

ALMA MATER STUDIORUM · UNIVERSITÀ DI BOLOGNA

SCUOLA DI SCIENZE
Corso di Laurea in Informatica

NPA-Exam:
**uno strumento per l'erogazione
di test universitari via browser**

Relatore:
Chiar.mo Prof.
Fabio Vitali

Presentata da:
Francesca Grillo

Sessione II
Anno Accademico 2014/2015

Alla mia famiglia ...

Introduzione

Nella facoltà di informatica gli esami di programmazione si svolgono con carta e penna. Questo problema, che ho personalmente riscontrato nel corso della mia carriera universitaria, non è ancora stato risolto. Lo scopo di questo progetto è quello di dimostrare che è possibile realizzare uno strumento che soddisfi le necessità sia degli studenti, fornendo un ambiente più consono dove poter svolgere l'esame, sia dei docenti, per dar loro la possibilità di configurare e gestire la prova di ogni studente dall'inizio alla fine.

Prendendo in considerazione gli esami di tecnologie web, quello che più mi ha fatto riflettere è che allo studente viene richiesto di creare una pagina web, gestendone la struttura e lo stile tramite l'utilizzo di codice HTML e CSS, oppure di scrivere dei frammenti di codice Javascript, senza la possibilità di testare ciò che si sta scrivendo. Potrebbe non essere un grosso problema se l'esercizio non richiede l'elaborazione di grosse quantità di codice, ma in caso contrario diventa quasi impossibile non commettere errori, anche solo di sintassi, come la chiusura di una parentesi.

Inoltre, molto spesso capita che riguardando il codice e ragionando sul funzionamento lo studente si renda conto di aver dimenticato qualcosa, quindi dal momento che su un foglio di carta non si può far uso di funzioni quali *copia*, *taglia* o *incolla*, inizi a scarabocchiare qua e là per aggiungere ciò che manca o correggere degli errori. Questo non aiuta il docente in fase di correzione, il quale si vede costretto ad interpretare prima di tutto quello che c'è scritto, anche perché non tutti gli studenti hanno una scrittura chiara e spesso non commentano il proprio elaborato, e poi a correggere eventuali errori.

Per evitare questi problemi viene richiesto di ricopiare il compito in bella copia, ma

non sempre c'è il tempo per farlo e generalmente l'obiettivo di ogni studente è quello di cercare di risolvere il maggior numero di esercizi prima che finisca il tempo a disposizione. Ecco perché sarebbe più semplice e vantaggioso far svolgere l'esame al computer piuttosto che su un foglio di carta.

In commercio esistono degli strumenti che permettono di testare in tempo reale l'output prodotto dal codice scritto, direttamente dal browser, e prendono il nome di *Web Playground*. Questo è ciò di cui hanno bisogno gli studenti per esaminare quello che stanno facendo e per rendersi conto degli errori commessi. D'altra parte però i web playground non forniscono la possibilità di implementare un servizio di validazione automatica, che sarebbe utile al docente per la correzione dei compiti.

Esistono, tuttavia, strumenti di questo tipo che permettono di fare la valutazione di una pagina web prendendo in input l'url della stessa, ma non possiedono un ambiente per poterne creare una.

Quindi, quello che serve è uno strumento che unifichi le due cose e per questo nasce l'idea di inventare e realizzare questo progetto, che prende il nome di NPA-Exam. C'è da dire che l'implementazione non parte da zero; infatti, esiste già una versione beta, chiamata ExamBin, che è stata creata per questo scopo. NPA-Exam è un'estensione pensata per risolvere alcune mancanze presenti nel vecchio progetto e per inserire nuove funzionalità allo stesso, al fine di facilitare sempre di più il lavoro degli studenti e del docente.

Il tutto si basa su un sistema di web playground, denominato JSBin, il quale risulta l'unico a poter essere installato localmente sul computer degli utenti e ad avere il codice completamente open source. ExamBin sfrutta le funzionalità offerte da JSBin per implementare la pagina nella quale lo studente dovrà svolgere l'esame. A questo si sommano le funzionalità aggiuntive messe a disposizione dello studente, quali la possibilità di identificarsi al sistema e di consegnare il compito svolto, e quelle messe a disposizione del docente, quali la possibilità di fornire in partenza frammenti di codice HTML, CSS e JavaScript, che gli studenti dovranno poi modificare, di gestire la durata della prova d'esame e di correggerne in un secondo momento il contenuto automaticamente.

NPA-Exam, come già detto, ha come scopo quello di estendere ExamBin inserendo le funzionalità che mancano a quest'ultimo. Tra le più importanti la possibilità di: svolgere più esami durante l'arco della giornata, configurare più domande per la prova d'esame, fornire una copia digitale del testo d'esame, mantenere i dati degli studenti registrati al sistema e degli studenti che hanno consegnato la prova all'interno di tabelle dinamiche.

Nei vari capitoli verranno analizzati meglio tutti i dettagli; in particolare, nel primo capitolo si parlerà del contesto scientifico e di quali sono stati i metodi adottati in passato per far svolgere gli esami al computer. Nel secondo capitolo si parlerà degli strumenti di web playground e nello specifico verrà descritto il funzionamento di ExamBin. Nel terzo capitolo si analizzeranno le funzionalità messe a disposizione da NPA-Exam e le differenze tra quest'ultimo ed ExamBin. Nel quarto capitolo verranno mostrati i dettagli implementativi più importanti, tramite l'uso di frammenti di codice, che caratterizzano NPA-Exam. Infine, nelle Conclusioni verrà fatto un quadro generale della situazione con l'aggiunta di eventuali sviluppi futuri.

Indice

Introduzione	i
1 Esami: programmazione su carta	1
1.1 Soluzioni proposte nel corso degli anni	2
1.1.1 Web Development	4
2 Web Playground	9
2.1 JSBin VS ExamBin	11
2.1.1 ExamBin: Funzionalità Docente	13
2.1.2 ExamBin: Funzionalità Studente	17
2.2 Le limitazioni che presenta ExamBin	18
2.2.1 Una sola domanda da configurare	19
2.2.2 Domande a risposta aperta e multipla non disponibili	20
2.2.3 Testo dell'esame cartaceo	21
2.2.4 Controllo manuale: studenti registrati e prove consegnate	22
2.2.5 Una sola prova d'esame durante la giornata	23
3 Da ExamBin a NPA-Exam	27
3.1 NPA-Exam risolve i problemi di ExamBin	28
3.2 Funzionalità Docente	30
3.2.1 Setup	31
3.2.2 Inizio Prova	34
3.2.3 Fine Prova	35
3.3 Funzionalità Studente	37

3.3.1	Login	38
3.3.1.1	In attesa di cominciare	38
3.3.2	Svolgimento	39
3.3.3	Consegna	40
4	Dettagli implementativi	43
4.1	Struttura file Json	45
4.2	Gestione lista esami	47
4.3	Inizializzazione tabelle dinamiche	48
4.3.1	Tabella degli studenti registrati	49
4.3.2	Tabella delle prove concluse	51
4.4	Temporizzazione	54
4.5	Gestione di più prove durante la giornata	58
4.6	Salvataggio	59
4.6.1	Salvataggio domande a risposta aperta	60
4.6.2	Salvataggio domande a risposta multipla	61
4.6.3	Salvataggio domande con JSBin	61
4.6.4	Le prove non consegnate	63
4.7	Correzione	64
4.7.1	Elaborazione dati JSBin	65
4.7.1.1	Validazione Statica	66
4.7.1.2	Validazione Dinamica	67
4.7.2	Elaborazione dati domande a risposta aperta e multipla	70
	Conclusioni	72
	Bibliografia	73
	Ringraziamenti	75

Elenco delle figure

2.1	Ambiente di sviluppo JSBin	11
2.2	Ambiente di sviluppo ExamBin	14
2.3	Pannello delle informazioni relative alla prova d'esame	15
2.4	Interfaccia pannello di correzione	17
2.5	Interfaccia ExamBin per lo studente	18
2.6	Form di configurazione della domanda	19
2.7	Interfaccia di esempio	20
2.8	Tabella Sandbox	23
3.1	Form per la creazione di domande in NPA-Exam	28
3.2	NPA-Exam: interfaccia di configurazione	31
3.3	NPA-Exam: pannello "inizio prova"	34
3.4	NPA-Exam: pannello "fine prova"	36
3.5	Durata overtime	36
3.6	Messaggio di avviso	37
3.7	Interfaccia di Login	38
3.8	Interfaccia di conferma Login	38
3.9	Intefaccia della prova d'esame	39
3.10	Messaggio di fine prova	40
4.1	Esempio di generazione del voto	65

Elenco listati di codice

2.1	Campo Url	24
3.1	Esempio codice HTML per la copertina	32
4.1	Frammento html	44
4.2	Frammento javaScript	44
4.3	Collegamento alla pagina javaScript	44
4.4	Struttura file Json	45
4.5	Funzione di validazione domande a risposta multipla	46
4.6	Caricamento file nella lista delle prove d'esame	48
4.7	Recupero dati Login	50
4.8	Recupero url dell'esame consegnato	52
4.9	Recupero dati dell'esame consegnato	53
4.10	Creazione directory per i risultati dell'esame	58
4.11	Frammento Json domande a risposta aperta	60
4.12	Frammento Json domande a risposta multipla	61
4.13	Salvataggio dati con JSBin	62
4.14	Frammento Json domande con JSBin	63
4.15	Funzione per cancellare le prove non consegnate	64
4.16	Unificazione codice con Cheerio	67
4.17	Creazione DOM	68
4.18	Test Mocha Chai in ExamBin	69

Capitolo 1

Esami: programmazione su carta

Uno dei principali problemi con i quali gli studenti convivono da sempre è il fatto di non avere la possibilità di svolgere gli esami di programmazione direttamente al computer. Di seguito verranno analizzate, nel dettaglio, le limitazioni riscontrate, alcune delle soluzioni proposte negli anni, con annessi vantaggi e svantaggi, e nello specifico l'idea che c'è alla base di NPA-Exam, il progetto che ho personalmente sviluppato allo scopo di minimizzare il più possibile il problema suddetto, ossia l'impossibilità di svolgere degli esami di programmazione direttamente al computer.

Durante un esame di programmazione, allo studente viene richiesto di scrivere dei programmi che possano risolvere un certo tipo di problemi.

La cosa più semplice sarebbe quella di analizzare un piccolo frammento di codice per cercare eventuali errori. Mentre un qualcosa di più elaborato potrebbe richiedere la realizzazione di un sito web o la scrittura di un programma partendo da zero.

Da ciò si evince che la difficoltà varia in base al tipo di problema proposto. Ma qual è la limitazione che viene imposta allo studente?

È molto semplice rispondere a questa domanda, soprattutto da parte di chi, come me, ha vissuto l'esperienza in prima persona. La limitazione riguarda gli strumenti forniti per svolgere la prova d'esame, che sono semplicemente carta e penna. Ciò fa comprendere che, non avendo a disposizione niente che permetta di testare in tempo reale il proprio

elaborato, sia decisamente facile commettere errori, anche molto banali, che sono la causa poi di valutazioni finali negative.

Per contro, anche il docente si ritrova svantaggiato, perché oltre a non rendersi conto della reale preparazione che ha lo studente, deve necessariamente testare manualmente ogni singolo esercizio per poter dare un voto. Questo fa perdere molto tempo e non è detto che la correzione venga fatta nel modo esatto; infatti, come spesso accade, non riscontra la presenza di qualche errore o interpreta male ciò che legge.

Da qui nasce l'esigenza di ricercare nelle nuove tecnologie uno strumento innovativo che faciliti il lavoro sia allo studente sia al docente.

1.1 Soluzioni proposte nel corso degli anni

Negli anni sono stati molti i ricercatori che si sono occupati di questo problema e nelle università sono stati sviluppati e testati diversi sistemi per svolgere esami interattivi, all'interno di laboratori opportunamente attrezzati.

Amruth [KUM10] ha condotto degli studi, proprio su quest'ultimo punto, per valutare l'effetto che le prove di laboratorio, obbligatorie e/o opzionali, hanno avuto sul rendimento degli studenti e per capire che cosa comporta combinare i laboratori con una prova d'esame online. Ad esempio, durante la prova, allo studente viene presentato un problema e gli viene chiesto di risolverlo scrivendo un programma, oppure gli viene già fornito un programma e lui ha il compito di cercare e risolvere eventuali errori sintattici o logici. È risultato che questo approccio, per la maggior parte dei casi, ha avuto un riscontro positivo tra gli studenti.

Anche Jacobson [JAC00] descrive una prova d'esame nella quale le capacità di ogni studente vengono valutate tramite dei test di laboratorio. La differenza è che in questo caso l'esercizio consiste semplicemente nel completare le parti mancanti di un programma già esistente e testarlo finché non funziona. Inoltre, lo studente ha la possibilità di

ricercare delle guide online che lo aiutino nella risoluzione del problema, ovviamente con delle limitazioni, per impedire che possa copiare da qualcuno o imbrogliare. Un approccio di questo tipo va sicuramente a favore dello studente, ma lo svantaggio è che così non vengono pienamente testate le capacità dello stesso, in quanto non gli viene richiesto di scrivere un programma per intero.

Jonsson [JLNT05], invece, introduce l'uso di AES (Authentic Examination System), tramite il quale gli studenti e i docenti possono collegarsi allo stesso sistema. Lo studente risponde alle domande che gli vengono fornite e al termine dell'esame gli viene comunicato il voto, che dipende dal numero di risposte giuste e dal tempo impiegato. Uno svantaggio di questo approccio è la necessità per i docenti di essere online durante l'esame. Inoltre, se il numero di studenti presenti alla prova è molto elevato, la fase di correzione viene rallentata notevolmente. Per testare questo sistema sono stati utilizzati dei questionari, compilati da 231 studenti, in un periodo di 3 mesi, per un totale di 4 esami.

Come si può notare, queste soluzioni risolvono alcune delle limitazioni citate precedentemente che lo studente si trova a dover affrontare. Tra queste, la possibilità di svolgere l'esame al computer gli permette di rendersi immediatamente conto degli errori che potrebbe aver commesso. Un altro vantaggio è quello di lavorare con editor di testo (per citarne alcuni: emacs¹, gedit², eclipse³) che per la stragrande maggioranza rilevano gli errori di scrittura, evidenziano il codice sorgente e sono dotati di auto completamento del codice. Nello specifico, grazie a quest'ultima funzionalità, appena si inizia a digitare una parola si apre una finestrella che suggerisce il comando o il prototipo della funzione da utilizzare. Come se non bastasse, tutto questo fa risparmiare parecchio tempo, ad esempio, quando si presenta la necessità di utilizzare pezzi di codice ripetuti è sufficiente fare copia e incolla nel punto giusto.

C'è da dire però che, se da un lato l'uso del computer semplifica molto la vita dello

¹<http://www.gnu.org/software/emacs/>

²<https://en.wikipedia.org/wiki/Gedit>

³[https://en.wikipedia.org/wiki/Eclipse_\(software\)](https://en.wikipedia.org/wiki/Eclipse_(software))

studente, dall'altro non risolve pienamente le limitazioni che si presentano al docente. Quest'ultimo, infatti, si ritrova comunque a dover correggere ogni singolo compito manualmente. Per ovviare al problema, avrebbe bisogno di un sistema di validazione automatica che, sulla base di regole prestabilite dal docente stesso, verifichi l'elaborato di ogni singolo studente in maniera rapida ed efficiente.

In questo capitolo, verranno analizzati sistemi di correzione automatizzata, riguardanti la programmazione web, prendendo in considerazione i principali linguaggi utilizzati, quali HTML e CSS, per modellare la parte grafica e statica di un sito Web, e Javascript per creare effetti dinamici sfruttando funzioni (o script).

1.1.1 Web Development

Con il termine, "*Web Development*⁴", si identifica l'ambiente di sviluppo di un sito web per Internet (World Wide Web) o Intranet (una rete privata). Riducendosi alla realizzazione di una semplice pagina HTML o allo sviluppo di applicazioni web più complesse. In questo campo, esistono diversi tool di validazione automatica, per testare singolarmente frammenti di codice o file.

Un esempio, è W3C⁵ (World Wide Web Consortium), usato principalmente per codice HTML⁶ e CSS⁷. Permette di effettuare la validazione in tre diversi modi: tramite l'utilizzo di un URI (indirizzo), caricando un file oppure inserendo manualmente del codice. JSLint⁸, invece, viene utilizzato per testare codice Javascript.

Più specificatamente, esistono degli strumenti, di classificazione automatica, che si adattano meglio all'interazione tra studente e docente. Risalente al 2008 troviamo, Web-CAT [ATEPQ08], il quale, valuta i progetti degli studenti e raccoglie una grande quantità di dati. Dal momento che non sono, però, accessibili è necessaria la presenza di un ulteriore strumento che li prenda in input e li rielabori.

