Ministry of Science and Higher Education of the Russian Federation

NATIONAL RESEARCH

TOMSK STATE UNIVERSITY (NR TSU)

Research and Education Center “Higher IT School” (HITs)

APPROVED BY

Head of the Main Educational Program

Professor, Doctor of Physical and Mathematical Sciences

\_\_\_\_\_\_\_\_\_\_ O.A. Zmeev,

*(signature)*

«\_\_\_\_\_»\_\_\_\_\_\_\_\_\_\_2023

**BACHELOR’S THESIS**

DESIGN AND DEVELOPMENT OF A PROTOTYPE FOR AN AUTOMATED TESTING SYSTEM FOR CHECKING AUTOMATED TESTS DEVELOPED BY SUTDENTS

Main educational program 09.03.04 – Software Engineering

Specialization “Software Engineering”

Yahya Abdul Majeed Khan

Bachelor’s Thesis Supervisor

Candidate of Science (Technology),

Associate Professor

\_\_\_\_\_\_\_\_\_\_\_\_\_ D.O.Zmeev

*(signature)*

«\_\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_2024

Written by

Student of Group No. 972005

\_\_\_\_\_\_\_\_\_ Y.Abdul Majeed Khan

*(signature)*

«\_\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_2024

Tomsk – 2024

ANNOTATION

CONTENTS

LIST OF FIGURES

INTRODUCTION

Software testing plays a crucial role in the software development lifecycle, ensuring that product releases adhere to requirements and remain free of defects. Consequently, the inclusion of software testing and quality assurance as part of the curriculum is essential for educating students and equipping them with industry-standard competencies.

In the traditional approach to teaching software testing, instructors manually evaluate each submitted solution. However, this method is often time-consuming and restricts the number of students a single teacher can effectively instruct. To address similar limitations in other areas of teaching, solutions have been devised to enhance such processes in the realm of IT education. One notable success story in this context is the implementation of online judges.

At present, a plethora of platforms incorporating online judges are available for training students in data structures and algorithms. These platforms span from public websites such as Leetcode, HackerRank, and CodeForces to more specialized private platforms developed by universities and institutions. Each platform employs its unique approach to the learning process, enhancing student engagement and making it more appealing.

However, the absence of platforms dedicated to automated software testing is evident. Consequently, the purpose of this work is to devise a conceptual framework for automated website testing—a subset of software testing. Then, with this framework as the foundation, create an educational website testing platform tailored for both students and educators.

Such a platform will have the following benefits:

* Reduced workload on teachers due to automatic marking,
* Improved student interaction due to gamification of the learning process, and competition with peers,
* Availability of metrics ( time, memory and other computer resources)

To achieve this goal, it is necessary to solve the following tasks:

* Design a concept for automated web testing
* Formulate requirements for an application to be developed
* Design the application being developed
* Implement the application
* Design a fault-tolerant deployment architecture for the application

GLOSSARY

Online Judge

Maintainer

Skeleton

1 DESIGNING A CONCEPT FOR AUTOMATED WEB TESTING

1.1 Defining a Task and Solution structure

To begin, a Task and its corresponding Solution structure must be established.

For web testing purposes, a Task refers to a set of directives that students are required to fulfill on a specific website. It should be named and accompanied by a detailed description that outlines the specific actions that need to be taken. This information is crucial for students to understand the purpose of the task and how it contributes to the overall testing process.

The Task should also include the URL of the website that students are required to test against. This allows them to access the site directly and begin their evaluation. Additionally, the Task should specify the number of points that will be awarded upon successful completion. This provides students with a clear incentive to complete the task accurately and thoroughly.

The Solution, on the other hand, is defined by the testing framework that students use to solve the Task. This framework can be used to infer the programming language used in the Solution. By knowing the testing framework, you can gain insights into the student's approach to the task and the tools they have at their disposal.

Furthermore, the content of the Solution encompasses the actual code provided by the student. This code is a critical component of the testing process, as it demonstrates the student's understanding of the website's functionality and their ability to apply their knowledge to solve the Task. By analyzing the code, you can assess the student's proficiency in the programming language and their overall testing skills.

If any packages or libraries are employed as part of the solution, they must be known in order to setup the environment for execution.

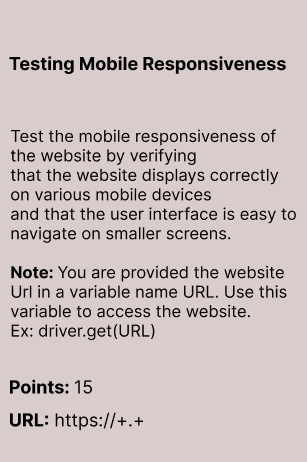


Figure 1 - Example Task Structure

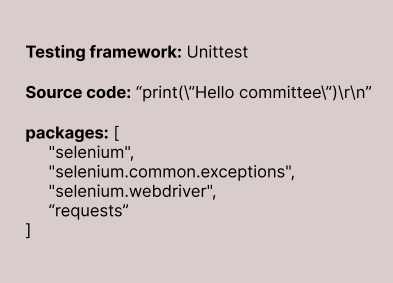


Figure 2 – Example Solution Structure

1.2 Testing a submitted solution

In order to make informed decisions regarding the acceptance or rejection of submitted solutions, it is imperative to establish well-defined criteria for what constitutes an acceptable solution. Any solution that falls short of meeting all of these specified criteria must be deemed ineligible for acceptance. The subsequent enumeration outlines the specific criteria that a solution must rigorously adhere to in order to be considered for acceptance.

1. **Correctness**: The solution should execute without producing any errors. It must also make certain assertions depending on the task it is submitted for, and these assertions must succeed.
2. **Completeness**: The solution should cover all the necessary aspects of the task. It should include all the required steps, checks, and validations to ensure thorough testing.
3. **Resource limits**: The solution should run within the constraints of the resource limits in terms of execution time and resource usage.
4. **Platform Compatibility**: The solution must be implemented using a programming language and test framework that are supported by the platform.

Having established the criteria for evaluating a solution, the subsequent step involves devising an automated methodology capable of systematically assessing each criterion. This process will determine whether the solution merits acceptance or rejection.

In order to ensure the correctness of a submitted solution, it is enough to run the code against the website and utilize the automation framework language binding library to determine whether the test has been successful or not. This library is a crucial component of the automation framework as it enables the execution of tests on the browser using the respective driver, and the evaluation of their results in a consistent and reliable manner. By using this library, it is possible to obtain a clear and objective assessment of whether the test in the submitted solution passed or failed.

The details of how resource limits will be enforced and evaluated will be discussed later on in this paper. For now, it is sufficient to know that this issue will be handled by the Linux kernel, which is responsible for managing the resources of the machine where the code execution engine is running.

