

Capitolo 3

Architettura generale

Il sistema Programmazione 2 Automatic Corrector, mostrato in figura 3.1, è composto da due parti:

- Il plugin Moodle, denominato **Prog 2 Automatic Corrector**, che estende le modalità di consegna per un'attività di tipo assignment aggiungendo le seguenti funzionalità:
 - Il caricamento dei file per la correzione dell'assignment, *correctionFiles*, da parte del docente;
 - Il caricamento dei file da correggere, *taskFiles*, e la ricezione di un feedback relativa alla consegna per uno studente.
- Il web service **Programmazione2AutomaticCorrector-backend** che si occupa della gestione dei *correctionFile* e dei *taskFile* e della loro elaborazione volta alla restituzione di un feedback riguardante la consegna di uno studente.

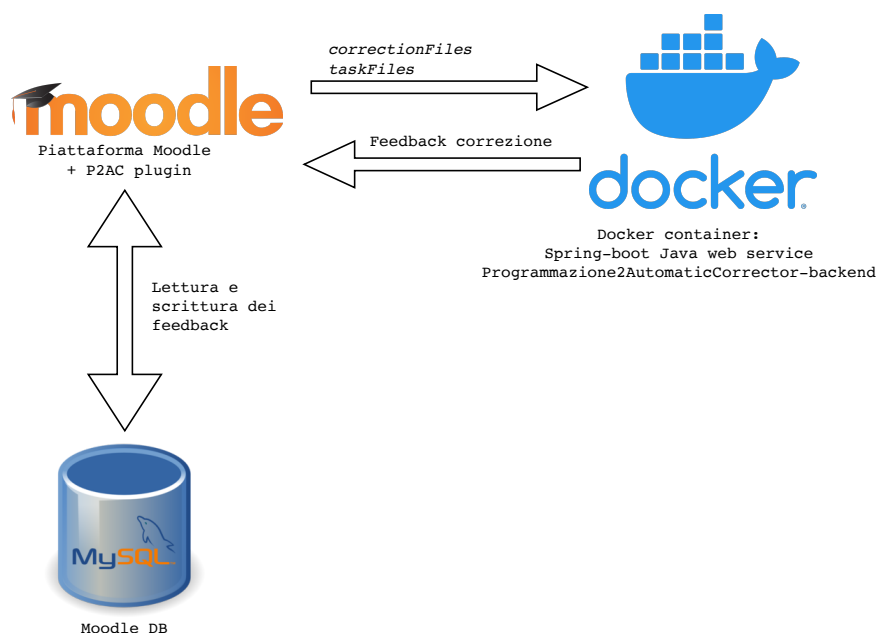


Figura 3.1: Architettura sistema

Da qui in seguito per chiarezza, i file di correzione caricati da un docente verranno rappresentati dal termine *correctionFiles* mentre i file caricati da uno studente dal termine *taskFiles*.

3.1 Casi d'uso

In questa sezione verranno introdotte e rappresentate, attraverso diagrammi d'attività, le varie operazioni che utenti, di tipo *docente* e *studente*, possono effettuare grazie al plugin **Programmazione 2 Automatic Corrector**. Le operazioni di default, messe a disposizione da parte di Moodle, non verranno trattate in quanto distrarrebbero il lettore dal tema principale.

3.1.1 Casi d'uso : Docente

Le operazioni principali che effettua il docente sono quelle di creare un assignment e di visualizzarne i risultati. Il plugin aggiunge, alle impostazioni presenti di default durante la creazione di un assignment, la modalità di consegna **Prog 2 Automatic Corrector**. La nuova modalità di consegna richiede, da parte del docente, il caricamento dei file necessaria alla correzione. La visualizzazione dei risultati di un assignment avviene tramite la sezione *grading table*.

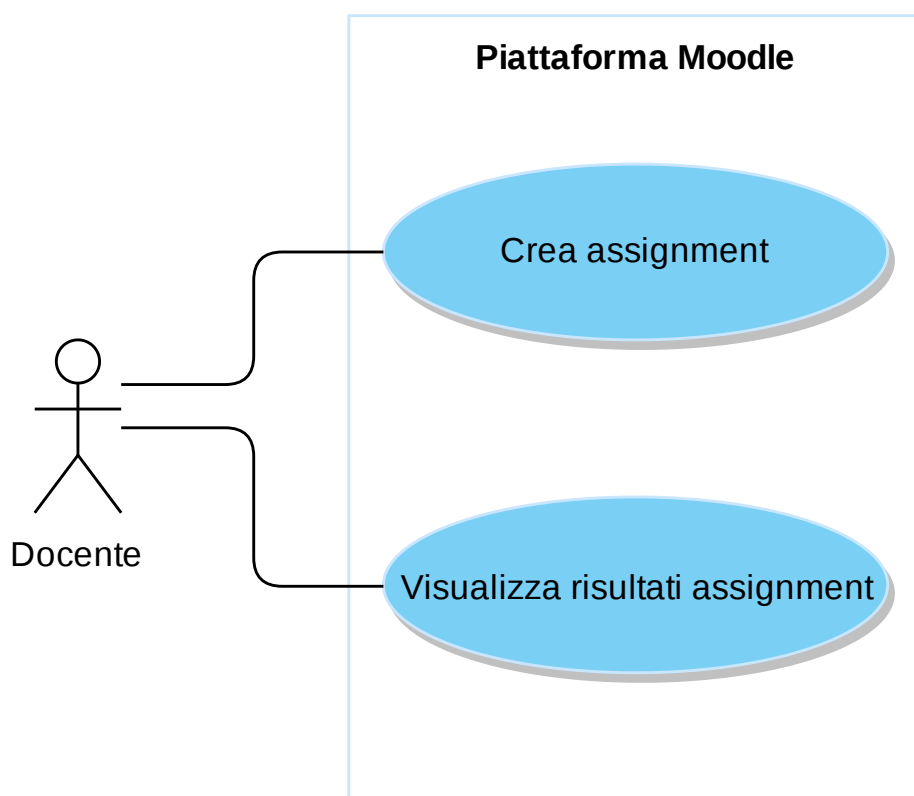


Figura 3.2: Casi d'uso docente

3.1.1.1 Docente - Creazione assignment

Un docente crea un assignment di tipo **Prog 2 Automatic Corrector** e carica, nell'ambiente predisposto alla consegna, i file necessari alla correzione.

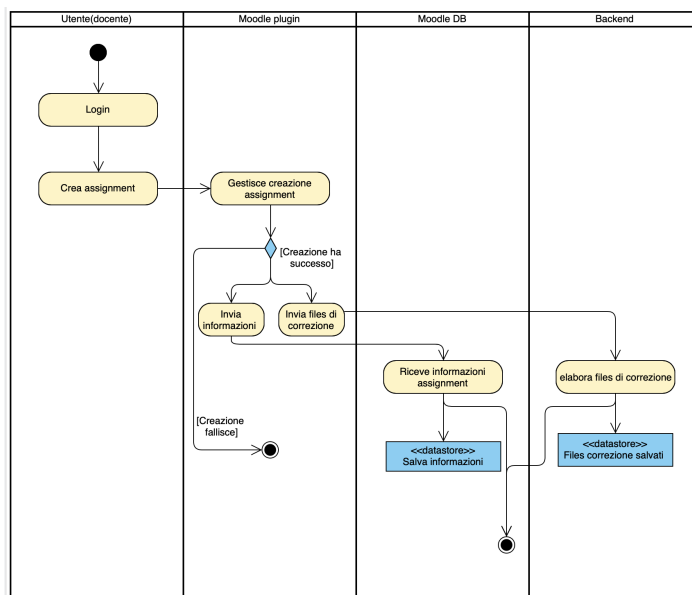


Figura 3.3: Diagramma attività creazione assignment

3.1.1.2 Docente - Visualizzazione risultati assignment

Un docente, dalla schermata riepilogativa di un assignment, può visualizzare tutte le consegne dell'assignment e i relativi feedback.