⁴https://en.wikipedia.org/wiki/Web_development

⁵https://en.wikipedia.org/wiki/W3C_Markup_Validation_Service

⁶<https://validator.w3.org/>

⁷<https://jigsaw.w3.org/css-validator/>

⁸<http://www.jshint.com/>

Il progetto Information Business Reporting Tools (BIRT), invece, è un motore di reporting open-source basato sul IDE Eclipse [TEF08]. BIRT riceve i dati da una sorgente e li rielabora tramite l'utilizzo di tabelle e grafici, generando report in diversi formati, tra cui PDF e HTML. In altre parole, quando lo studente termina la prova d'esame riceve un resoconto dettagliato riguardante l'analisi complessiva del test e il punteggio raggiunto.

Nello stesso anno viene presentato un prototipo di sistema, chiamato ProtoAPOGEE [FPQTL08], simile al precedente. Una differenza è che ProtoAPOGEE è in grado di generare una guida, passo dopo passo, che riproduce i casi di test falliti, cioè fornisce dei feedback informativi per aiutare gli studenti a comprendere meglio gli errori commessi.

In particolare, utilizza la libreria Watir⁹ di Ruby [TFH04], uno strumento, open-source, di testing delle applicazioni Web. Tramite il browser vengono simulate le azioni dello studente, valutato il progetto secondo norme stabilite dal docente e generato automaticamente il voto.

Un'estensione di ProtoAPOGEE, che prende il nome di APOGEE, comprende due nuovi moduli:

- Uno strumento di creazione visivo che permette ai docenti di sviluppare script di valutazione dei progetti.
- Un laboratorio virtuale che permette agli studenti di caricare e configurare in remoto i loro progetti, in un database personale, senza interferire gli uni con gli altri. Inoltre, se un progetto viene distribuito con successo, verrà caricato e classificato nel portale Web di APOGEE.

Similmente, anche AWAT[SQF08] sfrutta la libreria Watir di Ruby, ma in questo caso i test vengono effettuati tramite regole definite all'interno di un foglio Excel.

L'idea che c'è alla base è la seguente: il docente fornisce il problema da risolvere e si aspetta che gli studenti pubblicino l'URL della soluzione proposta. A questo punto, il docente fornisce all'applicativo AWAT un documento Excel, che contiene l'insieme di regole che serviranno per la valutazione, e un altro file contenente una lista di tutti gli

⁹<http://watir.com/>

URL consegnati dagli studenti. Ogni compito viene confrontato con i risultati attesi, specificati nel file Excel, e in base al numero di riscontri positivi il sistema formula un possibile voto.

È facile constatare che questi strumenti semplificano notevolmente il lavoro del docente, ma presentano lo stesso dei difetti. Ad esempio, il fatto di dover caricare dei file preconfigurati con una serie di regole. Ciò significa che una lieve modifica alla traccia dell'esame potrebbe comportare una necessaria modifica al file di correzione, fornito al sistema per la verifica. Inoltre, il docente non ha la certezza che il suo file non generi dei bug perché non ha modo di testarlo. Il docente potrebbe, in alternativa, svolgere una prova d'esame per conto proprio e verificare che tutto funzioni correttamente, ma questo implicherebbe una conoscenza approfondita dei linguaggi che si stanno utilizzando.

In generale, tutte le soluzioni fin'ora proposte, non sono ancora sufficienti per realizzare una vera e propria prova d'esame. Basti pensare che lo studente ha bisogno, innanzitutto, di un editor di testo dover poter sviluppare le sue idee.

Dal momento che stiamo analizzando nello specifico problemi riguardanti la programmazione web, diciamo che ha bisogno di un browser per testare il codice scritto, ma facendo un riferimento anche a linguaggi come C o C++ dovrà avere a disposizione un compilatore che gli permetta di eseguire i programmi sviluppati. Inoltre, in nessuno di questi sistemi è permessa la registrazione, cioè non esiste un database che registri tutte le informazioni principali (nome, cognome e matricola) che identificano univocamente ogni studente.

Quest'ultimo aspetto rappresenta uno svantaggio soprattutto per il docente che si vede costretto a mantenere una lista cartacea o anche digitale, salvata su un file, degli studenti iscritti all'esame e a verificare che il numero degli iscritti coincida con i presenti.

Facendo il punto della situazione, per un esame di programmazione web svolto al computer sono necessari almeno quattro strumenti diversi: un database per memorizzare i dati, un ambiente dopo poter scrivere del codice e uno per testarlo, uno strumento di correzione automatica. Tutti da utilizzare contemporaneamente, per far fronte a tutte le mancanze che si presentano nell'unicità.

Quindi, l'idea è quella di realizzare uno strumento innovativo che raggruppi tutte queste idee e sia facilmente utilizzabile anche da chi non è particolarmente ferrato in materia.

C'è da dire che in commercio esistono già dei tool che risolvono questo tipo di problema. Stiamo parlando dei cosiddetti Web Playground (o Code Playground, per tutti i linguaggi che non riguardano la programmazione web). Questo argomento verrà trattato nel dettaglio nel prossimo capitolo, dove verranno analizzati alcuni di questi ambienti di sviluppo.

Capitolo 2

Web Playground

Il termine “*Web Playground*¹”, come facilmente intuibile, identifica un ambiente di sviluppo web, cioè utilizzabile da un qualsiasi browser, che ha lo scopo di facilitare il più possibile l’uso di linguaggi quali HTML, CSS e JavaScript. Infatti, gli utenti hanno la possibilità di scrivere del codice e di vedere subito i risultati dell’esecuzione. Inoltre, alcuni di questi applicativi consentono agli utenti di creare degli account personali, salvare il proprio lavoro e condividerlo con altre persone.

Di seguito, ne verranno descritti alcuni tra quelli che permettono di testare tutti e tre i linguaggi suddetti:

- **Liveweave** [SET12] : è pensato per un team di sviluppatori e web designer. Infatti, la sua principale caratteristica è quella di permettere la collaborazione in tempo reale ad un team di persone, quindi è utile per testare, effettuare debug e gestire parti specifiche di codice. È dotato di code-hinting (suggerimenti sul codice), nonché completamento automatico con un semplice click. Sarebbe utile nel caso in cui il docente decidesse di far collaborare gli studenti per un progetto di gruppo.

- **JSFiddle** [OP09]: è stato uno dei primi web playground sviluppati. Fornisce un

¹https://en.wikipedia.org/wiki/Online_JavaScript_IDE

ambiente personalizzato (sulla base di selezioni dell'utente) per testare il proprio codice. Tra le principali funzionalità notiamo i comandi: “run” che permette di eseguire il codice, “fork” che dà la possibilità di creare un nuovo esempio duplicandone uno già esistente, e “clear” che svuota le aree di lavoro utilizzate. Una funzionalità più avanzata permette di simulare chiamate Ajax.

- **CodePen** [CTA12]: è molto simile a JSFiddle. Nella home sono visibili le realizzazioni più popolari, salvate da tutti gli utenti. Per iniziare si può cliccare su uno dei lavori già presenti per usare il codice come punto di partenza oppure creare un nuovo progetto cliccando sulla voce “new pen”. Anche qui è possibile fare una fork del codice, per fare una copia da modificare a piacimento, si può condividere il proprio progetto tramite il comando “share”, lo si può salvare con il comando “save” e, infine, si può modificare il nome e la descrizione con il comando “info”. Ne esiste una versione PRO, a pagamento, che permette la collaborazione tra più utenti.

Poiché CodePen dà la possibilità di salvare e condividere il proprio codice sarebbe utile per far realizzare agli studenti dei progetti comuni. Inoltre, offre un servizio di “*pair programming*”² e quindi sarebbe interessante sfruttarlo per creare una collaborazione tra studente e docente o tra studente e studente.

- **JSBin** [SHA08]: come i precedenti, è stato ideato per permettere l'elaborazione e il testing immediato di frammenti di codice. È del tutto gratuito ed open-source e, anche in questo caso, l'utente ha la possibilità di condividere il proprio elaborato e di salvarlo. Ma, rispetto agli altri applicativi elencati, JSBin presenta due grandi vantaggi: il salvataggio dei dati in automatico, ogni singola modifica viene registrata, e soprattutto, è possibile scaricare il codice sorgente ed installarlo sulla propria macchina su un server locale.

Questo permette di poter lavorare anche senza la rete internet e in una prova d'esame una funzionalità simile sarebbe l'ideale per impedire copiatore di ogni genere.

²https://en.wikipedia.org/wiki/Pair_programming

Quello che però mette insieme tutti gli elementi descritti nel capitolo precedente, è un applicativo realizzato recentemente da uno studente di Informatica per il Management, dell'Università di Bologna.

- **ExamBin** [BOR14]: come il nome forse suggerisce è un'applicazione, anch'essa open-source e localmente installabile, che mette insieme alcune delle caratteristiche principali di JSBin con l'aggiunta di nuove opzioni che permettono agli studenti di realizzare una prova d'esame di tecnologie web al computer e ai docenti di gestire al meglio la sicurezza e la validità della stessa.

Nella prossima sezione, verranno analizzati nello specifico gli ultimi due applicativi, sopra citati, per capire meglio che relazione c'è tra di loro.

2.1 JSBin VS ExamBin

JSBin³ è stato sviluppato nel 2008 da Remy Sharp, il suo codice è open-source ed è disponibile al relativo account github⁴.

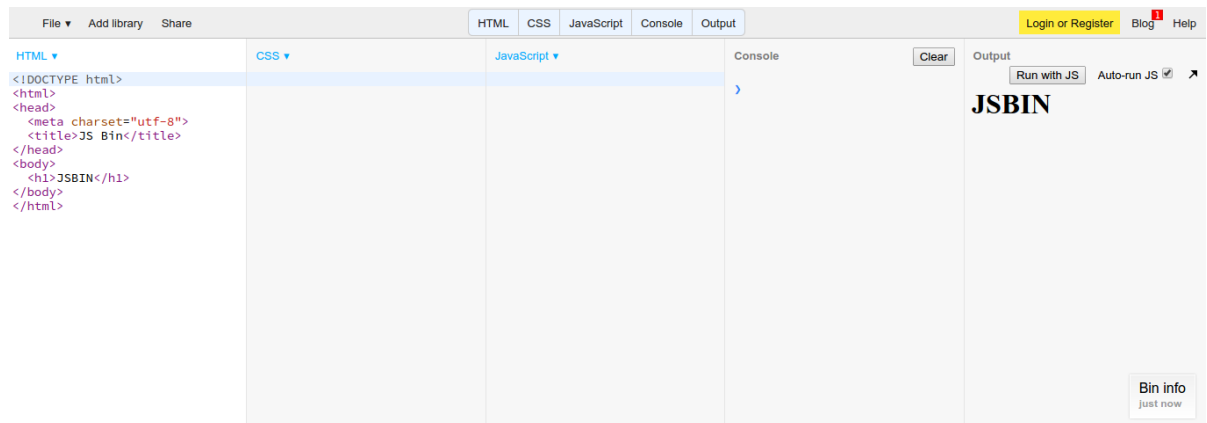


Figura 2.1: Ambiente di sviluppo JSBin

JSBin, nasce come un'applicazione web progettata per testare frammenti di codice JavaScript, HTML e CSS, e per il debug del codice in modo collaborativo. Consente di

³<http://jsbin.com/about>

⁴<http://github.com/remy/jsbin>

ricaricare l'URL mantenendo lo stato del codice, in quanto è dotata di un sistema di salvataggio automatico, mentre molte delle applicazioni citate non lo fanno.

Una volta soddisfatti del proprio lavoro è possibile salvare ed, eventualmente, condividere l'URL con qualcuno. Inoltre, è presente un servizio di registrazione che permette a chiunque di creare un proprio account personale.

Inizialmente l'idea scaturisce da una conversazione con un altro sviluppatore, nel tentativo di aiutarlo a eseguire il debug di un problema Ajax. Lo scopo originale era quello di costruire JSBin utilizzando il motore di Google App, ma successivamente l'intera soluzione è stata realizzata in JavaScript, con l'aggiunta di jQuery e di LAMP⁵, quest'ultimo per il processo di salvataggio.

Come è possibile osservare in Figura 2.1 l'ambiente è suddiviso in 5 sezioni differenti, che possono essere disabilitate/abilitate con un semplice click:

- Le prime 3 sezioni per elaborare, rispettivamente, codice HTML, CSS e JavaScript.
- Una Console per visualizzare eventuali errori sintattici.
- L'ultima sezione di Output per rendersi conto, in tempo reale, dei risultati ottenuti.

Grazie alle molteplici funzionalità offerte nasce l'idea di realizzare un nuovo strumento, ExamBin, sfruttando le caratteristiche di base di JSBin ed estendendole per permettere agli studenti di poter realizzare esami di tecnologie web al computer e al docente di configurare e, successivamente, gestire ogni prova per tutta la sua durata. Inoltre, il fatto che il codice sia disponibile e che l'applicativo sia installabile localmente ha convinto, ulteriormente, chi ha realizzato il progetto ad usufruire di questa piattaforma come rampa di lancio.

Vediamo quali sono le principali funzionalità⁶ che ExamBin mette a disposizione:

- Il docente fornisce una domanda di HTML e/o CSS e/o JavaScript, allegando del codice parzialmente completo o con degli errori.

⁵[https://en.wikipedia.org/wiki/LAMP_\(software_bundle\)](https://en.wikipedia.org/wiki/LAMP_(software_bundle))

⁶<https://github.com/mborghi/ExamBin>

- Un ambiente di lavoro JSBin, privato di internet e collegato ad un server locale, che gli studenti avranno la possibilità di utilizzare per rispondere alla domanda, dopo aver effettuato il login al sistema.
- La possibilità, per lo studente, di consegnare il compito in qualsiasi momento. Allo scadere del tempo la consegna diventa automatica.
- Il contenuto di ogni compito viene mantenuto in una copia privata, all'interno del sistema, la quale è identificata da nome, cognome e matricola dello studente. Il salvataggio viene attivato automaticamente ad ogni modifica apportata.
- Al termine della prova d'esame, a rete riattivata, il docente può correggere/validare i compiti di una certa prova d'esame in maniera automatizzata, standard o custom, eseguendo dei test e generando una possibile valutazione complessiva, sulla base di regole preconfigurate.

Di seguito verranno analizzate, separatamente, le differenze tra i servizi forniti al docente e allo studente.

2.1.1 ExamBin: Funzionalità Docente

Nella Figura 2.2 troviamo l'interfaccia di partenza, visibile solo al docente, che permette di configurare la prova d'esame. La quale, è suddivisa in 3 sezioni:

- **Setup:** Inizialmente questo è l'unico pannello attivo. Nel primo form, che è quello relativo alla gestione dei file del compito, il docente ha la possibilità di caricare tre file, rispettivamente con i formati HTML, CSS, JavaScript. JSBin presenta dei file di default, compatibili con i formati appena citati, che inizializzano l'ambiente (in Figura 2.1) utilizzato dagli studenti per svolgere l'esame. Il tasto "aggiorna" sovrascrive il contenuto di questi file, i quali, conterranno le nuove direttive stabilite dal docente. Il tasto "attuale", come il nome suggerisce, mostra il valore corrente. Infine, il tasto "pulisci" cancella contemporaneamente il contenuto di tutti e tre i file.

The screenshot displays the 'Pannello di controllo del Professore' (Teacher Control Panel) for ExamBin. At the top, there is a navigation bar with 'Guida' (Guide), 'Setup', 'Start', and 'Exam Info' tabs. A 'Pagina di Correzione' (Correction Page) button is also visible. The main content area is titled 'caricamento delle informazioni per INIZIARE l'esame' (loading information to start the exam). It is divided into two main sections:

- Gestione File compito** (Task File Management): This section allows the teacher to upload files for the task. It includes three rows for 'File Html', 'File Css', and 'File Javascript', each with a file selection button and a status indicator ('Nessun file selezionato'). Below these are buttons for 'Aggiorna' (Update), 'Attuale' (Current), and 'Pulisci' (Clear).
- Gestione Temporizzazione compito** (Task Timing Management): This section allows the teacher to set the exam duration and over-time. It features two spinners for 'Durata esame*' (Exam duration) and 'Durata Over Time*' (Over-time duration), with a note '* in minuti' (in minutes). An 'Aggiorna valori Clock del compito' (Update task clock values) button is located below the spinners.

At the bottom center, there is a large blue button labeled 'Permetti agli studenti di Loggarsi' (Allow students to log in).

Figura 2.2: Ambiente di sviluppo ExamBin

Nel secondo form, invece, relativo alla temporizzazione del compito, il docente può visualizzare i valori di default configurati per la durata e per l'over time (eventuale tempo aggiuntivo, che viene concesso allo studente dopo il termine del tempo regolamentare) della prova d'esame.

Una volta configurati questi valori, il grande tasto centrale, "Permetti agli studenti di loggarsi" permette di abilitare il login agli studenti, disabilitare la sezione corrente e abilitare quella di seguito descritta.