Furthermore, if the solution is not compatible with the platform, it is likely that the code execution engine will encounter errors or exceptions during its execution. In such cases, the code execution engine will return an error and the test will fail.

Henceforth, we will be focusing on checking the completeness of the submitted code. Throughout our research, various distinct methods have been identified for this purpose, including:

Note: the following methods for testing for completeness are performed after the submission has passed the test for correctness, except for method ..

Also, from this point on, most examples will be listed for the Selenium WebDriver and it’s language bindings in dotnet. However, the concepts translate over to other languages and test automation frameworks.

1. **Static code analysis and manipulation**

Static code analysis is a method generally used in software development and quality assurance teams to scan source code for security vulnerabilities, coding standard violations etc. It involves examining the code without executing the program. A linter is an example of a popular static code analyzer.

Incorporating static code analysis can prove beneficial for verifying submitted user code against predefined rule sets. This approach necessitates the inclusion of a well-defined set of rules when establishing a new task, which dictates whether specific methods, properties, string values, keywords, and other task-specific attributes (e.g., web element IDs, page URLs) are permissible or not. By implementing static code analysis in this context, one can maintain adherence to established guidelines and ensure uniformity in handling various tasks.

When a user submission is being checked for this task, these predefined rules must be provided as custom rules to the static code analyzer, which meticulously examines the user's solution against the established guidelines, ensuring adherence to the desired standards and practices. Some general examples of described scenarios are as follows:

* **Checking the correct URL**: Verify whether the user is using the correct URL in their code by looking for instances where the user specifies the website URL (e.g., driver.get("https://example.com")). Ensure that the URL matches the expected domain or path.
* **Validating website components/task criteria**: Define a list of required components or criteria, and then perform static analysis on the test code to identify relevant method calls or interactions with web elements. Look for specific methods (e.g., findElement, click, getText) that correspond to the required components, or checking whether the user’s test script contains lines like driver.findElement(By.id("someId")).
* **Checking for certain Methods/Properties**: If you want to enforce the use of specific methods or properties, maintain a list of allowed or disallowed methods/properties to compare against.

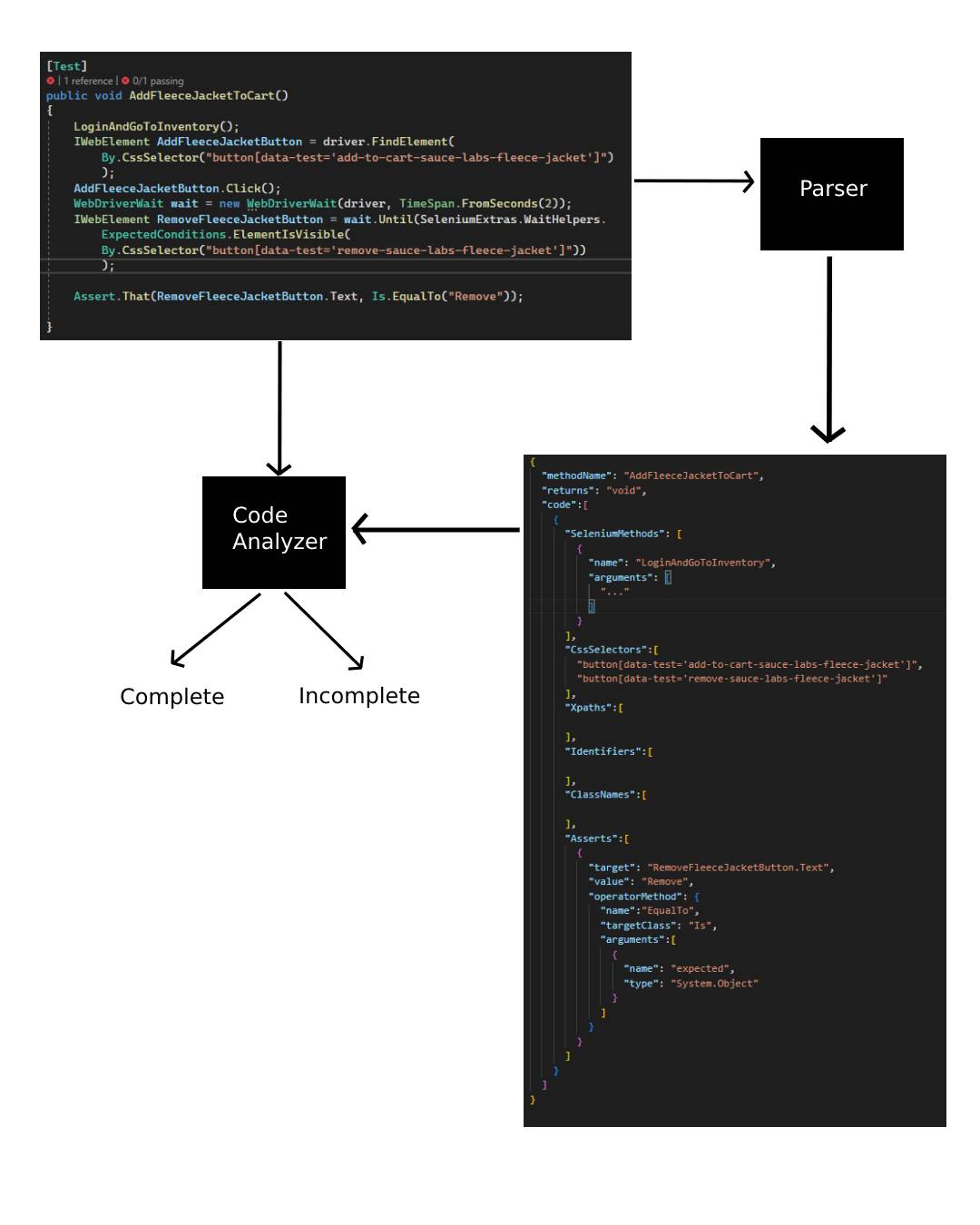


Figure 3 - Steps in static code analysis for determining completeness of a submitted solution

1. **Checking the logs/ tracking events**

Using Selenium's functionalities like command listeners, or directly injecting event listeners into the browser, we can log all the events that take place during the test. We can check these logs automatically and compare them to the expected ones.

In this case, we must provide the user with the driver to use for

1. Fault-based testing/Mutation testing
2. Checking the required test output
3. Comparison using Abstract Syntax trees

2 REQUIREMENTS GATHERING

Requirements gathering is the precursor to the design stage. During this stage, all the requirements that will serve as the foundation upon which the application will be built are collected.

To ensure that all relevant information is captured, it is prudent to study the functionalities of existing online judges for data structures and algorithms. By observing their features and capabilities, we can identify key requirements that will contribute to the success of our own system for website testing. This approach allows us to learn from successful implementations and incorporate best practices into our design.

