service[5]: per testare il codice CSS;
- modulo node Esprima[6]: per testare il codice Javascript;

Per le chiamate ai primi due validatori, si è utilizzato il modulo node w3c-validate[7] e in particolare il suo metodo validate. Questa funzione permette di richiedere la validazione su di un frammento di codice, HTML o CSS, attraverso i due servizi sopra citati e inoltre realizza un primo parsing dei risultati, riconsegnadoli in formato JSON.

Per la validazione del codice Javascript invece, non si è fatto riferimento ad un servizio esterno, ma bensì ad un modulo node appositamente progettato per tale scopo. Il modulo in questione è denominato Esprima, parser conforme agli standard ECMAScript ad alte prestazioni. Ho scelto questo tipo di servizio, poiché a differenza dei normali validatori di sintassi non si preoccupa molto di stili di codifica e formattazione, ma svolge un'analisi che potremmo definire anche semantica sotto un certo punto di vista, che gli permette di identificare un maggior numero di potenziali situazioni d'errore.

Infatti se ad esempio nel codice è presente un'istruzione di "return" non interna ad una funzione questa viene identificata come un errore dallo strumento in questione e di norma ignorata, come ho potuto constatare, da gran parte di altri validatori.

I risultati ottenuti dalla validazione statica ci permettono solamente di capire se il codice di uno studente è scritto in maniera corretta o meno,

[4]http://validator.w3.org/
[5]http://jigsaw.w3.org/css-validator/
[6]https://www.npmjs.org/package/esprima
[7]https://www.npmjs.org/package/w3c-validate

segnalandoci in che sezioni sono stati riscontrati i problemi. Il docente con questo tipo di analisi non potrà mai capire se ciò che un alunno ha scritto è conforme al testo dell'esame sia a livello stilistico, parlando di stile degli elementi HTML, sia a livello comportamentale, valutando le singole funzioni Javascript.

Per questo si è deciso di implementare una validazione di tipo dinamico, che desse effettivamente la possibilità di eseguire il codice redatto da uno studente.

4.5.2 Validazione dinamica

Per la realizzazione della validazione dinamica, o "custom", come a dir si voglia, si è dovuto far interoperare tra loro diversi moduli node.

L'idea era quella di poter far definire alcune regole al professore in un file, testandole per ciascun compito, ma a differenza di molti strumenti ad oggi presenti online, si voleva far in modo che il docente potesse testare al di fuori del sistema il file che aveva redatto per questo tipo di validazione. Tutto ciò per fare in modo che l'insegnante possa scrivere la sua soluzione al compito e aggiungerle il file di testing che ha intenzione di inserire nel sistema per la validazione "custom", in modo tale da poter valutare direttamente dal proprio web browser se effettivamente il file in questione verifica in maniera corretta i requisiti specificati nel compito.

Per permettere questo tipo di operazione si è pensato di ricreare lato server il DOM (Document Object Model) del compito, per poi testarlo con strumenti con cui normalmente si verificano le pagine web lato client. Quindi si è deciso di utilizzare framework di unit test già presenti sul mercato che potessero essere usati sia lato client che server, e non di creare un servizio ad hoc da zero per la validazione.

I moduli node utilizzati in questa fase sono stati i seguenti:

- Cheerio: per unificare le sezioni di codice;
- Jsdom: per generare il DOM server side;

- Mocha: per eseguire una suite di test;
- Chai: per scrivere asserzioni complesse.

Prima di tutto è stato necessario unificare il contenuto dei tre script di codice da cui il compito era formato, mettendo insieme quindi HTML, CSS e Javascript in un unico oggetto. Questo è stato possibile utilizzando il modulo node Cheerio, che fornisce lato server le funzionalità principali del framework Javascript jQuery[8] per poter elaborare in maniera semplice e flessibile il DOM.

