University of Helsinki

Faculty of Science

Department of Computer Science

TestMyCode Eclipse

Project Architecture

Juhani Heliö

Ville-Pekka Hämäläinen

Nikke Kostiainen

Erkka Kääriä

Leo Leppänen

Joel Nummelin

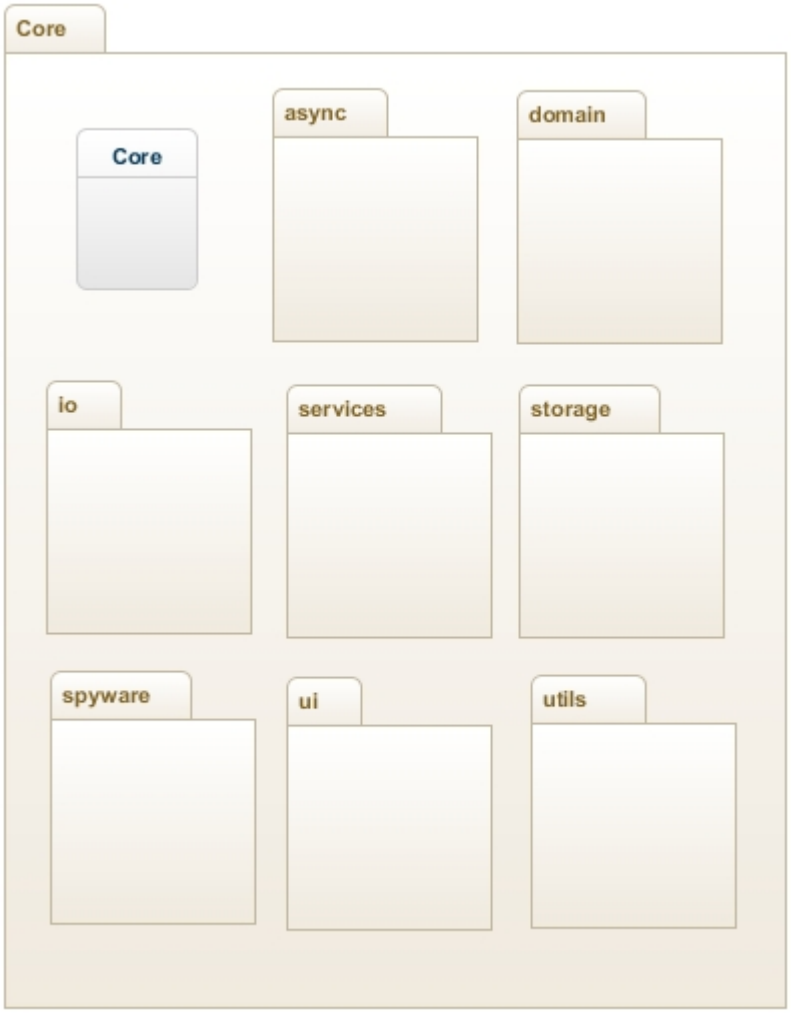
June 24, 2014

Contents

1 Overview

The TestMyCode Eclipse plugin consists of two discrete main components: An IDE independent TestMyCode Core and an Eclipse plugin that wraps the Core component and implements necessary IDE specific features.

2 TestMyCode Core



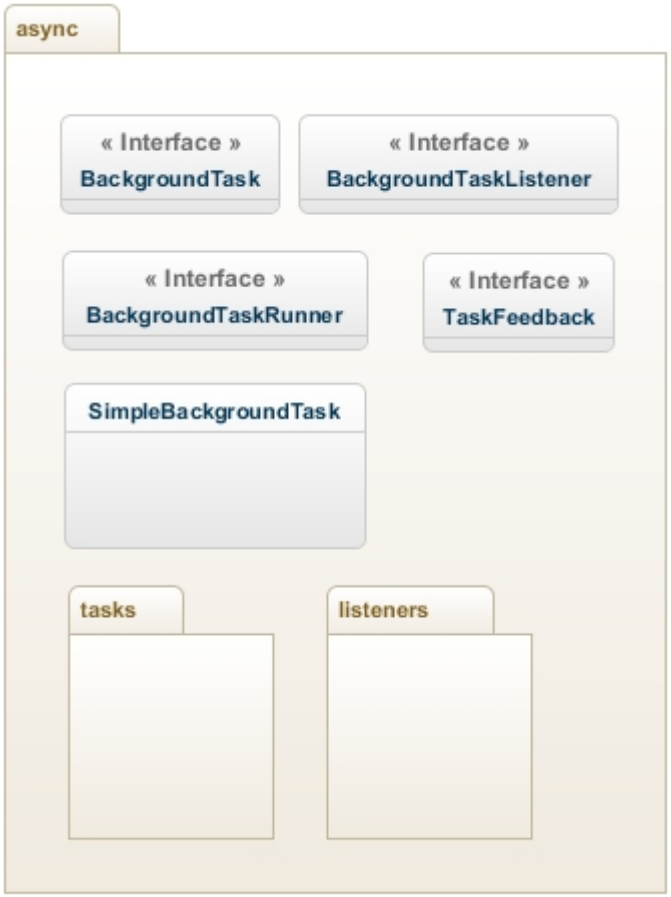
TestMyCode Core is the IDE independent component of the project. It consists of IDE independent code that contains most of the projects logic.