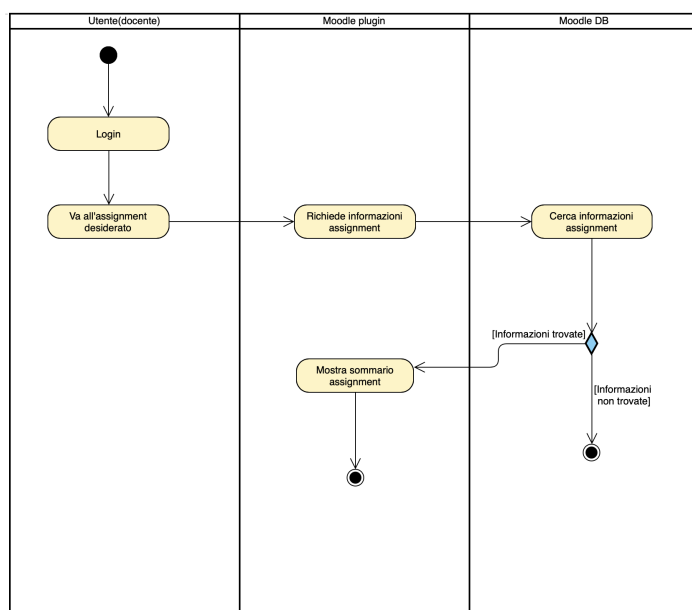


Figura 3.4: Diagramma attività visualizzazione assignment

3.1.2 Casi d'uso : Studente

Uno studente può consegnare un'assignment, creato precedentemente da un docente, e ricevere un feedback immediato riguardo la sua consegna (se la modalità di consegna impostata è *Prog 2 Automatic Corrector*). Dopo aver consegnato un'assignment lo studente può visualizzare il feedback, nella schermata di consegna dell'assignment stesso, finché la consegna non viene rimossa e nel caso venisse modificata visualizzerà il feedback relativo alla nuova consegna.

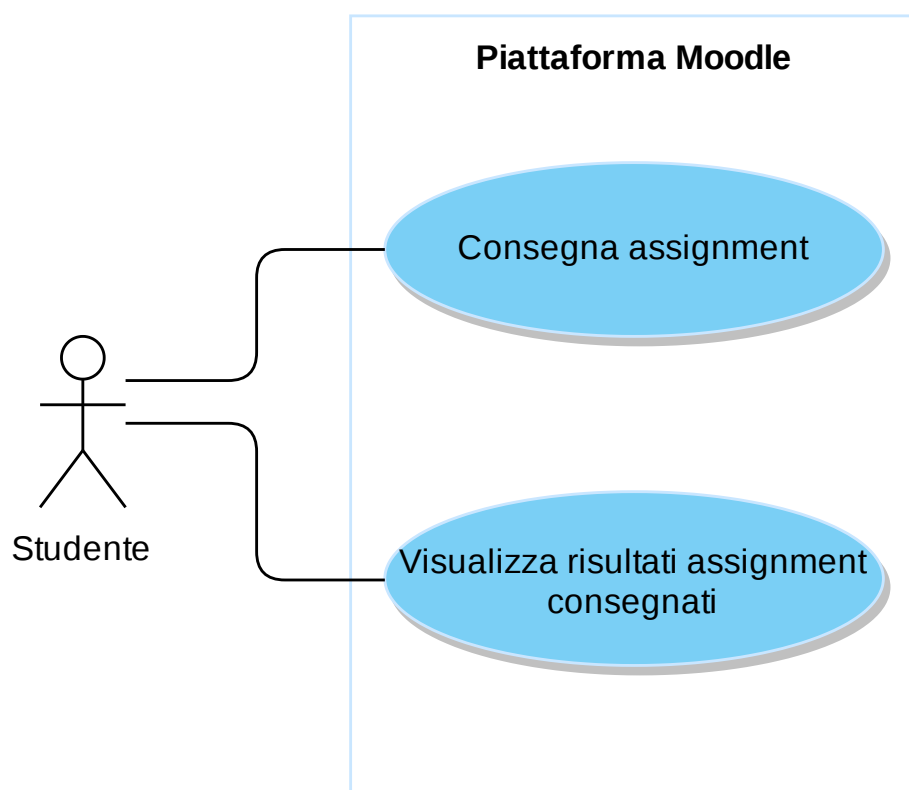


Figura 3.5: Casi d'uso studente

3.1.2.1 Studente - Consegna assignment

Uno studente, dall'apposita schermata, nell'attività assignment precedentemente creata può consegnare la sua soluzione per l'assignment e ricevere un feedback immediato riguardo la sua consegna.

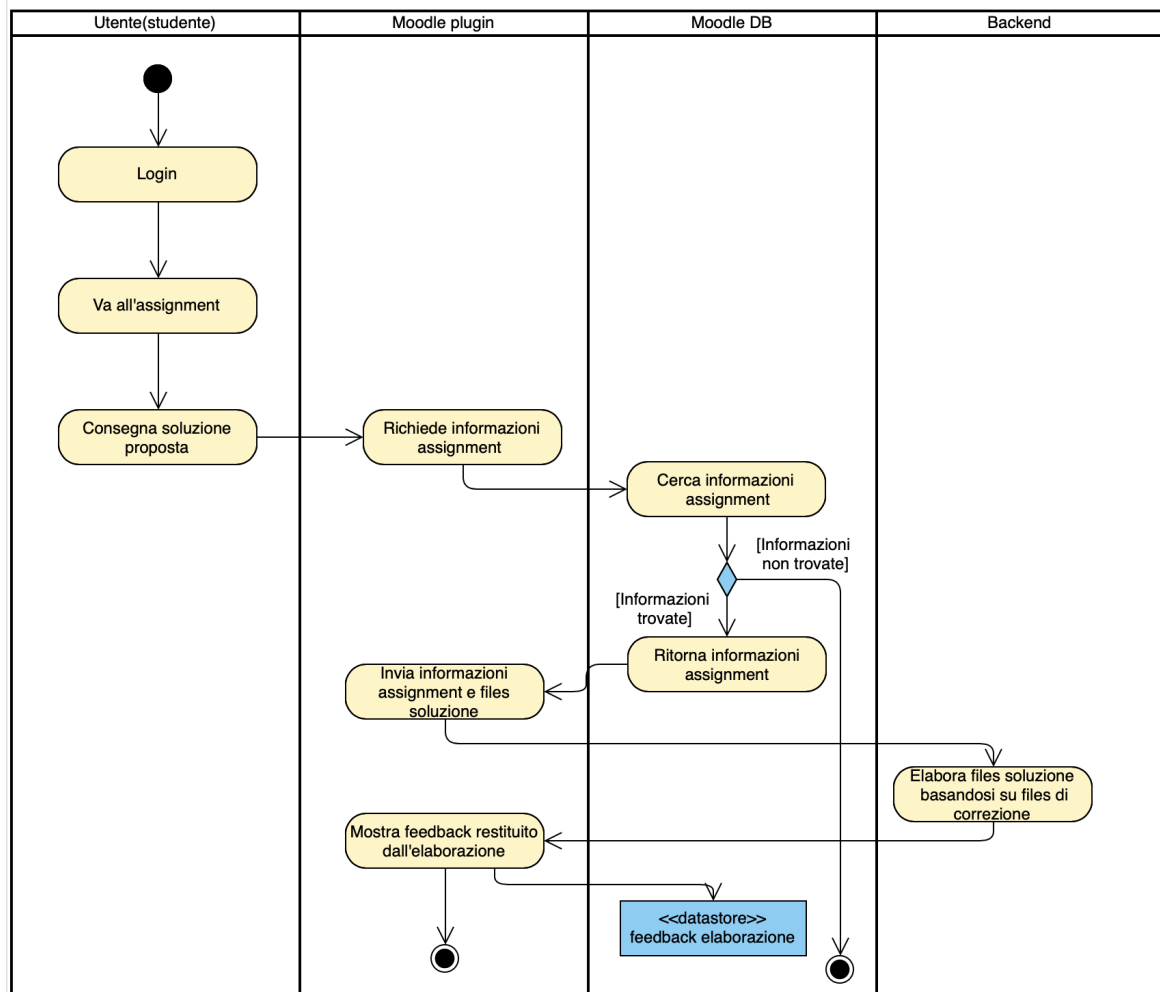


Figura 3.6: Diagramma attività consegna assignment

3.1.2.2 Studente - Visualizzazione feedback assignment

Uno studente può visualizzare il feedback di un assignment andando nella schermata dell'assignment della quale si desidera vedere il feedback, quest'ultimo sarà presente solo se lo studente ha consegnato l'assignment e non lo ha rimosso.