2.1 Analyzing CodeHedgeHog

When conducting research on online judges, there exists a multitude of options available, each offering similar fundamental functionalities and services. However, for the sake of analysis, CodeHedgeHog, which is an online judge for automatically marking solutions to data structures and algorithm problems, was selected as it is an already implemented platform in the Higher IT School (HITS) of Tomsk State University (TSU).

Main business cases of our interest:

* **Problem Sets**: CodeHedgeHog offers a large collection of coding problems, known as the TestBank, with an exposed API, categorized by themes and related tasks. We can consider providing a similar problem set for website testing, categorized by tasks and functionality areas.
* **Online Judge**: It also has an online judge system that automatically checks the submitted code against predefined test cases. We can implement a similar system for our website, where the submitted scripts are run against a particular website and the results are displayed to the students.
* **Classes**: CodeHedgeHog has the concept of classes, which is described as “a system class that combines users (groups of students or individual students and teachers), topics and related tasks. By its nature it resembles a classroom and may belong to several departments or not to anyone.” [1]
* **Queue**: It also has a personal submission queue for each user, where they can view the history of all their past submissions, along with other info.
* **Rating**: This feature shows the rankings of all the users in a class.

CodeHedgeHog has other neat features for users, such as the ability to view submission metrics like time, status etc., with the ability to filter them by language, result, date etc. It also allows teachers to post-moderate submissions, resulting in more fine grain control.

2.2 Recognizing the main business process

In order to successfully undertake any project, it is crucial to identify its primary feature, which is essentially the core process that addresses the problem the system is being designed to solve. This involves pinpointing the key aspect or defining characteristic that distinguishes the project and outlines its purpose.

In the case of this project, the main business case revolves around automating the testing process of automated web tests submitted by students. This business process can be visualized in Figure 4.

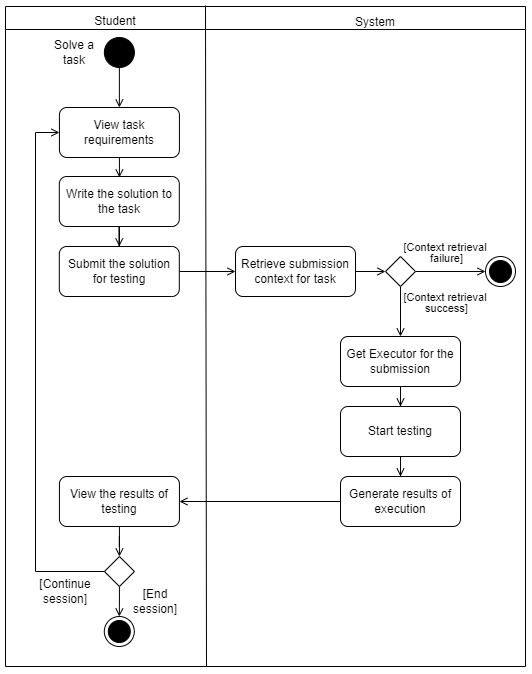


Figure 4 - Activity diagram showing the main business process

2.3 Functional and Non-functional requirements

**Functional requirements**

**Task Management:** The system should provide functionality for maintainers to create, update and delete tasks in the Task Manager. Similarly, it should also allow teachers to perform these actions upon platform specific tasks that have been duplicated from the Task manager.

**Group Management:** This feature enables teachers to create and manage groups of students within the platform. The teachers can add or remove students from the group, merge groups together, or delete them altogether. Grouping students help them separate by their skill level, interest or by their actual groups assigned to them by the institution they are a part of.

**Contest Management:** Teachers should be able to manage contests on the platform. Contests can be created for various purposes and occasions, adding or removing multiple different groups and tasks to these contests accordingly. These contests must have a start and an end time, with each contest independent from the others, with its own results and rankings.

**Script execution:** This feature is what enables the system to automatically execute the scripts submitted by students. These scripts are run on Selenium Grid, and the results are determined on the resultant execution of the scripts compared to the expected outcomes defined for them in task.

**Viewing submission history:** Each Student must be able to view the complete submission history for all their past script submissions, in all the contests. Teachers on the other hand, must be able to view the submissions history of all their students, in all the contests they handle. For each submission, the task is was for, the time of submission, and the result must be shown.

**Ranking system:** The rankings amongst the participants of the contest, from all the groups combined must be calculated and displayed. The ranking is solely based on the number of points accumulated for the tasks solved within a contest.

**Non-functional requirements**

**Secure:** Since the application will be running arbitrary code submitted by users, it is important to put rigid security measures in place to make sure that script submissions with malicious intent are not successful. Besides this, the application should also uphold standard security measures such as data integrity and privacy, preventing unauthorized access and secure storage and transformation of user data.

**Performant:** It is expected for the application users to grow overtime. Therefore, the application should be designed and deployed in a manner such that it’s performance in terms of user experience and response times do not degrade or decay under certain conditions. The application should specifically be fast with its execution times for submitted user scripts, as this is the main business process. In general, the application should be efficient in use of resources, have minimal latency and fast load and interaction times and varying loads and data volumes.

**Scalable:** As mentioned earlier, it is predicted for the application to grow overtime in terms of users and the data managed. As a result, it is prudent for the various components of the application to be designed such that they can be upscaled or downscaled in order to handle a growing amount of work without impacting performance. This includes the ability of expanding its capabilities by addition of new instances of deployed components.

**Reliable:** As the application primarily focuses on providing educational services, potentially to multiple different institutions, it is of utmost importance that it should function consistently and correctly under specified conditions. It should have consistent availability of service, and should be recoverable from failures without losing data or causing extended downtime.

**Usable:** The application should be intuitive and easy to use for all users, including maintainers, teachers and students. This includes industry standard requirements such as clear navigation, user-friendly interface, and use of features without confusion or difficulty.

2.4 Formalizing requirements

Having an idea of what needs to be achieved to deem the project successful, it is important to formalize the requirements. This involves transforming of the requirements into a specification with unique interpretation such that all ambiguities are resolved, and the project can be assessed for consistency, correctness and attainability. Good formalization of requirements also reduces corrective measures during the later stages of the system development.

2.3.1 Use cases

Figure 5 shows the general functionality that the application in its entirety must provide. From this figure, it is evident that the main use case is submitting solutions. This, as expected, aligns with the main business process of the application identified earlier.

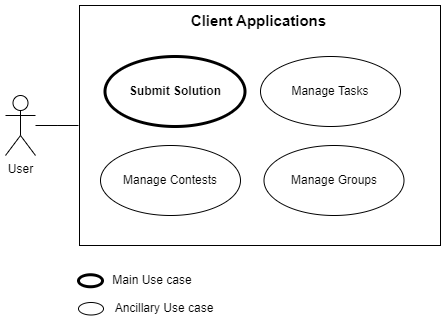


Figure 5 – Use case diagram showing common functionality of the application

Next, let’s look at the details of each of these generic use cases.