Come possiamo notare nel frammento di codice sottostante (Listato 4.3), per prima cosa in una variabile denominata "$" viene caricato il codice HTML presente nel record relativo al compito attraverso l'utilizzo della funzione load di cheerio che restituisce un oggetto arricchito contenente il DOM. Successivamente viene aggiunto al tag head, in un blocco style, il codice relativo alla parte CSS e in un blocco script, prima della chiusura del tag html, viene inserito il codice Javascript del compito. Infine, come possiamo alla riga 4, viene inserito un riferimento alla libreria JQuery, per permettere al professore di utilizzare questo Framework in fase di test.

Per ottenere la stringa contenente l'intero codice del compito, basta richiamare il metodo html privo di parametri fornito da cheerio nell'oggetto arricchito "$".

```
1 var $ = cheerio.load(record.html);
2 $("head").append("\n<style>\n" + record.css + "\n</style>");
3 $("html").append("\n<script>\n"+ record.javascript +"\n</
    script>");
4 $("html").append("<script src='https://ajax.googleapis.com/
    ajax/libs/jquery/1.11.0/jquery.min.js'></script>");
5 var html = $.html(); //questo è un commento
```

Listato 4.2: Unificazione codice compito con modulo Cheerio

L'utilizzo del modulo cheerio permette di ricreare il DOM e manipolarlo server side, come abbiamo visto, ma non permette di lavorare in un ambiente

[8] http://jquery.com/

browser-like sul server per effettuare in maniera efficace test funzionali sul DOM ricreato, funzionalità di cui il sistema aveva necessariamente bisogno per effettuare la validazione dinamica.

Per questo si è deciso di utilizzare il modulo node JSDOM che permette di ovviare al problema sopra citato ricreando lato server un ambiente browser-like perfetto a livello di creazione del DOM.

La domanda sorge spontanea: perché non utilizzare fin da subito JSDOM? La risposta è semplice, per lavorare in maniera corretta questa libreria ha bisogno che gli venga dato in input l'intero codice della pagina web. Ecco perché si è resa necessaria una prima iterazione con l'utilizzo di cheerio che ha fatto in modo di inglobare in una stringa tutto il risultato ottenuto dall'unione di codice HTML, CSS e Javascript.

Come si puo notare nel Listato 4.3 fornendo in input la variabile html, ottenuta precedentemente grazie all'utilizzo di cheerio, al metodo jsdom dell'oggetto JSDOM (contenente l'implementazione della libreria stessa), seguita dai parametri presenti nell'oggetto features, verrà fornito come risultato una variabile contenente nel suo attributo parentWindow il DOM dell'intera pagina web redatta da un singolo studente.

```
1  var doc = jsdom.jsdom(html, null, {
2        features: {
3            FetchExternalResources   : ['script'],
4            ProcessExternalResources : ['script'],
5            MutationEvents           : '2.0',
6            QuerySelector            : false
7        }
8    });
9  var window = doc.parentWindow;
10 window.addEventListener('load',  function(){...}
```

Listato 4.3: Creazione DOM con modulo JSDOM

Si aggiunge un listener al DOM appena creato sull'evento load, per attendere il caricamento di tutti gli script presenti nella pagina web.

Dopo aver ricreato il DOM lato server e avendolo posto in una variabile denominata window, siamo pronti ad effettuare la vera validazione custom utilizzando i moduli node mocha e chai a cui è dedicata un'intera sezione, dove verrà inoltre spiegato il modo di redigere un file di validazione dinamica.

4.5.2.1 Mocha e Chai

Per la realizzazione della funzionalità di validazione dinamica, come precedentemente accennato, non si è creato un meccanismo ad hoc per testare le pagine web redatte dagli studenti, ma si è voluto utilizzare framework di unit test che potessero essere utilizzati sia lato client che server. Questa scelta per dare la facoltà al professore di provare il file di test da lui generato anche client side provandone l'effettiva correttezza utilizzando le stesse tecnologie utilizzate lato server per eseguire il processo di validazione custom.