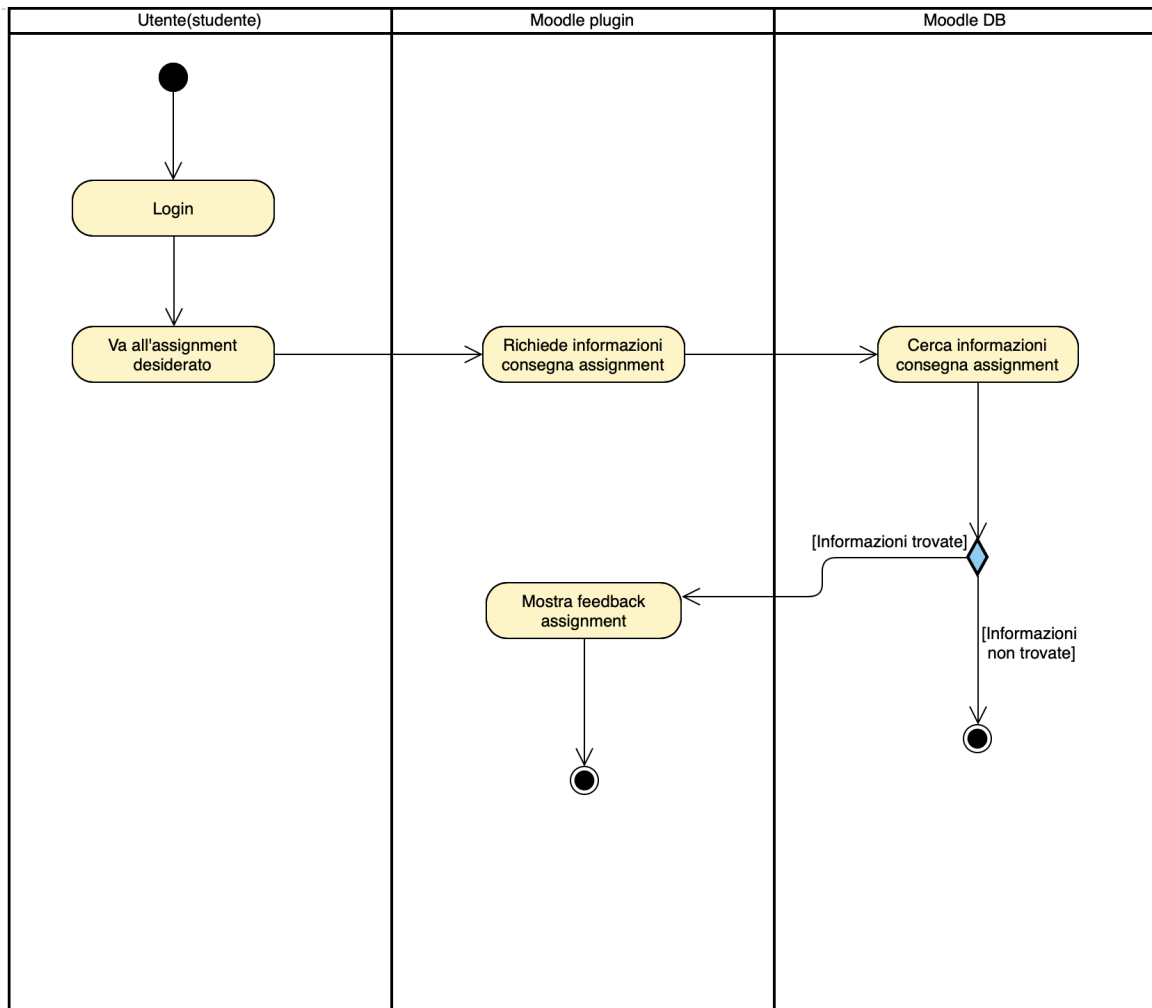


Figura 3.7: Diagramma attività visualizzazione feedback

Capitolo 4

Implementazione

Prima di procedere con l'implementazione del backend, è stato necessario studiare il framework Spring, in particolare Spring MVC^[5] per lo sviluppo di applicazioni web. La pagina ufficiale di Spring mette a disposizione molte guide utili all'apprendimento del framework stesso, in particolare è stata molto utile la guida^[6] per conoscere e testare, attraverso lo sviluppo di web service di test, le funzionalità di un web service sviluppato attraverso Spring MVC.

Lo sviluppo del frontend è avvenuto dopo uno studio della documentazione per sviluppatori fornita da Moodle^[7]. E' stata studiata più nel dettaglio la documentazione relativa allo sviluppo di un plugin del tipo '*Assignment submission*'^[8].

4.1 Backend

Il backend è stato scritto in Java con l'ausilio del framework Spring MVC, il suo scopo è quello di gestire le richieste provenienti dalla piattaforma Moodle, nello specifico dal plugin **Proge 2 Automatic Corrector**.

Come detto in precedenza, Spring MVC si basa sul pattern Model View Controller, dunque i componenti principali del backend sono appunto i controller e il model (la view è stata implementata nel plugin Moodle in quanto la visualizzazione del feedback è compito del plugin).

E' stato scelto l'approccio RESTful per il web service cosicché il plugin Moodle possa comunicare col backend tramite i metodi HTTP, in particolare il metodo HTTP **Post** il quale permette al plugin di inviare i files caricati dal docente e dagli studenti. In seguito, nella sezione inerente allo sviluppo del plugin, verrà mostrato in che modo avviene la comunicazione lato plugin.

Il formato dei dati scelto per la comunicazione è JSON: dopo aver elaborato i files in ingresso, il backend restituirà una risposta, il feedback per lo studente, in formato JSON al plugin Moodle.

Le funzioni principali del backend sono essenzialmente:

- La ricezione e la gestione dei *correctionFiles* e dei *taskFiles*.
- L'elaborazione dei *taskFiles* e la restituzione di un feedback.

La ricezione e la gestione dei file avviene tramite due Controller, uno che si occupa delle richieste del docente e uno che si occupa di quelle provenienti dagli studenti.

Dato che si tratta di un web service RESTful, i controller vengono raggiunti tramite il loro endpoint. I REST endpoints offerti dal web service sono mostrati nella seguente tabella 4.1:

Metodo	URL	Parametri richiesti
POST	upload/teacher	inputFile e assignmentID
DELETE	delete/teacher	assignmentID
POST	upload/student	inputFile, assignmentID e studentID

Tabella 4.1: Endpoints

Di seguito, in figura 4.1, una panoramica delle classi che compongono il backend, le quali verranno analizzate, in maniera più accurata, nelle sezioni successive.

