Master Test Plan

1. Evaluation Mission and Test Motivation

This Master Test Plan report mainly focuses on the test approach to the Find Your Job - Online Job Recruitment System. It contains the testing done to archive the mission of the system. The mission of this system is to provide a user friendly online application for managing job recruitment for the companies. So in order to give a better user experience, we have to design and test the system very thoroughly.

Since this is a web application that helps the people's future, it must contain exact details since a single bug or a security loophole can lead to a disaster. The User Interfaces and the system logic must maintain the quality in order to give the user what is expected by them. Since all the requirements must match with the system's functionalities a well-organized testing process must be carried out to check whether the system delivers what is expected.

The system is developed with respect to the MVC architecture. So the testing plan must include the testing methods for every Model, View and Controller.

The main objectives of the testing process are listed below:

- Finding as many bugs as possible and correct them before the system is being deployed to the users. A special thorough search for bugs will be carried out to ensure the confidentiality and the integrity of the system because the system deals with sensitive information like CVs of job appliers and Company details.
- Find important problems of design and implementation that poses a threat to the quality of the system and that does not meet user requirements. These problems will be corrected before the deliverance of the system.
- Identifying the risks that are associated with the project. Here the risks involving the project implementation are identified, tested and refactored before they happen. Risks like failing user acceptance can be forecasted by deploying a prototype among a little party and getting their feedbacks to change the system.
- The system must meet the quality standards and keep up with the standards of the existing systems. To achieve this a performance testing is done along with a user acceptance testing to check whether the system meets the required standards.
- The system must meet its requirements. So a proper testing process is carried out in order to check whether the system meets all the functional and non-functional requirements. A testing is done to check whether the system is moving away from the desired objectives, wasting time on useless requirements.
- A user acceptance testing is done to check whether the stakeholders are satisfied with the system that is built and to check whether the quality which is desired by the stakeholders is present in the system.
- The system should fulfill the process mandates. A testing process must be carried out to check the legal state of the system. Here every legal violation must be identified and resolved before the deployment of the system.
- The users must be advised about the testing too. When the user acceptance is carried out, the stakeholders must be notified that the system is a Beta version

2. Target Test Items

Since this Find Your System is an online web application, the system should have higher security and high quality user interfaces. The system should be using minimum resources and minimum database calls in order to perform well. As this application is based on the MVC architecture, the test items are much similar to the Models Views and Controllers. So The listing below identifies those test items: software, hardware, and supporting product elements that have been identified as targets for testing.

- User Interfaces:

  These are the views of the system. Here the quality of the views is taken into consideration with the time that the requested page is taken to load. Selenium IDE is used to test the user interfaces in this application.

- Data models and components:

  This is the most important part of the system since data models is basically the database of the system. So the database performance and its durability must be tested continuously. The database should be backed up well too.

- Functions:

  These are the controllers of the system. Since this is the business logic of the system, it must be thoroughly tested and validated in order to check whether the controllers meet the expected requirements. PHP Unit testing can be used to test the functions of the system.

3. Test Approach

The test approach presents the recommended strategy for designing and implementing the required tests. Here is the approach for testing for the Online Find Your Job System.

- Data and Database Integrity Testing

  The entities and the database connections were made by Doctrine ORM. When using Doctrine ORM, the database developer manually creates the database using the command prompt which can lead to a lot of bugs. So the database connections must be checked regularly if a change to the database is made.

  The performance of the database is tested using the MySQL workbench tools such as Performance dashboard and Performance Schema Reports.

- Function Testing

  According to the MVC architecture, Function Testing includes the testing done to the controllers in the system. Since the controllers contain the main functionality of the system, it must be regularly tested to check whether they meet the user requirements.

  PHPUnit is used for this function testing. For validating the data which are passed from the Views to the database through the controllers, Symfony validator service is being used.

- User Interface Testing

  User Interface Testing includes the testing done for the views of the system. This testing is done after the function testing because the functionalities must give the right business logic for the views to implement them.

  The UI Testing is done using the Selenium IDE which is integrated in Firefox. Form submissions and other UI functionalities are orderly tested using the Selenium IDE.

- Performance Profiling

  Performance Testing is very crucial since a lot of users may create accounts and have vacancies and applications. So a lot of database space is needed so the performance can be a considerable issue. Database performances are tested through MySQL workbench Performance Dashboard and Firefox performance tool.

- Load Testing

  Load Testing is the testing done for the system under different data loads. MySQL queries are tested and rewritten to minimize the load on the system.

- Security and Access Control Testing

  Security and Access Control Testing is done manually by checking entering the username and password combinations. The Symfony Debug toolbar is used to test whether the right user is logged onto the system and to test the roles assigned to the specific users meet the expected requirements.

- Failover and Recovery Testing

  This is also done manually to test how to recover from packet losses and database failures. Database is constantly backed up in order to recover quickly from the failures.

- Configuration Testing

  The system needs to run on different platforms and different servers. So the system must be tested and configured to run on different platforms like Operating Systems and should be able to run on many servers depending on the server the project is going to be deployed.