The Core also interfaces with other TestMyCode components such as the TestMyCode Server[1] and the TestMyCode JUnit Runner[2].

2.1 Core

The main class of TestMyCode Core is the aptly named *Core*. This class provides the actual plugin access to many other features of the core such as the user's settings, the server interface, data access objects and other core components.

2.2 Background tasks

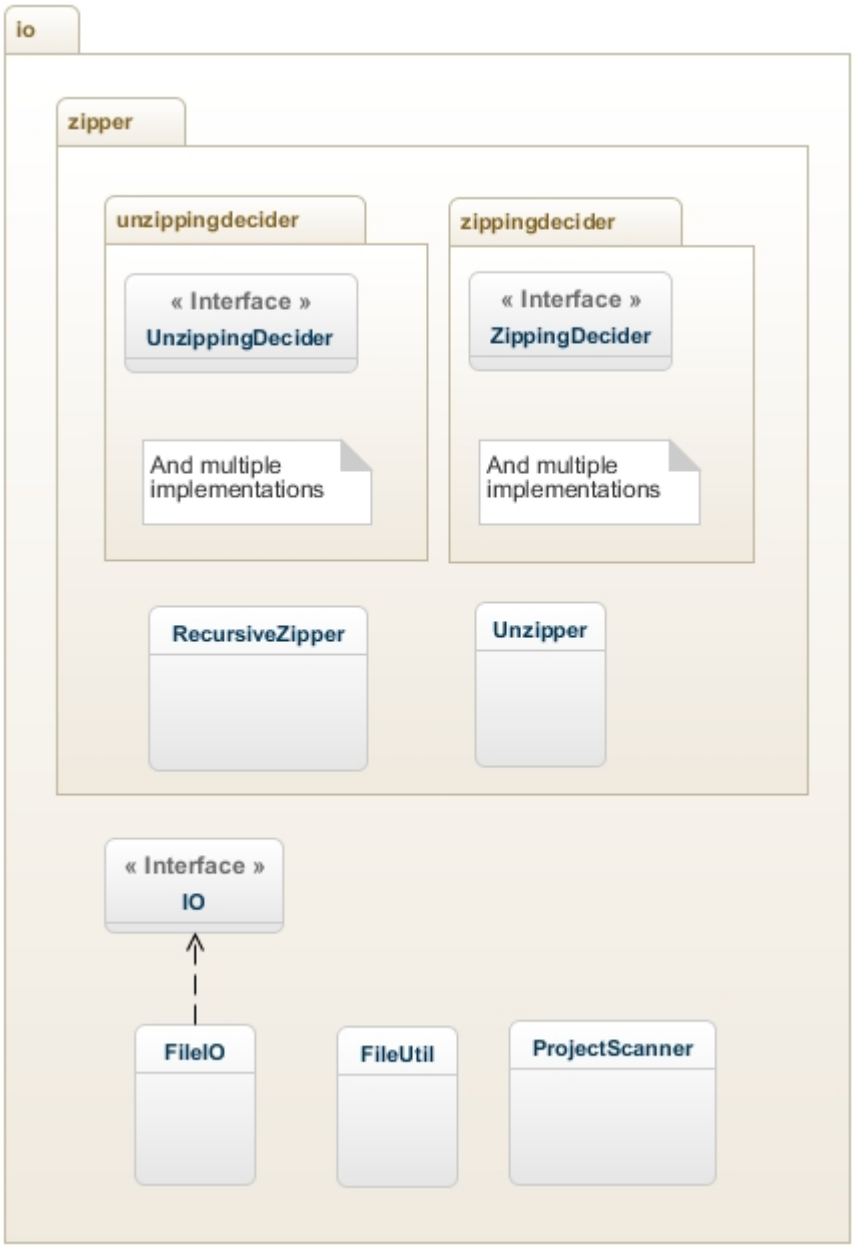


[1]https://github.com/testmycode/tmc-server
[2]https://github.com/testmycode/tmc-junit-runner

Any actual processing the plugin needs is done asynchronously with implementations of the interface *core.async.BackgroundTask*. Some of these tasks are associated with a relevant implementation of *core.async.BackgroundTaskListener* that describes reactions to different states of the task.