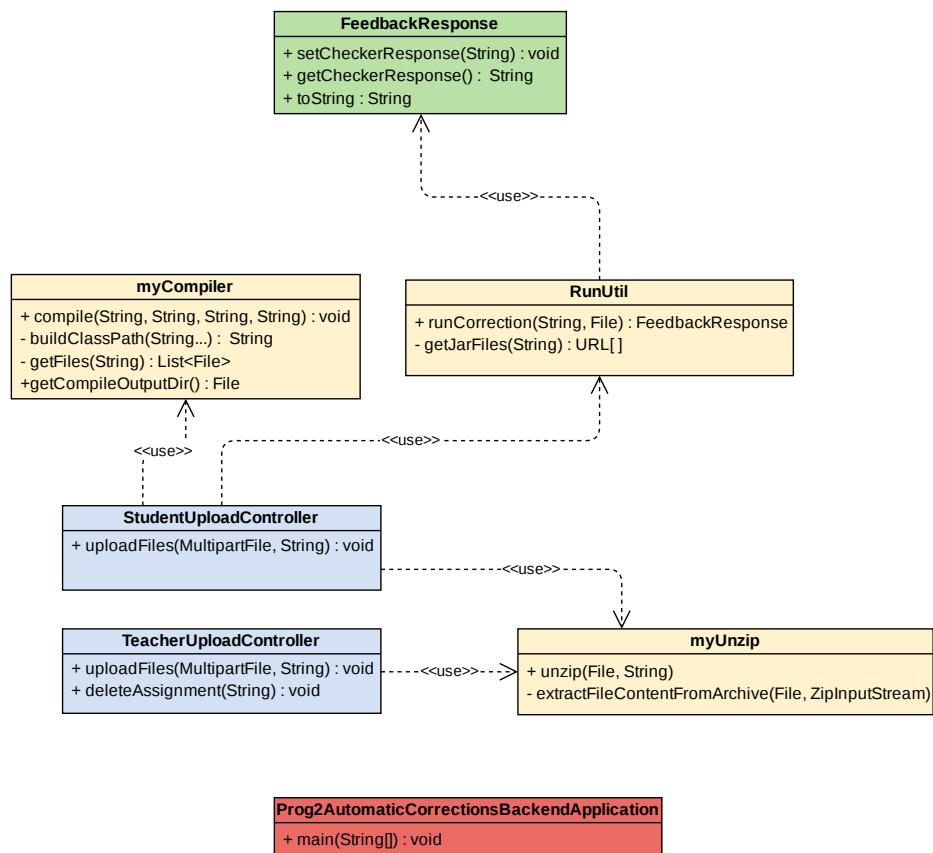


Figura 4.1: UML Classi Backend

4.1.1 I Controller

4.1.1.1 Docente

`TeacherUploadController` è il controller delegato alla gestione delle richieste del docente, la classe presenta due metodi principali.

Il primo metodo analizzato è `uploadFiles`:

```

1  @RequestMapping(value = "/upload/teacher" ,method = RequestMethod.POST)
2  public void uploadFiles(@RequestParam("file") MultipartFile inputFile ,
3                          @RequestParam("assignmentID") String assignmentID)

```

il quale si occupa della gestione delle richieste provenienti dal plugin `Prog 2 Automatic Corrector` al momento della creazione di un nuovo assignment; richiede due parametri:

- `inputFile`: indica i *correction files*, in un file zip, caricati dal docente, durante la creazione dell'assignment;
- `assignmentID`: valore univoco che serve a rappresentare l'assignment appena creato, viene generato da Moodle.

Alla ricezione del file .zip dal plugin, il controller, con il supporto della classe `myUnzip`, decompresse e salva tutti i *correction files* caricati dal docente nella directory, creata dal controller, corrispondente all'assignment appena creato dal docente tramite il plugin Moodle.

Il comportamento del controller è rappresentato dal diagramma di sequenza nella figura 4.2.

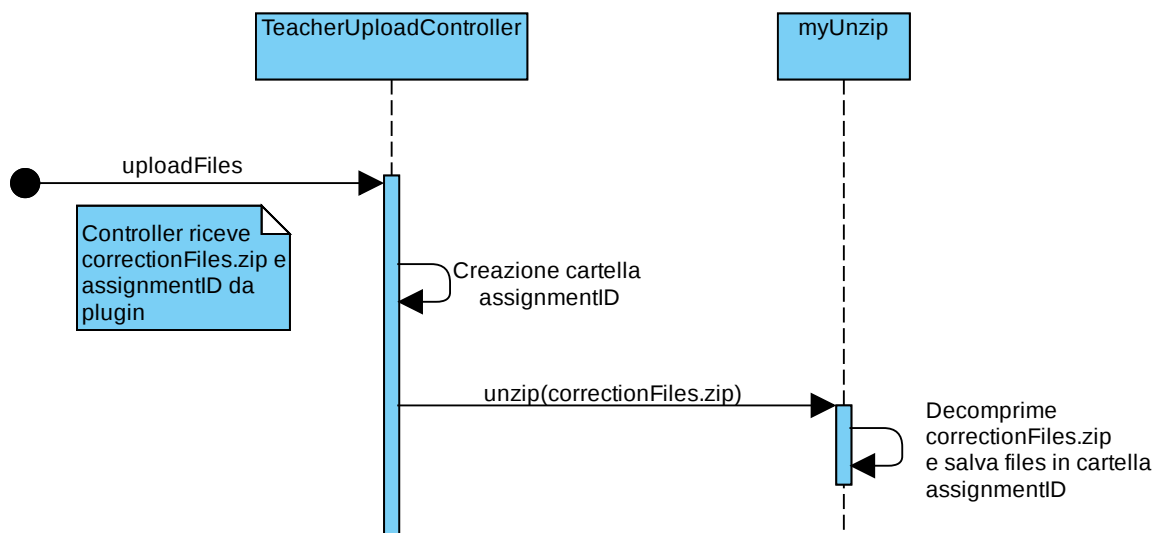


Figura 4.2: Diagramma sequenza docente creazione assignment

Il metodo `deleteAssignment`:

```
1 @RequestMapping(value = "/delete/teacher", method = RequestMethod.DELETE)
2 public void deleteAssignment(@RequestParam("assignmentID") String assignmentID)
```

viene chiamato quando un docente elimina, dalla piattaforma Moodle, un assignment avente come tipo di consegna `Prog 2 Automatic Corrector`, in modo da garantire consistenza eliminando dal backend la cartella corrispondente all'assignment. L'unico parametro richiesto è `assignmentID`, valore univoco che rappresenta l'assignment da eliminare.

4.1.1.2 Studente

Il controller designato a gestire le richieste degli studenti è `StudentUploadController` e presenta un solo metodo.

Il metodo `UploadFiles`:

```
1 @RequestMapping(value = "/upload/student", method = RequestMethod.POST)
2 public FeedbackResponse uploadFiles(@RequestParam("file") MultipartFile inputFile,
3                                     @RequestParam("assignmentID") String assignmentID,
4                                     @RequestParam("studentID") String studentID)
```

in primis è responsabile della gestione del file .zip, ricevuto dal plugin al momento della consegna di un assignment da parte di uno studente.

I parametri richiesti sono:

- `inputFile`: file .zip contenente la consegna dello studente per l'assignment;
- `assignmentID`: valore univoco rappresentate l'assignment alla quale si riferisce la consegna, generato da Moodle;
- `studentID`: ID che rappresenta in modo univoco uno studente, fornito da Moodle; è usato per garantire univocità alla correzione.

Con l'ausilio della classe `myUnzip` vengono decompressi tutti i file nella cartella dell'assignment corrispondente all'`assignmentID` ricevuto.

La classe `myCompiler`, dopo la decompressione dei file, si occupa della compilazione dei *correction files* e dei *task files*.

Grazie alla fase di compilazione i file sono pronti alla correzione; la quale avviene grazie alla classe `RunUtil` che utilizza le API del software `Prog2AutomaticCorrector`, sviluppato dal prof. Denaro, per correggere i files consegnati dallo studente.

La correzione genera un feedback che da un indicazione volta al miglioramento del progetto consegnato, nel caso ce ne sia bisogno, altrimenti un feedback positivo riguardo il progetto consegnato.

Il comportamento appena descritto è rappresentato dal diagramma di sequenza riportato dalla figura 4.3.