Gli strumenti utilizzati per andare a sviluppare questo tipo di servizio, sono stati il framework di unit test Mocha.js[9] e la libreria di asserzioni Chai.js[10]. La scelta è ricaduta su questi due pacchetti, poiché presentano una perfetta integrazione sia con il browser (client side) sia con gli applicativi creati con l'utilizzo di node.js (nel nostro caso la parte server side).

Mocha è un framework di unit test Javascript ricco di funzionalità a livello di esecuzione, che rendono semplici e flessibili i test asincroni su pagine web. Test mocha eseguiti in serie, consentono di ottenere un report preciso e puntuale. Il suo codice sorgente è presente in Github[11].

Chai è una libreria di asserzioni creata per ambienti node e browser che può essere deliziosamente accoppiata con qualsiasi framework di test Javascript. Questo strumento consente di creare affermazioni complesse con diversi stili, dando la possibilità all'utente di testare in maniera efficace e puntuale qualsiasi tipo di comportamento sia a livello sintattico che semantico.

[9] http://mochajs.org/
[10] http://chaijs.com/
[11] https://github.com/mochajs/mocha

Vediamo ora come possiamo utilizzare questi due strumenti per creare test case che utilizzano asserzioni complesse (Listato 4.4).

```
1 describe("Nome raggruppamento", function() {
2   it("Titolo test", function() {
3     var i = 10;
4     //mi aspetto di trovare il valore 10 nella variabile i
5     chai.expect(i).to.equal(10); //asserzione
6   });
```

Listato 4.4: Esempio creazione test case con Mocha.js e Chai.js

Con il metodo "describe" di mocha viene data la possibilità all'utente di definire semplicemente un raggruppamento dei vari test case, che può nidificarsi in profondità e in cui il primo parametro fa riferimento al nome del raggruppamento. I singoli test case vengono definiti dal metodo "it", dove il primo parametro fa riferimento al nome del test e il secondo alla funzione dove è definito il corpo del test.

All'interno di ogni test case si possono definire delle asserzioni che nel nostro caso vengono realizzate con l'utilizzo degli strumenti messi a disposizione da chai.js. Ogni asserzione consente di testare un certo oggetto, nel caso in figura ci si aspetta che la variabile denominata i, presenti al suo interno il valore 10. Nello specifico per fare questo tipo di controllo ci si serve della libreria expect del modulo chai a cui viene passato come parametro l'oggetto su cui si vuole fare la verifica su cui verrà invocato il metodo to.equal con il primo parametro contenente il valore numerico 10. Relativamente a tale test case un buon titolo potrebbe essere: "La variabile i dovrebbe avere il valore 10". Ovviamente, come ci si può aspettare, il test andrà a buon fine.

Il servizio di validazione automatica server side è stato creato sfruttando questo tipo di tecnologie in modo tale da poter testare qualsiasi componente e comportamento del compito redatto da ogni singolo alunno.

Come discusso precedentemente per prima cosa viene ricreato il DOM lato server relativo al compito redatto dall'alunno, poi su tale oggetto vengono effettuati tutti i test case definiti dal professore nel file da lui presentato al sistema per la validazione custom.

Per fare in modo che il file contenente i test case definiti dal professore, fosse valido sia client side che server side, si è deciso di adottare un piccolo accorgimento. Infatti per accedere ad una qualsiasi variabile del DOM, si richiede al docente di scrivere: "window.nomeVariabile" anziché solo "nomeVariabile". Questo poiché server side, nell'oggetto window è contenuto il DOM del compito appena creato. Tale meccanismo non danneggia l'esecuzione del test client side da parte dell'insegnante, poiché anche nell'oggetto window del browser è contenuto l'intero DOM della pagina, ma essendo questo precaricato come riferimento iniziale, non si ha necessità di scrivere "window.nomeVariabile" per accedere al contenuto di una variabile, basta semplicemente il suo nome.

```
1 describe("Nome Test Suite", function() {
2   it("Valore variabile i pari a 10", function() {
3     chai.expect(window.i).to.equal(10);
4   });
5   it("Nessun elemento html con attributo class", function() {
6       chai.expect(window.$("[class]").length).to.equal(0);
7   });
8 });
```

Listato 4.5: Test case con Mocha.js e Chai.js valido per ExaM Bin

Nel frammento di codice precedente è possibile osservare una test suite valida per la correzione di un esame sulla piattaforma ExaM Bin.