3.1 Testing Techniques and Types

3.1.1 Data and Database Integrity Testing

The databases and the database processes should be tested as an independent subsystem. This testing should test the subsystems without the target-of-test's User Interface as the interface to the data. Additional research into the Database Management System (DBMS) needs to be performed to identify the tools and techniques that may exist to support the testing identified in the following table.

Technique Objective:	Database creation is done using the Doctrine Object Relation Mapping tool for symfony. This is done through the command line interface, and the Doctrine system guides through the creation of the entities and creating the database. So this will enable testing automatically by giving the incorrect database entities.

Technique:	• Invoke every entity and relationship with Doctrine ORM along with the getters and setters for each and every attribute of the system. ORM notations are being used to connect the database and access the database. • Inspect the database to ensure the data has been populated as intended and all database events have occurred properly, or review the returned data to ensure that the correct data was retrieved for the correct reasons. • The functionality of the database can be tested by using PHPMyAdmin and using the reverse engineer tool in the MySQL Workbench.
Oracles:	The database access can be tested by showing the data and the fields that the data is injected in the same code the data is transferred to the database. This way the developer can know whether the system behaves as expected as it enters data with respect to the database attributes. But this has a probable failure when two similar kinds of tables are present in the database. This way the retrieved data may not be erroneous even though they are not in the expected table.
Required Tools:	• Doctrine ORM tool to generate tables. • PHPUnit framework for generating test cases • Symfony Framework to create relationships between tables • Mozilla Firefox for testing the application • PHPMyAdmin to view the database, tables and the content. • MySQL workbench to verify the database schema and relationships
Success Criteria:	• All the methods that are used in this system are very popular standard methods • All the testing that is done has a proper documentation maintained in order for the future reference. • Every test is guaranteed to produce a reasonable, satisfactory outcome and testing is continued until then.
Special Considerations:	• Database should be normalized in order to do the testing well • MySQL Workbench and PHPMyAdmin are observed manually for any data integrity violation and for any database design faults.

3.1.2 Function Testing

Function testing of the target-of-test should focus on any requirements for test that can be traced directly to use cases or business functions and business rules. The goals of these tests are to verify proper data acceptance, processing, and retrieval, and the appropriate implementation of the business rules. This type of testing is based upon black box techniques; that is verifying the application and its internal processes by interacting with the application via the Graphical User Interface (GUI) and analyzing the output or results. The following table identifies an outline of the testing recommended for each application.

Technique Objective:	Exercise target-of-test functionality, including navigation, data entry, processing and retrieval to observe and log target behavior. Here every functionality is tested with its business logic to check whether all the functions can fulfill all the functional and non-functional requirements.
Technique:	Here PHPUnit testing framework Symfony Validator is used to execute each use-case scenario's individual use case flows or functions and features, using valid and invalid data to verify that: • The expected results occur when valid data is used • The appropriate error or warning messages are displayed when invalid data is used • Each business rule is properly applied and each of them provides the expected user requirement • Correct data is entered to the forms and they are properly validated before writing them to the database. • The functionalities are only shown to the authorized people and only has access to the authorized people. • Backup the code before changing the functionality to make sure if something happens a recovery is possible The testing of the login - logout functionality of the system is done manually by checking the usernames and passwords. The symfony debugger tool is used to verify that the right user is logged on to the system with the right User Role.
Oracles:	Automated testing can be done by PHPUnit framework by providing assertions for the controller tests. Here we can create an input and determine the output based on the functionality of the function and then create an assert to check whether the actual outcome of the function is what is expected. Then we can change the functionality if the function according to result of the test.
Required Tools:	• PHPUnit Framework for testing the functions • Doctrine ORM to access the database within the functionality • GIT Hub for backing up the code • Symfony Debugger tool to check authentication
Success Criteria:	• All the tests carried out to the identified use cases are working properly with no errors. • All automated tests are guaranteed to give a reasonable and a validated result
Special Considerations:	There are special instances where certain use cases are not authorized to certain users. So it is better to create every user instance to check the functionality of the system.

3.1.3 User Interface Testing

User Interface (UI) testing verifies a user's interaction with the software. The goal of UI testing is to ensure that the UI provides the user with the appropriate access and navigation through the functions of the target-of-test. In addition, UI testing ensures that the objects within the UI function as expected and conform to corporate or industry standards.