- **Start:** In questa sezione, sono presenti solo due bottoni: "Ritorna al setup" che permette la riabilitazione del pannello precedente, qualora il docente si rendesse conto di aver commesso degli errori in fase di configurazione, e "Inizia compito" che dà il via all'esame, disabilita il pannello corrente e reindirizza gli studenti alla schermata dove dovranno svolgere la prova. Ulteriori dettagli su questo punto verranno forniti nella prossima sezione.
- **Exam Info:** Questo è l'ultimo pannello, in Figura 2.3, ad essere attivato e rimarrà tale fino al completamento della prova d'esame. È presente un countdown,



Figura 2.3: Pannello delle informazioni relative alla prova d'esame

lo stesso che sarà visibile anche allo studente, che tiene conto del tempo residuo. Inizialmente il bottone “Termina definitivamente la sessione” non è visibile, compare nell'istante in cui il countdown arriva a zero. Questo permette, appunto, al docente di terminare la sessione d'esame e ritornare alla schermata di setup per configurarne una nuova.

Il bottone sottostante consente al docente di visualizzare la lista di studenti che hanno dichiarato di aver concluso la prova d'esame.

L'ultima informazione presente è quella sullo stato interno del server, relativo alla temporizzazione. Quest'ultima viene gestita tramite la configurazione di un modulo, chiamato “timer”, suddiviso in 7 stati, ognuno dei quali identifica una particolare fase dell'esame. Questo modulo viene interrogato da ogni singolo computer con una frequenza iniziale di 3/5 secondi. Il passaggio da uno stato all'altro può essere manuale o automatico. Nello specifico, sono i seguenti:

1. **NoTest**: Non c'è nessun esame in corso.
2. **Setup**: L'esame sta per iniziare e gli studenti hanno la possibilità di loggarsi, successivamente, verranno reindirizzati ad una pagina intermedia, diversa da quella di login, in attesa che la prova abbia inizio. Il passaggio dallo stato di NoTest allo

stato di Setup è manuale, avviene quando il docente clicca sul pulsante “Permetti agli studenti di loggarsi”.

3. **Ready:** Questo è lo stato, che precede l’inizio vero e proprio della prova, durante il quale viene caricato l’indirizzo della pagina d’esame e resa più frequente l’interrogazione al sistema (0.5/1 secondi) da parte della pagina in cui si trova in attesa lo studente. Così facendo tutti inizieranno a meno di un secondo gli uni dagli altri. Il passaggio di stato avviene quando il docente clicca sul pulsante “Inizia Compito”.
4. **Start:** Dopo 5 secondi dall’inizio del Ready il server effettua un cambio di stato automatico. In questo modo comunica alle pagine in attesa che è il momento di iniziare. Quindi, gli studenti vengono reindirizzati alla pagina d’esame e viene fatto partire il countdown. La frequenza diventa 5/10 secondi.
5. **OverTime:** A 15 secondi dalla fine il server passa automaticamente in questo stato, dove viene ridotta nuovamente la frequenza (4/5 secondi). Questa fase ha una durata pari all’overtime configurato inizialmente dal docente.
6. **AlmostOver:** A 5 secondi dalla fine dell’overtime avviene, di nuovo, il passaggio automatico in questo stato che riduce ulteriormente la frequenza delle interrogazioni (0.5/1 secondi).
7. **Over:** Si arriva in quest’ultimo stato automaticamente al termine del periodo di overtime (che è zero di default). Se ci sono ancora sessioni attive vengono consegnate automaticamente dal sistema. Quando il docente clicca sul tasto “Termina definitivamente la sessione” si ritorna nello stato di NoTest.

Ad ogni interrogazione viene restituito un Json tipo: { status: “currentValue” }

Con 2 eccezioni: se lo status è Ready ci sarà un campo in più, “url”, che identifica l’indirizzo della prova d’esame. Se lo status è Start, il Json restituito sarà lo stesso dello status Ready con l’aggiunta di un nuovo campo “timeout” che identifica la durata dell’esame.

L’ultima funzionalità messa a disposizione del docente è il pannello di controllo, in Figura 2.4, per la correzione automatica dell’esame.

← Home **Pannello correzione compiti del Professore**

Caricamento info per **CORREGGERE** l'esame

File validazione CUSTOM

Abilita Validazione Custom

Gestione Pesì per proposta di voto

form setup pesi validazione

Peso Html standard :

Peso Css standard :

Peso Javascript standard :

Peso Custom (Mocha) :

Voto di partenza :

Aggiorna Pesì per validazione

Scegli data compito : **Correggi Esame**

Figura 2.4: Interfaccia pannello di correzione

Anche in questo caso si nota la presenza di due form. Uno per l'attribuzione dei pesi ad ogni errore, i quali, presentano dei valori di default che possono essere cambiati in qualsiasi momento dal docente e aggiornati tramite il relativo bottone. L'altro form, invece, è adibito alla validazione personalizzata. Il docente può caricare un file contenente precise regole che saranno poi testate dal sistema su ogni singolo compito. Se non viene caricato alcun file i test verranno effettuati su regole di default già preconfigurate.

2.1.2 ExamBin: Funzionalità Studente

Per quanto riguarda lo studente, le funzionalità di cui dispone sono molto semplici. Inizialmente, gli viene richiesto di registrarsi al sistema, tramite un'interfaccia di login, specificando nome, cognome e numero di matricola. Prima di dare conferma deve aspettare che il docente abiliti il relativo bottone.

Una volta superata la fase di login, viene reindirizzato automaticamente in una pagina intermedia, dove rimarrà in attesa di iniziare l'esame. Durante questa fase il client (in questo caso la pagina in cui si trova in attesa lo studente) invierà delle richieste al server,

con le frequenze specificate nella sezione precedente, per avere informazioni sullo stato del timer. Quando quest'ultimo avrà come valore “start” ci sarà un nuovo reindirizzamento automatico, alla pagina dell'esame, perché significa che il docente ha dato il via alla prova.

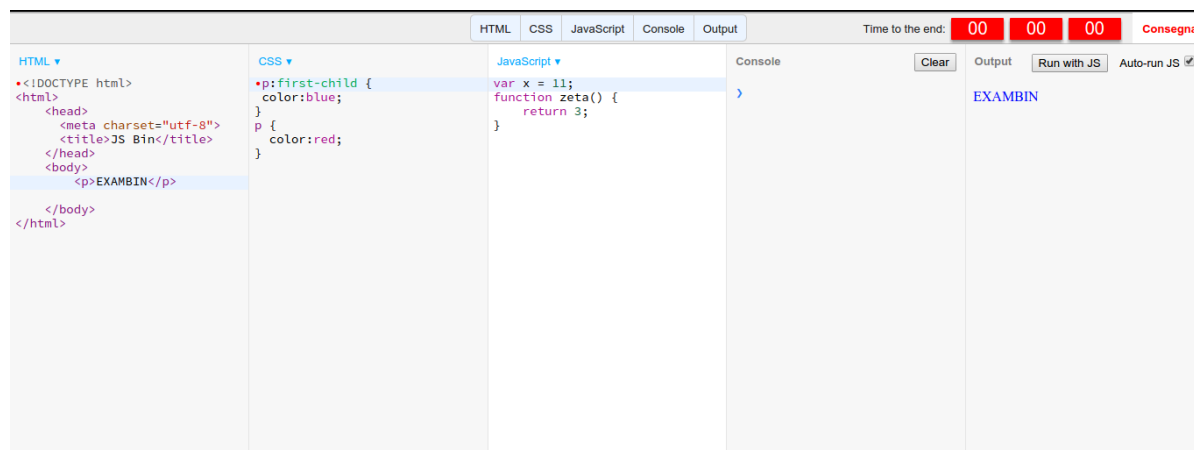


Figura 2.5: Interfaccia ExamBin per lo studente

L'ambiente che si troverà davanti lo studente è quello mostrato in Figura 2.5. Si può dire che è lo stesso di quello di JSBin, con l'aggiunta del countDown e del bottone “consegna” che permetterà allo studente di dichiarare conclusa la sua prova d'esame. Questo strumento garantisce, inoltre, l'impossibilità di copiare, perché ogni partecipante viene reindirizzato ad un url univoco, generato con le credenziali inserite in fase di login. Quindi, di fatto, solo lo studente in questione può accedervi.

2.2 Le limitazioni che presenta ExamBin

ExamBin è uno strumento che favorisce, da un lato lo studente, il quale, finalmente si ritrova a poter svolgere l'esame nell'ambiente più confortevole possibile, con tutti gli strumenti di cui ha bisogno per testare e modificare il proprio elaborato, dall'altro il docente, che può controllare meglio cosa succede durante la prova d'esame ed ha a disposizione un sistema che valuta automaticamente, con delle piccole configurazioni, ogni compito consegnato.

Purtroppo, però, anche in questo caso si presentano delle limitazioni che verranno descritte nelle prossime sezioni.

2.2.1 Una sola domanda da configurare

La limitazione più evidente che presenta ExamBin è l'impossibilità di configurare più domande per la prova d'esame. Per capire meglio verrà data una descrizione più dettagliata del form che consente di configurare la singola domanda.

Riassumendo quanto già detto, al docente viene data la possibilità di creare la domanda caricando nel sistema dei file già configurati, nel form in Figura 2.6, con estensione rispettivamente HTML, CSS e JavaScript, il cui contenuto andrà a sostituire, inizialmente, le informazioni già presenti nei file di default di JSBin, con le omonime estensioni.

Si è scelta un'implementazione di questo tipo per consentire al docente di riutilizzare, in una nuova prova d'esame, gli stessi file senza doverli necessariamente ricaricare; cosa che dovrebbe fare nel caso volesse apportare delle modifiche, anche minime.

Inoltre, il docente è obbligato a caricare ogni volta tutti e tre i file o nessuno; questo significa che per aggiornarne anche solo uno dei tre deve

ricaricare anche la vecchia versione dei file che vuole riutilizzare, nonostante i dati siano già presenti nel sistema. I pulsanti "Attuale" e "Pulisci" vengono utilizzati rispettivamente per visualizzare e per cancellare il contenuto presente nei file di default. Nel caso della cancellazione gli studenti si troveranno a lavorare in un ambiente JSBin "pulito", ciò significa che il docente ha deciso di non mettere a disposizione alcun frammento di codice indicativo o da modificare, quindi gli studenti dovranno scrivere tutto partendo da zero. L'impossibilità di creare più domande sta nel fatto di avere a disposizione un solo form di configurazione. Questo perché JSBin fornisce tre soli file di default che inizializzano l'area di lavoro.



Figura 2.6: Form di configurazione della domanda

Per risolvere il problema bisognerebbe conoscere il numero di domande che il docente ha intenzione di creare e generare di conseguenza lo stesso numero di form da riempire. Questo però comporta un ulteriore problema, perché oltre alla creazione di nuovi form, si dovrebbero creare all'interno del sistema tre file di default, uno per ogni estensione, da associare ad ogni domanda, altrimenti si va a sovrascrivere sempre la stessa area di memoria. Si potrebbe considerare una buona soluzione, ma dal momento che non si sa a priori quante domande il docente ha intenzione di configurare non è funzionale.

Il problema si pone anche nella parte che riguarda gli studenti, i quali hanno a disposizione come unica area di lavoro l'ambiente JSBin. Quindi per inserire più domande nella prova d'esame sarebbe necessaria la presenza di una lista che le contenga tutte, in modo da consentire agli studenti di avere una visione complessiva del lavoro che dovranno svolgere, e più istanze di JSBin per differenziare le varie domande.

2.2.2 Domande a risposta aperta e multipla non disponibili

Svolgere un esame di programmazione non significa solo ed esclusivamente modificare dei frammenti di codice o scriverne di nuovi. Il docente potrebbe anche decidere di fare delle domande teoriche per verificare ancora di più il livello di conoscenza della materia da parte dei propri studenti. L'attuale implementazione di ExamBin non permette di fare domande di questo tipo, poiché si basa strettamente sulle funzionalità di JSBin che non è stato pensato e configurato per questo scopo.

Si consideri il frammento XML della precedentemente domanda. Cosa fa il seguente blocco CSS?

```
p { text-align:justify ; font-size:12pt; background-color: yellow; }
```

Modifica lo stile di un paragrafo giustificando il testo, inizializzando la dimensione di quest'ultimo a 12 point e cambiando il colore di sfondo in giallo.

Quale fra i seguenti elementi HTML non è un componente di un form?

- Input
- Label
- Button
- Title
- Fieldset

Figura 2.7: Interfaccia di esempio

Sarebbe necessario l'inserimento di un nuovo form o la modifica di quello già esiste, per consentire al docente di creare delle domande di questa tipologia, oppure dare la possibilità di caricare un file già configurato da memorizzare nel sistema. Per contro, lo studente dovrebbe avere a disposizione una schermata, che potrebbe essere simile a quella mostrata in Figura 2.7, dove poter visionare le domande e dare le risposte.

Tuttavia, risulterebbe un po' scomodo utilizzare l'interfaccia di JSBin per gestire domande a risposta aperta e/o multipla, in quanto bisognerebbe inserire al suo interno una nuova sezione riservata mentre, le tre sezioni già esistenti, non verrebbero utilizzate poiché non necessarie. Quindi sarebbe utile, ad esempio, configurare una nuova interfaccia, in aggiunta a quella già esistente, per avere una migliore gestione complessiva.

2.2.3 Testo dell'esame cartaceo

Il problema di cui si è parlato sin dall'inizio nasce dal fatto di dover svolgere un esame di programmazione su carta, con tutti gli svantaggi che comporta. Per questo motivo, l'obiettivo è quello di rendere completamente interattiva la prova.

Non si può definire tale un esame nel quale viene concessa agli studenti la possibilità di rispondere ai quesiti proposti utilizzando un computer, se gli stessi quesiti sono presentati sotto forma cartacea. Questa è un'altra limitazione che presenta l'applicativo ExamBin.

Come già detto, fornisce agli studenti come piano di lavoro l'ambiente JSBin, per modificare a piacimento il proprio codice, però, questo ambiente non mette a disposizione un'interfaccia dove poter caricare il testo della domanda, perciò, il docente si vede costretto a consegnare un testo cartaceo o al massimo fornire una spiegazione verbale di quello che gli studenti sono tenuti a fare.

Ciò che si evince anche dalle sezioni precedenti è che risolvere questi problemi non è impossibile. È necessario adottare una logica diversa in fase di configurazione, creando un form più esteso che dia la possibilità al docente di differenziare le domande, in base alla tipologia, inserendo una traccia o una breve descrizione da associare all'esercizio.

D'altra parte, nella sezione dedicata agli studenti, diventa a questo punto obbligatorio

realizzare una pagina che contenga la lista di tutti quesiti da risolvere, con collegamenti all'ambiente JSBin solo quando necessario.

2.2.4 Controllo manuale: studenti registrati e prove consegnate

Quello che potrebbe rendere ancora più completa una prova d'esame di questo tipo e che sicuramente potrebbe agevolare il lavoro del docente sarebbe un sistema automatico configurato adeguatamente per memorizzare i dati degli studenti che si registrano al sistema e soprattutto degli studenti che hanno consegnato la prova.

Generalmente succede che il giorno dell'esame, prima di iniziare la prova, il docente deve impiegare un po' del suo tempo per fare l'appello ed assicurarsi che tutti gli studenti che si sono iscritti all'esame siano effettivamente presenti. Successivamente deve controllare nuovamente la lista di iscrizioni, quando necessario, per segnare via via tutti gli studenti che hanno consegnato.

Adottando la soluzione precedentemente descritta, il docente avrebbe una visione più chiara degli studenti che stanno svolgendo l'esame e di quelli che effettivamente hanno deciso di consegnare. Quindi, per verificare che tutto funzioni correttamente gli basta andare a confrontare i dati raccolti in fase di login e in fase di consegna ed accertarsi che coincidano; sarebbe quindi opportuno implementare un contatore che tenga il conto automaticamente, in entrambi i casi. Ciò non esclude in fatto che il docente possa comunque decidere di controllare la sua copia cartacea delle iscrizioni, per essere totalmente sicuro, ma avendo a disposizione uno strumento del genere risparmierebbe più tempo.

In realtà ExamBin possiede una tabella in cui vengono registrati i dati delle prove d'esame concluse, ma non essendo una tabella dinamica per visualizzare i risultati il docente deve ogni volta cliccare su un apposito pulsante. Quindi, quello che si ripropone di implementare sono due tabelle dinamiche per memorizzare i dati degli studenti registrati e di quelli che hanno consegnato la prova, un modo da rendere il tutto più semplice ed efficiente.

2.2.5 Una sola prova d'esame durante la giornata

L'ultima e forse la più importante limitazione che ExamBin presenta è il fatto di non poter svolgere più di una prova d'esame durante l'arco della giornata. Per capire meglio il perché è necessario analizzare in dettaglio il metodo utilizzato per recuperare e salvare i dati relativi alla prova d'esame.