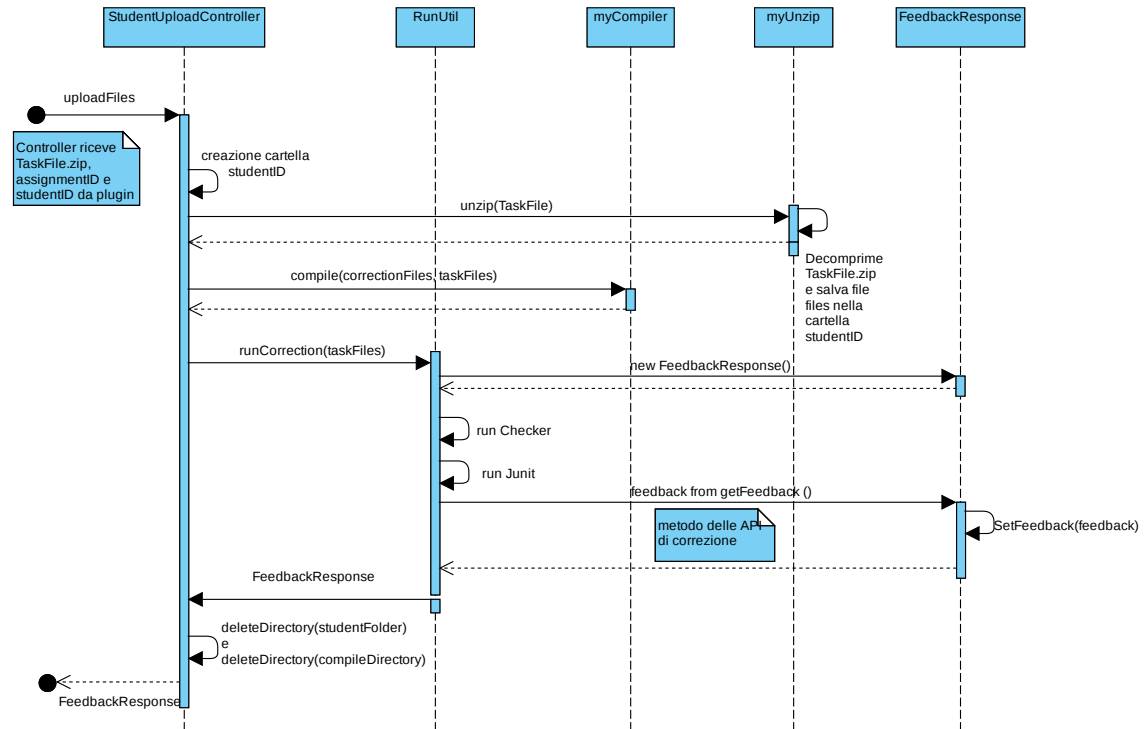


Figura 4.3: Diagramma sequenza studente consegna assignment

4.1.2 Classi di supporto

In questa sezione verrà fornita una breve descrizione delle classi di supporto usate dai Controller per la decompressione, la compilazione e la correzione dei file.

4.1.2.1 myUnzip

La classe `unZip` decompri i file `.zip` provenienti dall'assignment e in aggiunta li salva nel cartella indicata dal Controller.

4.1.2.2 myCompiler

La classe `myCompiler` compila i *correction files* e i *task files* tramite l'API `JavaCompiler`. La funzione `buildClassPath()` è molto importante in quanto permette di creare un *classpath* personalizzato per il *JavaCompiler* delegato alla compilazione dei file. La creazione di un *classpath* personalizzato è necessaria poiché i *correction files* hanno bisogno delle dipendenze `JUnit.jar` e `Prog2AutomaticCorrector.jar` per essere compilati correttamente.

4.1.2.3 RunUtil

`RunUtil` è la classe delegata alla correzione dei *task files*. Per farsi che il processo di correzione avvenga correttamente è necessario configurare un `ClassLoader` personalizzato per la correzione che abbia le dipendenze alle API `Programmazione2AutomaticCorrector` e che carichi i *task files* e i *correction files* compilati. La correzione effettiva, dei *task files*, avviene per mezzo dei *correction files*, i quali implementano le API `Programmazione2AutomaticCorrector`.

4.1.3 Configurazione backend

Il backend utilizza il file `application.properties` che non è altro che un semplice file di testo, contenente coppie di chiave-valori, per configurare le proprietà dell'applicazione. Come mostrato nel listato 4.1, il file *application.properties* è usato per definire i prefissi delle cartelle usate per il salvataggio dei file ricevuti e dei file compilati. Oltre a definire i prefissi delle cartelle, definisce il percorso delle librerie necessarie alla compilazione e alla correzione dei file.

```

1  #Path directory gestione file ricevuti
2  upload.dir.parent=/tmp
3  upload.dir.assignment=assignment_
4  upload.dir.test=correctionFiles
5  upload.dir.student=consegna_
6  upload.dir.compiled=bin_
7
8  #Path directory con librerie
9  compile.dir.lib=./src/main/resources/compilelib/
10 run.dir.lib=./src/main/resources/runlib

```

Listing 4.1: File di configurazione `application.properties`

Le proprietà descritte nel file `application.properties` combinate con l'annotazione `@value` permettono di sfruttare la *dependency injection* per 'iniettare' i valori delle proprietà nelle variabili.

Il listato 4.2 mostra come viene 'iniettato' il valore della proprietà `upload.dir.parent` nella variabile `parentDir`.

```
1 @Value("${upload.dir.parent}")
2 private String parentDir;
```

Listing 4.2: Esempio utilizzo proprietà

4.1.4 Deployment

Ho deciso di usare docker per la fase di *deployment* dell'applicazione poiché garantisce il funzionamento correttamente dell'applicazione indipendentemente dal sistema sottostante, rendendola di fatto indipendente dall'hardware. La creazione del immagine del docker, trattato nella sezione(2.4), avviene per mezzo del *Dockerfile* mostrato nel listato 4.3 in seguito:

```
1 #base image containing Java runtime
2 FROM openjdk:8-jdk-alpine
3
4 # Add a volume pointing to /tmp
5 VOLUME /tmp
6
7 # The application's jar file
8 ARG JAR_FILE=target/Prog2AutomaticCorrections-1.1.0-SNAPSHOT.jar
9
10 #Add the application's jar to the container
11 ADD ${JAR_FILE} p2ac.jar
12
13 # Prepare environment.
14 # Create needed folders
15 RUN mkdir /home/p2ac && \
16     mkdir /home/p2ac/libs && \
17     mkdir /home/p2ac/libs/compilelib && \
18     mkdir /home/p2ac/libs/runlib
19
20 # Add needed libs for compile and run
21 ADD ./src/main/resources/compilelib /home/p2ac/libs/compilelib
22 ADD ./src/main/resources/runlib /home/p2ac/libs/runlib
23
24 RUN sh -c 'touch /p2ac.jar'
25
26 # Run the jar file
27 ENTRYPOINT ["java", "-Djava.security.egd=file:/dev/./urandom", "-Dspring.profiles.active=server", "-jar", "/p2ac.jar"]
```

Listing 4.3: Dockerfile per backend

Di seguito verrà analizzato il listato 4.3 per capirne il funzionamento:

- La prima istruzione `FROM` serve per scegliere un immagine di base; in questo caso è stata scelta l'immagine `openjdk:8-jdk-alpine` poiché è un immagine molto leggera di openJDK8.
- Con l'istruzione `VOLUME` viene mappato il percorso `temp` perché l'applicazione dovrà scrivere dati.

- L'istruzione `ADD` serve per copiare il file `.jar`, corrispondente all'applicazione, nell'immagine `docker`. Viene inoltre usato per copiare, nell'immagine `docker`, le librerie necessarie per la corretta esecuzione della fase di correzione.
- L'applicazione necessita di determinate cartelle per il corretto funzionamento, tramite l'istruzione `RUN` e il comando `mkdir` le cartelle vengono create.
- L'ultima istruzione `ENTRYPOINT` permette di eseguire il backend.

Prima di creare l'immagine `docker`, bisogna assicurarsi di aver impacchettato l'applicazione sotto forma di un file `jar` usando Maven. L'impacchettamento viene eseguito con il seguente comando:

```
1 $ mvn clean package
```

Il comando per la creazione dell'immagine `docker` è il seguente:

```
1 $ docker build -t <nome_jar_backend> .
```

Ora è possibile avviare l'immagine `docker` del server di backend tramite il comando:

```
1 $ docker run -p 8080:8080 <nome_jar_backend>
```

4.2 Frontend

Questa sezione descrive come è stato sviluppato il plugin. Iniziando con una panoramica del tipo di plugin scelto e perché, seguirà la descrizione della struttura dei file del plugin e verranno analizzati i file più importanti.