Technique Objective:	Exercise the following to observe and log the standards conformance and target behavior: • Render web pages with the minimum memory and data usage and with the higher quality. • Web system should have higher user experience, which means understanding the system and controlling the system must be easier. • Index page should be eye catching and should explain what the system is all about • Web pages must be platform independent, which means they must run on a lot of devices with different platforms. • Web page tabs and buttons should give a good understanding about the actions and the content. • Buttons should render the exact required page, which means the routing should work without bugs. • Submitting forms should work as expected without errors.
Technique:	• Use test cases and PHPMyAdmin to check whether the form submissions work properly. Check whether the data added to correct fields. • Use Selenium IDE along with the fire bug tool for Firefox to generate web requests to see whether the User Interface behaves as expected • Use PhpStorm to check for any miss spelling renders and to check any errors in the view files. • Use Firefox Developer tools and Symfony Debugger tool to check the data usage and the performance of the rendering of the web pages. • User Experience is managed through User Acceptance Testing and getting their feedbacks on the User Interfaces.
Oracles:	Only form submissions can be automated for User Interface Testing, along with the miss spelling errors and routing errors. Other testing cannot be automated because the user behaves differently

Required Tools:	• Selenium IDE to test the requests send to the web page • PhpStorm IDE to debug the errors in the code and routing • Mozilla Firefox to test the performance • Symfony Debugger tool to check the memory usage
Success Criteria:	• All the pages and routes are working well. • There are no miss spelling or broken routes that fails the User Interface • The system is user friendly with an easily understandable user interface
Special Considerations:	Most of the user interfaces cannot be tested accordingly because the components like images are loaded at the runtime.

3.1.4 Performance Profiling

Performance profiling is a performance test in which response times, transaction rates, and other time-sensitive requirements are measured and evaluated. The goal of Performance Profiling is to verify performance requirements have been achieved. Performance profiling is implemented and executed to profile and tune a target-of-test's performance behaviors as a function of conditions such as workload or hardware configurations.

Technique Objective:	Exercise behaviors for designated functional transactions or business functions under the following conditions to observe and log target behavior and application performance data: System's performance under normal anticipated workload System's performance under anticipated worst-case workload
Technique:	• Take every functionality into consideration and run one at a time and measure the memory usage and the time taken to execute. • Do multiple transactions simultaneously and check the memory usage and the time taken to execute. • Run every transaction simultaneously and check memory usage and the time to execute. • Take multiple instances of users and do all the transactions for each and test the above three steps for the whole system.
Oracles:	The results of automated tests are taken into consideration while executing the functionalities to check the performance. But this results are platform dependent and also depends on the internet connection of the user.

Required Tools:	• MySQL Workbench Performance dashboard to test the performance of the database. • Symfony Debugger Tool to check the memory usage of the application. • Windows Task Manager to check the memory usage of the computer system.
Success Criteria:	The technique supports testing: • Single Transaction or single user: Successful emulation of the transaction scripts without any failures due to test implementation problems. • Multiple transactions or multiple users: Successful emulation of the workload without any failures due to test implementation problems.
Special Considerations:	Comprehensive performance testing includes having a background workload on the server. There are several methods that can be used to perform this, including: • Use multiple physical clients, each running test scripts, to place a load on the system. • Use MySQL Workbench Performance Dashboard to directly measure the performance of the database • Increase the load incrementally and check the performance of the system and the database. Performance testing should be performed on a dedicated machine or at a dedicated time. This permits full control and accurate measurement. The databases used for Performance Testing should be either actual size or scaled equally

3.1.5 Load Testing

Load testing is a performance test that subjects the target-of-test to varying workloads to measure and evaluate the performance behaviors and abilities of the target-of-test to continue to function properly under these different workloads. The goal of load testing is to determine and ensure that the system functions properly beyond the expected maximum workload. Additionally, load testing evaluates the performance characteristics, such as response times, transaction rates, and other time-sensitive issues

Technique Objective:	Exercise designated transactions or business cases under varying workload conditions to observe and log target behavior and system performance data. This includes: • Observation of the behavior of the system under normal workload. • Observation of the behavior of the system under worst case workload • Observation of the behavior of the system with a considerable amount of concurrent users under normal and worst case workload.
Technique:	• Use test scripts and Symfony Debugger tool to check the time taken to load the requested pages and the memory usage. • Use PhpStorm to modify the code to increase the number of transactions to increase the load on the system. We can use loops for this scenario. • Should test the systems daily weekly and monthly for peek loads. User acceptance testing is useful here. • Workloads should include spiky Peaks and regular peaks use Selenium IDE to create a large number of web requests to test the load. • The workload variations must be tested in different environments and different platforms too.
Oracles:	The system's performance may vary on different platforms. This cannot be achieved by means of manual terms because it is not practically possible to wear down a computer. So the code itself must be updated and changed to apply a huge load on the system.
Required Tools:	• PHPUnit framework to test the load on the functionalities • PhpStorm to modify the code for applying the load • MySQL Workbench to measure the database performance.
Success Criteria:	• Exhausting the system with the worst case scenario will help to determine the system performance boundaries and requirements. This will help to determine whether the system can behave as expected for an expected workload. • Average transactions on single user and multiple users should work perfectly without any errors. • System should be able to withstand spiky peek loads and regular peeks without failing the system.
Special Considerations:	• Load testing should be performed on a dedicated machine or at a dedicated time. This permits full control and accurate measurement. • The databases used for load testing should be either actual size or scaled equally • Database Indexes must be used wherever necessary in order to avoid a performance bottleneck in the database while retrieving large datasets. • Should avoid Database queries with multiple joins because they automatically pose a huge workload on the system • Create virtual users to load and simulate many clients and check the performance of the system for at least hundred users. Remote Terminal Emulation tools are also used load the network with workload.