Con la prima clausola it si va a testare il contenuto della variabile 'i' del DOM accedendovi attraverso il campo 'i' della variabile window che, come illustrato in precedenza, contiene il DOM generato server side del compito. La seconda clausola it è più interessante, poiché mostra com'è possibile verificare regole a livello stilistico. Nello specifico si vuole controllare se non esiste nel codice nessun elemento html che contenga l'attributo class. Per svolgere questo controllo il docente può utilizzare l'oggetto $ del framework jQuery che viene messo a disposizione dal sistema per la scrittura dei test attraverso il campo $ della variabile window. Con il selettore "[class]" applicato a tale oggetto verranno restituiti tutti gli elementi html che contengono l'attributo

class in un array. Per fare in modo che il test vada a buon fine, come si può notare alla riga 6, quest'ultimo oggetto non deve contenere elementi, quindi avere una lunghezza pari a zero.

Dopo aver spiegato come redigere un file per la validazione custom, ora ci addentriamo nel dettagli che ne hanno reso possibile l'esecuzione da parte della piattaforma.

All'interno del server per poter eseguire in maniera corretta e nel giusto contesto le regole definite dal docente nel file di validazione personalizzata, si è resa necessaria la creazione di un modulo per la configurazione del test e si sono dovute aggiungere una serie di istruzioni all'inizio di tale file.

```
1 var exam = null;
2 function getExam(){ return exam; }
3 function setExam(examToTest){ exam = examToTest; }
4 var testConfig = { getExam: getExam,setExam: setExam };
5 module.exports = testConfig;
```

Listato 4.6: Modulo TestConfig.js

Per prima cosa si è dovuto creare un'apposita libreria per la configurazione del test personalizzato, il cui codice è presente nel Listato 4.6. Questo modulo presenta al suo interno la sola variabile exam, che è predisposta per contenere l'oggetto al cui interno è presente il DOM del compito ottenuto precedentemente grazie all'utilizzo del modulo jsdom. A livello di metodi troviamo solamente la funzioni getExam, che restituisce il valore della variabile exam, e setExam che invece consente di cambiarne il valore.

Oltre alla creazione del modulo per la configurazione del test si è dovuto inserire all'inizio del file di test redatto dal professore una serie di istruzioni per permettere al framework mocha di eseguire il test utilizzando i giusti parametri.

```
1 var chai = require('chai');
2 var config = require('./testConfig');
3 var window = config.getExam();
```

Listato 4.7: Istruzioni configurazione file test mocha

Queste istruzioni vengono inserire automaticamente dal sistema, dopo che il professore ha fatto l'upload del file di validazione personalizzata e fanno riferimento al Listato 4.7. Come si può notare, le prime due istruzioni permettono di utilizzare nel file di test i moduli chai e testConfig; il primo consente di eseguire in maniera corretta le asserzioni scritte con l'utilizzo di tale libreria invece il secondo consente, attraverso la chiamata del metodo getExam, di acquisire il DOM del compito che si deve testare. Come si può notare alla riga 3, questo viene inserito all'interno della variabile window, per fare in modo che il meccanismo precedentemente citato per l'accesso alle variabili (window.nomeVariabile) vada a buon fine sia client che server side.