4.2.1 Assignment submission plugin

Date le specifiche, ho scelto di implementare un plugin di tipo *Assignment submission* poiché permette di aggiungere una nuova modalità di *consegna* per un'attività di assignment e in particolare permette di aggiungere un *form* per la consegna degli studenti personalizzato. Il tipo *Assignment submission* ha inoltre il pieno controllo sulla visualizzazione del risultato del assignment, permettendo così la modifica delle informazioni visualizzate dopo la consegna del assignment: l'aggiunta del feedback.

Il nome scelto per il plugin è *Prog 2 Automatic Corrector*, abbreviato in *P2AC* dato che le regole di Moodle impongono che il nome del plugin sia lungo massimo 11 caratteri.

4.2.2 Struttura plugin

La struttura di un plugin *assign submission*, come indicato nella sezione sviluppatori di moodle^[7], è rappresentata nella figura 4.4.

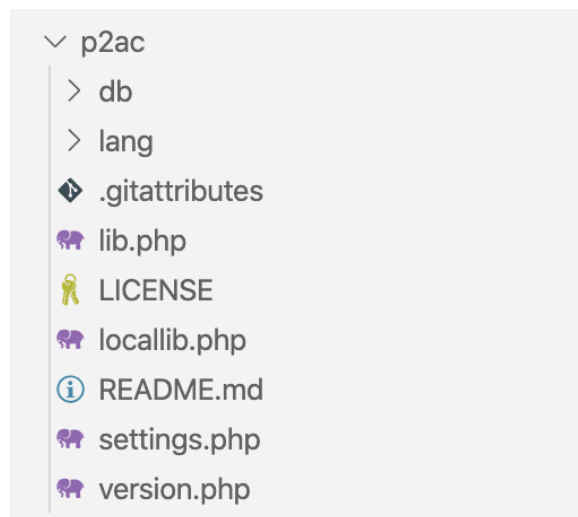


Figura 4.4: Struttura file plugin assignsubmission

Nelle sezioni successive verranno spiegati i file più importanti per l'implementazione del plugin.

Per far sì che il plugin sia visibile alla piattaforma Moodle, tutti i file del plugin *Prog 2 Automatic Corrector* devono trovarsi in *mod/assign/submission/p2ac* all'interno della cartella root di Moodle.

4.2.2.1 Lang

La cartella *lang* contiene i file della lingua per il plugin. A seconda della lingua da supportare, i file risiedono in una diversa sottocartella di *lang*. Il plugin *Prog 2 Automatic Corrector* supporta l'inglese e l'italiano:

- `lang/en/assignsubmission_p2ac.php`
- `lang/it/assignsubmission_p2ac.php`

Il nome del file della lingua deve avere la seguente struttura: `<tipo_plugin>_<nome_plugin>`. Un file di lingua è composto da chiavi-valori: `string ["key"] = "Value"`; dove la chiave è la stessa in tutti i diversi file di lingua e il valore dipende dalla lingua. Moodle fornisce un'API String dedicata che consente - con una chiave - il recupero del valore in base alla lingua selezionata.

4.2.2.2 Database

I file relativi all'interazione col database di Moodle si trovano nella cartella *db*, contenuta nella cartella *p2ac*, cartella del plugin (fig:4.4).

Il file `db/install.xml` viene utilizzato durante l'installazione del plugin, serve per definire le tabelle del database associate. Per ciascun plugin, il database deve avere una tabella principale con lo stesso nome del plugin stesso.

Il codice 4.4 mostra come la tabella P2AC, del plugin, è stata definita, con tutti i suoi campi e le definizioni di chiave primaria ed esterna.

```

1 <TABLES>
2   <TABLE NAME="assignsubmission_p2ac" COMMENT="Info about P2AC submissions for
3     assignments">
4     <FIELDS>
5       <FIELD NAME="id" TYPE="int" LENGTH="10" NOTNULL="true" SEQUENCE="true"/>
6       <FIELD NAME="assignment_id" TYPE="int" LENGTH="10" NOTNULL="true" DEFAULT="0"
7         SEQUENCE="false"/>
8       <FIELD NAME="submission_id" TYPE="int" LENGTH="10" NOTNULL="true" DEFAULT="0"
9         SEQUENCE="false"/>
10    </FIELDS>
11    <KEYS>
12      <KEY NAME="primary" TYPE="primary" FIELDS="id" COMMENT="unique id submission."/>
13      <KEY NAME="fk_assignment" TYPE="foreign" FIELDS="assignment_id" REFTABLE="assign
14        " REFFIELDS="id" COMMENT="assignment instance this submission relates to"/>
15      <KEY NAME="fk_submission" TYPE="foreign" FIELDS="submission_id" REFTABLE="
16        assign_submission" REFFIELDS="id"/>
17    </KEYS>
18  </TABLE>
19  <TABLE NAME="P2AC_feedback" COMMENT="Info about P2AC feedback for assignments">
20    <FIELDS>
21      <FIELD NAME="id" TYPE="int" LENGTH="10" NOTNULL="true" SEQUENCE="true"/>
22      <FIELD NAME="message" TYPE="char" LENGTH="255" NOTNULL="true" SEQUENCE="false"
23        COMMENT="checker feedback"/>
24      <FIELD NAME="P2AC_id" TYPE="int" LENGTH="10" NOTNULL="true" SEQUENCE="false"
25        COMMENT="fk for assignsubmission_p2ac table"/>
26    </FIELDS>
27    <KEYS>
28      <KEY NAME="primary" TYPE="primary" FIELDS="id"/>
29      <KEY NAME="fk_P2AC" TYPE="foreign" FIELDS="P2AC_id" REFTABLE="
30        assignsubmission_p2ac" REFFIELDS="id" COMMENT="fk assignsubmission_p2ac"/>
31    </KEYS>
32  </TABLE>
33 </TABLES>

```

Listing 4.4: Creazione tabelle assignment

La struttura generale dei dati è illustrata nel modello ER nella figura 4.5

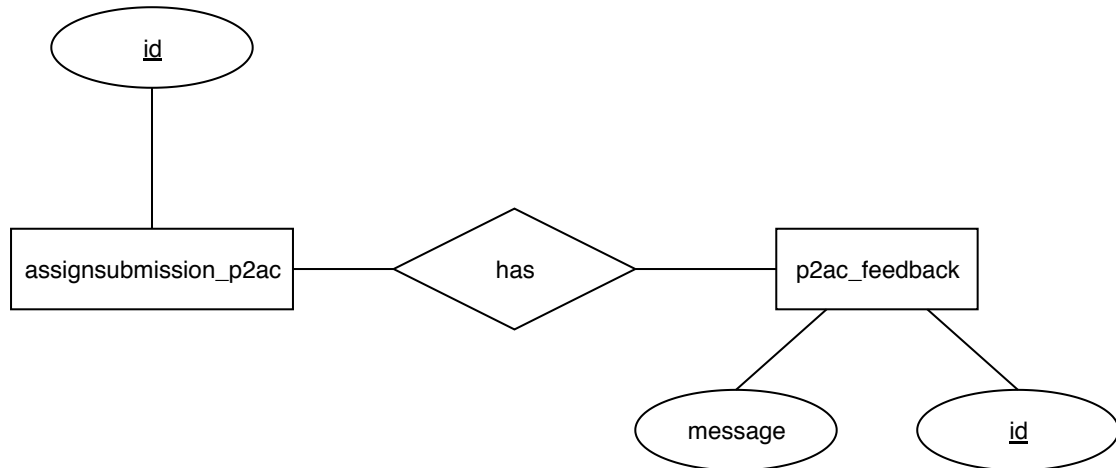


Figura 4.5: Diagramma-ER plugin

Ogni volta che un assignment, con consegna di tipo **Prog 2 Automatic Corrector**, viene creato dalla piattaforma Moodle, una nuova istanza dell'entità' *assignsubmission_P2AC* viene creata.

La consegna di un *taskfile*, da parte di uno studente, porta all'elaborazione della sua consegna da parte del backend il quale restituirà il feedback in formato stringa-JSON. Il plugin si occuperà di parsare tale stringa e salvarla nella tabella *P2AC_feedback* associata all'assignment. Questo passaggio è necessario poiché permette di mostrare il feedback relativo all'assignment al docente e allo studente anche successivamente alla consegna.