3.1.6 Security and Access Control Testing

Security and Access Control Testing focuses on two key areas of security:

- Application-level security, including access to the Data or Business Functions
- System-level Security, including logging into or remotely accessing to the system.

Based on the security you want, application-level security ensures that actors are restricted to specific functions or use cases, or they are limited in the data that is available to them. In this Find Your Job System, every job seeker and job recruiter can create and update accounts but only managers can delete them.

System-level security ensures that only those users granted access to the system are capable of accessing the applications and only through the appropriate gateways. System level security ensures that application level can only perform certain tasks.

Technique Objective:	Exercise the target-of-test under the following conditions to observe and log target behavior: • Application-level Security: An actor can access only those functions or data for which their user type is provided permissions. This ensures that a user cannot enter the URL and access the data without having the access to them. • System-level Security: Only those actors with access to the system and applications are permitted to access them. This will guarantee that Application Level can only perform only certain tasks and an intruder who broke into the application level cannot get to the database level without the system access level.
Technique:	• Application-level Security: Identify and list each user type and the functions or data each type has permissions for. ▪ Create tests for each user type and verify each permission by creating all the functionalities that is specific to each user type. ▪ Give warnings and forbidden messages when a user tries to access a function which is not authorized for that user ▪ Modify user type and re-run tests for same users. In each case, verify those additional functions or data are correctly available or denied. ▪ System-level Access: Create an account granting System Level Access. Then write database queries to access and change the database. Test accessing them from a user level access account.

Oracles:	An automated testing method for testing the access cannot be created. All the testing must be done manually using Symfony Debugger tool. Since there are always some security loopholes that are left unanswered, the system is always vulnerable to attacks. But the security provided by symfony security will be sufficient for this system since this application does not handle highly sensitive data.
Required Tools:	• Symfony debugger tool to test the user login and access roles. • PHPUnit to test the forbidden and access denied messages • SQL Map tool to test the system for SQL Injection.
Success Criteria:	System will log on the user to the system granting the relevant privileges to that user If a user tries to access a functionality which is not the specific user is not granted access, then access denied message is given to the user.
Special Considerations:	The specific access levels are assigned to the job recruiter and the job seeker accounts, so the user is granted the specific privileges when they create the account for themselves. SQL Injection security loopholes can be neglected since Doctrine ORM manages the database access. So no SQL query is needed to access the database. Doctrine Query Builder safely passes the SQL queries to the database

3.1.7 Failover and Recovery Testing

Failover and recovery testing ensures that the target-of-test can successfully failover and recover from a variety of hardware, software or network malfunctions with undue loss of data or data integrity.
For those systems that must be kept running failover testing ensures that, when a failover condition occurs, the alternate or backup systems properly "take over" for the failed system without any loss of data or transactions.
Recovery testing is an antagonistic test process in which the application or system is exposed to extreme conditions, or simulated conditions, to cause a failure, such as device Input/output (I/O) failures, or invalid database pointers and keys. Recovery processes are invoked, and the application or system is monitored and inspected to verify proper application, or system, and data recovery has been achieved

Technique Objective:	Simulate the failure conditions and exercise the recovery processes (manual and automated) to restore the database, applications, and system to a desired, known, state. Program failures may occur due to many reasons. The following types of conditions are included in the testing to observe and log target behavior after recovery: • Power interruption to the client or to the server • Communication interruption via network servers due to network failures • Interruption, communication, or power loss to DASD (Dynamic Access Storage Devices) and DASD controllers • Incomplete cycles such as an interrupted transaction, filtering process while executing a query interrupted, data synchronization interrupted • Invalid database pointers or keys which will interrupt the controllers and database flow • Invalid or corrupted data elements in database due to corrupted hard disks • System Failures due to high workload on the system
Technique:	The tests already created for Function and Business Cycle testing can be used as a basis for creating a series of transactions to support failover and recovery testing, primarily to define the tests to be run to test that recovery was successful. • Power the PC down if a power interruption to the client PC happened. • Simulate or initiate power down procedures for the server if the sever gets a power loss. • Simulate or initiate communication loss with the network if an interruption to the network servers occurs. Here communication wires are physically disconnected or network servers or routers powered down. • Simulate or physically eliminate communication with one or more DASDs or controllers if DASD and DASD controllers are interrupted or a communication loss or a power loss happened. • Transactions are rolled back to the previous state if an interruption or a power loss occurs in the middle of a transaction Additional transactions should be executed once the above conditions or simulated conditions are achieved and, recovery procedures should be invoked when the second test point state is reached. Database processes are aborted or prematurely terminated if a cycle is interrupted or became an incomplete cycle. The testing method is similar to the above. Testing for the following conditions requires that a known database state be achieved. Several database fields, pointers, and keys should be corrupted manually and directly within the database using database tools and additional transactions are executed using the tests from Application Function and Business Cycle Testing and full cycles executed. Backing up of the system/database should also be tested by reloading the system/database and seeing if it works fine.