Figure 6 details the “Submit Solution” goal of the general use cases diagram. With regards to submitting a solution to a task, it is expected for the user to be able to view task details. The student should also have access to all the submissions they have made for this particular task. While submitting a solution, it is necessary to choose the testing framework that was used while writing the solution script. It must also be possible to perform multiple submissions for the same task.

The teacher, on the other hand, must be able to mark these submissions, and filter them on various fields, such as contest, task theme, submission date etc.

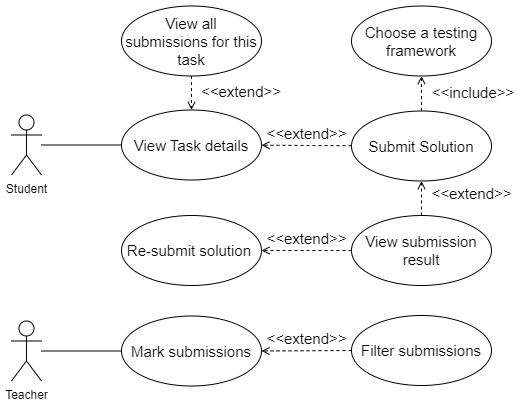


Figure 6 - Use case diagram for the "Submit Solution" goal of the general use case diagram

Figure 7 visualizes the details for the “Manage tasks” goal of the general use case diagram. Only the Maintainer and the Teacher user are involved in this case. The maintainer is the only one able to maintain the TaskBank. Creating a task involves, more importantly, choosing a skeleton for modification of the submitted script, and the websites to test the script against.

The teacher working on a specific platform, meanwhile, can create a task by pulling from the TaskBank, or editing an already existing task. The CRUD Website use case is further explained.

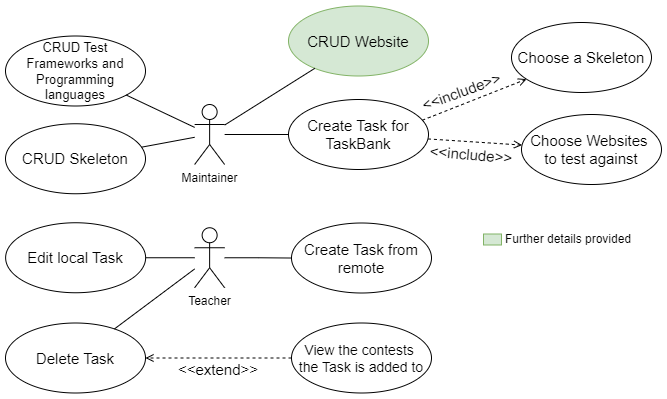


Figure 7 - Use case diagram for the "Manage Tasks" goal of the general use case diagram

As Figure 8 further demonstrates, the CRUD Website use case is further composed of the ability to add bugs to the different variants of the website. The maintainer should also be able to view the bugs that exist for a given website.

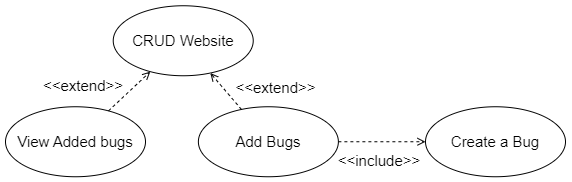


Figure 8 - Details for the "CRUD Website" goal of the Manage Tasks use case diagram

When managing contests, by extension, the teacher can add or remove groups and tasks to a contest, besides being able to update the various attributes, or delete it. The Student, on the other hand, can view the contests that they are enrolled in and the rankings in each contest. All of this can be seen in Figure 9.

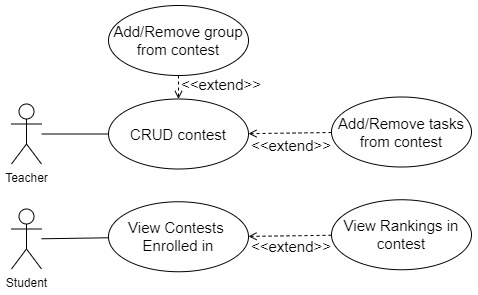


Figure 9 - Use case diagram for the "Manage Contests" goal of the general use case diagram

Lastly, when managing the groups on a platform, the teacher is able view all the users registered on the platform. They can then choose users to add or remove from a group. There is also the possibility of merging groups by adding them to a new group.

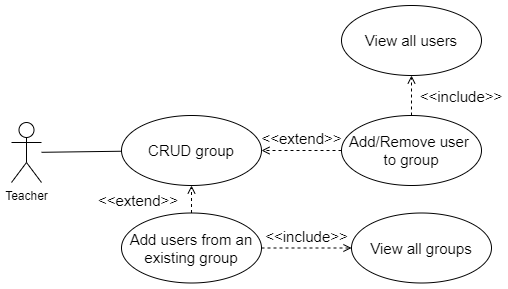


Figure 10 - Use case diagram for the "Manage Groups" goal of the general use case diagram

2.3.2 Problem Domain Model

During further design, a problem domain model of the application being developed was created, describing the main entities, their structure and connection between them, necessary for further implementation of the application. The domain model is shown in Figure 9.

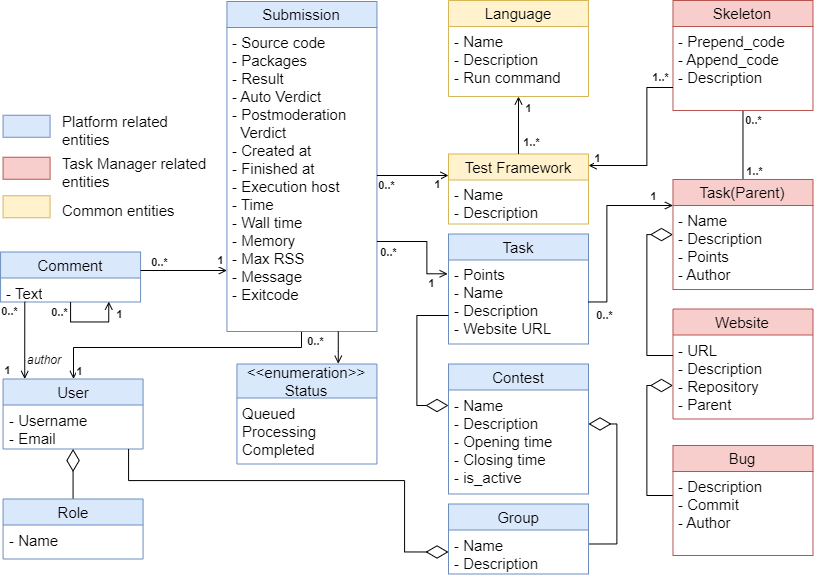


Figure 14 - Problem Domain Model

**Explanations for the diagram**

**Submission** – class, designed to represent the submission made by a student for a task. It stores the source code submitted, along with the packages used in the source code. Knowing the packages used is important for setting up the environment for the execution of the source code successfully. It also has a Result field, which is a composed form of the result of the execution in a format ready to be displayed. Furthermore, it stores the time for when the submission was created, and when its execution finished, along with the verdict, both automatic and post-moderated. It also has a Status, which can be used to poll for the submission status at any moment. Finally, it has a few other execution metrics, which will be discussed later.