Quindi, dopo aver aggiunto queste istruzioni, abbiamo finalmente creato il file su cui il sistema è pronto ad eseguire la suite di test relativi alla validazione dinamica. Vediamo ora come la piattaforma esegue effettivamente questo tipo di test sul compito di un singolo alunno, andando a spiegare il codice presente nel Listato 4.8.

```
1 testConfig.setExam(window);
2 var mocha = new Mocha();
3 mocha.addFile("test/myjsbin/testExamFile");
4 var passed = [], failed = [];
5 mocha.run(function(){
6   delete require.cache[pathCacheFileTest];
7   }).on('fail', function(test){ failed.push(test.title);
8   }).on('pass', function(test){ passed.push(test.title); });
```

Listato 4.8: Esecuzione test mocha in maniera programmatica

Per prima cosa viene chiamato il metodo setExam del modulo testConfig che permette di impostare come valore dell'oggetto exam il DOM del compito contenuto nella variabile window (ottenuta precedentemente con JSDOM, vedi Listato 4.8). Successivamente viene creato un nuovo oggetto mocha e attraverso il metodo addFile gli viene aggiunto il file "TestExamFile", generato come precedentemente descritto, che contiene la suite di test da far eseguire alla macchina e inizialmente le istruzioni per la sua corretta configurazione.

Alla riga 5 viene fatta eseguire effettivamente al sistema in maniera programmatica la validazione custom attraverso l'utilizzo della funzione run dell'oggetto mocha. Per intercettare i risultati prodotti dal test l'applicativo si serve della funzione on. In questa maniera, in base al risultato di "pass" o "fail", il sistema inserisce il titolo del test all'interno di un apposito array, "passed" (che conterrà il nome di tutti i test andati a buon fine) o "failed"(che conterrà il nome di tutti i test falliti). Questi due array vengono riempiti per poter dare un feedback al professore relativamente ai risultati ottenuti dalla validazione personalizzata di un certo compito.

Infine, alla riga 6 troviamo l'unica istruzione della funzione passata come parametro al metodo run, che viene eseguita una volta terminati tutti i test. Qui si effettua un'operazione di cancellazione della cache di Mocha e in particolare del file inserito precedentemente nell'oggetto mocha attraverso l'utilizzo della funzione addFile. E' stato necessario inserire questo tipo di comportamento, poiché per un presunto bug all'interno del modulo mocha, quest'ultimo senza tale accorgimento non permetteva di eseguire la stessa suite di test in maniera programmatica più di una volta all'interno di un applicativo node.

La realizzazione del servizio di analisi dinamica non è stata affatto semplice poiché come detto si è dovuto affrontare una serie di problematiche dovute all'integrazione di più moduli tra loro e alla difficoltà di utilizzare strumenti di analisi creati per essere eseguiti sia lato client nel browser, sia lato server all'interno di applicativi node.

Moduli che tra l'altro non è stato facile installare all'interno della macchina poiché richiedono singolarmente molte dipendenze che necessitano a loro volta di ulteriori installazioni. Un esempio lampante è quello del modulo JSDOM che, necessitando del compilatore C e di Python, mi ha costretto ad installare, nella mia macchina windows, Visual Studio e Python.

In questo caso non sono riuscito a tenere fede alla prerogativa relativa alla semplicità di installazione dell'applicativo, ma sono stato costretto ad utilizzare tali moduli software poiché sono gli unici che mi consentivano di

realizzare il servizio di validazione custom di cui necessitava ExaM Bin.

4.6 Sicurezza

L'ultimo aspetto che andiamo ad introdurre a livello di sviluppo della piattaforma ExaM Bin è la sicurezza. Ogni servizio per essere implementato ha avuto bisogno di una gestione degli accessi in modo tale da garantire la corretta fruizione delle funzionalità solamente a coloro a cui erano rivolte. Questo tipo di meccanismo è stato creato attraverso l'utilizzo dei cookie di sessione e di alcune variabili di configurazione.

Per identificare la macchina adibita ad uso esclusivo del professore è stato utilizzato un apposito campo del file di configurazione del sistema, dove è stato posto l'indirizzo di rete di tale computer. Quindi si è fatto in modo a livello implementativo che ogni servizio dedicato a questo tipo di utente funzionasse in maniera corretta solamente per richieste provenienti dall'url precedentemente configurata.