4.2.2.3 Locallib.php

E' il file nel quale tutte le funzionalità del plugin vengono definite. Tutti i plugin di tipo *submission* devono definire una classe che abbia come nome il nome del plugin e che estenda *assign_submission_plugin*, come mostrato nel listato 4.5.

```

1  class assign_submission_P2AC extends assign_submission_plugin {
2      ... code
3  }
  
```

Listing 4.5: Definizione classe plugin

Un nuovo plugin di tipo *assign submission* deve rispettare la gerarchia di classi mostrata in figura 4.6:

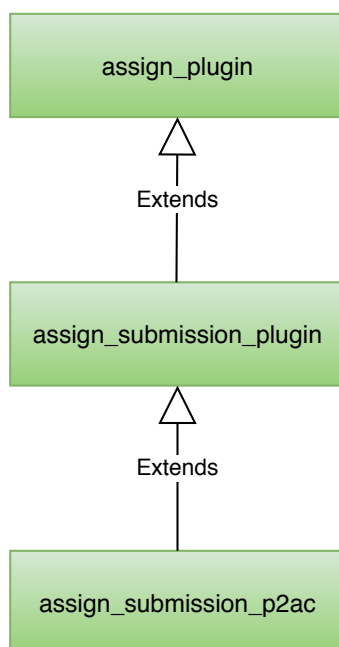


Figura 4.6: Gerarchia classi assignment

La classe *assign_plugin* è una classe astratta ed è la classe di base per tutti i plugin assignment.

La classe *assing_submission_plugin* è anche essa una classe astratta e ogni plugin di tipo submission deve estenderla. Presenta un limitato numero di funzioni aggiuntive esclusive per i plugin di tipo *submission*.

Queste due classi forniscono una serie di funzioni pubbliche, i cosiddetti *hook*, che possono essere sovrascritte per implementare le funzionalità necessarie.

Di seguito verranno descritte brevemente alcune funzioni selezionate per dare un'idea di come sono state implementate le funzionalità aggiuntive del plugin.

- La funzione **get_settings()** viene chiamata durante la creazione della pagina delle impostazioni per l'assignment. Consente al plugin di aggiungere un elenco di impostazioni al form. Nel caso del plugin P2AC viene aggiunto un file manager per consentire ai docenti di caricare il loro file zip *correctionFiles*.
- La funzion **save_settings()** viene chiamata quando viene salvata la pagina delle impostazioni dell'assignment, per un nuovo assignment o durante la modifica di uno esistente. Nel plugin P2AC questa funzione salva lo zip di *correctionFiles* caricato dall'insegnante e invia il file a *Programmazione2AutomaticCorrector-backend*, il quale lo elabora.

La comunicazione con il backend avviene come mostrato nel seguente listato 4.6:

```
1 $curl = new curl();
2 $curl->post($url, $params, $options);
```

Listing 4.6: comunicazione con il backend

- La funzione **get_form_elems_for_user()** viene chiamata durante la creazione del form di invio e consente (come la funzione `get_settings` per le impostazioni) di aggiungere un elenco di elementi al form di invio. Nel caso del plugin P2AC questa funzione aggiunge il file manager per consentire agli studenti di caricare il loro file zip *taskFiles*.
- La funzione **save()** viene chiamata per salvare la consegna di uno studente. Con il plugin P2AC la funzione:

- Salva il file zip *taskFiles* appena caricato nel database Moodle;
- Invia il file zip a *Programmazione2AutomaticCorrector-backend*, il quale lo elabora e fornisce un responso ¹. L'invio del file .zip al backend avviene come mostrato nel listato 4.6;
- Riceve ed elabora la risposta del backend come mostrato in seguito dal listato 4.7:

```
1 $response = $curl->post($url, $params, $options);
2 $feedback = json_decode($response);
```

Listing 4.7: elaborazione responso lato plugin

- Salva la risposta del backend, il feedback, nel database Moodle come mostrato dal listato 4.8 in seguito:

```
1 $p2ac_feedback = new stdClass();
2 $p2ac_feedback->message = $feedback->checkerResponse;
3 $p2ac_feedback->p2ac_id = $p2acsubmission->id;
4 $p2ac_feedback->id = $DB->insert_record(TABLE_P2AC_FEEDBACK,
    $p2ac_feedback);
```

Listing 4.8: salvataggio risposta backend lato plugin

- La funzione **view_summary()** è usata per visualizzare il riepilogo della submission sia per un docente che per uno studente. Per gli studenti il riepilogo sarà mostrato nella tabella di stato della submission mentre per i docenti sarà mostrato in una colonna nella tabella di valutazione relativa all'assignment.

¹L'elaborazione e il responso avvengono come è stato mostrato nella sezione Controller-Studente 4.1.1.2

- La funzione `delete_backend_files()` viene invocata al momento della cancellazione di un assignment. Il suo compito è quello di cancellare, dal database Moodle, tutti i dati relativi all'assignment, nel caso del plugin P2AC elimina in aggiunta i dati relativi al feedback. Per garantire consistenza viene inviata una richiesta *HTTP Delete*, come mostrato nel listato 4.9, al backend in modo che vengano cancellati anche i *correctionFiles* relativi all'assignment.

```
1      $url = $wsbaseaddress . "/delete/teacher?assignmentID=" . $assignmentid;  
2      $curl = new curl();  
3      $curl->delete($url);
```

Listing 4.9: comunicazione con il backend per eliminazione dati

Capitolo 5

Demo

5.1 Configurazione iniziale

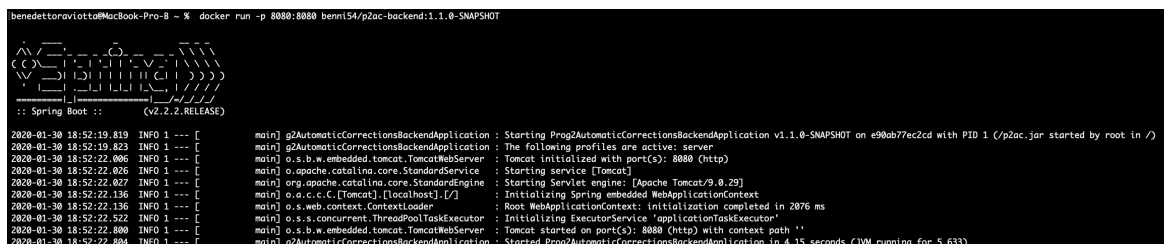
5.1.1 Backend

Il Programmazione2AutomaticCorrector-backend viene fornito, sotto forma di *immagine docker*, con una configurazione completa e pronto all'utilizzo. Per avviare il backend basta eseguire, da terminale, il seguente comando:

```
1 $ docker run -p 8080:8080 benni54/p2ac-backend:1.1.0 -SNAPSHOT
```

Listing 5.1: Avvio server backend da terminale

il quale produrrà, se tutto è andato correttamente, la seguente schermata 5.1:



```
benedettoraviotta@MacBook-Pro-B ~ % docker run -p 8080:8080 benni54/p2ac-backend:1.1.0 -SNAPSHOT
:: Spring Boot ::      (v2.2.2.RELEASE)

2020-01-30 18:52:19.819 INFO 1 --- [main] g2AutomaticCorrectionsBackendApplication : Starting Prog2AutomaticCorrectionsBackendApplication v1.1.0-SNAPSHOT on e90ab77ec2cd with PID 1 (/p2ac.jar started by root in /)
2020-01-30 18:52:19.823 INFO 1 --- [main] g2AutomaticCorrectionsBackendApplication : The following profiles are active: server
2020-01-30 18:52:22.006 INFO 1 --- [main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat initialized with port(s): 8080 (http)
2020-01-30 18:52:22.026 INFO 1 --- [main] o.apache.catalina.core.StandardService : Starting service [Tomcat]
2020-01-30 18:52:22.027 INFO 1 --- [main] org.apache.catalina.core.StandardEngine : Starting Servlet engine: [Apache Tomcat/9.0.29]
2020-01-30 18:52:22.136 INFO 1 --- [main] o.a.c.c.C.[Tomcat].[localhost].[/] : Initializing Spring embedded WebApplicationContext
2020-01-30 18:52:22.136 INFO 1 --- [main] o.s.web.context.ContextLoader : Root WebApplicationContext: initialization completed in 2076 ms
2020-01-30 18:52:22.522 INFO 1 --- [main] o.s.s.concurrent.ThreadPoolTaskExecutor : Initializing ExecutorService 'applicationTaskExecutor'
2020-01-30 18:52:22.800 INFO 1 --- [main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat started on port(s): 8080 (http) with context path ''
2020-01-30 18:52:22.804 INFO 1 --- [main] g2AutomaticCorrectionsBackendApplication : Started Prog2AutomaticCorrectionsBackendApplication in 4.15 seconds (JVM running for 5.633)
```