**Contest** – class, designed to represent a logical contest within the Platform component. A contest is simply a collection of tasks and groups, with a name and description. More importantly, a contest has an opening time and a closing time. A contest is only visible to the users when it is set to active.

**Group –** class, which helps group user together. This group can then be collectively added or removed from a contest, or moved between contests, and reused for other contests. In the context of educational institutions, this group represents the group assigned to a student by the respective institution.

**Task –** class, within the Platform component, which represents a task to be solved. It is imperative that this class have a reference to a parent task in the Task manager component, as this reference is used to find the context for execution (the websites and skeletons), and decide whether a submission should pass or fail. Each task has a name and description, points, and the URL for the website for the students to view.

**Task (Parent) –** class, within the Task manager component, which stores the original tasks created by the maintainers of the Task data store. Each task can have multiple skeletons, each for a supported testing framework. This class is also associated with multiple websites, where one of them is a primary website which the user views, without bugs, while the others are variants of the primary website with bugs introduced for testing purposes. This class, additionally, stores the author of the task.

**Skeleton –** class, which stores two strings of code. One of the strings is prepended to the code that the user submits. This can be used for providing ready to use objects or variables like URL, driver, or make some packages available by default. The other string is appended to the code. This can be used to perform cleanup functions such as quitting the driver if the user forgets to, perform some calculations, or run certain modules of the testing framework. It also has a description, and preferably a name to identify by.

**Website –** class, which represents an actual website. Each website is identified by its URL. The class also stores the repository of the website where its source code is stored, and is accompanied by a description of the website. Finally, it also stores a reference to its parent if it is a variant of another website, or this field is *NULL* if the website is primary.

**Bug –** class, which provides information about the different bugs introduced into variants of a website. It stores a description of the bug, stating the reason it was introduced, and stores the link to the commit when the bug was introduced for each access to version control. Finally, the author that introduced the bug is also stored. Each bug is associated with a single website only.

**Test Framework and Language –** These classes exist in both the Task manager and Platform components. They dictate the support of the application for the different test frameworks and programming languages.

**Status –** enumeration, for the status of a submission. A Queued status mean the submission has been added to the queue of the message broker, however, has not yet been received by the code execution engine. A status of processing means the solution has been received by the code execution engine and is currently executing. A completed status means the execution has finished and a result generated, ready to be viewed.

3 CHOOSING THE DEVELOPMENT STACK

3.1 Dotnet

To implement the web APIs and the long-running background service, .NET 6.0 was chosen as the framework – a cross-platform, high-performance open source framework for creating web applications, which includes simplified development experience, more productivity & a unified block of libraries & platform. The language of choice was C#, which is a simple, modern, object-oriented, and type-safe programming language. [1]

ASP.NET Core supports two types of web APIs – minimal and controller based. Controllers in a web API are classes that derive from ControllerBase. Controllers are activated and disposed on a per request basis. [2]

Without delving into the details, the controller based approach was chosen due to the familiarity of the developer with this approach, and its object oriented design towards developing APIs. Using dependency injection is also easier and more intuitive with the controller based approach. This approach also provides a structured way to organize endpoints, models, and business logic within dedicated controller classes.

On the other hand, background tasks can be implemented as hosted services. A hosted service is a class with background task logic that implements the IHostedService interface. With ASP.NET Core, background services can quickly be put together using the worker service template from the Microsoft.NET.Sdk.Worker SDK and a reference to the Microsoft.Extensions.Hosting package.

The IHostedService interface defines two methods for objects that are managed by the host:

* StartAsync(CancellationToken)
* StopAsync(CancellationToken)

BackgroundService is an abstract base class for implementing a long running IHostedService. It implements the IHostedService and IDisposable interfaces. This class requires a single method, ExecuteAsync(CancellationToken), to be implemented by the subclass, which is called to run the background service.

When implementing a background service using the worker template from the SDK, it is important to remember to enable the server garbage collection . This can be done by adding the ServerGarbageCollection node to the project file as such:

<PropertyGroup>

<ServerGarbageCollection>true</ServerGarbageCollection>

</PropertyGroup>

Enabling server garbage collection has the following benefits:

* Efficient memory management: Automatically reclaims unused memory to prevent memory leaks and optimize resource usage.
* Long-term stability: Helps maintain stable performance in long-running services by managing memory over extended periods.
* Reduced maintenance: Minimizes the need for manual memory management, simplifying maintenance.
* Predictable behavior: Contributes to consistent and predictable application behavior.

[3]

Since hosted services are logging, configuration and dependency injection (DI) ready, it is however important to note that no scope is created for a hosted service by default, and hence, we must create a scope to consume a service.

3.2 Dapper

Dapper is an open-source object-relational mapping (ORM) library for .NET and .NET Core applications. The library allows developers quickly and easily access data from databases without the need to write tedious code. Dapper allows you to execute raw SQL queries, map the results to objects, and execute stored procedures, among other things. It is available as a NuGet package.

Dapper falls into a family of tools known as micro-ORMs. These tools perform only a subset of the functionality of full-blown Object Relations Mappers, such as Entity Framework Core, but Dapper is known for its speed and simple implementation compared to others.

Dapper has no real expectations about the schema of your database. It is not reliant on conventions in the same way as Entity Framework Core, so Dapper is also a good choice where the database structure isn't particularly normalized. [4]

With dapper, you can execute query and non-query commands, both synchronously and asynchronously. Querying data using Dapper is quite simple. You just need to provide the dapper select query and parameters, and after that, Dapper will automatically map the resulting columns to their corresponding properties in your model type.