The actual tasks are ran by an implementation of the *core.async.BackgroundTaskRunner* that the IDE dependent plugin component needs to provide. These tasks can also invoke the IDE user interface via a IDE dependent implementation of the *ui.IdeUIInvoker*.

2.3 Input-Output

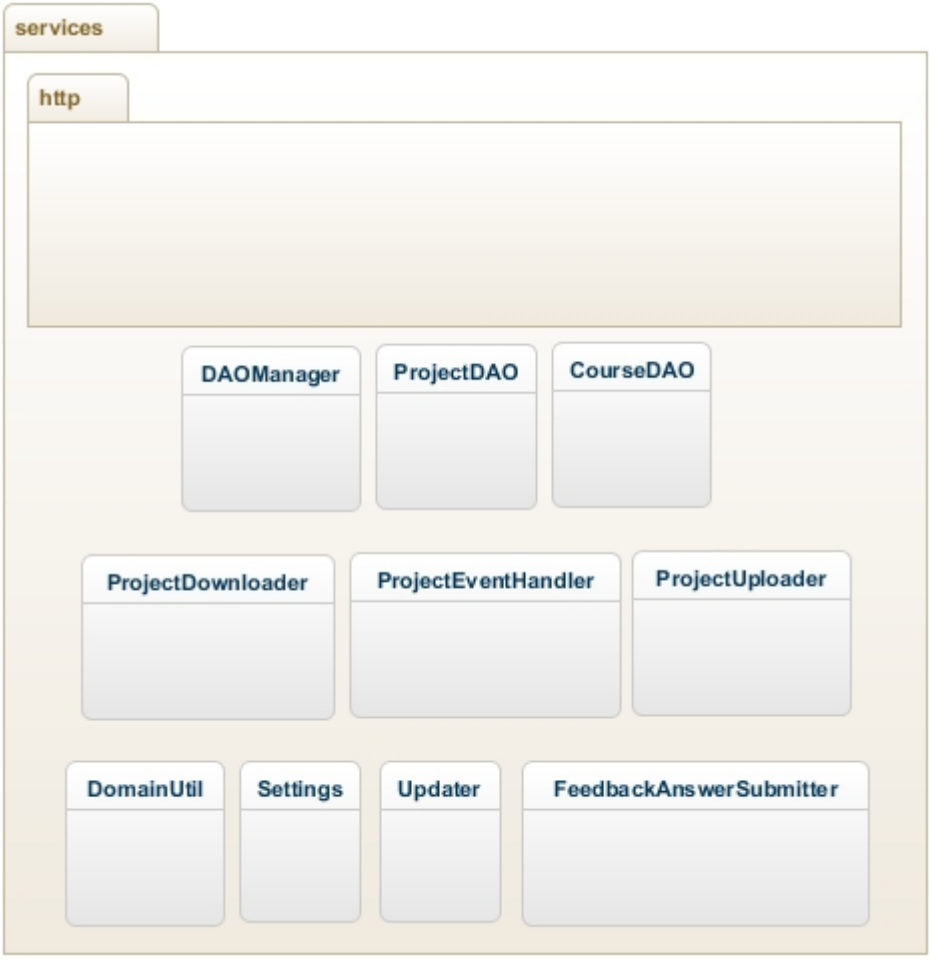


The project needs to interface with the file system in multiple ways. We need to zip and unzip projects and save details of the exercises and projects the user has downloaded.

The package *io* contains a *core.io.FileIO* class that contains the actual logic used to interface with the file system.

For zipping and unzipping, *core.io.zipper.RecursiveZipper* and *core.io.zipper.Unzipper* are used. These classes utilize different implementations of *core.io.zipper.unzippingdecider.UnzippingDecider* and *ZippingDecider* that tell the zippers what files to (un)zip and what to leave untouched. This allows us to for example download full projects when updating an exercise without overwriting the student's progress.

2.4 Services



The *services* package contains our data access objects and other miscellaneous classes that other services and the plugin can utilize to accomplish different tasks.

This package also contains the subpackage *core.services.http* that contains the code required to interface with the TestMyCode server. The main interface between this HTTP stack and other components is the *core.services.http.ServerManager* class that provides simple methods for actions that require sending or receiving data from the TestMyCode server.