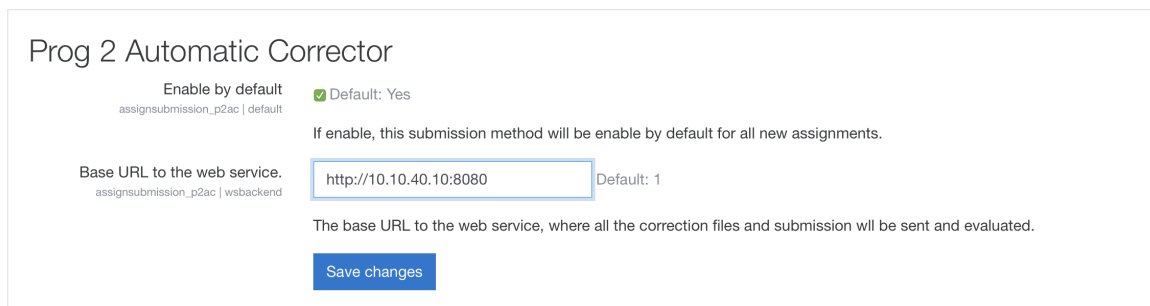
Figura 5.1: Avvio server backend

5.1.2 Frontend

Per farsi che il plugin P2AC possa comunicare con web service è necessario configurare il *web service* di backend per il plugin tramite Moodle:

Site administration → Plugins → Prog 2 Automatic Corrector

L'immagine seguente 5.2 mostra la schermata visualizzata dopo avere seguito il percorso indicato e mostra come configurare l'*URL* del backend.



Prog 2 Automatic Corrector

Enable by default
assignsubmission_p2ac | default

☒ Default: Yes

If enable, this submission method will be enable by default for all new assignments.

Base URL to the web service.
assignsubmission_p2ac | wsbackend

Default: 1

The base URL to the web service, where all the correction files and submission will be sent and evaluated.

[Save changes](#)

Figura 5.2: Configurazione URL backend da Moodle

5.2 Creazione assignment

I file necessari allo svolgimento dell'assignment, come il testo e le eventuali classi già scritte dal docente, possono essere caricate, nella sezione *Additional files*, durante la creazione dell'assignment come mostrato nell'immagine 5.3.

The screenshot shows the 'Adding a new Assignment' interface. At the top, the title 'Programmazione 2' is displayed, followed by a breadcrumb trail: 'Home / My courses / Prog2 / Programmazione 2 - P2AC / Adding a new Assignment'. The main heading is 'Adding a new Assignment' with an 'Expand all' link on the right. A sidebar on the left contains a 'General' tab and two sections: 'Assignment name' with a text input field and a red error icon, and 'Description' with a rich text editor. The 'Additional files' section is at the bottom, featuring a 'Files' button, a dashed box for file uploads with a blue arrow pointing down and the text 'You can drag and drop files here to add them.', and a note 'Maximum size for new files: Unlimited'.

Figura 5.3: Caricamento file necessari per svolgimento assignment

Il docente per rendere possibile la correzione delle consegne degli studenti, deve selezionare come tipo di consegna **Prog 2 Automatic Corrector** come mostrato nella schermata 5.4 in seguito:

Programmazione 2

[Home](#) / [My courses](#) / [Prog2](#) / [Programmazione 2 - P2AC](#) / Adding a new Assignment

Adding a new Assignment

[Expand all](#)

- [General](#)
- [Availability](#)
- [Submission types](#)

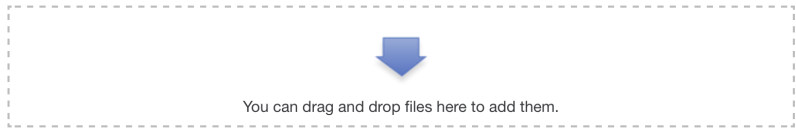
Submission types

☐ File submissions [?](#) ☐ Online text [?](#) ☒ Prog 2 Automatic Corrector [?](#)

Correction files [?](#)

Maximum file size: Unlimited, maximum number of files: 1

Files



You can drag and drop files here to add them.

Accepted file types:

Archive (ZIP) .zip

Figura 5.4: Creazione di un assignment

Figura 5.5: Caricamento file di correzione assignment

5.3 Consegna assignment

La seguente schermata 5.6 mostra la pagina visualizzata dallo studente durante la consegna di un assignment, e l'area nella quale dovrà caricare la sua consegna.

Programmazione 2

[Home](#) / [My courses](#) / [Prog2](#) / [Programmazione 2 - P2AC](#) / [Biblioteca](#) / [Edit submission](#)


Biblioteca

Exercise ZIP

?

Maximum file size: 8MB, maximum number of files: 1

Files



studentTask....

Accepted file types:
Archive (ZIP) .zip

Save changes

Cancel

Figura 5.6: Consegna dello studente per l'assignment

5.4 Visualizzazione risultato assignment

In questa sezione verranno mostrate le due differenti viste riguardanti la visualizzazione dei risultati relativi ad un assignment.

5.4.1 Studente


Uno studente, subito dopo aver consegnato i suoi files relativi all'assignment, potrà vedere il feedback relativo alla sua consegna. Come mostrato in seguito in figura 5.7:

Biblioteca

Submission status

Attempt number	This is attempt 1.
Submission status	Submitted for grading
Grading status	Not graded
Due date	Friday, 7 February 2020, 12:00 am
Time remaining	3 days 10 hours
Last modified	Friday, 31 January 2020, 3:14 pm
Submission comments	Comments (0)

Prog 2
Automatic
Corrector

 [studentTask.zip](#) 31 January 2020, 3:11 pm

Feedback about your submission:

La classe 'Libro' non dispone del metodo 'get' per le varibiali private: 'isbn' ed 'editore'

Edit submission

Remove submission

You can still make changes to your submission.



Figura 5.7: Visualizzazione feedback da parte di uno studente

5.4.2 Docente

Un docente visualizzerà tutti i feedback, relativi all'assignment, come mostrato nella seguente schermata 5.9:

Biblioteca

Grading action Choose...

Select	User picture	First name / Surname	Email address	Status	Grade	Edit	Last modified (submission)	Submission comments	Prog 2 Automatic Corrector	Last modified (grade)	Fee con
<input type="checkbox"/>		Aldo Rossi	student@moodle.it	Submitted for grading	Grade	Edit	Friday, 31 January 2020, 3:25 pm	Comments (0)	Q La classe 'Libro' non dispone del metodo 'get' per le varibiali private: 'isbn' ed 'editore'	-	
<input type="checkbox"/>		Matteo Bianchi	student3@moodle.it	Submitted for grading	Grade	Edit	Friday, 31 January 2020, 3:42 pm	Comments (0)	Q La classe 'Libro' non dispone di un suo metodo 'Equals'.	-	

With selected... Lock submissions Go

Figura 5.8: Visualizzazione di tutti i feedback da parte di un docente

Un docente può visualizzare più nel dettaglio la consegna di uno studente con il relativo feedback, figura 5.9.

Biblioteca

 studentTask.zip 31 January 2020, 3:25 pm

Feedback about your submission:

La classe 'Libro' non dispone del metodo 'get' per le varibiali private: 'isbn' ed 'editore'

[Back](#)

Figura 5.9: Visualizzazione dettagliata di una consegna da parte di un docente