Here is an example of using Dapper to perform multiple SQL statements in one query to get the submission context.

public async Task<SubmissionContext> GetSubmissionContext(int assignment\_id,int testing\_fw\_id)

{

var parameters = new {assignment\_id,testing\_fw\_id };

using (var connection = \_context.CreateConnection())

{

using (var multi = await connection.QueryMultipleAsync(Queries.GetSubmissionContext, parameters))

{

var skeleton = multi.ReadFirst<Skeleton>();

var websites = multi.Read<ContextItem>();

return new SubmissionContext

{

Websites = websites.ToList(),

code\_skeleton = skeleton

};

}

}

}

And the corresponding SQL query:

public const string GetSubmissionContext =

"SELECT code\_before, code\_after FROM assignment a " +

"LEFT JOIN a\_assignment\_skeleton aas ON a.assignment\_id = aas.assignment\_id " +

"LEFT JOIN skeleton s ON s.skeleton\_id = aas.skeleton\_id " +

"WHERE a.assignment\_id = @assignment\_id and s.testing\_fw\_id = @testing\_fw\_id " +

"SELECT hosted\_at as url, aaw.should\_pass FROM website w " +

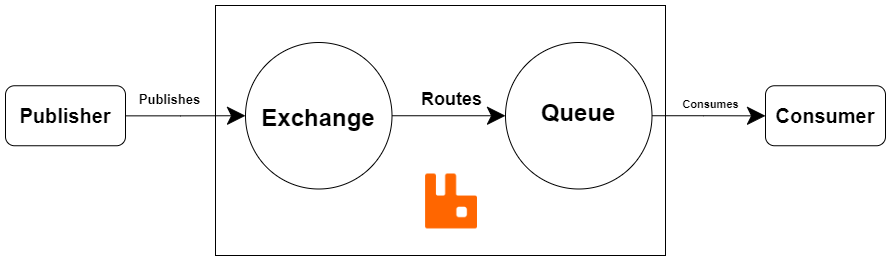
"LEFT JOIN a\_assignment\_website aaw ON w.website\_id = aaw.website\_id " +

"WHERE assignment\_id = @assignment\_id ";

3.3 RabbitMQ

RabbitMQ is an open-source and lightweight message broker which supports multiple messaging protocols. It can be deployed in distributed and federated configurations to meet high-scale, high-availability requirements. In addition, it's the most widely deployed message broker. [5]

RabbitMQ uses the AMQP protocol. The AMQP 0-9-1 Model has the following view of the world: messages are published to exchanges. Exchanges then distribute message copies to queues using rules called bindings. Then the broker either deliver messages to consumers subscribed to queues, or consumers fetch/pull messages from queues on demand.



AMQP exchanges can be of four types:

1. Direct exchange
2. Fanout exchange
3. Topic exchange
4. Headers exchange

Each of these exchanges can be declared for different use cases, and configured with a number of attributes. Details at <https://www.rabbitmq.com/tutorials/amqp-concepts>

**RPC pattern over RabbitMQ**

In our case, we want to return the result to the submission back to the user in minimal time, after the execution on a remote computer. For this, we will be using the Remote Procedure call or RPC pattern.

In general doing RPC over RabbitMQ is easy. A client sends a request message, along with a correlation Id and a server replies with a response message to a specific queue known as the ‘callback’ queue. The correlation Id is used to determine which request the response belongs to.

Such is the implementation of a callback mapper:

First, we declare a dictionary, which keeps track of the correlation Id and the corresponding TaskCompletionSource.

private readonly ConcurrentDictionary<string, TaskCompletionSource<TestResult>> callbackMapper = new();

Then, when publishing a message, we provide a random guid as the correlationId for the IBasicProperties.

var correlationId = Guid.NewGuid().ToString();

Finally, once a message is received from the code execution engine, we check for the corresponding TaskCompletionSource of the correlationId in the BasicDeliverEventArgs, and set the result.

consumer.Received += (model, ea) =>

{

if (!callbackMapper.TryRemove(ea.BasicProperties.CorrelationId, out var tcs))

return;

var body = ea.Body.ToArray();

var message = Encoding.UTF8.GetString(body);

var response = JsonSerializer.Deserialize<TestResult>(message);

tcs.SetResult(response);

};

In this way, the client acts as both the publisher and the consumer.

3.4 Reactjs, JQuery, EasyUI

React is an open source Javascript library for developing user interfaces. It is mainly used for building single page applications that are decoupled from the backend. It speeds up development by allowing us to create reusable UI components.

jQuery is a fast, small, and feature-rich JavaScript library. It makes things like HTML document traversal and manipulation, event handling, animation, and Ajax much simpler with an easy-to-use API that works across a multitude of browsers. [6]

EasyUI is a collection of user-interface components based on jQuery, Angular, Vue and React.

**Creating a window with Easyui**

To create a window for admins to manage contest, first, we have to declare a div in the index.html

<div id="winContest" title="Contests" minimizable='false' style="display:none;"> </div>

Then, once the DOM is ready for the javascript to execute, we call the following function:

function createContestWindow(event){

let win = $('#winContest');

if( win.css('display') != 'none') {

win.window('open')

}else{

win.window({

width:1000,

height:600,

modal:true

});

win.show();

}

}

It is important to note that when using Easyui jquery components with React, we must not change the root div which is managed by the react virtual DOM, This can cause unpredictable results, and hence, all the windows should be placed in a separate root element.

If you wish to create a new window each time the contest window is clicked, you have to generate the contest div on click, instead of placing is statically in the html. In this case, it is also important to remove the div from the DOM once the window is closed. This can be done by adding the following handler to the data-options attribute of the window div.

function closeHandler(event){

$(this).window('destroy')

}

4 DESIGN AND DEVELOPMENT OF THE APPLICATION

4.1 System architecture

The Component diagram in Figure 10 shows the different components involved in making up the system. This helps us break down the entire system into various high levels of functionality. Each component is responsible for one clear aim within the entire system and only interacts with other essential elements on a need-to-know basis, using the required protocols. The green component symbolizes that it’s an external component.