ExamBin sfrutta molto le caratteristiche di JSBin, in questo caso, tra tutte le tabelle presenti nel database ne gestisce in particolare una; la tabella Sandbox, raffigurata in Figura 2.8, che generalmente viene usata per salvare i lavori realizzati dagli utenti, viene leggermente modificata per poter salvare i compiti degli studenti.

```
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  
  `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 N  
  `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 2.8: Tabella Sandbox

I campi più utilizzati sono in particolare “html”, “css” e “javascript”, che vengono utilizzati per memorizzare il codice scritto, mentre, il campo “url” è stato modificato per identificare univocamente ogni studente.

L'applicativo JSBin genera questo campo tramite il metodo **shortCode** definito nel file “utils.js” della directory “lib”, per farlo si serve di una funzione **random()**, che presi in input delle lettere e dei numeri, genera un *url* in maniera del tutto casuale. In ExamBin, invece, il metodo **shortCode** utilizza i dati recuperati dal login e genera un *url* univoco per ogni studente registrato. Il prototipo del nuovo *url* generato è presentato a seguire nel Listato 2.1.

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

Listato 2.1: Campo Url

La prima informazione identifica la data, in formato americano, in cui si svolge l'esame; a seguire il nome, il cognome e il numero di matricola dello studente, e infine un id autoincrementante, utilizzato per altri scopi non rilevanti nel caso che si sta analizzando. La prima volta che lo studente effettua una modifica al proprio codice, viene creato un record all'interno di questa tabella, tramite dei servizi offerti da JSBin. Ad ogni nuova modifica, utilizzando sempre le stesse funzioni già configurate all'interno del sistema, il record precedentemente creato viene aggiornato con le ultime informazioni aggiunte.

Ciò che impedisce la realizzazione di più esami nello stesso giorno è legato al modo in cui l'esame viene identificato. Infatti, ExamBin recupera dal campo *url* creato la data in cui si svolge la prova e la considera un id univoco per la stessa. Così facendo è impossibile distinguere il numero di prove svolte durante il giorno; sarebbe necessario modificare la tabella Sandbox aggiungendo un nuovo campo come id dell'esame o cercare un'alternativa alla logica di gestione adottata in questo caso.

Con l'obiettivo di risolvere questi problemi e l'intenzione di aggiungere nuove funzio-

nalità per migliorare sempre di più l'applicazione nasce l'idea di sviluppare una nuova versione di ExamBin, che ho personalmente implementato. Le caratteristiche e le modifiche apportate, graficamente e strutturalmente, verranno esaminate meglio nel prossimo capitolo.

Capitolo 3

Da ExamBin a NPA-Exam

NPA-Exam è l'acronimo di *“No Paper Anymore for Exams”*, un modo molto semplice per dire che non ci sarà più bisogno di svolgere gli esami, di programmazione si intende, su carta. Nasce come estensione di ExamBin, al fine di colmare alcune delle mancanze che quest'ultimo presenta, le quali sono state descritte nella sezione precedente. Lo scopo è quello di modellare quest'applicazione, renderla semplice, efficace e sicura, per permettere la corretta configurazione ed esecuzione di un esame di programmazione web a livello universitario.

Quello che però mi ha particolarmente convinta a contribuire alla realizzazione di questo progetto è stata, senza alcun dubbio, l'esperienza vissuta in prima persona. Realizzare un esame di programmazione su carta non è per niente semplice. È già difficile pensare ad un algoritmo che risolva il problema proposto in un tempo limitato (2-3 ore nella maggior parte dei casi), se poi a questo si sottrae il tempo che serve per cercare di eseguire mentalmente il proprio codice alla ricerca di errori, commentare e spiegare le funzioni utilizzate, per rendere più chiari i ragionamenti e facilitare il docente in fase di correzione, diventa quasi impossibile fornire una versione funzionante senza la presenza di errori sintattici.

Quindi avere a disposizione uno strumento di questo tipo sicuramente ha i suoi vantaggi. Lo studente può gestire meglio il tempo che ha a disposizione e rendersi conto di quello che sta facendo, il docente invece ha il pieno controllo sulla prova d'esame e un sistema di correzione automatica da usare come meglio crede.

Un altro aspetto importante, da sottolineare, è l'aver mantenuto il codice open-source, per consentire a chiunque, in futuro, di scaricarlo e modificarlo a piacimento, aggiungendo nuove funzionalità o utilizzandolo come base per un nuovo progetto.

Nella prossima sezione verranno analizzate principalmente le soluzioni proposte per risolvere i problemi legati ad ExamBin, con una breve descrizione di ciò che è stato realizzato.

3.1 NPA-Exam risolve i problemi di ExamBin

NPA-Exam nasce per completare ExamBin. Di seguito verranno elencati i principali problemi riscontrati e le soluzioni adottate per risolverli, i cui dettagli implementativi saranno trattati meglio più avanti.

1. **Singola domanda all'esame:** per risolvere questo problema si è scelto di modificare completamente la grafica relativa alla pagina di configurazione del compito, cambiando anche la logica implementativa. Exambin disponeva di unico form che consentiva di caricare tre file differenti, per gestire rispettivamente codice HTML, CSS e JavaScript.



Figura 3.1: Form per la creazione di domande in NPA-Exam

In NPA-Exam viene inserito un nuovo form strutturalmente diverso, raffigurato in figura Figura 3.1, che permette al docente di definire, innanzitutto, la tipologia della domanda che andrà a configurare e successivamente di compilare i rispettivi

campi in base alla tipologia scelta; i dettagli su come viene effettivamente realizzata la domanda sono descritti nella sezione 3.2.1.

Questo risolve il problema della singola domanda poiché il docente può creare un numero di istanze, dello stesso form, pari al numero di domande che ha intenzione di fare all'esame.

2. **Domande a risposta aperta e multipla:** l'inserimento del nuovo form risolve anche questo problema, in quanto come già detto è possibile definire la tipologia della domanda da fare, che può essere "text" (risposta aperta), "radio" (risposta multipla), "jsbin" (necessità l'uso dell'ambiente JSBin). Anche in questo caso maggiori dettagli si troveranno nella sezione appena citata.

3. **Testo dell'esame cartaceo:** come suggerito nel capitolo precedente, nella sezione dedicata agli studenti è stata inserita una pagina che contiene la lista di tutte le domande configurate da parte del docente, tramite l'uso del suddetto form, con collegamenti a JSBin quando necessario.

Così facendo non ci sarà più bisogno di consegnare una copia cartacea del testo d'esame agli studenti, i quali ritroveranno le domande da svolgere direttamente nella pagina dell'esame.

4. **Login Studenti e prove consegnate:** ExamBin non prevede l'uso di una tabella per mantenere i dati degli studenti registrati al sistema ma fornisce soltanto una tabella degli studenti che hanno consegnato la prova. Al contrario, NPA-Exam mette a disposizione due tabelle dinamiche, per gestire entrambe le situazioni e per velocizzare il recupero dei dati.

La prima tabella è stata inserita nella fase "*Inizio Prova*" descritta nella sezione 3.2.2 e raffigurata in Figura 3.3; essendo una tabella dinamica viene aggiunta una nuova riga ogni qualvolta uno studente si registra al sistema.

La seconda tabella è stata inserita nella fase "*Fine Prova*" descritta nella sezione 3.2.3 e raffigurata in Figura 3.4; anche in questo caso la tabella è dinamica e viene aggiunta una nuova riga quando uno studente decide di consegnare definitivamente la prova. Entrambe le tabelle vengono svuotate quando si da inizio ad una nuova prova d'esame.

5. **Un solo esame durante la giornata:** questo è l'ultimo dei problemi più evidenti riscontrati in ExamBin. Infatti, non è possibile effettuare più prova d'esame nell'arco della stessa giornata. Questo succede perché ExamBin crea un record, per ogni studente, all'interno del database dove vengono salvate le informazioni riguardanti l'esame; ogni record è identificato dalla data odierna, quindi non c'è distinzione tra un esame che ad esempio si svolge la mattina e uno che si svolge al pomeriggio.

Per risolvere il problema si è scelto di non creare un record all'interno del database, quindi di non modificare la struttura della tabella Sandbox, ma di salvare i dati all'interno di un file Json. Nello specifico, ad ogni studente viene associato un file e questi file sono contenuti in una directory identificata da un nome simile a "data+numeroprova". Così facendo, ad esempio, i dati degli studenti che hanno svolto l'esame di mattina saranno nella cartella "data+prova1", invece, per l'esame svolto di pomeriggio saranno nella cartella "data+prova2". Maggiori dettagli su questo punto verranno forniti nel Capitolo 4.

Questi sono i punti fondamentali che distinguono i due progetti, ma l'idea di fondo è la stessa; da un lato vi è la presenza del pannello di controllo del docente e dall'altro dell'ambiente di sviluppo fornito allo studente. Nelle prossime sezioni verranno descritte separatamente le funzionalità riservate rispettivamente al docente e allo studente.

3.2 Funzionalità Docente

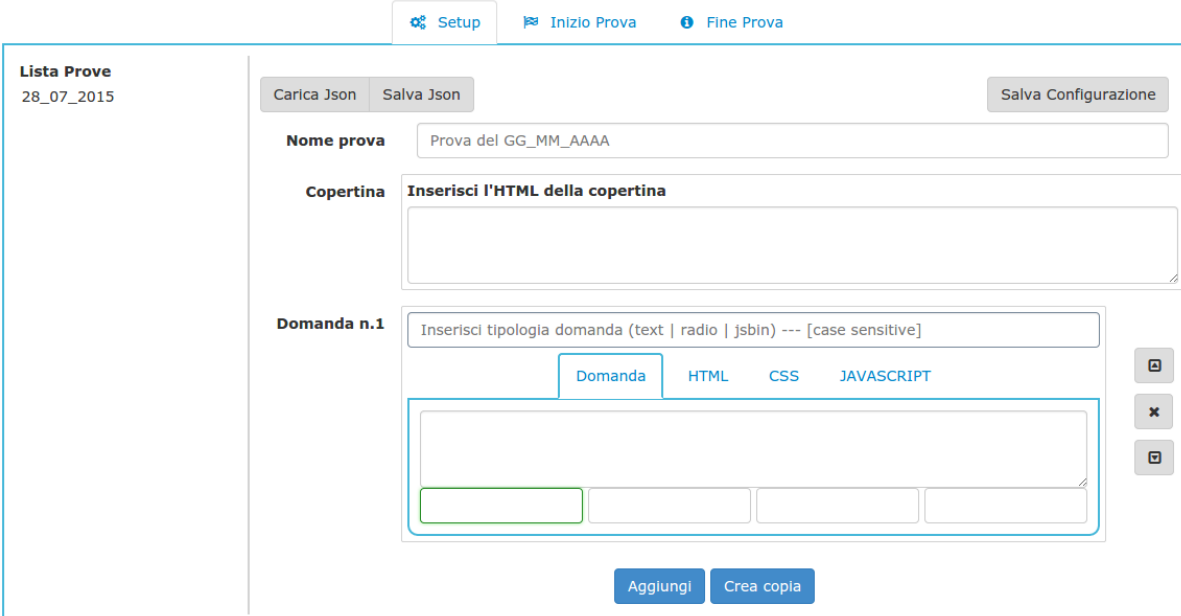
L'interfaccia di configurazione della prova d'esame è stata progettata appositamente per essere gestita dal docente. Ci si accede dal browser, utilizzando la seguente sintassi "**http://domain:port/path**", dove *domain* rappresenta l'indirizzo fisico della macchina sulla quale è installato l'applicativo e può essere un nome o un indirizzo IP, *port* identifica il numero di porta, configurato per questo servizio, e *path* identifica la risorsa vera e propria. In questo caso, l'interfaccia è disponibile all'indirizzo "/professor", quindi ci si può accedere digitando, ad esempio, "**http://localhost:7002/professor**".

È suddivisa in tre macro sezioni, che identificano tre diverse fasi di gestione; a grandi linee distinguiamo:

- **Setup:** dove il docente ha la possibilità di configurare le domande da sottoporre agli studenti, manualmente o caricando un file già pronto.
- **Inizio Prova:** dove il docente può decidere quanto deve durare l'esame, abilitare il login agli studenti e dare il via alla prova.
- **Fine Prova:** dove il docente può controllare l'andamento della prova d'esame e, se lo ritiene necessario, aumentarne la durata o terminare definitivamente.

I tre pannelli rimangono attivi per tutta la durata d'esame. Ma, una volta dato il via alla prova non è più possibile cambiare le configurazioni iniziali. Per non incorrere in eventuali errori sono stati inseriti dei messaggi di avvertimento o, in alternativa, disabilitati alcuni bottoni. Di seguito, verranno analizzate nel dettaglio le singole sottosezioni.

3.2.1 Setup



The screenshot shows the 'Setup' phase of the NPA-Exam configuration interface. At the top, there are three tabs: 'Setup' (active), 'Inizio Prova', and 'Fine Prova'. Below the tabs, there are buttons for 'Carica Json', 'Salva Json', and 'Salva Configurazione'. The main configuration area is divided into sections:

- Nome prova:** A text input field containing 'Prova del GG_MM_AAAA'.
- Copertina:** A large text area with the placeholder text 'Inserisci l'HTML della copertina'.
- Domanda n.1:** A section for configuring a question. It includes a text input field with the placeholder 'Inserisci tipologia domanda (text | radio | jsbin) --- [case sensitive]'. Below this are four tabs: 'Domanda' (selected), 'HTML', 'CSS', and 'JAVASCRIPT'. A large text area is provided for the question content, with a small input field below it. To the right of this area are three buttons: a copy icon, a close icon (X), and a refresh icon.

At the bottom of the configuration area, there are two buttons: 'Aggiungi' and 'Crea copia'.

Figura 3.2: NPA-Exam: interfaccia di configurazione

A sinistra è visibile la lista, che rimarrà tale in tutte e tre le interfacce, delle prove d'esame già configurate e presenti nel sistema, che possono essere utilizzate come base per creare una nuova prova. A destra, invece, è presente il vero e proprio pannello di configurazione. Il docente può decidere di caricare un file Json, la cui struttura verrà poi mostrata nel dettaglio nel capitolo successivo, già preconfigurato e di salvarlo, servendosi dei due pulsanti in alto a sinistra. Oppure, può costruire il file manualmente riempiendo i relativi campi del form.

La prima cosa da specificare è il nome della prova, come suggerito; la data inserita, diventerà il nome del file Json da generare.

Nel secondo campo viene richiesto l'inserimento del codice html di una "copertina", visibile nella pagina alla quale verrà reindirizzato lo studente una volta effettuato il login, in attesa di iniziare la prova d'esame. Un esempio potrebbe essere quello raffigurato nel Listato 3.1.