Oracles:	This Failure and recovery testing needs some functions to be perform manually. Powering down the PC or the server manually in the middle of a transaction by switching them off or unplugging to test the results can be categorized into those conditions. But testing for huge workload can be automated but as mentioned previously, the code must be modified accordingly to test it. Most importantly the database and the system should be backed up occasionally. This can be automated or done manually. The backing up can be set to be done once a week to be ready for a data corruption or a system failure.
Required Tools:	• Backup and recovery tools such as PHPMyAdmin and MySQL Workbench • External power supplies such as UPS devices. • External hard disks for backing up data • Installation monitory tools such as Task Manager
Success Criteria:	• Successful recovery of data on a Client or a Server power down situation. • Successful rollback of the operation when an interruption occurs in the middle of a transaction. • Successful backing up of database and successful recovery of database if the backed up database is needed to be reloaded.
Special Considerations:	• Recovery testing is highly intrusive. Procedures to disconnect cabling, simulating power or communication loss may not be desirable or feasible. Alternative methods, such as diagnostic software tools may be required. • The Failure and recovery testing needs a lot of attention since the system failure for a very short period of time can be highly costly. So the recovery testing must be done in a proper way. • Resources from the Systems or Computer Operations, Database and Networking groups are required. • These tests should be run after hours or on an isolated machine to get the perfect results.

3.1.8 Configuration Testing

Configuration testing verifies the operation of the target-of-test on different software and hardware configurations. In most production environments, the particular hardware specifications for the client workstations, network connections, and database servers vary. Client workstations may have different software loaded at any one time, many different combinations may be active using different resources

Technique Objective:	Exercise the target-of-test on the required hardware and software configurations to observe and log target behavior under different configurations and identify changes in configuration state. Configure the system to run on minimum memory usage and minimum data usage but with higher performance because this system does not need higher graphics.
Technique:	• Use Function Test scripts to test the functionalities and database connections and database queries. • Test the performance of the software while working with various non-target-of-test related software, such as Microsoft Excel and Word applications, either as part of the test or prior to the start of the test. • Execute selected transactions while interacting with other target-of-test and the non-target-of-test software processes. • Repeat the above process, minimizing the available conventional memory on the client workstation.
Oracles:	Since this is a web application, configuration testing is not very much important because the Symfony Framework and the Doctrine ORM does the configuration part for the system minimizing memory usage and the data usage. The memory usage can be seen by the Symfony Debugger toolbar by running the application on a localhost server.
Required Tools:	• Apache Wamp Server with PHP and MySQL enabled • MySQL workbench to configure the database • Symfony Debugger toolbar to test the memory usage
Success Criteria:	The system runs with all the user requirements with a user friendly environment and a better user experience while consuming only a reasonable amount of data usage and memory usage.
Special Considerations:	If the configuration costs a lot of resources than the benefit of that configuration, that configuration is avoided. If the configuration's changes to the system does not create a reasonable impact, those configurations are neglected. Since this is a web application, unlike in games and desktop applications, users are not allowed to do configurations on their own to update the system.

4. Deliverables

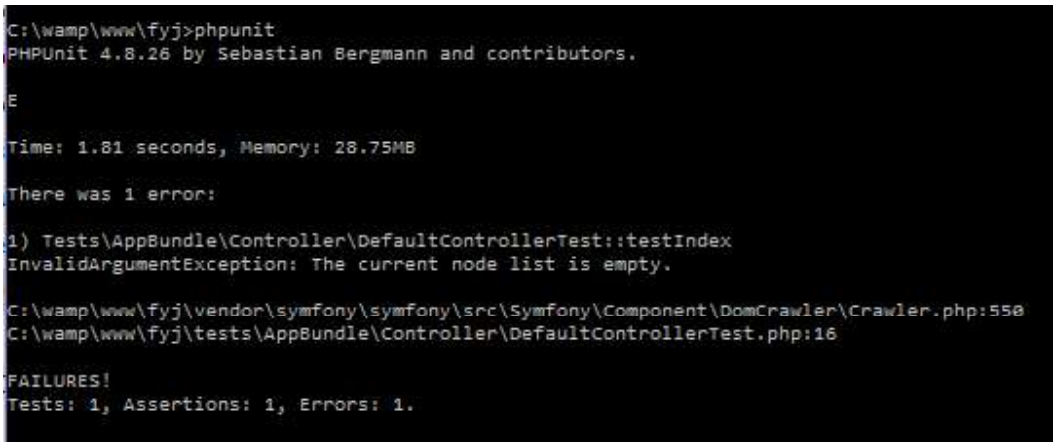
In this section, various artifacts that will be created by the test effort that are useful deliverables to the various stakeholders of the test effort are taken into account. Here are the deliverables that give direct, tangible benefit to a stakeholder and those by which you want the success of the test effort to be measured:

- Test logs
- User Interface Test Details
- Code Inspections
- Database performance statistics
- Browser Performance Details
- Symfony Debugger tool details

4.1 Test Evaluation Summaries

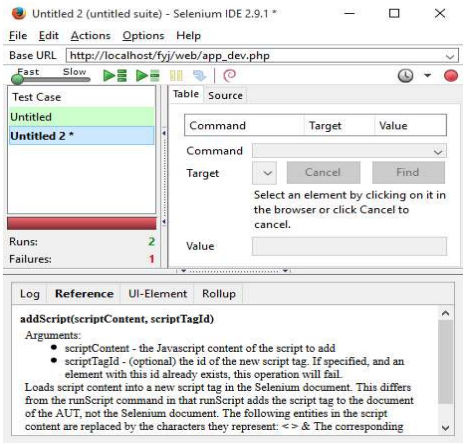
- Test Logs

These are the detailed logs that give the outcome of every test that is written for the functionalities. If even a single test log is given the output as a failure, the that functionality is recorrected and all the tests are run again and the test logs are checked for faults. Test logs will be provided very often soon after a new functionality is added or a change to an existing functionality is made.



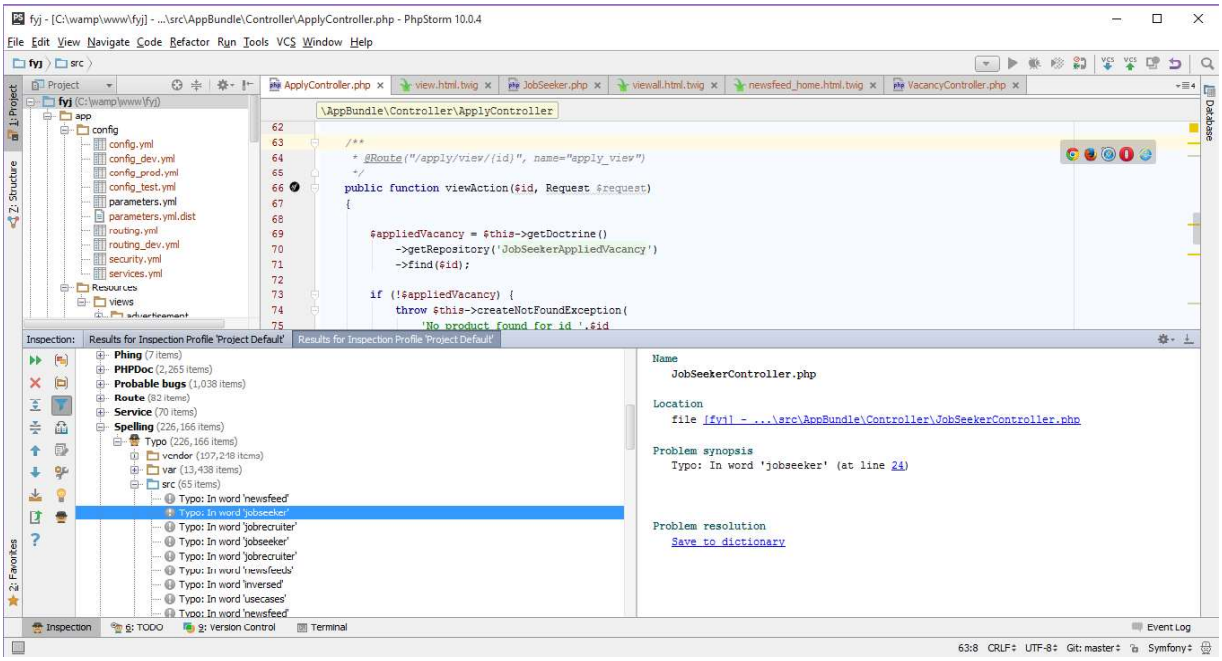
- User Interface Test Details

The User Interface is tested using the Selenium IDE and the Fire Bug tool for Firefox. Tests will be carried out to check for the faults in form submissions and working actions. User Interface Test Details will be provided after the system is done coding and is being checked for User Interface errors.



