Software Quality Deliverable 2.3

**Parking Garage** \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Fixed/Improved Analysis Design

*[we used the same design*​*!*]

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Quality Report

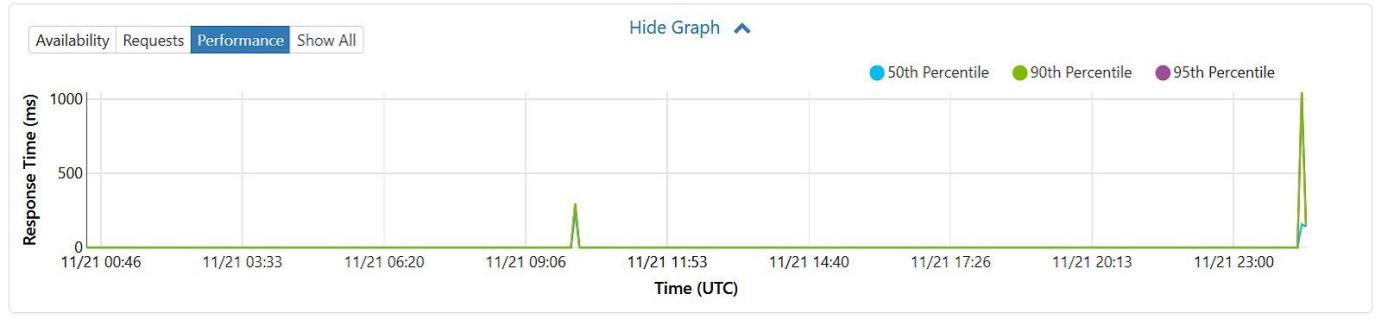
**Product Quality Metrics**

**Quality:** ​Availability

**Goal:**​ The system shall work continuously.

**Metrics:**​ The system should have 99% availability on the day.

**Validation Results:** ​According to the Azure portal, there were no failed requests, either to the web front-end or to the SQL database backend. Most responses were returned from the server on the order of milliseconds, though there were some latency spikes on the order of seconds at the 95th percentile. No timeouts were observed under normal load. Additionally, Azure is configured to dynamically scale up or down various resources under load. Alert emails are sent to service maintainers for upcoming maintenance and high CPU usage.

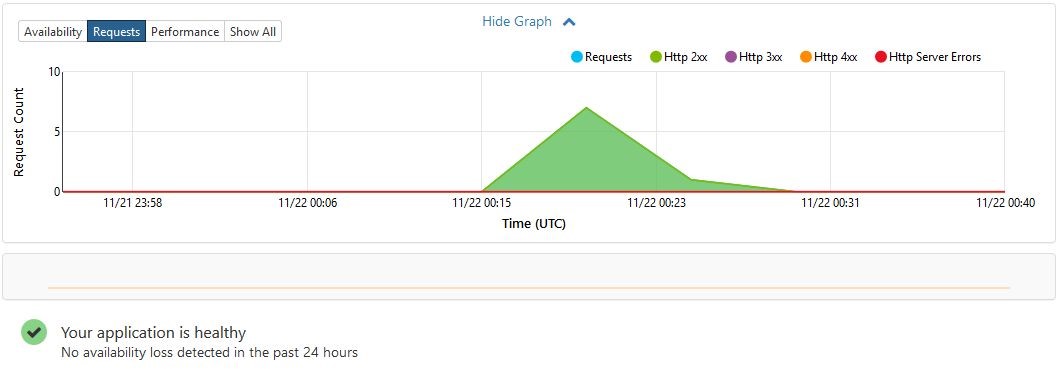


**Quality:** ​Reliability

**Goal:** ​The system shall work correctly at all times.

**Metric:**​ The error rate for operations should be below 1%.

**Validation Results:** Azure web portal shows 100% availability for the backend SQL database. For the web app itself, no requests returned internal service errors.



During the prototyping phase, some user queries incorrectly returned HTTP 400 error codes due to a bug in how the current list of checked out books were retrieved from the database. Once that bug was fixed, the 400 errors stopped.

**Quality:**​ ​Robustness

**Goal:** ​The system should handle a variety of inputs.

**Metric:** ​There should be zero errors that are a function of input data.

**Validation results:** All stories where a user can enter data were tested on Unicode inputs by hand and in test, with no failures. This was expected, given that C# strings support Unicode by default and all string fields in the database are specified as NVarChars which also support Unicode. Excessively long input strings could generate failures for librarians adding a book to the database - some validation logic should be added for such inputs.

**Kailun 03/31**

**Quality:** Learnability

**Goal:** The system should be easy to learn and to use.

**Metric:** A new user should be able to use the system less than a day’s training.

**Validation results:** new users of the system learned in approximately half hours how to use all of the functionality of the prototype: namely, registering, logining in, and making a reservation.

**Quality:** Usability

**Goal:** The system should be user friendly.

**Metric:** There should be no more than three complaints from users when trying to learn the system.

**Validation results:** Users were dissatisfied with the lack of detail in register. When you fail to register, the system does not notice any reason. Work should be scheduled to add the necessary details.

**Quality:** Efficiency

**Goal:** The system should quickly solve the intended problems.

**Metric:** The system should finish all data query and manipulation in less than one second.

**Validation results:** Most responses were returned from the server on the order of seconds. There are certain operations in the Database Accessor that could be done asynchronously where they are currently synchronous which could improve end to end latencies.