Per gli alunni invece si è utilizzato un meccanismo di cookie creato ad hoc. Una volta che lo studente ha fatto login il sistema crea un cookie per lui e ne memorizza al suo interno i dati con cui si è autenticato al sistema: nome, cognome e matricola. Controllando se questo oggetto è presente o meno nelle richieste successive si è in grado di capire che tipo di utente sta tentando di accedere ad un determinato servizio.

Tale meccanismo inoltre consente di gestire gli accessi a livello di ogni singolo elaborato. Dato che per accedere ad un singolo compito l'indirizzo è del tipo "/campoUrl/campoVersion" e come abbiamo visto in precedenza il campo url viene generato con i dati personali di ogni singolo studente, non è difficile verificare se un alunno sta cercando di entrare in una schermata di realizzazione esame non sua. Infatti al sistema basta confrontare i valori nel campo url con i valori presenti nel cookie per capire se lo studente che sta tentando di accedere alla pagina è effettivamente colui che l'ha realizzata. In caso di tentato plagio lo studente verrà reindirizzato in una pagina diversa

rispetto a quella che ha richiesto, dove gli verrà notificato il comportamento illecito.

Ovvimente il professore può accedere a qualsiasi compito degli studenti anche in fase di elaborazione, poiché la sua richiesta a tale servizio viene gestita in maniera particolare e gli fornisce la facoltà di visualizzare l'interfaccia risultante senza alcuna difficoltà.

Il cookie relativo ad un certo alunno svanirà una volta consegnato l'esame e di conseguenza non sarà più possibile accedere ad alcun servizio da parte di tale utente.

Questo tipo di gestione della sicurezza effettuata in maniera personalizzata sicuramente può comportare maggiori rischi di errori e infiltrazioni di codice malevolo rispetto a soluzioni in cui si ricorre all'utilizzo di specifici strumenti per creare un ambiente asettico per ogni richiesta ricevuta dal server; ma viste le mie esigue conoscenze in questo ambito è stata l'unica che potessi implementare con semplicità e in un breve lasso di tempo.

Conclusioni

In questo documento è stato presentato ExaM Bin, una piattaforma open source per la gestione di una prova d'esame di tecnologie web. Con l'analisi di tale progetto si è voluto evidenziare il potenziale dell'utilizzo delle recenti risorse nel campo delle tecnologie web per la creazione di un applicativo atto a gestire una prova d'esame in questa disciplina.

L'idea innovativa deriva dal fatto di voler conciliare in un'unica piattaforma tutti i servizi e funzionalità necessarie per fare esami di questo tipo. ExaM Bin nasce in risposta a questo problema integrando strumenti open source gratuiti presenti attualmente sul mercato, in fatto di web playground, unit test e storage, in un unico progetto.

Questo tipo di tool come si evince nella trattazione offre un importante benefit per gli studenti, liberi di interagire con un ambiente confortevole per la scrittura del codice che gli consente di testarlo in tempo reale potendone quindi valutare visivamente le caratteristiche stilistiche e i comportamenti dinamici, usufruendo di una vera e propria console dei comandi.

I docenti, dal canto loro, relativamente ad una singola prova d'esame, possono configurarne i dati di partenza a livello di codice, impostarne la durata a livello di temporizzazione e infine verificarne l'intero contenuto in maniera automatizzata. Quest'ultima funzionalità permette al professore di lasciare il peso della correzione all'applicativo sfruttando il suo meccanismo di validazione automatica che consente di effettuare un'analisi sia a livello statico, eseguendo un controllo della sintassi, che dinamico, testando il comportamento dell'elaborato in qualsiasi caso d'uso.

Il fatto che la piattaforma unisca in un unico applicativo le funzionalità relative a studenti e docenti è l'elemento qualificante del progetto e tale da renderlo praticamente unico nel suo genere.