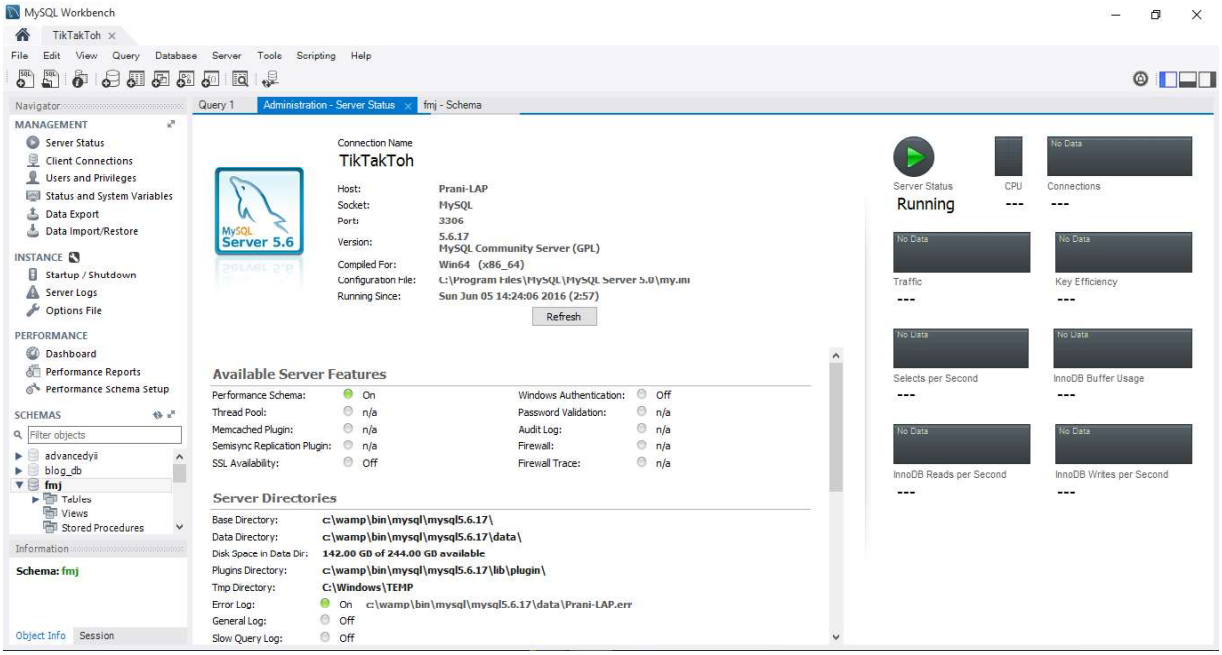
- Code Inspection Tool:

Code Inspection details can be got after running the inspect code test in the PhpStorm. This will notify about the spelling mistakes, empty tags, unused items to help to refactor the code for better performance. These details will be provided for about once a week and after the whole project ends.



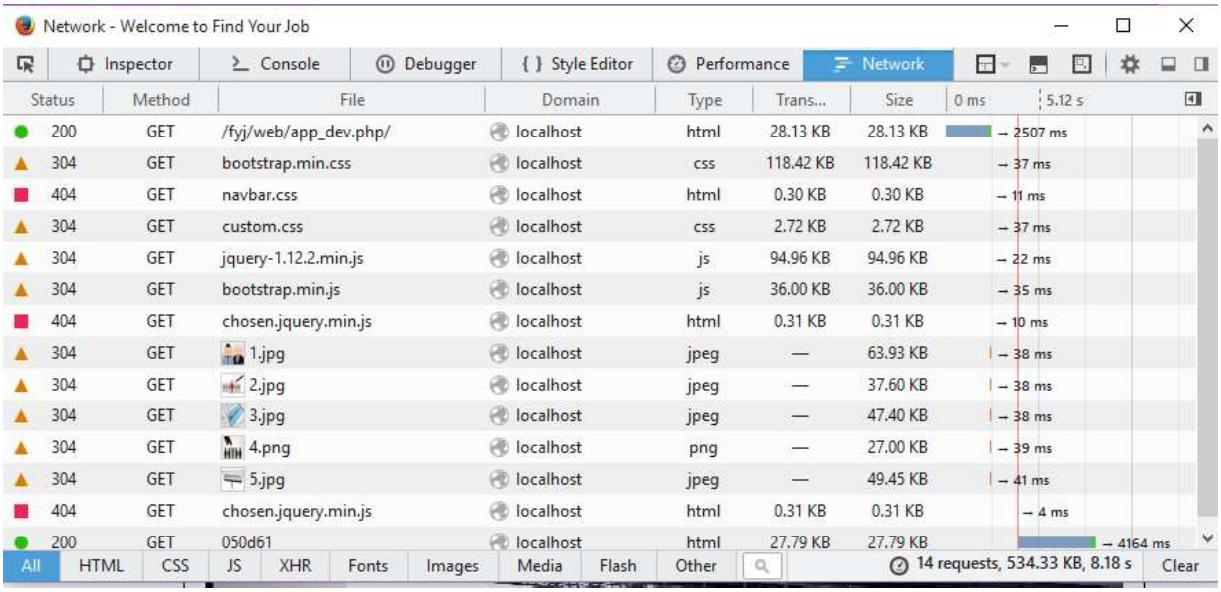
- Database performance statistics

Database performance statistics can be a very good method to check the database performance. It checks the memory usage for the database and the size of the database. Database performance statistics are provided after each database query made. Furthermore, DB performance details are provided after deploying the system, to check the storage of the system.