2.5 Spyware

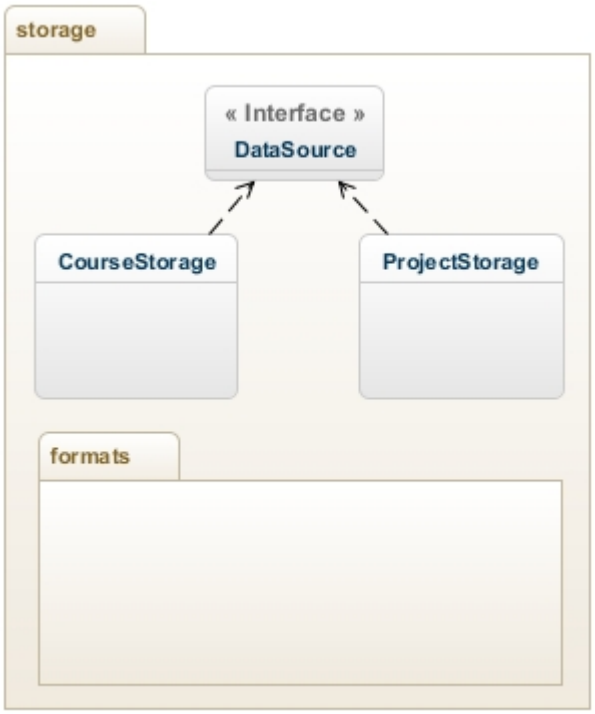
The 'spyware' component that resides in the *core.spyware* package contains the logic for logging the student's actions for analysis of the student's progress and process of learning. Provided that the user has consented to sending snapshots of his study, the component sends the snapshots to TestMyCode Spyware Server[3] for further analysis of the whole class' progress as a population.

This component is rather highly decoupled from the rest of the plugin and is quite self-contained. Due to being dependent on - for example - the HTTP stack, it is an integrated component of the Core instead of being a discrete project.

The spyware component also has a 'hidden' dependency to the Google diff match patch component[4]. This library is provided as *core.spyware.utility.diff_match_patch*. While a third party Maven implementation of the library exists, it was decided against using it until the other spyware tools are updated to utilize that as well.

[3]https://github.com/testmycode/tmc-spyware-server
[4]https://code.google.com/p/google-diff-match-patch/

2.6 Storage



The *core.storage* package contains classes that utilize the *core.io.FileIO* to write information about the user's courses and exercises to the disk so that this information can be accessed over multiple sessions.

2.7 User interface

The *core.ui* package mainly consists of the *core.ui.IdeUIInvoker* interface, that the IDE specific plugin must implement. This interface defines the methods of UI-interaction that the core requires. This allows us to invoke dialogs and other UI elements from the background tasks and their listeners, allowing higher decoupling between the Core and the plugin.

3 Eclipse plugin

The Eclipse plugin is basically an Eclipse dependent wrapper over the actual TestMyCode Core. It provides the required IDE specific implementations that are needed for the add-on to really work. The core is provided to the plugin as a shaded .jar -file, that is located in the *lib/* folder.

3.1 Activator

The *tmc.eclipse.activator.CoreInitializer* defines the start-up behavior of the plugin, initializes the Core and schedules the always present background tasks such as scheduled checks for new code reviews.

3.2 Handlers

The package *tmc.eclipse.handlers* contains the classes that define how the plugin reacts to different user interface events, such as clicking on menu items. The *tmc.eclipse.handlers.listeners* packages contains classes that react to different user interface events, such as changing the focus. For example the class *tmc.eclipse.handlers.listeners.SelectionListener* allows us to disable the 'submit' button whenever a non-TestMyCode project is selected.

3.3 Services

The package *tmc.eclipse.services* contains classes that open different downloaded projects in the IDE with correct project natures, f.ex. as a Maven project or as an Ant project.

3.4 Spyware

The *tmc.eclipse.spyware* package handles listening to events within the IDE editors and passes them onwards to the Core for further handling and (depending on whether the user has given consent) possibly sends the snapshot data to the TestMyCode Spyware server.

3.5 Tasks

The *tmc.eclipse.tasks* package contains the IDE specific implementations of the task running related interfaces defined by the Core.

It also contains the class *tmc.eclipse.tasks.TaskStarter* that contains the required logic for launching new background tasks from within the plugin component.

3.6 User interface

The *tmc.eclipse.ui* package contains all the custom dialogs and windows needed by the plugin component.

3.7 Utilities

The class *tmc.eclipse.util.WorkbenchHelper* is an utility class that contains code needed in multiple places within the plugin. It allows - for example - for resolving the 'active project' (a term not native to Eclipse IDE), and saving of all open files.

Also present in the *tmc.eclipse.util* package are the *project natures* that allow for the exercise status indicators (the small colored dots) shown in the project explorer.