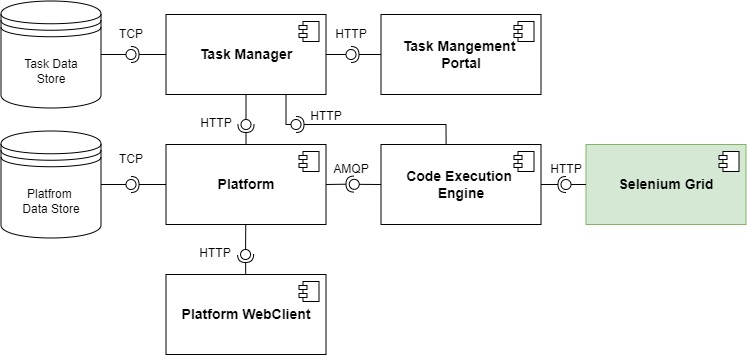
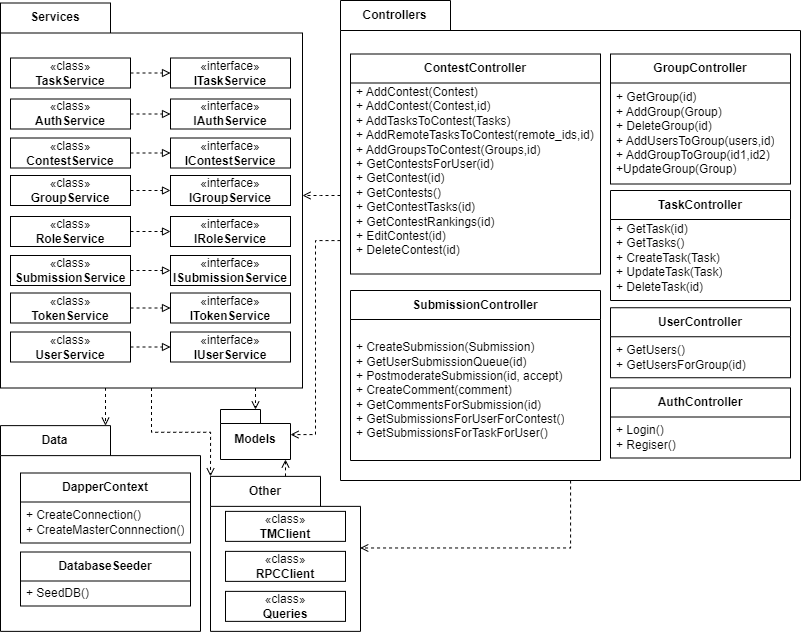


Figure 15 - Component Diagram for the system architecture

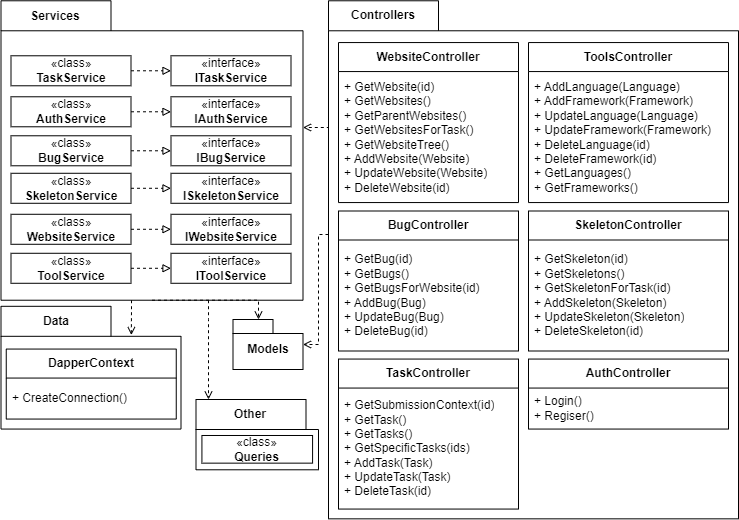
The entire system is composed of 6 components, along with 2 databases as infrastructure components. Each of these components is further described below:

**Platform –**Web API, written in .NET 6. This component is institution specific, and hence, has its own database. As part of the system, it performs authentication and authorization, provides an API for creating submissions, managing contests, groups, users, and tasks.



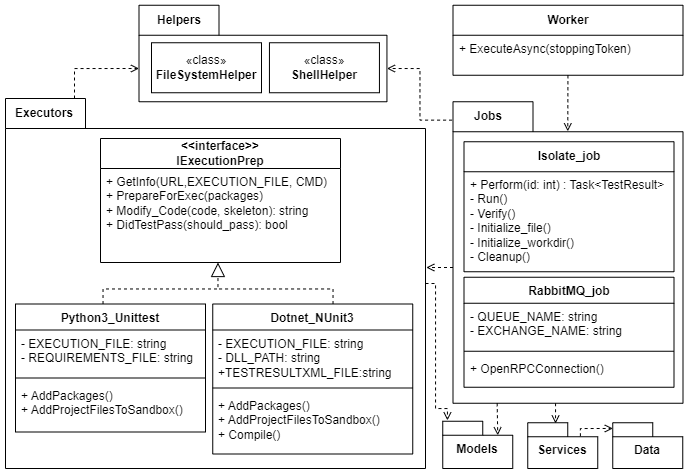
The platform has 6 controllers which provide API endpoints to perform actions relating to the main entities. These controllers, in turn, depend on their correlating services, which provide the business logic for the specific action. The controllers only interact with interfaces instead of the concrete implementations of the services, hence using dependency inversion. The services in turn depend on the Data package, which provides the interface with the database, using the queries defined in the Queries class. The TMClient provides endpoints that retrieve data from the Task Manager component, such as remote tasks, or submission context. The TMClient is implemented using the Refit library for interacting with the Task Manager API. The RPCClient helps the component connect to RabbitMQ and send and receive messages.

**Task Manager –** Web API, written in .NET 6. This component is central, and hence, communicates with all the different institution specific platforms. This component is mainly used by the maintainers of the task data store, and hence, this component has its own database too. It perform authentication and authorization for maintainers, and provides and API for managing Websites, Skeletons, Bugs, Tasks, and Tools.



The Task Manager has 6 controllers, which provide endpoints relating to the maintenance of the task data store. These controllers in turn depend on services which provide the business logic for these actions. The controllers use dependency inversion, and hence only interact with interfaces, which in turn have concrete implementations. The services perform database operations using the interfaces provided by the Dapper package, using the Queries class for the specific queries.

**Code Execution Engine –** Long-running service, written in .NET 6. The primary job of this component is to receive submissions, execute them with the help of Selenium Grid, and return the result to the calling platform. It receives the submission context from the Task manager based on the parent Id of the task.



In this component, an executor is a single class that provides all the functionality needed to execute a script written in a specific language and testing framework. Any new executors added to the system must inherit from the IExecutionPrep interface.

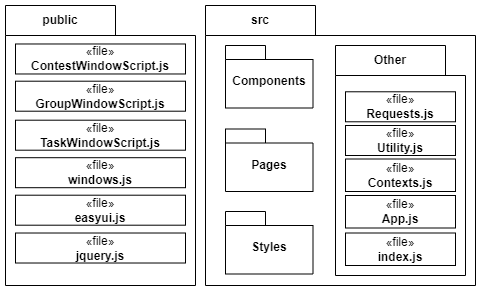
The RabbitMQ\_job performs all the functionality relating to receiving messages from and sending responses to the message broker. It receives a submission Id, and calls the perform method of the Isolate\_job class, passing the submission Id. When the result is ready, it returns the result.

The Isolate\_job class chooses the correct executor to run the script, creates a sandboxed environment, and runs the script against all the websites. It then produces a result file to be returned, and updates the database with the execution metrics. It communicates with the database through services, where the business logic is defined.

The ShellHelper class provides a single method, which allows the program to execute bash commands.

The FileSystem helper has multilple methods to help the program with manipulating files, changing directories etc.