Yu Nong 3/30

**Quality:**​ ​Integrity

**Goal:** They data should be integrated for more than 3 years

**Metric:**​ The data leakage accidents should be less than 1 in 3 years

**Validation results:** There has not been full audit done. The connection between the web server and the database management system is based on TLS 1.3, which strongly encrypts the data between the web server and the database management system. However, the password for logging in the database system is short and easy-to-guess. Moreover, the web application in our system is based on HTTP protocol directly without any encryption. All the packets sent between the clients and the server can be easily captured and parsed. The system has no SQL injection checking to the content of submitted forms. Thus, An SQL injection attacking is possible to our system.

**Quality:**​ ​Portability

**Goal:** The system should be easily moved to a new platform

**Metric:**​ The system should work correctly on different devices, operating systems, database systems.

**Validation results:** This is a web-based application. All devices and operating systems which have a web browser can access the system without any issues. All queries and operations to the database system are based on standard SQL statements. There is no issue for different relational database systems. However, the system cannot moved to non-relational database systems without modifying the database query and operation statements.

Wen Li 4/5

**Process Quality Metrics**

**Quality:**​​Maintainability

**Goal:** ​The system should be easy to maintain and it should be ensured changes would not lead to defects.

**Metric:**

# <1> The program should not contain lexical and design anti-patterns[3][6], and it should be well modularized, and follow the principle of high cohesion and low coupling.

<2> the program should contains error correction mechanism, such as automated test unit to ensure each new change would not break down previous functions.

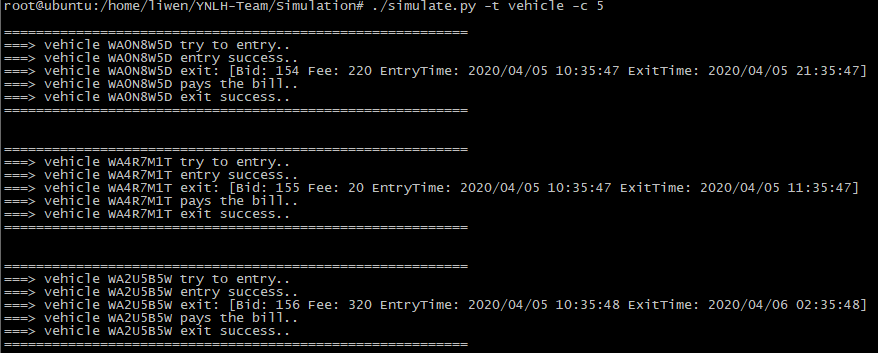
<3> the program should be implemented with low cyclomatic complexity.

**Validation results:**

# <1> we utilize Spring-MVC as our base model, the whole design is layered, it is easy to integrated new features.

<2> we utilize Junit as our unit test tool to ensure correction of functions, which also helps avoid defects brought by changes during maintaining.

<3> we utilize automated test script (python) to construct error correction mechanism, which helps avoid defects at system (requirement) level. (an example showed as below)



**Quality:**​​Testability

**Goal:** ​The system should support automated test and be easily testable.

**Metric:**

<1> Test cases should be complete, including functional cases and non-functional cases.

<2> Feature correction metric: each feature (requirement) should be tested through manual operation and automated test script, including normal/abnormal cases.

<3> Non-feature correction metric:

The program should provide performance test script to verify the throughput.

The program should provide stress testing script to test [system](C:/Users/liwen/AppData/Local/youdao/dict/Application/8.9.0.0/resultui/html/index.html" \l "/javascript:;) [stability](C:/Users/liwen/AppData/Local/youdao/dict/Application/8.9.0.0/resultui/html/index.html" \l "/javascript:;).

**Validation results:**

<1> we integrate manual test operations in an testing-purpose web page, which is easy to use.

<2> we provide python scripts to test all function/non-function features.

# Quality Goals

*See the table of quality goals below.*

|  |  |  |  |
| --- | --- | --- | --- |
| Product Quality | Quality Goals | Quality Metrics | Strategy |
| Product Operation |  |  |  |
| Correctness | The number of correctness issues should be less than 5 | The functionality issue reports should less than 5 in the whole lifecycle | Provide beta version to allow customers to experience and provide feedback |
| Reliability | The system shall work 99% accurately | The crashing frequency should be less than 3 per year | Stress testing |
| **Learnability** | The system should be easy to learn and to use. | A new user should be able to use the system less than a day’s training | See the detailed description above. |
| Integrity | They data should be integrated for more than 3 years | The data leakage accidents should be less than 1 in 3 years | Unit test on normal cases and boundary cases |
| **Efficiency** | The system should quickly solve the intended problems. | The system should return search results in less than one second | See the detailed description above. |
| **Usability** | The system should be user friendly. | The system should have well designed interface. | See the detailed description above. |
| Product Revision |  |  |  |
| Maintainability | The system should be easy to maintain and it should be ensured changes would not lead to defects. | *See the detailed*  *description above* | Code Review,  Static code analysis tools  Error correction tools |
| Flexibility | The system can be easily changed | The changing in some parts should not affect the whole running. | Utilize Spring-MVC model |
| Testability | The system should support automated test and be easily testable. | *See the detailed*  *description above* | Mutation and Unit testing tools  Automated test scripts |
| Product Transition |  |  |  |
| Reusability | Parts of the system should be able to be reused in another system | Modules in the system can be used in another system in less than 1 week configuring | Standard interface design and Objected Oriented Design |
| Portability | The system should be easily moved to a new platform | The system should work correctly on different devices, operating systems, database systems | Standard interface design |
| Interoperability | The system should be able to interaction with other systems | The system should be able to work with other data management systems | Standard interface design |