```
1 <h1> Esame di Tecnologie Web del 28_07_2015 </h1>
2 <h2> Corso di laurea in Informatica A.A. 2014/2015 </h2>
```

Listato 3.1: Esempio codice HTML per la copertina

L'ultimo campo, il più importante, è quello relativo alla creazione delle domande. Di default è presente una sola domanda, ma è possibile aggiungerne di nuove semplicemente cliccando sul tasto "aggiungi". In questo modo comparirà un nuovo campo, identico a quello già presente con degli id diversi, per non creare collisioni in fase di salvataggio. Per definire una domanda bisogna, innanzitutto, specificarne la tipologia, che può essere "text", "radio" o "jsbin". È importante ricordare che il testo è "case sensitive", quindi, text è diverso da Text; un errore di questo tipo non permette la corretta creazione della domanda.

A questo punto, è necessario riempire correttamente le relative aree di input, coerentemente alla tipologia scelta. Distinguiamo 3 casi:

1. **text**: identifica una domanda a risposta aperta. È sufficiente scrivere la domanda nella textarea presente nel tab "domanda".

2. **radio**: identifica una domanda a risposta multipla. In questo caso non basta scrivere solo la domanda, ma, bisogna inserire le 4 opzioni negli input del suddetto tab, tenendo presente che l'input evidenziato in verde è quello dove va inserita la risposta corretta, in modo da facilitare la fase di correzione. Vengono utilizzati i *radio button*¹, al posto dei *checkbox*², perché lo studente deve effettuare una scelta esclusiva, cioè non può dare più di una risposta per ogni domanda.
3. **jsbin**: identifica una domanda di pratica, che necessita della piattaforma JSBin per essere risolta. Questa tipologia prevede l'inserimento di una domanda testuale, così come per i precedenti casi, nella textarea del tab "domanda" e se la traccia richiede l'elaborazione di frammenti di codice, al docente viene data la possibilità di scrivere manualmente all'interno delle textarea relative ai tab "HTML", "CSS" e "JAVASCRIPT", ciò che lo studente andrà poi a modificare.

Una volta inseriti tutti i dati all'interno del form è possibile salvare le modifiche cliccando sul tasto "Salva Configurazione". Così facendo verrà generato in automatico un file Json contenente le informazioni richieste.

Da notare, la presenza di tre pulsanti, a lato di ogni domanda. Quello centrale consente di cancellare la domanda corrente, mentre, quello in alto e quello in basso, danno la possibilità di scambiare, la stessa, rispettivamente, con la domanda precedente o successiva.

L'ultima funzionalità messa a disposizione in questa sezione è quella di creare una copia, di uno qualsiasi, degli esami presenti nel sistema. In questo modo il docente può riciclare una prova già fatta in passato, apportando qualche piccola modifica. Per farlo basta selezionarne una dalla lista e cliccare sul tasto "crea copia"; quest'azione comporterà una serie di operazioni in sequenza:

- ◇ generazione di un file identico al precedente.
- ◇ il nome del nuovo file sarà lo stesso del vecchio con l'aggiunta alla fine di un contatore, che identifica il numero di copia.

¹https://en.wikipedia.org/wiki/Radio_button

²<https://en.wikipedia.org/wiki/Checkbox>

Ad esempio se il file di partenza è “28_07_15” la sua copia sarà “28_07_15_c1”, dove “c1” definisce la copia numero uno, proseguendo per logica, “28_07_15_c2” sarà la copia numero due, etc..

- ◇ aggiunta della copia alla lista delle prove disponibili.

Fatto questo, sarà sufficiente selezionare dalla lista la copia appena creata, i dati contenuti nel file verranno caricati nei rispettivi campi del form e si potranno modificare a piacimento. Successivamente si potrà salvare utilizzando sempre il tasto “Salva Configurazione”. Ovviamente è possibile creare una copia della copia e via dicendo, seguendo lo stesso procedimento.

3.2.2 Inizio Prova

In questa fase il docente stabilisce la durata della prova d’esame, che per default è stata inizializzata a 120 minuti, successivamente abilita il login agli studenti. La novità è che ha a disposizione un elenco di tutti gli studenti che si registrano al sistema, mantenuto all’interno della tabella rappresentata in Figura 3.3.

Setup Inizio Prova Fine Prova

Lista Prove
28_07_2015

Prova esame del 28 luglio 2015

Studenti loggati 2.

postazione	studente	matricola
computer A	Paolo Paolini	667788
computer B	Mario Rossi	123456

Durata esame (in minuti)

Aggiorna valori Clock del compito

Permetti Login Inizia Il Compito

Figura 3.3: NPA-Exam: pannello “inizio prova”

La tabella si riempie a run time recuperando le informazioni presenti nel database e presenta tre campi:

- **postazione:** identifica il computer sul quale si trova a lavorare ogni singolo studente, il modo in cui riempire questo campo viene concordato verbalmente tra studenti e docente.
- **studente:** identifica il nome e il cognome dello studente.
- **matricola:** identifica la matricola dello studente.

Inoltre, è stato configurato un contatore che tiene traccia dell'effettivo numero di studenti loggati, risparmiando così al docente la fatica di doverli contare personalmente. Inizialmente la tabella è vuota, il contatore è inizializzato a zero e il pulsante "Inizia il Compito" risulta disabilitato, in quanto, non è concesso dare il via ad una prova d'esame se nessuno studente è presente. Quindi, cliccando sul pulsante "Permetti Login", questo viene automaticamente abilitato, ma, rimane comunque l'impossibilità di far partire l'esame finché, almeno, uno studente non si è registrato.

Avendo a disposizione una lista di iscrizioni all'esame e un contatore all'interno del sistema, è facile, per il docente, capire quando tutti gli studenti si sono correttamente registrati. A quel punto non gli resta che dare il via alla prova, cliccando sul tasto "Inizia il Compito", e spostarsi alla sezione successiva che verrà descritta di seguito.

3.2.3 Fine Prova

In Figura 3.4 viene riportata la struttura dell'ultimo pannello messo a disposizione del docente. Dove gli vengono fornite le informazioni relative alla prova d'esame in corso. La tabella, inizialmente vuota, è la stessa di quella illustrata in Figura 3.3 con l'aggiunta di un campo che definisce lo stato della prova d'esame di ogni studente; appena quest'ultimo decide di consegnare comparirà un messaggio di conferma: "prova consegnata".

Quindi, è facilmente intuibile che lo scopo è quello di mantenere i dati di tutti gli studenti che consegnano l'elaborato. Come nel caso precedente, anche qui vi è un contatore che tiene traccia del numero di studenti che hanno terminato correttamente la prova.



Figura 3.4: NPA-Exam: pannello “fine prova”

Sulla sinistra, sotto la tabella troviamo un’indicazione sull’istante in cui parte la prova d’esame, cioè esattamente quando lo stato del server assume il valore “start”. Nello stesso istante viene fatto partire anche il countdown, sulla destra, che identifica il tempo residuo. Inoltre, viene mantenuta l’informazione che riguarda lo stato del server nelle varie fasi dell’esame.


Il tempo regolamentare verrà esteso di: 10 min

Figura 3.5: Durata overtime

Al docente viene data la possibilità di estendere, qualora lo ritenga necessario, il tempo regolamentare, di 1, 5 o 30 minuti, servendosi dei pulsanti in basso a sinistra. Ovviamente se si vuole incrementare il tempo di 10 minuti basta cliccare due volte sul pulsante centrale, in quanto, esiste un contatore interno al sistema che somma le quantità. Inoltre, comparirà un messaggio, come in Figura 3.5, che identifica il tempo totale stabilito; questo avviso è stato inserito a scopo informativo per evitare dimenticanze e concedere più tempo del previsto.

Infine, può decidere di terminare la prova d'esame in qualsiasi momento, cliccando sul pulsante "Forza fine prova adesso", in fondo a destra. Questo comporterà il cambiamento di stato del server da *start* a *over*, che a sua volta causerà la consegna automatica di tutte prove attive.

È importante tenere presente che i pannelli, "setup" e "inizio prova", descritti nelle sezioni precedenti, rimangono sempre attivi; ma, al docente non è consentito effettuare modifiche di alcun tipo finché la prova d'esame non è conclusa. Per evitare di incorrere in errori, sono stati disabilitati alcuni pulsanti e/o inseriti dei messaggi indicativi, un esempio in Figura 3.6, in entrambi i pannelli.



PROVA D'ESAME IN CORSO ... IMPOSSIBILE EFFETTUARE MODIFICHE

Figura 3.6: Messaggio di avviso

Riassumendo quanto detto, il docente ha la possibilità di configurare la prova d'esame, manualmente o con l'aiuto di un file già pronto, abilitare il login agli studenti, tener conto di chi effettivamente si è registrato, mediante l'uso di una tabella dinamica, dare il via alla prova d'esame con la possibilità di gestirne la temporizzazione, verificare quali sono gli studenti che hanno consegnato grazie alla presenza di un'altra tabella dinamica e, infine, terminare se vuole la prova in qualsiasi momento.

Nel prossima sezione verranno, invece, analizzate le fasi che caratterizzano lo sviluppo della prova d'esame da parte degli studenti.

3.3 Funzionalità Studente

Questa sezione è interamente dedicata alle funzionalità messe a disposizione degli studenti. Il ciclo di sviluppo della prova d'esame si suddivide generalmente in tre fasi principali: il login, lo svolgimento e la consegna del compito. In particolare, tra le prime due si contrappone una fase intermedia di attesa, che precede l'inizio della prova.

3.3.1 Login

L'interfaccia di login è disponibile all'indirizzo `/student` della macchina sulla quale è installato l'applicativo.

Allo studente viene richiesto di inserire il nome della postazione in cui si trova, generalmente i computer sono numerati o identificati da un nome, nel caso non lo fossero il dato da inserire in questo campo è da concordare verbalmente con il docente, e successivamente nome, cognome e numero di matricola. Una volta compilato il form, visibile in Figura 3.7, allo studente non resta che concludere la registrazione cliccando sul omonimo pulsante, il quale, verrà abilitato dal docente una volta completata la fase di configurazione.



The image shows a web form titled "Pagina di login dello studente". Under the heading "Inserimento dati", there are four input fields: "Postazione:", "Nome:", "Cognome:", and "Matricola:". A "Registrati" button is located below the fields.

Figura 3.7: Interfaccia di Login

3.3.1.1 In attesa di cominciare

Completata la fase di login, lo studente verrà automaticamente reindirizzato in una pagina intermedia, disponibile all'indirizzo `/confirmStudent`, di conferma dell'avvenuta registrazione.

Come già detto, in questa sede, viene anche visualizzata la copertina, rappresentata in Figura 3.8 dalle ultime due righe e relativa all'esempio proposto nel Listato 3.1, che il docente stabilisce.



Figura 3.8: Interfaccia di conferma Login

Lo scopo di questa pagina è solo ed esclusivamente quello di far attendere allo studente l'inizio della prova d'esame. L'attesa non è definita a priori ma varia a seconda delle condizioni che si vengono a verificare, ad esempio, può dipendere dal tempo impiegato da ogni persona per registrarsi al sistema.

Quindi, appena il docente ha la certezza che tutto sia pronto può dare il via all'esame e lo studente verrà reindirizzato automaticamente alla pagina in cui dovrà svolgere il compito.

3.3.2 Svolgimento

Questa è la fase più importante per lo studente perché si ritroverà di fronte ad una pagina, della quale viene mostrato un esempio in Figura 3.9, che contiene la lista delle domande alle quali dovrà rispondere e un countDown che identifica il tempo ancora disponibile.



The screenshot shows an exam interface with a blue border. At the top, a timer displays 'Tempo rimasto: 01 59 53'. Below the timer is the title 'LISTA DOMANDE'. The first question asks which of the following does not refer to Semantic Web, with radio buttons for 'Transmission Control Protocol', 'Resource Description Framework', 'Web Ontology Language', and 'Linked Data'. The second question asks for a CSS rule to associate white text on a black background to all span elements of class 'invert', with the text 'JSBIN' below it. The third question asks for a brief definition of AJAX, with a large text input field below it. The fourth question asks which HTML marker should be used to create an ordered list, with radio buttons for 'ul', 'ol', 'table', and 'p'. At the bottom, there are two buttons: 'Consegna' and 'Ritirati'.

Figura 3.9: Interfaccia della prova d'esame

Per tutte le domande viene fornita una traccia e degli strumenti per rispondere in modo adeguato, che variano in base alla diversa tipologia:

1. **Risposta aperta:** lo strumento messo a disposizione è una semplice textarea da utilizzare liberamente per una risposta testuale.

2. **Risposta multipla:** vengono fornite 4 possibili risposte, di cui solo una è quella corretta. Il caricamento delle stesse è randomizzato per impedire che tutti si ritrovino con la stessa sequenza e, soprattutto, perché dal momento che in fase di configurazione viene considerata come corretta la prima risposta inserita, questo procedimento non permette agli studenti di notare il trucco.
3. **JSBin:** in questo caso viene creato un collegamento all'ambiente JSBin, rappresentato in Figura 2.1, modificato appositamente per questo scopo con l'aggiunta di un `countDown`, in modo che lo studente possa sempre tenere sotto controllo il tempo che gli rimane.

L'ultima tipologia viene resa disponibile nel caso in cui, allo studente, venga richiesto di scrivere del codice di sua iniziativa, per risolvere un problema, e/o modificare dei frammenti già esistenti, per correggere eventuali errori o completare delle parti mancanti.

3.3.3 Consegna

L'ultima fase è quella relativa alla consegna dell'elaborato. Lo studente può decidere di consegnare in qualsiasi momento cliccando sul tasto "consegna", situato in basso a sinistra nella pagina con la lista delle domande, oppure è libero di ritirarsi dalla prova cliccando sul tasto "ritirati", situato in basso a destra nella stessa pagina. In quest'ultimo caso è come se avesse consegnato un compito in bianco.



Figura 3.10: Messaggio di fine prova

Dopo la consegna o il ritiro, ci sarà un reindirizzamento automatico in una nuova pagina, tempo in più. In quel caso, il `countDown` cambia colore e diventa rosso, questo sta ad indicare che il tempo regolamentare è finito ma che lo studente può continuare la sua prova.

Dopo la consegna o il ritiro, ci sarà un reindirizzamento automatico in una nuova pagina,

Allo scadere del tempo, se lo studente non ha ancora consegnato il suo elaborato, il sistema provvede a consegnare automaticamente per lui, a meno che il docente non abbia deciso di concedere del

all'indirizzo */finishExam*, dove verrà visualizzato un messaggio di conferma, visibile in Figura 3.10, dell'avvenuta conclusione della prova. Lo scopo è quello di chiudere definitivamente la sessione d'esame, impedendo allo studente di poter continuare ad apportare modifiche al proprio compito.

Riassumendo, lo studente ha la possibilità di registrarsi al sistema, comunicando i propri dati personali e la postazione in cui si trova, successivamente si mette in attesa finché il docente non dà il via alla prova d'esame, dopo di che può svolgere quest'ultima in un ambiente più consono che gli fornisce tutti gli strumenti di cui ha bisogno, ed infine, può consegnare il compito, dichiarando ufficialmente conclusa la propria prova oppure se non è soddisfatto del proprio lavoro può anche ritirarsi.

Nel prossimo capitolo verranno trattati i dettagli implementativi e mostrati i frammenti di codice più significativi che caratterizzano l'elaborato descritto.

Capitolo 4

Dettagli implementativi

Per la realizzazione di quest'applicazione bisogna, innanzitutto, prendere coscienza di quello che si ha già a disposizione. Quindi ho dovuto studiare attentamente le caratteristiche di JSBin, ma soprattutto di ExamBin, capire la logica che c'è dietro ad ogni funzione ed acquisire dimestichezza con il codice, prima di poter iniziare ad apportare delle modifiche. Inoltre, è importante avere una buona padronanza con la piattaforma Node.js¹ e i motori di template, con i quali è stato realizzato il tutto.

Node.js è una piattaforma costruita per realizzare facilmente applicazioni di rete veloci e scalabili. Invece, i motori di template sono delle librerie Javascript, utilizzate per rielaborare porzioni di testo, generalmente HTML, tramite l'uso di *template* (o modelli), cioè espressioni che vengono risolte dinamicamente a seconda dei dati ricevuti. Di seguito, viene descritto un esempio per capire come i suddetti modelli vengono elaborati nello sviluppo di quest'applicazione, prendendo in considerazione i frammenti di codice che configurano la durata della prova d'esame.

Nel frammento html, visibile nel Listato 4.1, il modello da prendere in considerazione è identificato dall'espressione “`{{durationExam}}`”, che viene risolta a run time nel frammento javascript, nel Listato 4.2. Nello specifico, viene fatta una chiamata

¹<https://nodejs.org/>

ajax² di tipo POST, all'indirizzo */setDataClockAula* che inizialmente, tramite la funzione `getDurationInMinutes()`, recupera il valore di default configurato e lo sostituisce nell'espressione sopra citata. Questa funzione restituisce un frammento Json con due campi, *exam* e *overTime*, che identificano rispettivamente la durata complessiva della prova d'esame e il tempo aggiuntivo che il docente può concedere; in questo caso, viene selezionato il campo "exam".

Se il docente decide di cambiare questo valore, cliccando sul pulsante di conferma, genera una nuova richiesta POST allo stesso indirizzo che permette di sovrascrivere il valore di default restituendo un nuovo Json aggiornato.

```

1 <input type='number' id='
   inputDurationTest' name='
   inputDurationTest'
2 value='{{durationExam}}' />

```

Listato 4.1: Frammento html

```

1 var duration = moduleTimer.
   getDurationInMinutes();
2 res.render("controlPanelProf",{
3   durationExam: duration.exam
4 });

```

Listato 4.2: Frammento javascript

Un'istruzione importante, da non trascurare, è il rendering alla pagina HTML "controlPanelProf", perché in questo modo si comunica al sistema qual è la pagina che contiene l'espressione da risolvere. Di contro, nella pagina HTML, va creato un collegamento all'omonima pagina javascript dove viene gestita la richiesta POST, includendo uno script simile a quello raffigurato nel Listato 4.3.

```

1 <script src="{{static}}/js/myjsbin/controlPanelProf.js"></script>

```

Listato 4.3: Collegamento alla pagina javascript

Ulteriori strumenti utilizzati per lo sviluppo di questo progetto sono stati il framework Bootstrap³ per il front-end dell'applicazione, le librerie jQuery⁴ di javascript e un database MySQL⁵ per il salvataggio dei dati relativi ad ogni studente.

²[https://en.wikipedia.org/wiki/Ajax_\(programming\)](https://en.wikipedia.org/wiki/Ajax_(programming))

³[https://en.wikipedia.org/wiki/Bootstrap_\(front-end_framework\)](https://en.wikipedia.org/wiki/Bootstrap_(front-end_framework))

⁴<https://en.wikipedia.org/wiki/JQuery>

⁵<https://en.wikipedia.org/wiki/MySQL>

Nelle prossime sezioni verranno descritte le modifiche più significate che hanno portato alla realizzazione di NPA-Exam.

4.1 Struttura file Json

Come già detto viene generato un file Json che contiene le informazioni riguardanti la prova d'esame stabilite dal docente durante la fase iniziale di configurazione. La struttura, della quale viene presentato un esempio nel Listato 4.4 , è molto semplice.