Lo sviluppo di tale soluzione ha richiesto una profonda analisi del panorama scientifico per poter capire quali tipi di strumenti fornissero le migliori potenzialità per essere sfruttati nella creazione di tale piattaforma. Il progetto è generato da un adattamento dell'applicativo di web playground JSBin, dato che quest'ultimo è l'unico nel suo genere installabile localmente nella macchina dell'utente e open source.

A questa base sono state aggiunte tutte le funzionalità per gestire nella sua interezza una prova d'esame. In particolare è stato creato un servizio di validazione automatizzata dei compiti che, diversamente dagli strumenti attualmente presenti sul mercato, è stato sviluppato servendosi di framework di unit test utilizzabili sia client, che server side. Questo permette al professore di controllare e verificare con mano, senza interagire con il sistema, le regole a cui sottoporre ogni singolo compito redatto dagli studenti.

Vista la natura open source del progetto e degli strumenti utilizzati nella sua creazione, un ulteriore punto a suo favore è dato dal fatto che il suo codice sorgente è disponibile online per l'intera comunità all'indirizzo `https://github.com/mborghi/jsbin`.

Parlando di sviluppi futuri si ha come primo obiettivo quello di portare ExaM Bin ad un effettivo utilizzo in ambito accademico per consentire ai docenti di effettuare esami di tecnologie web con diversi gradi di difficoltà. La distribuzione dell'applicativo potrebbe avvenire solamente dopo che lo si è testato nella maniera più completa possibile, ora siamo solo alla fase iniziale, in modo tale da scovare tutte le possibili situazioni d'errore non gestite, modificando o creando nuove funzionalità che consentano di superarle in facilità.

A livello di funzionalità si pensa di poter dare la possibilità al professore di definire più esercizi per ogni singola prova d'esame e di eseguirne più di una nella stessa giornata. In ambito architetturale si potrebbe riorganizzare

il database, creando tabelle per identificare in maniera puntuale studenti e prove d'esame, snellendo il meccanismo di salvataggio del compito togliendo parte della complessità nella creazione del campo url della tabella adibita alla memorizzazione dei compiti.

Vista l'attuale organizzazione interna del progetto si vuole fare in modo che possa supportare la fruizione in diverse lingue oltre all'italiano creando le traduzioni delle interfacce grafiche presenti.

In conclusione, con l'utilizzo di ExaM Bin è quindi possibile fare in maniera proficua esami di tecnologie web all'università senza carta e penna.

Bibliografia

[ABT97] Ashworth P., Bannister P., Thorne P., Guilty in whose eyes?University students' perceptions of cheating and plagiarismin academic work and assessment, Studies in HigherEducation, 22, 1997

[ALA05] Ala-Mutka K., A survey of automated assessment approaches for programming assignments. Computer Science Education, 15(2):83102, 2005.

[CEA03] Carter J., English J., Ala-Mutka K., Dick M., Fone W., Fuller U., and Sheard J. ITICSE working group report: How shall we assess this? SIGCSE Bulletin, 35(4):107123, 2003.

[CG02] Califf, M.E. and Goodwin, M. Testing skills and knowledge:Introducing a laboratory exam in CS1. SIGCSEBulletin, 34.1 (2002), 217-221.

[CJ01] Chamillard, A. T. and Joiner, J. K. Using Lab Practica to Evaluate Programming Ability. SIGCSE Bulletin, 33.1 (2001), 159-163.

[CW10] Coffman J. and Weaver A. C. Electronic commerce virtual . In SIGCSE 10: Proceedings of the 41st ACM technical symposium on Computer science education, pages 9296, New York, NY, USA, 2010. ACM.

[DBP04] Daniels M., Berglund A., Pears A., Fincher S., Five myths of assessment, Proceedings of the Sixth Australasian Conference on Computing Education, p.57-61, January 01, 2004, Dunedin, New Zealand

[DLO05] Douce C., Livingstone D., and Orwell J., Automatic test-based assessment of programming: A review. J. Educ. Resour. Comput., 5(3):4, 2005.