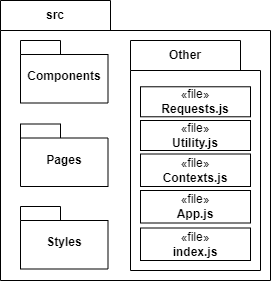
**Platform WebClient –** ReactJS application, which provides a user interface for the Platform component, for the students and teachers to works with.



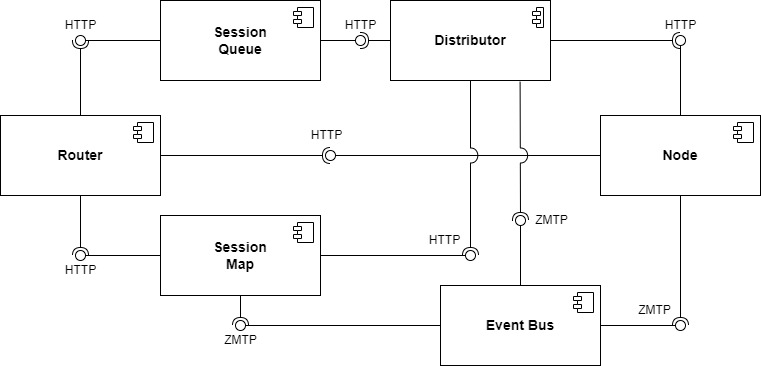
Besides multiple node packages, this component also makes use of two other libraries. Easyui.js is used to make the various windows for the admin functionalities. It needs jquery.js as a dependency. Jquery is also a dependency for DataTables, which is used to implement the submission queue.

Windows.js has functions which initialize the windows themselves, while the other window scripts initialize the elements within the specific windows for Contests, groups and tasks.

**Task Management Portal –** ReactJS application, which provides a user interface for the Task manager component, for the maintainers to work with.



**Selenium Grid –** External component, used for automating multiple different browsers, on multiple different operating systems, all in parallel. It does this by routing commands sent by the client to remote browser instances.



Let’s look into each component of Selenium Grid in more detail.

**Router**

The Router serves as the gateway to the Grid, accepting all incoming requests and directing them to the appropriate component.

When a new session request is received by the Router, it is routed to the New Session Queue.

If a request is associated with an existing session, the Router consults the Session Map to identify the Node ID where the session is active, and subsequently, the request is directly forwarded to that Node.

By distributing requests to the most capable component, the Router ensures efficient load balancing within the Grid, preventing unnecessary strain on any component not involved in the process.

**Distributor**

The Distributor is tasked with two primary duties:

1. **Node Registration and Capability Tracking**

A Node initiates its registration with the Distributor by dispatching a Node registration event via the Event Bus. Upon receiving this, the Distributor attempts to establish an HTTP connection with the Node to verify its presence. If the connection is successful, the Distributor registers the Node and maintains a record of the capabilities of all Nodes within the GridModel.

1. **New Session Queue Polling and Session Request Processing**

When the Router receives a new session request, it is directed to the New Session Queue and placed in line. The Distributor periodically checks the New Session Queue for any outstanding new session requests and identifies an appropriate Node where the session can be initiated. Once the session is established, the Distributor records the association between the session id and the Node executing the session in the Session Map.

**Session Map**

The Session Map serves as a data store that maintains the link between the session id and the Node executing the session. It aids the Router in directing a request to the appropriate Node. To do this, the Router queries the Session Map for the Node linked to a specific session id.

**Session Queue**

The New Session Queue holds all incoming session requests in a first-in, first-out (FIFO) sequence. It has adjustable settings for defining the request timeout and the interval for checking the timeout.

The Router places the new session request into the Session Queue and awaits a response. The New Session Queue consistently verifies if any request in the queue has exceeded its timeout, and if so, the request is promptly declined and removed.

The Distributor routinely verifies if a slot is open. If a slot is available, the Distributor retrieves the first corresponding request from the New Session Queue and tries to initiate a new session.

When the requested capabilities align with the capabilities of any of the available Node slots, the Distributor tries to secure the open slot. If all slots are occupied, the Distributor returns the request to the queue. If a request times out during the retry process or while being added to the front of the queue, it is declined.

Once a session is successfully established, the Distributor forwards the session details to the New Session Queue. This information is then relayed back to the Router, and ultimately to the client.

**Node**

A Grid has the capacity to house numerous Nodes. Each Node is responsible for managing the slots for the browsers available on the machine where it operates.

The Node self-registers to the Distributor via the Event Bus, and its configuration details are transmitted as part of the registration message.

By default, the Node automatically registers all browser drivers found on the machine's path where it operates. It also establishes one slot for each available CPU for Chromium-based browsers and Firefox. For Safari, only a single slot is created. With a specific configuration, it has the ability to run sessions in Docker containers or relay commands.

A Node solely carries out the commands it receives, without making evaluations, judgments, or controlling anything beyond the flow of commands and responses. The machine where the Node operates does not need to share the same operating system as the other components. For instance, a Windows Node may have the ability to provide IE Mode on Edge as a browser option, a feature not possible on Linux or Mac. A Grid can house multiple Nodes configured with Windows, Mac, or Linux.

**Event Bus**

The Event Bus serves as a communication path between the Nodes, Distributor, New Session Queue, and Session Map. The Grid does most of its internal communication through messages, avoiding expensive HTTP calls. When starting the Grid in its fully distributed mode, the Event Bus is the first component that should be started. [1]

# Bibliography

|  |  |
| --- | --- |
| [1] | З. Д. Олеговна, *РАЗРАБОТКА КЛИЕНТСКОГО ВЕБ-ПРИЛОЖЕНИЯ ДЛЯ ЦИФРОВОЙ ПЛАТФОРМЫ ОБУЧЕНИЯ ПРОГРАММИРОВАНИЮ <<CODEHEDGEHOG>>,* Tomsk, 2021. |
| [2] | «What is .Net?,» [В Интернете]. Available: https://dotnet.microsoft.com/en-us/learn/dotnet/what-is-dotnet. |
| [3] | C. w. A. w. A. Core. [В Интернете]. Available: https://learn.microsoft.com/en-us/aspnet/core/web-api/?view=aspnetcore-8.0&WT.mc\_id=dotnet-35129-website. |
| [4] | «Worker Services in .NET,» [В Интернете]. Available: https://learn.microsoft.com/en-us/dotnet/core/extensions/workers. |
| [5] | «What is Dapper?,» [В Интернете]. Available: https://www.learndapper.com/. |
| [6] | «RabbitMQ,» [В Интернете]. Available: https://docs.nestjs.com/microservices/rabbitmq. |
| [7] | «What is jQuery?,» [В Интернете]. Available: https://jquery.com/. |
| [8] | «Grid Components,» [В Интернете]. Available: https://www.selenium.dev/documentation/grid/components/. |