```
1 {
2   "name": "Prova del 28_07_2015",
3   "cover": [],
4   "question": [
5     {
6       "type": "radio",
7       "text": ["Quanto fa cinque per quattro?"],
8       "options": ["20", "16", "3", "giallo"]
9     },
10    {
11      "type": "text",
12      "text": ["Come si fa un paragrafo in HTML?"]
13    },
14    {
15      "type": "jsbin",
16      "text": ["Crea un paragrafo con il testo rosso."],
17      "html": ["<html>", "</html>"],
18      "css": [""],
19      "javascript": ["" ]
20    }
21  ]
}
```

Listato 4.4: Struttura file Json

Esistono 3 campi principali:

- **name:** identifica univocamente la prova d'esame.
- **cover:** è un array di stringhe HTML (per essere precisi ogni stringa corrisponde ad un'intera riga) e identifica la copertina che comparirà nella pagina di attesa dello studente.
- **question:** è un array di oggetti Json; ogni oggetto è identificato da una tipologia, definita all'interno del campo *type*.

La tipologia "text" è la più semplice, ha in aggiunta solo un campo omonimo che conterrà il testo della domanda, a risposta aperta. La tipologia "radio" presenta un campo in più rispetto alla precedente, *options*, un array che conterrà le 4 opzioni, tra le quali, lo studente dovrà scegliere la risposta corretta alla domanda fornita, che anche in questo caso si trova all'interno del campo *text*. In questo è importante precisare che le opzioni devono essere esclusivamente 4, quindi non è possibile configurarne solo una, due o tre. Per non incorrere in errori è stata creata una funzione, nel listato, che effettua una verifica.

```
1 function validateMoreChoise(idChoise){
2     var countNotNull = 0;
3     for (var j = 0; j < 4; j++) {
4         if (idChoise[j] !== '') {
5             countNotNull++;
6         }
7     }
8     if(countNotNull === 4){
9         return true;
10    }
11    else{
12        return "tutti i 4 campi devono essere riempiti";
13    }
14 }
```

Listato 4.5: Funzione di validazione domande a risposta multipla

La funzione riceve in input un array che contiene gli id dei 4 campi relativi alle opzioni, con un ciclo *for* si controlla ogni campo per confermare che sia inizializzato, se lo è il contatore viene incrementato. Al termine del ciclo si fa un controllo sul contatore, se è esattamente 4 allora la funzione restituisce *true*, altrimenti un messaggio di avviso.

L'ultima tipologia disponibile è "jsbin", che ha un campo *text* per la domanda testuale e altri 3 campi, rispettivamente *html*, *css* e *javascript*, che conterrà le porzioni di codice che lo studente dovrà modificare all'esame.

Se il docente non inizializza nessuno degli input associati in fase di configurazione questi 3 campi non vengono generati e l'oggetto Json sarà simile a quello di una domanda a risposta aperta, cioè con i soli campi *type* e *text*, ma con una tipologia differente.

4.2 Gestione lista esami

Nella pagina riservata al docente è presente una lista delle prove d'esame passate, questo è molto comodo perché è possibile riutilizzarle per crearne una nuova partendo già da qualcosa di pronto. Il meccanismo di caricamento prevede un'interrogazione al server, tramite una richiesta POST all'indirizzo */appendFile*, per ricercare all'interno della cartella "exams", che si trova nella directory views del progetto e il cui path relativo è *NPA-Exam/views/exams*, i file disponibili.

Per prima cosa viene fatto un controllo per verificare che la cartella esiste, se non esiste viene creata altrimenti si va a controllare se al suo interno ci sono dei file, nel qual caso questi file vengono caricati nella sezione riservata e saranno visibili nella schermata di gestione del docente.

Il caricamento vero e proprio viene fatto utilizzando una funzione, **appendFilePresent()**, richiamata all'inizio appena viene fatto partire l'applicativo, che prende in input i dati restituiti dalla chiamata ajax e tramite un ciclo *for*, raffigurato nel Listato 4.6, carica uno per uno i file presenti nella cartella.

```
1 for(i=0; i<response.len; i++) {  
2   $('<code>.list</code>').append(<code><option id="select">selectNumber data-link=response</code>  
   <code>.result[i]>response.file[i]</code></option>);  
3 }
```

Listato 4.6: Caricamento file nella lista delle prove d'esame

Quindi se non ci sono stati errori e il “response” contenente i dati viene ricevuto correttamente, verranno caricati tutti i file presenti della cartella exams. Ognuno di questi, avrà un *id* dato dall'unione della stringa “select” più un numero, identificato da “selectNumber”, cioè un contatore che parte da 1 e si incrementa ad ogni iterazione, in questo modo viene garantita l'univocità dell'identificativo. Inoltre, viene inserito un campo “data-link” che conterrà il path del file, in modo da poterne recuperare il contenuto al momento opportuno, ed infine, *file[i]* è il nome dell'i-esimo file caricato, privato dell'estensione “.json”.

Quello che si ottiene è una stringa HTML di questo tipo:

```
<code><option id="select1" data-link="./views/exams/28_07_2015.json">28_07_2015</option></code>
```

Le volte successive, quando il docente crea un nuovo file viene richiamata una nuova funzione, **appendLastFile()**, molto simile a quella appena descritta, che prima di effettuare il caricamento controlla che il file non esista già nella lista. Questo impedisce, in caso di errori, di creare più istanze dello stesso file.

4.3 Inizializzazione tabelle dinamiche

La gestione viene fatta tramite un modulo, configurato appositamente all'interno del progetto, chiamato “**studentData**” che contiene una serie di funzioni utili per recuperare i dati di ogni studente, in fase di login e in seguito alla terminazione della prova. In questo caso, vengono utilizzate quattro delle funzioni implementate, nello specifico due per ogni tabella.

Per l'inizializzazione della tabella, riservata al mantenimento dei dati degli studenti loggati, vengono utilizzate le funzioni `setDataStudent()` e `getDataStudent()`, rispettivamente per salvare e poi recuperare i dati.

`setDataStudent()` prende in input le informazioni recuperate dal form del login (nome, cognome, matricola e postazione) di ogni studente e crea un oggetto Json, come segue:

```
var data = {
    'studentName' : newName,
    'studentSurname': newSurname,
    'studentRegistrationNumber': newRegistrationNumber,
    'studentPost': newPost
};
```

Ogni oggetto viene salvato in un array e recuperato in un secondo momento dalla funzione `getDataStudent()`, la quale, restituisce gli oggetti singolarmente.

Similmente, per l'inizializzazione della tabella, riservata al mantenimento dei dati degli studenti che hanno terminato l'esame, vengono utilizzate le funzioni `setUrlExamComplete()` e `getUrlExamComplete()`. La prima prende in input l'url della prova che è stata consegnata e lo salva in un array, successivamente, la seconda funzione permette di recuperare, uno per uno, i valori memorizzati nell'array.

Nelle sezioni seguenti, verrà descritta nel dettaglio la logica del funzionamento e le circostanze in cui vengono utilizzate le funzioni appena descritte.

4.3.1 Tabella degli studenti registrati

La tabella, in Figura 3.3, viene aggiornata ogni qualvolta lo studente clicca sul pulsante "Registrati", nel form in Figura 3.7. In quel momento si scatenano una serie di eventi, sia nella sezione dedicata agli studenti che in quella dedicata al docente.

Nel primo caso, lo studente viene reindirizzato, dalla pagina di login alla pagina di attesa. Durante questo passaggio i dati vengono recuperati e memorizzati, tramite una richiesta POST all'indirizzo `/confirmStudent`, nel modulo precedentemente citato, richiamando la funzione `setDataStudent()`.

Nel secondo caso, ogni 10 secondi viene richiamata la funzione `updateTable()` che fa una richiesta POST all'indirizzo `/getDataStudent`. Lo scopo è quello di richiamare costantemente la funzione `getDataStudent()` per verificare se qualcuno si è registrato al sistema. I dati restituiti da questa chiamata vengono elaborati ed inseriti nella tabella, con il frammento di codice visibile nel Listato 4.7.

```
1  if(data){
2    var tableRow;
3    var $tbody = $("#tableStartExam tbody");
4    var $spanStudentLog = $("#studentLog");
5
6    tableRow = "<tr>";
7    tableRow += "<td>" + data.studentPost + "</td>";
8    tableRow += "<td>" + data.studentName + " " + data.studentSurname
9                + "</td>";
10   tableRow += "<td>" + data.studentRegistrationNumber + "</td>";
11   tableRow += "</tr>";
12   $tbody.append(tableRow);
13   studentLog++;
14   $spanStudentLog.text(studentLog);
15 }
```

Listato 4.7: Recupero dati Login

Inizialmente vengono recuperate le posizioni degli elementi, rispettivamente, della tabella da riempire e dello span che conterrà il contatore degli studenti registrati correttamente al sistema. Quindi viene creato il frammento HTML della nuova riga, dove la prima colonna conterrà la postazione in cui si trova a lavorare lo studente, la seconda colonna contiene il nome e il cognome dello studente, separati da uno spazio e la terza ed ultima colonna identifica il numero di matricola dello studente.

La funzione `append` permette di aggiungerlo definitivamente alla tabella. Infine, viene incrementato il numero di studenti loggati al sistema e sovrascritto questo nuovo valore nello span definito all'inizio.

È importante sottolineare che la funzione `updateTable()` viene richiamata finché lo stato interno del server ha valore "Setup". Infatti, nel momento in cui viene dato il via

alla prova e lo stato assume il valore di “Start”, non è più necessario continuare a fare dei controlli poiché non è più consentito effettuare il login ad esame già iniziato.

4.3.2 Tabella delle prove concluse

La tabella, in Figura 3.4, viene aggiornata ogni qualvolta lo studente clicca sul pulsante “Consegna” o sul pulsante “Ritirati”, nella schermata in Figura 3.9. Anche in questo caso, si scatenano una serie di eventi, sia nella sezione dedicata agli studenti che in quella dedicata al docente.

Quando lo studente decide di consegnare il compito, di ritirarsi o quando il tempo a disposizione è scaduto, viene fatta una richiesta POST, all’indirizzo */deliveryExam*. Lo scopo è quello di recuperare l’url della pagina, contenente la prova d’esame appena conclusa, e salvarlo nel sistema; l’istruzione successiva è quella di reindirizzare lo studente alla pagina di fine esame, in Figura 3.10, per impedirli di apportare ulteriori modifiche. C’è da fare una precisazione; poiché gli studenti hanno la possibilità di consegnare o di ritirarsi, bisogna distinguere i due casi. Nel momento in cui viene fatta la richiesta POST, all’indirizzo sopra citato, l’url recuperato dalla pagina dell’esame viene modificato prima di essere salvato. Si distinguono due casi:

- Se lo studente decide di consegnare viene richiamata la funzione **deliveryExam()** che aggiunge alla fine dell’url la stringa “_delivery”. L’url restituito è simile al seguente:

```
listPage_2015-08-17_Mario_Rossi_123456_computerA_delivery
```

- Se lo studente decide di ritirarsi viene richiamata la funzione **notDeliveryExam()** che aggiunge alla fine dell’url la stringa “_notdelivery”. L’url restituito è simile al seguente:

```
listPage_2015-08-17_Mario_Rossi_123456_computerA_notdelivery
```

La funzione che si occupa di salvare questo url prende il nome di **setUrlExamComplete()**. Per prima cosa è necessario eliminare tutti gli underscore per poter recuperare

i dati, questo compito viene affidato alla funzione `split()` che divide la stringa iniziale in un array di stringhe e restituisce il nuovo array. Dal momento che i primi due campi corrispondono rispettivamente al nome della pagina e alla data dell'esame, non sono necessari in questo caso e quindi vengono ignorati; invece, con i dati restanti viene creato un frammento Json, come mostrato nel Listato 4.8, del quale il campo "status" diventa fondamentale per il corretto aggiornamento della tabella, poiché identifica lo stato della sessione conclusa.

Questo frammento viene poi salvato in un array denominato "*urlExamComplete*", presente sempre all'interno del modulo `studentData`, che contiene l'elenco di tutte le prove concluse o non consegnate.

```
1  var dataToSend = {};  
2  var urlArray = strUrlExam.split("_");  
3  dataToSend.name = urlArray[2];  
4  dataToSend.surname = urlArray[3];  
5  dataToSend.registrationNumber = urlArray[4];  
6  dataToSend.post = urlArray[5];  
7  dataToSend.status = urlArray[6];  
8  urlExamComplete.push(dataToSend);
```

Listato 4.8: Recupero url dell'esame consegnato

Per quanto riguarda la sezione del docente, similmente a quanto succedeva prima, ogni 5 secondi viene richiamata la funzione `updateTableFinish()` che fa una richiesta ajax di tipo POST all'indirizzo `/getFinishStudent` richiamando costantemente la funzione `getUrlExamComplete()` per verificare se qualcuno ha consegnato l'esame o se ha deciso di ritirarsi. Questa funzione restituisce l'array costruito precedentemente, il quale viene rielaborato per creare la nuova riga da inserire nella tabella, come si può notare nel Listato 4.9.

La creazione e l'inserimento del frammento di codice HTML relativo alla riga, così come l'aggiornamento del contatore, sono analoghi al caso descritto nella sezione precedente. Quello che cambia sono i controlli fatti prima di aggiornare la riga, poiché i dati vengono gestiti in modo diverso.

```
1  var $tbody = $("#tableEndExam tbody");
2  var $rows = $('tableEndExam tbody tr');
3  var $spanStudentEnd = $("#studentEnd");
4  var tableRow;
5  for (var i = 0; i < res.length; i++) {
6      var boolean = 1;
7      var tmp = res[i];
8      $rows.each(function () {
9          var matricola = $(this).find("td").eq(2).html();
10         if (matricola === tmp.registrationNumber) { boolean = 0; }
11     });
12     if(boolean && tmp.status === "delivery"){
13         tableRow = "<tr>";
14         tableRow += "<td>" + tmp.post + "</td>";
15         tableRow += "<td>" + tmp.name + " " + tmp.surname + "</td>";
16         tableRow += "<td class=\"" + ".mat" + "\">" +
17             tmp.registrationNumber + "</td>";
18         tableRow += "<td>" + "Consegnata" + "</td>";
19         tableRow += "</tr>";
20         $tbody.append(tableRow);
21         studentEnd++;
22     }
23     else if(boolean && tmp.status === "notdelivery"){
24         /* stesso codice dell'if precedente */
25         /* ma viene aggiunto lo stato "Non consegnata" */
26         examNotDelivery.push(tmp.registrationNumber);
27     }
28 }
29 $spanStudentEnd.text(studentEnd);
30 }
```

Listato 4.9: Recupero dati dell'esame consegnato

Dal momento che viene restituito un array di oggetti Json è necessario utilizzare un ciclo per scandirli tutti e ad ogni iterazione l'oggetto corrente viene salvato in una variabile temporanea. Inoltre, poiché i dati contenuti nell'array non vengono mai cancellati per tutta la durata della prova d'esame, per evitare di caricare nella tabella delle informazioni

già presenti, si effettua un controllo sul campo “matricola” che identifica univocamente uno studente e per questo l’informazione non può essere duplicata.

Quindi per ogni riga della tabella si confronta il campo “matricola” di quest’ultima con l’omonimo campo presente nell’oggetto Json corrente preso in esame; se i valori sono uguali vuol dire che i dati relativi a quello studente sono già presenti nella tabella e non devono essere ricaricati, per questo motivo la variabile booleana viene settata a 0. Così facendo le condizioni dei due if risultano false e non viene fatto alcun aggiornamento.

L’altro controllo viene fatto sullo status della sessione; se l’esame è stato consegnato lo status avrà valore “delivery” e quindi nella tabella, nella sezione riservata, verrà inserita la stringa “Consegnato”, viceversa se l’esame non è stato consegnato lo status avrà valore “notdelivery” e nella tabella verrà inserita la stringa “Non consegnato”. In quest’ultimo caso viene memorizzata in un array, denominato *examNotDelivery*, la matricola dello studente che non ha consegnato l’esame.

Quindi al termine della prova in questo array saranno presenti le matricole di tutti gli studenti che si sono ritirati; informazioni che verranno utilizzate in un secondo momento, ma dettagli verranno forniti nelle prossime sezioni.

4.4 Temporizzazione

La temporizzazione dell’intera prova d’esame, come per il vecchio applicativo Exam-Bin, viene gestita tramite un modulo configurato appositamente, denominato *timer*, al quale sono state apportate lievi modifiche. Al suo interno sono stati definiti sette stati, che identificano diverse fasi dell’esecuzione della prova stessa, e ad ogni interrogazione fatta a questo servizio viene restituito un Json che contiene le informazioni necessarie a comprendere lo stato raggiunto.

Il passaggio da uno stato all’altro può essere manuale e/o automatico e può scatenare una serie di eventi. Inizialmente le interrogazioni vengono fatte con una frequenza pari a “**3000 + Math.random(2000)**”, un numero casuale tra 3000 e 5000 millisecondi (3/5 secondi), per evitare che più richieste fatte nello stesso momento provochino sovrapposizioni di tempi; questa frequenza può variare a seconda dello stato in cui ci si trova.

Di seguito, verranno analizzati nello specifico tutti e sette gli stati:

1. **Notest:** è lo stato iniziale, configurato come valore di default all'interno del modulo, indica che non c'è nessun esame in corso. Durante questa fase il docente ha il compito di configurare la prova d'esame. Viene restituito un Json del tipo:

```
{
    status : "notest"
}
```

2. **Setup:** il passaggio a questo stato è manuale; si ottiene nel momento in cui il docente abilita il login agli studenti, cliccando sul tasto "Permetti Login". Nello stesso istante viene riattivato il tasto, inizialmente disabilitato, che permette ad ogni studente di identificarsi al sistema. Viene restituito un Json del tipo:

```
{
    status : "setup"
}
```

3. **Ready:** anche in questo caso il passaggio di stato è manuale; si ottiene quando il docente dà il via alla prova, cliccando sul pulsante "Inizia Il Compito". In questa fase viene diminuita la frequenza con la quale vengono fatte le interrogazioni, passando da 3/5 secondi a 0.5/1 secondi, da parte della pagina di attesa in cui sono stati reindirizzati gli studenti dopo il login. Così facendo tutti inizieranno la prova a meno di un secondo gli uni dagli altri.

Inoltre, viene aggiunta un'informazione in più al frammento Json da restituire, che identifica l'indirizzo della pagina nella quale si svolge l'esame, il risultato è il seguente:

```
{
    status : "ready",
    url: "http://localhost:7002/urlEsame"
}
```

4. **Start:** il passaggio in questo stato avviene automaticamente dopo 5 secondi dall'inizio dello stato precedente. In questa fase, che identifica l'inizio dell'esame, gli studenti vengono reindirizzati alla pagina, nella quale, dovranno svolgere la prova e la frequenza delle interrogazioni viene aumentata, passando da 0.5/1 secondi a 5/10 secondi. Una nuova informazione viene aggiunta al frammento Json e corrisponde al tempo residuo a disposizione, espresso in millisecondi, che alla prima interrogazione risulterà, ovviamente, uguale al tempo complessivo della prova configurato dal docente (il valore di default è inizializzato a 10 minuti).

Il Json restituito è il seguente:

```
{
  status : "start",
  url: "http://localhost:7002/urlEsame",
  timeout: "60000"
}
```

5. **Overtime:** il passaggio automatico in questo stato avviene quando mancano 15 secondi al termine dello stato di "Start". La durata di questa fase (che può anche essere nulla) varia in base alle decisioni prese dal docente, che a runtime, decide se estendere o meno il tempo regolamentare, servendosi dei pulsanti presenti nel pannello "*Fine Prova*", raffigurato in Figura 3.4.

La frequenza delle interrogazioni viene ridotta a 4/5 secondi e il Json restituito è il seguente:

```
{
  status : "overtime"
}
```

6. **AlmostOver:** il passaggio, nuovamente automatico, in questo stato può avvenire se lo stato precedente ha valore 0 oppure quando mancano 5 secondi al suo termine. Il solo scopo di questo cambiamento è quello di rendere ancora più frequente il numero delle interrogazioni. Quindi, come succedeva nel caso dello stato "Ready",

si ritorna a 0.5/1 secondi.

Viene restituito un Json di questo tipo:

```
{
    status : "almostover"
}
```

7. **Over:** il passaggio in quest'ultimo stato può avvenire sia manualmente che automaticamente, a seconda degli eventi.

Distinguiamo due casi:

- ◇ generalmente avviene in automatico al termine del tempo regolamentare, se non è stato definito alcun *overtime*.
- ◇ manualmente quando il docente clicca sul pulsante “Forza terminazione esame”, raffigurato sempre nel pannello “*Fine Prova*”, in Figura 3.4.

In entrambi i casi, il raggiungimento di questo stato sta ad indicare che la prova d'esame si è conclusa; in particolare, nei primi due casi, se gli studenti non hanno ancora consegnato le loro prove, provvede automaticamente il sistema a farlo per loro. Inoltre, dopo 5 secondi dal termine di questo stato si ritorna automaticamente nello stato di “Notest”, in modo che il docente possa cominciare una nuova prova d'esame, qualora lo volesse. Il Json restituito è il seguente:

```
{
    status : "over"
}
```

Le interrogazioni di cui si è parlato vengono chiaramente fatte, sia dalle pagine configurate per il docente che da quelle per gli studenti, tramite chiamate Ajax al servizio disponibile all'indirizzo `/getClockAula`, il quale, ad ogni richiesta richiama l'omonima funzione `getClockAula()` che identifica lo stato attuale e restituisce il Json corrispondente, come descritto. Mentre, per effettuare le modifiche manuali dello stato, è necessario interrogare il servizio disponibile all'indirizzo `/setClockAula`, il quale, richiama l'omonima funzione `setClockAula()` che prende in input il nuovo valore e lo sostituisce al vecchio.

4.5 Gestione di più prove durante la giornata

Per poter effettuare più di una prova durante la giornata è stato modificato il modo di identificare la prova stessa. Come già detto più volte, nella versione di Examin l'esame era identificato dalla data in cui veniva svolto. In NPA-Exam invece alla data viene associato una nuova informazione che identifica il numero della prova svolta.

Quando il docente crea il file Json, cliccando sul tasto “Salva Configurazione”, viene fatta una richiesta ajax di tipo POST all'indirizzo “*createFileJson*” che si occupa di creare appunto il file e di inserirlo nella directory “./views/exams/” contenente la lista di tutti i Json configurati. Oltre a questo però si occupa di creare anche una nuova directory, all'interno di “./views/result/”, che prenderà il nome di “**dataesame_numeroprova**” e conterrà tutti i dati delle prove consegnate dagli studenti. Per avere un'idea più chiara il codice viene riportato nel Listato 4.10.

```
1   currentExam = currentDateExam+"_P"+numberExam;
2   var dir = './views/result/';
3   var dirResultToday = './views/result/'+currentExam;
4   if (!fs.existsSync(dirResultToday)) {
5       createDirectory(dirResultToday);
6   }
7   else{
8       var filename = [];
9       var list = fs.readdirSync(dir);
10      list.forEach(function (file) {
11          filename.push(file);
12      });
13      var lastDir = filename.pop();
14      var numberlastDir = lastDir.slice(-1);
15      numberExam = parseInt(numberlastDir) + 1;
16      currentExam = currentDateExam+"_P"+numberExam;
17      var newDirResultToday = './views/result/'+currentExam;
18      createDirectory(newDirResultToday);
19  }
```

Listato 4.10: Creazione directory per i risultati dell'esame

La prima variabile modificata corrisponde al nome della nuova directory da creare, il quale, è composto dalla data dell'esame e dal numero della prova (inizializzato a uno); tutte e tre le variabili prese in considerazione sono globali, poiché vengono utilizzate anche in altre funzioni all'interno del sistema.

Le due istruzioni successive identificano il path della directory padre e della nuova directory che dovrebbe essere creata al suo interno. In poche parole se quest'ultima non esiste allora viene creata, altrimenti, si fa una visita all'interno della directory padre per cercare l'ultima che è stata inserita.

Ad esempio, supponendo che la directory padre sia vuota e che quella da creare sia "1_1_2014_P1", allora la condizione dell'if risulterà vera e quindi verrà fatto l'inserimento. Se viene configurato un nuovo file, per iniziare una nuova prova lo stesso giorno, la condizione questa volta sarà falsa, quindi verrà recuperato il numero dell'ultima directory creata ed incrementato di uno e la nuova sarà "1_1_2014_P2" e così via. In questo modo, vi è la possibilità di distinguere prove d'esame svolte in momenti diversi della giornata.

4.6 Salvataggio

Per salvare i compiti consegnati, non si utilizza più un record all'interno del database ma un semplice file Json da associare ad ogni studente. Per farlo è stato necessario implementare delle nuove funzioni ed apportare qualche modifica a quelle già presenti nell'applicativo JSBin.

Ogni volta che uno studente si registra al sistema viene richiamata la funzione **create-FileResultStudent()** che prende in input il numero di matricola e crea un file Json, il cui nome sarà proprio la matricola in modo da garantire l'univocità, e che è molto simile a quello configurato dal docente, ma ad ogni modifica apportata verranno aggiornate le informazioni al suo interno. Questo file verrà poi salvato nella cartella corrispondente alla prova d'esame in corso che sarà creata come descritto nella sezione precedente.

Avendo a disposizione tre diverse tipologie di domande le funzioni implementate sono anch'esse tre e verranno descritte singolarmente nelle prossime sezioni.

4.6.1 Salvataggio domande a risposta aperta

Per le domande a risposta aperta lo studente ha a disposizione una textarea dove poter scrivere liberamente tutto ciò che ricorda. Ogni textarea ha un id, ad esempio “*textarea2*”; la scelta non è casuale, infatti in questo caso viene identificata la textarea associata alla domanda numero due.

Questa informazione viene sfruttata dalla funzione di salvataggio per ricercare all’interno del file il punto esatto in cui salvare i dati ricevuti, tenendo in considerazione però che si a disposizione un array di oggetti Json e che quindi la domanda numero due si troverà in posizione “numerodomanda-1”.

Tenendo presente sempre lo stesso esempio, ad ogni modifica apportata viene fatta una richiesta all’indirizzo */saveValueTextarea*; prima di tutto viene recuperata la matricola dello studente per poter ricercare il file corretto, una volta trovato viene selezionato l’oggetto Json corrispondente alla seconda domanda e a questo aggiunto un nuovo campo “**answer**” in cui verranno salvati i dati recuperati nella textarea.

Il Json restituito è visibile nel Listato 4.11.

```
1 {
2   "type": "text",
3   "text": [
4     "Come si fa un paragrafo in HTML?"
5   ],
6   "answer": [
7     "<p>",
8     "</p>"
9   ]
10 }
```

Listato 4.11: Frammento Json domande a risposta aperta

4.6.2 Salvataggio domande a risposta multipla

Per le domande a risposta multipla lo studente ha a disposizione quattro possibili opzioni tra le quali scegliere. Ogni opzione ha un suo id, un po' diverso da quello relativo alla textarea, un esempio è “*choise31*” che identifica la prima opzione della domanda numero tre.

Similmente a quanto succedeva prima, quando lo studente seleziona un'opzione viene fatto una richiesta all'indirizzo *saveRadioChecked*; viene recuperata la matricola dello studente che andrà ad identificare il file corretto, viene recuperato il frammento Json in cui è definita la domanda e viene salvato nel campo “options” la risposta selezionata. Se lo studente decide di cambiare, il valore salvato viene sovrascritto con il nuovo valore.

Il Json restituito è visibile nel Listato 4.12.

```
1 {
2   "type": "radio",
3   "text": [
4     "Quanto fa cinque per quattro?"
5   ],
6   "options": "20"
7 }
```

Listato 4.12: Frammento Json domande a risposta multipla

4.6.3 Salvataggio domande con JSBin

Le domande che necessitano dell'uso di JSBin vengono gestite in una maniera un po' più articolata. In questo caso non è stata creata una nuova funzione ma è stato modificato un metodo già esistente che si occupa di salvare i dati.

Il metodo in questione è denominato **createRevision** ed è possibile trovarlo nella directory “*lib/handlers*” all'interno del file *bin.js* dell'applicativo JSBin. Generalmente ad ogni modifica effettuata i dati vengono salvati all'interno di un record creato nel database, in questo caso invece la nuova configurazione permette di salvare i dati all'interno

del file Json. Le modifiche apportate al modulo sono visibili nel Listato 4.13.

```
1 var Jsbinchecked = studentData.getJsbinChecked();
2 var urlFile = studentData.getUrlFileStudent();
3 var content = fs.readFileSync(urlFile, "utf8");
4 var obj = JSON.parse(content);
5
6 if(panel === "html"){
7     obj.question[Jsbinchecked - 1].html = params[panel].split("\n");
8 }
9 else if(panel === "css"){
10    obj.question[Jsbinchecked - 1].css = params[panel].split("\n");
11 }
12 else{
13    obj.question[Jsbinchecked - 1].javascript= params[panel].split("\n");
14 }
15
16 var json = JSON.stringify(obj, null, 2);
17 fs.openSync(urlFile, "w");
18 fs.appendFileSync(urlFile, json);
```

Listato 4.13: Salvataggio dati con JSBin

Nella pagina dello studente sono presenti i collegamenti a JSBin, ogni link è identificato da un id, ad esempio “*link3*” sta ad indicare che si tratta del collegamento presente nella domanda numero tre. La prima cosa che viene fatta nella funzione riportata è recuperare l’ultima parte di questo id, cioè solo il numero, tramite la funzione **getJsbinChecked()** definita nel modulo **studentData**, per poter modificare la domanda corretta; successivamente viene recuperato l’url del file del file da modificare e in particolare il suo contenuto.

JSBin utilizza un array “*params*” per salvare il contenuto del tre sezioni (HTML, CSS e JavaScript) fornite, mentre, “*panel*” contiene semplicemente il nome delle stesse. Ovviamente non possono essere modificate tutte nello stesso momento, quindi vengono distinti i tre casi separatamente. Se la sezione corrente è “html” allora in *params[panel]* ci sarà il valore scritto al suo interno e questo valore viene salvato nella corrispondente sezione

“html” del file Json. Analoghi sono gli altri due casi.

Per rendere definitive le modifiche si utilizza la funzione **stringify()** di Json, che converte i valori ricevuti in stringhe Json, le successive istruzioni aggiornano il contenuto del file. Il frammento Json restituito in questo caso è simile a quello raffigurato nel Listato 4.14.

```
1 {
2   "type": "jsbin",
3   "text": [
4     "Creare un documento HTML+CSS con grassetto e corsivo!"
5   ]
6   "html": [
7     "<p>",
8     "paragrafo",
9     "</p>"
10  ]
11  "css": [
12    "font-weight: bold;",
13    "font-style: italic;"
14  ]
15 }
```

Listato 4.14: Frammento Json domande con JSBin

4.6.4 Le prove non consegnate

Può capitare che lo studente decida di non consegnare la prova, quindi dal momento che è come consegnare in bianco non ha senso conservare il file vuoto. Quindi è stato configurato un array denominato “*examNotDelivery*”, che viene aggiornato ogni volta che uno studente si ritira dall’esame; per farlo si utilizza la funzione **push()**, definita nella funzione che aggiorna la tabella dinamica, raffigurata nel Listato 4.9.

Quando il server raggiunge lo stato di “*over*” significa che l’esame è definitivamente concluso. In quel momento viene richiamata la funzione `deleteExamNotDelivery()` che si occupa di cancellare i file degli studenti che non hanno consegnato. Il codice è raffigurato nel Listato 4.15.