[ENG02] English, J. Experience with a computer-assisted formal programming examination. Proc. 7th Annual Conference on Innovation and Technology in Computer Science Education (2002, Aarhus, Denmark) 51-54.

[EP08] Edwards S. H. and P´erez-Quinones M. A. Web-cat: automatically grading programming assignments. In ITiCSE 08: Proceedings of the 13th annual Conf. on Innovation and technology in computer science education, pages 328328, New York, NY, USA, 2008. ACM.

[FM06] Feng, M. Y. and McAllister, A. A Tool for Automated GUI Program Grading, 36th ASEE/IEEE Frontiers in Education Conference, October 28-31, 2006, Session S1F

[FPQ08] Fu X., Peltsverger B., Qian K., Tao L., and Liu J. Apogee:automated project grading and instant feedback system for web based computing. In SIGCSE 08: Proceedings of the 39th SIGCSE technical symposium on Computer science education, pages 7781, New York, NY, USA, 2008. ACM.

[GUT06] Gutierrez F. Stingray: a hands-on approach to learning information security. In SIGITE 06: Proceedings of the 7th Conf. on Information technology education, pages 5358, New York, NY, USA, 2006. ACM.

[HOL60] Hollingsworth J., Automatic graders for programming classes, Communications of the ACM, v.3 n.10, p.528-529, Oct. 1960 [doi 10.1145/367415.367422]

[HS05] Haghighi P. D., Sheard J., Summative Computer Programming Assessment Using Both Paper and Computer, Proceedings of the 2005

conference on Towards Sustainable and Scalable Educational Innovations Informed by the Learning Sciences: Sharing Good Practices of Research, Experimentation and Innovation, p.67-75, May 14, 2005

[IAK10] Ihantola P., Ahoniemi T., Karavirta V., Seppälä O. Review of recent systems for automatic assessment of programming assignments, Proceedings of the 10th Koli Calling International Conference on Computing Education Research, p.86-93, October 28-31, 2010, Koli, Finland [doi¿10.1145/1930464.1930480]

[KI10] Karavirta V. and Ihantola P. Serverless automatic assessment of Javascript exercises. Proceedings of the fifteenth annual conference on Innovation and technology in computer science education Pages 303-303, New York, NY, USA, 2010. ACM.

[REE89] Reek K. A., The TRY system -or- how to avoid testing student programs, ACM SIGCSE Bulletin, v.21 n.1, p.112-116, Feb. 1989 doi:10.1145/65294.71198

[SHA14] Shan P. (2014), JSbin, JSfiddle or Codepen, which one to use and why?, (http://voidcanvas.com/jsbin-jsfiddle-or-codepen-which-one-to-use-and-why/)

[SJ04] Sun, H. and Jones, E.L. Specification-driven automated testing of GUI-based Java programs, Proceedings 42nd ACM Southeastern Regional Conference, Huntsville, Alabama, USA, April 2-3, 2004, pp. 140-145.

[SQF08] Sztipanovits M., Qian K. and Fu X., The automated web application testing (awat) system. In ACM-SE 46: Proceedings of the 46th Annual Southeast Regional Conf., pages 8893, New York, NY, USA, 2008. ACM.

Ringraziamenti

Qui ho la possibilità di ringraziare tutto il mondo o anche solo chi, da quel lontano 20 agosto del '92, fa parte del mio e mi ha consentito di arrivare a scrivere queste pagine.

Ringrazio relatore, co-relatore, docenti, panchina, amici, parenti, nonni, zii, compagni di squadra, compagni di università, compagni di progetto, tecnici di laboratorio, segretarie, assistenti, allenatori, presidenti, guardalinee, massaggiatori, fisioterapisti, conoscenti e ultima, ma non certo per importanza, la mia famiglia.