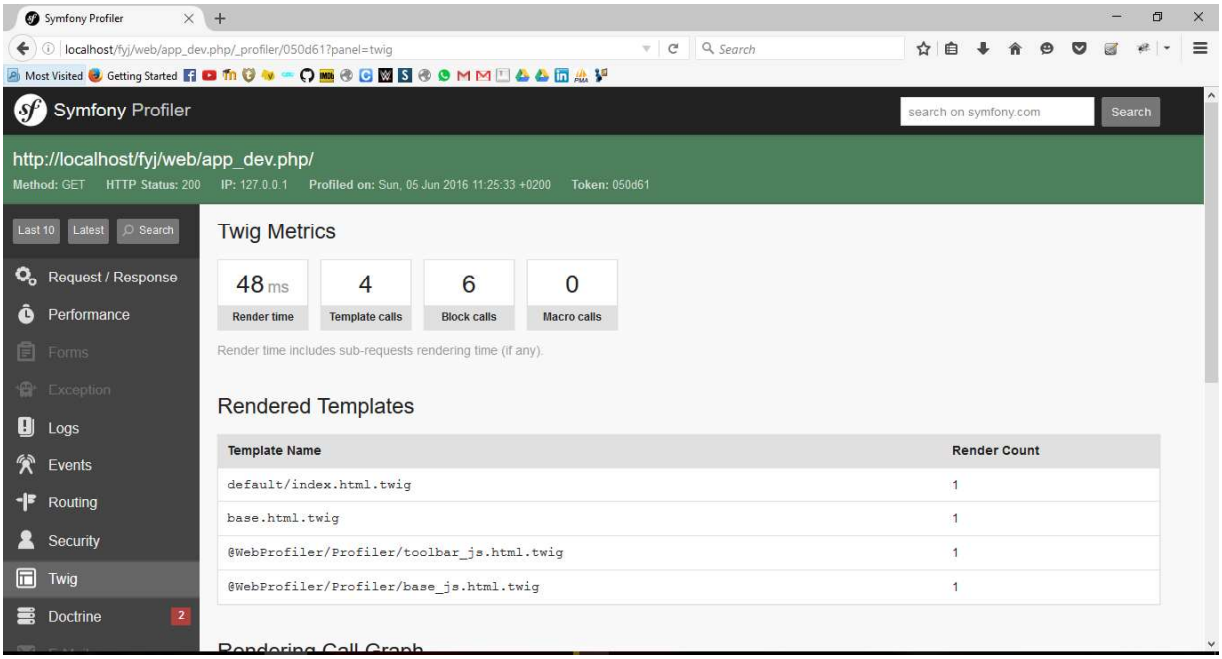
- Browser Performance Details

Browser performance details are checked to test the memory usage for the browser while the application is running as well as the memory usage for the application. The size of the page loaded can be also checked by the browser performance details, so that the data usage for the application can be calculated. These Details are provided if a major DB query is made or a page with a lot of images is being loaded to the screen.



- Symfony Debugger tool details

Symfony Debugger tool is used to calculate the memory usage for the sole application. Also the time taken for a page to load from server to the user's screen can be checked by this tool too. This toolbar is always present while developing the system, so these details can be acquired whenever needed.



4.2 Reporting on Test Coverage

Test Logs will be provided daily in the development stage of the system since unit testing is done regularly after every different made to a functionality. User Interface test and Code inspections will be checked once a week in the development phase of the system, then at the user acceptance testing phase, User Interface is tested regularly with the feedbacks from the users and tested daily.

Database performance is checked only if a large query is executed, but once the application is deployed, the database capacity against its performance is checked once a week and reports are made accordingly. Browser performance details and Symfony debugger details are only checked for pages with a lot of images are loaded into the browser. In Find Your Job system, these reports are made only when the index page is developed because it is the only page that consists images.

A generic report for the testing details contains the following:

- Date
- Logged in user
- Test case field
- No. of test cases executed
- No. of pass
- No. of fail
- Pass percentage
- Fail percentage
- Comments

These reports must be made for every successful and unsuccessful test case and an extra effort should be put to the test cases of the use cases and non-functional requirements like security.

5. Risks, Dependencies, Assumptions, and Constraints

Risk	Mitigation Strategy	Contingency (Risk is realized)
Prerequisite entry criteria is not met.	• Developer will define the prerequisites that must be met before Load Testing can start. • Users will endeavour to meet prerequisites indicated by Developers.	• Meet outstanding prerequisites • Consider Load Test Failure
Test data proves to be inadequate.	• Users will ensure a full set of suitable and protected test data is available. • Developer will indicate what is required and will verify the suitability of test data.	• Redefine test data • Review Test Plan and modify • components (that is, scripts) • Consider Load Test Failure
Database requires refresh.	• Admin or the Developer will endeavour to ensure the Database is regularly refreshed as required by Users	• Restore data and restart • Clear Database
Data is corrupted	• System Admin backs up the database regularly in order to prevent data losses	• Restore the latest backup to the database server

6. References

- PHPUnit Testing Framework available at https://phpunit.de/
- Selenium IDE available at https://addons.mozilla.org/en-US/firefox/addon/selenium-ide/
- PHPMyAdmin available at https://www.phpmyadmin.net/
- Wamp Server available at http://www.wampserver.com/en/