```
1 var dirResultToday = './views/result/'+currentExam;
2 var list = fs.readdirSync(dirResultToday);
3 for (var i = 0; i < ExamNotDelivery.length; i++) {
4   list.forEach(function (file) {
5     var tmp = ExamNotDelivery[i] + ".json";
6     if (tmp === file){
7       var filePath = dirResultToday + "/" + file;
8       fs.unlink(filePath, function(err) {
9         if (err) {
10          console.log(JSON.stringify(err));
11        }
12      });
13    }
14  });
15 }
```

Listato 4.15: Funzione per cancellare le prove non consegnate

In pratica si fa un controllo incrociato tra tutti i file presenti nella cartella delle prove svolte e i file contenuti nell’array; appena si trovano due file che coincidono significa che quello presente nella cartella deve essere cancellato. Poiché nell’array sono contenuti i nomi del file è stato necessario aggiungere l’estensione “.json” per effettuare il controllo nel modo corretto; per l’eliminazione, invece, basta recuperare il path del file e passarlo alla funzione `unlink()`, contenuta nel modulo `fs` di JSBin;

4.7 Correzione

La sezione dedicata alla correzione è disponibile all’indirizzo `/teacher/correction` del server su cui è installato l’applicativo NPA-Exam. È accessibile dalla pagina di configurazione dedicata al docente e disponibile all’indirizzo `/teacher`.

L'interfaccia e le configurazioni interne rimangono le stesse di ExamBin poiché non sono ancora state aggiornate e rese compatibili con NPA-Exam. Però, verranno comunque analizzate nel dettaglio sia le funzionalità presenti sia quelle che dovranno essere aggiunte, mostrando nel primo caso qualche frammento di codice significativo e nel secondo elencando quali potrebbero essere le idee da sviluppare per gli aggiornamenti.

Quest'analisi verrà suddivisa principalmente in due sezioni. Una dedicata alla correzione dei dati elaborati con JSBin, che è la parte già configurata in ExamBin, e l'altra dedicata all'elaborazione dei dati delle domande a risposta aperta e multipla, che è la parte da aggiungere.

4.7.1 Elaborazione dati JSBin

Nella versione di ExamBin viene gestito unicamente il caso delle domande che necessitano della piattaforma JSBin per essere svolte. Come visibile in Figura 2.4, il docente ha a disposizione due form principali, che gli permettono di configurare le regole da testare durante la correzione, e uno più piccolo che gli permette di selezionare la prova da correggere e di dare il via al definitivo test.

Il form sulla destra gli permette di dare un peso ad ogni errore, il quale sarà poi determinante per la valutazione finale del compito. Ogni errore identificato viene moltiplicato per il peso ad esso attribuito e il valore ottenuto viene sottratto al voto di partenza.

Peso HTML standard = 0.5	errori HTML = 1
Peso CSS standard = 0.5	errori CSS = 2
Peso JavaScript standard = 0.5	errori JavaScript: 1
Peso CUSTOM standard = 2	errori CUSTOM = 0
Voto di partenza = 30	
Voto Proposto = $30 - (0.5 * 1 + 0.5 * 2 + 0.5 * 1 + 2 * 0) = 28$	

Figura 4.1: Esempio di generazione del voto

Sulla base di questi calcoli verrà generata una proposta di voto, come mostrato in Figura 4.1, la quale sarà presentata al docente una volta completata la correzione, all'interno di un file di report.

Il form sulla sinistra gli dà la possibilità di scegliere se fare una valutazione statica o dinamica della prova. Scegliendo la valutazione statica, non è necessario configurare nulla, ma vengono utilizzate delle regole assolute già presenti che vanno a testare la parte sintattica dell'esercizio senza eseguirlo. Con la valutazione dinamica, invece, è proprio il docente che deve caricare un file contenente una lista di test che andranno poi eseguiti su tutti i compiti consegnati. Nell'interfaccia questo tipo di gestione è identificato con il nome di "valutazione custom". Per l'implementazione di entrambe le funzionalità sono stati utilizzati diversi moduli e servizi che verranno analizzati separatamente nelle prossime due sezioni.

4.7.1.1 Validazione Statica

Per la realizzazione di questo servizio sono stati utilizzati tre validatori:

- W3C Markup Validation Service⁶ per testare il codice HTML.
- W3C CSS Validation Service⁷ per testare il codice CSS.
- module node Esprima⁸ per testare il codice JavaScript.

Utilizzando il modulo node **w3cvalidate**⁹ e il suo metodo *Validate* vengono analizzati frammenti di codice HTML e CSS, servendosi dei primi due servizi elencati. Il terzo, invece, è stato utilizzato per analizzare frammenti di codice JavaScript; si è scelto proprio il modulo **Esprima** poiché in seguito a dei confronti effettuati con servizi simili si è rivelato il più potente nell'identificare il maggior numero di errori.

I risultati ottenuti danno così la possibilità di comprendere se gli studenti hanno commesso degli errori sintattici e di identificare il punto esatto in cui si manifestano. Vista però la limitazione presentata da questo servizio, ExamBin, mette a disposizione anche un servizio di validazione dinamica, di seguito descritto.

⁶<http://validator.w3.org/>

⁷<http://jigsaw.w3.org/css-validator/>

⁸<https://www.npmjs.org/package/esprima>

⁹<https://www.npmjs.org/package/w3c-validate>

4.7.1.2 Validazione Dinamica

Per questo servizio sono stati utilizzati dei framework di testing già esistenti e disponibili sia lato client sia lato server. In particolare, vengono utilizzati quattro moduli node:

- **Cheerio:** per unificare le sezioni di codice (HTML, CSS e JavaScript).
- **Jsdom:** per generare il DOM (Document Object Model) lato server.
- **Mocha:** per eseguire una serie di test.
- **Chai:** per scrivere asserzioni complesse.

Il primo modulo ad essere utilizzato è chiaramente Cheerio che, come raffigurato nel Listato 4.16, mette insieme le tre sezioni precedentemente citate.

```
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/
5      jquery/1.11.0/jquery.min.js'></script>");
6  var html = $.html();
```

Listato 4.16: Unificazione codice con Cheerio

Per prima cosa viene salvato nella variabile “\$” il contenuto della sezione HTML di JSBin, recuperato dal record nel database. Successivamente nel campo “head”, in un blocco “style”, viene inserito il contenuto della sezione CSS e rispettivamente vengono inseriti nel campo “html”, in un blocco “script”, il contenuto della sezione JavaScript e un riferimento ad una libreria JQuery. L’ultima istruzione permette di salvare il tutto in un’unica variabile, richiamando il metodo “html()” fornito da Cheerio sull’oggetto “\$”.

La limitazione più evidente presentata da questo modulo è quella di non consentire di effettuare dei test sui dati recuperati e per risolvere questo problema viene utilizzato il modulo Jsdom, il quale, tuttavia, ha bisogno di ricevere in input tutto il codice della

pagina HTML prima di fare i test; per questo motivo è stato introdotto l'uso di Cheerio piuttosto che utilizzare direttamente Jsdom per fare entrambe le cose. Un esempio di come vengono gestiti i dati con Jsdom è raffigurato nel Listato 4.17.

```
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 4.17: Creazione DOM

Una volta preso in input il giusto valore, sfruttando un array di features, viene generato il DOM lato server e salvato in una variabile windows. Inoltre, viene aggiunto anche un listener sull'evento "load" in attesa che tutti gli script vengano correttamente caricati. Fatto questo è possibile effettuare i veri e propri test, per farlo si utilizzano gli ultimi due moduli citati, mocha e chai.

Mocha¹⁰ è un framework di Javascript che permette di eseguire dei test asincroni su pagine web e restituisce un report ben definito con tutte le informazioni relative ai dati analizzati.

Chai¹¹ è una libreria di asserzioni compatibile con qualsiasi framework simile a Mocha. Per asserzioni si intendono delle espressioni anche molto complesse scritte per effettuare dei test efficaci sia a livello sintattico sia a livello semantico.

Nel Listato 4.18 è possibile osservare una test suite, cioè un insieme di regole, configurata per correggere un esame sulla piattaforma ExamBin.

¹⁰<http://mocha.js.org>

¹¹<http://chaijs.com>

```
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 4.18: Test Mocha Chai in ExamBin

Il metodo **describe** definisce l'insieme dei test che si possono effettuare, il primo parametro è il nome che si vuole attribuire a questo insieme e il secondo è la funzione che conterrà le definizioni dei vari test; i quali, sono a loro volta definiti dal metodo **it**, dove il primo parametro rappresenta il nome del test e il secondo parametro è la funzione che lo definisce.

All'interno di ogni metodo **it** vengono definite le asserzioni che consentono di esaminare un certo oggetto. Ad esempio, nel primo test mostrato nel frammento di codice ci si aspetta che la variabile `i` contenga il valore 10, mentre, il secondo test effettua un controllo su i tag "class" presenti nel documento HTML e ci si aspetta che il numero di tag trovati sia pari a 0, ciò significa che non possono essere usati.

Per permettere la riuscita di questi test è stato configurato un modulo, chiamato "test-Config", all'interno del quale sono definite varie regole che permettono al framework mocha di eseguire i test prendendo in input i giusti parametri. Queste configurazioni vengono caricate automaticamente una volta che il docente ha caricato il file di valutazione personalizzato.

Una volta effettuati i test i risultati verranno caricati in una tabella, visibile sempre in Figura 2.4, che riassume il tutto. Inoltre, è possibile anche visualizzare gli stessi risultati in formato Json, cliccando sul tasto "Vedi report in Json" che comparirà nella stessa schermata al termine della valutazione complessiva.

Per rendere compatibile questa gestione con NPA-Exam basta semplicemente modifica-

re il punto in cui vengono recuperati i dati, cioè non devono essere cercati nel record creato all'interno nel database ma nel file Json configurato appositamente per salvare gli elaborati degli studenti.

4.7.2 Elaborazione dati domande a risposta aperta e multipla

La gestione di queste due tipologie di domande non è ancora stata implementata nella parte che riguarda la valutazione automatica della prova d'esame. D'altra parte per quanto riguarda le domande a risposta aperta non è pensabile la configurazione di regole automatiche per la correzione; questo perché essendo domande con testo libero potrebbero avere contenuti diversi ma avere tutte lo stesso significato.

Ad esempio, ad una domanda del tipo “Di che colore è questo paragrafo?” si potrebbe rispondere in due modi diversi ma entrambi corretti, come: “Il paragrafo è di colore rosso” o “È rosso”. Quindi per realizzare un correttore automatico per questa tipologia di domanda si dovrebbero conoscere a prescindere tutte le possibili risposte, per poter poi decidere quali valutare positivamente oppure no; in questo caso specifico è consigliabile un metodo di valutazione differente da quello già proposto ed implementato in ExamBin.

Un ragionamento simile si può fare per le domande a risposta multipla, il sistema di valutazione già presente non è necessario in quanto nel file Json configurato dal docente per la prova d'esame questa tipologia viene definita in modo particolare; cioè la prima opzione inserita è quella giusta, quindi ciò che dovrebbe fare il validatore automatico è confrontare la risposta salvata nel file dello studente con quella presente nel file del docente, se coincidono la risposta viene dichiarata corretta altrimenti no.

Conclusioni

In questo scritto è stato presentato l'applicativo NPA-Exam, le sue caratteristiche e i dettagli implementativi più rilevanti. NPA-Exam è stato progettato per risolvere i problemi legati agli esami di programmazione di tecnologie web.

Come è stato descritto offre la possibilità agli studenti di lavorare in un ambiente confortevole e veloce, dove poter scrivere e testare il proprio codice, leggere e rispondere alle domande fornite. D'altra parte offre la possibilità al docente di controllare pienamente la prova d'esame, avere una chiara visione degli studenti che vi partecipano e di quelli che consegnano o si ritirano.

Ciò che lo rende unico è il fatto di mettere a disposizione delle funzionalità che nessun altro applicativo presenta fin ad oggi. Infatti, anche la sua vecchia versione, ExamBin, presentava delle lacune, come l'impossibilità di configurare più di una domanda per la prova d'esame e svolgere più prove durante la giornata.

NPA-Exam risolve questi problemi e aggiunge nuove funzionalità come la possibilità di inserire anche domande a risposta aperta e multipla e di tenere traccia degli studenti che si registrano al sistema, ai quali, viene consentito di ritirarsi dalla prova in qualsiasi momento. Inoltre, ottimizza alcune delle funzioni di ExamBin, come il recupero dei dati degli studenti che hanno concluso l'esame, grazie all'introduzione di una tabella dinamica che velocizza l'aggiornamento delle informazioni.

L'applicativo è completamente open-source ed è disponibile al mio indirizzo github: <https://github.com/fgrillo21/NPA-Exam>. Quindi chiunque può scaricarlo ed utilizzarlo come base per un nuovo progetto.

Tuttavia, non risulta ancora completo; negli sviluppi futuri si ripropone di aggiornare la configurazione riguardante la valutazione automatica dei compiti, implementata nella versione di ExamBin, ed adattarla alle nuove funzionalità. In aggiunta l'obiettivo principale è sempre quello di migliorare costantemente l'applicativo, tenerlo aggiornato e magari fare in modo che si possa utilizzare per riconoscere linguaggi diversi da HTML, CSS e JavaScript, per poterlo sfruttare anche in esami diversi.

In conclusione si può affermare che l'obiettivo principale è stato raggiunto, cioè con NPA-Exam è possibile realizzare esami di tecnologie web al computer, senza la necessità di usare carta e penna.

Bibliografia

- [ATEPQ08] Anthony Allevato, Matthew Thornton, Stephen H. Edwards, Manuel A. Pérez-Quiñones, “Mining Data from an Automated Grading and Testing System by Adding”, Educational Data Mining 2008, the 1st international conference, Montreal, Canada, June 20-21, 2008
- [BOR14] Matteo Borghi, Un’applicazione dei web playground all’insegnamento dell’informatica, p. 1-73, Bologna, 16 Dicembre, 2014.
- [CTA12] Chris Coyier, Tim Sabat, Alex Vasquez, “CodePen”, <http://codepen.io/>, 2012
- [FPQTL08] Xiang Fu, Boris Peltserger, Kai Qian, Lixin Tao, Jigang Liu, “APOGEE: automated project grading and instant feedback system for web based computing”, proceedings of the 39th SIGCSE Technical Symposium on computer science education, Portland, OR, USA, March 12-15, 2008
- [JAC00] Norman Jacobson, “Using on-computer exams to ensure beginning students’ programming competency”, SIGCSE Bulletin (Association for Computing Machinery, Special Interest Group on Computer Science Education) 32(4), 53-56 (2000)
- [JLNT05] Jonsson T, Loghmani P, Nadjm-Tehrani S, “Evaluation of an authentic examination system (AES) for programming courses”, http://www.hgur.se/activities/projects/financed_projects/i-j/jonsson_torbjorn_00_slutrapport.pdf, Swedes, August 12, 2005

- [KUM10] Amruth N. Kumar, “Closed labs in computer science I revisited in the context of online testing”, proceedings of the 41st ACM technical symposium on computer science education, Milwaukee, WI, USA, March 10-13, 2010
- [SET12] Amit Set, “Liveweave”, <http://liveweave.com>, 2012
- [SHA08] Remy Sharp, “JSBin”, <http://jsbin.com>, 2008
- [SQF08] Mate’ Sztipanovits, Kai Qian, Xiang Fu, “The automated web application testing (AWAT) system”, proceeding ACM of the 46th Southeast Regional Conference 2008: 88-93, New York, USA, 2008
- [TEF08] The Eclipse Foundation, “BIRT Project: Business Intelligence and Reporting Tools”, Available at: <http://www.eclipse.org/birt/>, 2008
- [TFH04] Thomas D, Fowler C, Hunt A, “Programming Ruby: The Pragmatic Programmers’ Guide”, ISBN-10: 0974514055, Pragmatic Bookshelf, 2004.
- [OP09] Piotr Zalewa, Oscar Krawczyk, “JSFiddle”, <http://jsfiddle.net/>, January 2009

Ringraziamenti

Ringrazio il mio relatore, Fabio Vitali, per essersi fidato di me ed avermi affidato la realizzazione di questo progetto, nonché per avermi dato i giusti consigli e per la disponibilità dimostrata.

Ringrazio tutta la mia famiglia. In particolare, mia sorella, che mi ha sempre incoraggiata e per aver letto la mia tesi, anche se quasi sicuramente non ci ha capito niente, e i miei genitori, che mi hanno sostenuta economicamente e soprattutto moralmente nonostante le difficoltà, impedendomi di mollare.

Ringrazio Flavia Delli e Melissa Angelini, che sono state le prime persone che ho conosciuto e nonostante tutti i problemi che si sono presentati in questi anni sono riuscite a capirmi e ad aiutarmi quando ne ho avuto bisogno.

Ringrazio Matteo Borghi, che ha avuto la pazienza di sopportarmi in questi mesi di duro lavoro e mi ha aiutata dandomi dei consigli, quando non sapevo dove sbattere la testa, pur avendo altri impegni.

Ringrazio Francesca Sarti e Giulia Baccolini, per tutti i bei momenti trascorsi, per le pazze giornate in laboratorio dove passavamo più tempo a scherzare e urlare che a studiare, per le figuracce fatte in giro per Bologna e per quelle che verranno. Non ci conosciamo da molto ma il futuro promette bene.

Ringrazio Mary Hachfeld per il sostegno che mi ha dato dall'inizio, per i pranzi, per le cene e un po' meno per i caffè, anche se col tempo sono migliorati. Molta gente ha provato a metterci i bastoni tra le ruote ma non siamo state noi a cadere.

Ringrazio il mio cuoco preferito, Mirko Mucaria, che mi ha impedito di morire di fame, soprattutto negli ultimi mesi. Per le serate organizzate all'ultimo minuto, per la giusta dose di insulti (giornalieri) e per i consigli, anche se a volte non so come mai ma

“non mi risponde”.

Ringrazio l'ultima persona, ma non certo per importanza, la mia migliore amica, Giulia Naponiello, che è senza dubbio la persona che mi sopporta di più da quasi 4 anni ormai. La ringrazio prima di tutto per aver trovato il tempo di leggere la mia tesi, per esserci stata nei momenti belli ma soprattutto in quelli brutti, per la forza e il coraggio che mi ha sempre dato, in particolare negli ultimi mesi quando avevo perso le speranze e credevo di non farcela. Cosa più importante grazie per non avermi tolto il saluto sentendo le mie freddure quasi giornaliere da #maiunagioia.

Ringrazio tutti quelli che non hanno mai creduto in me e che hanno fatto di tutto per ostacolarmi, farmi passare per la persona che non sono e mettermi contro la gente. Grazie perché mi avete fatto capire che ho degli amici meravigliosi e questo traguardo lo dedico anche a voi rosiconi.

Mi scuso se qualcuno sperava di essere nominato e non l'ho fatto ma non posso fare 50 pagine di ringraziamenti, vi ringrazio lo stesso tutti. Un saluto speciale al club delle CCI che porta avanti il suo nome con orgoglio.

Francesca Grillo