**一張含有 標誌 的圖片

自動產生的描述COMP S350F**

**Software Engineering**

**Test Plan Document**

**Group 17**

**Group members**

Mick LEE Kwun Sy – 12888141

CHUNG Fuk Sheng Ford – 12894147

Shaw CHEN Jiawei – 12790696

Elvis ZHAO Tianyu – 12807228

Niko CHEN Zhenyuan – 12814649

Context

1. DOCUMENTATION OF CHANGES
   1. Changes in Functions
   2. Changes in System
2. TEST PLANS
   1. Validation testing

2.1.1 User login

2.1.2 Student Information management

2.1.3 Students check their grades

2.1.4 Teachers modify scores

2.1.5 Logout

2.2 Defect testing

2.2.1 Student grade detection test

2.2.2 Password length detection test

2.2.3 Duplicate course selection detection test

2.2.4 Validation code time limit detection test

2.2.5 Password secondary validation test

2.3 Development test

2.3.1 Divide the system into components

2.3.2 Divide the components into units

1. **DOCUMENTATION OF CHANGES**

As mentioned in the system modeling document before, we will provide another document to list the changes in requirements and more. However, rather than listing out the changes one by one, we think it is better to use some updated abstract models to represent the changes and to also showcase what the system currently looks like.

**1.1 Changes in Functions**

The use-case diagrams attached in the system modeling document before already showcased the functions derived from the updated functional requirements.

Please zoom in to see the details.

A diagram of a user interface

Description automatically generated

*Figure 1 functions available to students*

A diagram of a user system

Description automatically generated with medium confidence

*Figure 2 functions available to teaching staff*

A diagram of a diagram

Description automatically generated

*Figure 3 functions available to system administrators*

**1.2 Changes in System**

Instead of following the original requirement to build almost every component of the system up from scratch, we decided to implement a system level reuse to catch up with the tight schedule.

The system we have reused is a student management system originally built for colleges in mainland China, and our job is to understand the architecture of the system and to make adaptations such that the system delivers the expected functionalities in our client’s environment (translating UI, modifying functions etc.).

After performing some analysis on the source-code of the system, we found that the system follows a layered architecture as below.

A computer server diagram with words

Description automatically generated

*Figure 4 architecture of the system*

To cut a long story short, this kind of architecture ensures the independence of each layer, such that the impact of modifying components in a layer can be minimized.

The updated ER diagram below shows the objects in the system after adaptations are made.

A diagram of a computer

Description automatically generated

*Figure 5 ER diagram of the system*

The updated sequence diagram below shows how the components in the system communicate with each other for a specific operation, after adaptations are made.

[A diagram of a diagram

Description automatically generated](file:///C:\Users\cklks\Pictures\Image_20231209233151.jpg)

*Figure 6 sequence diagram of “modify student’s score” operation*

***Note: diagrams 6 and 7 are too big to be put in this word document so we have uploaded the two figures to our GitHub repository separately, please check out the folder named figures.***

Lastly, the updated interfaces in the system are shown below.

A white background with black text

Description automatically generated

*Figure 7 admin management interface*

A screenshot of a computer code

Description automatically generated

*Figure 8 student management interface*

A screenshot of a computer program

Description automatically generated

*Figure 9 selected course management interface*

A screenshot of a computer program

Description automatically generated

*Figure 10 teaching staff management interface*

A screenshot of a computer program

Description automatically generated

*Figure 11 score(grade) management interface*

A screenshot of a computer program

Description automatically generated

*Figure 12 course management interface*

1. Test Plan

**2.1 Validation Test**

2.1.1 User Login

Test Procedure: Enter the verifiable id and password (id should start with s with 7 digits), correctly enter the verification code, then click the login button.

**Result:** Successful login to the system.

2.1.2 Student Information Management

Test Procedure: After logging in to the system, click the student information button to click the student list button in the navigation menu to view personal information.

**Result:** The system correctly displays students' personal information.

2.1.3 Students check their grades

Test Procedure: After logging in to the system, students click on the assessment information button to click score list to view their grades.

**Result:** The system correctly displays the student's own records.

2.1.4 Teachers modify scores

Test Steps: After logging in to the system, the teacher clicks on the score list button to view and ‘modify’ button to modify the student's score.

**Result:** The system modifies the students' scores

2.1.5 Logout

Test Procedure: After logging in to the system, click the secure exit button.

**Result**: Successfully logged out of the system

**2.2 Defect Test**

2.2.1 Student Grade Detection Test

* **Purpose**: To test whether the system will add negative numbers to student grades
* **Process description**: When adding student grades, the student grades are intentionally set to negative numbers and the system still adds negative student grades to the grade table.
* **Result**: the system will add negative numbers to student scores.

2.2.2 Password length detection test

* **Purpose**: To test whether the system has enabled password length restriction.
* **Process description**: When changing a password, set a new password larger than 32 digits, but the system still shows that the password is changed successfully.
* **Result**: The system does not enforce the password length restriction.

2.2.3 Duplicate course selection detection test

* **Purpose**: To test if the system double-checks the selection of courses.
* **Process description**: When selecting a course, intentionally choose a course that has already been selected, the system will still show that the selection is successful.
* **Result**: The system does not perform duplicity checks on course selection.

2.2.4 validation code time limit detection test

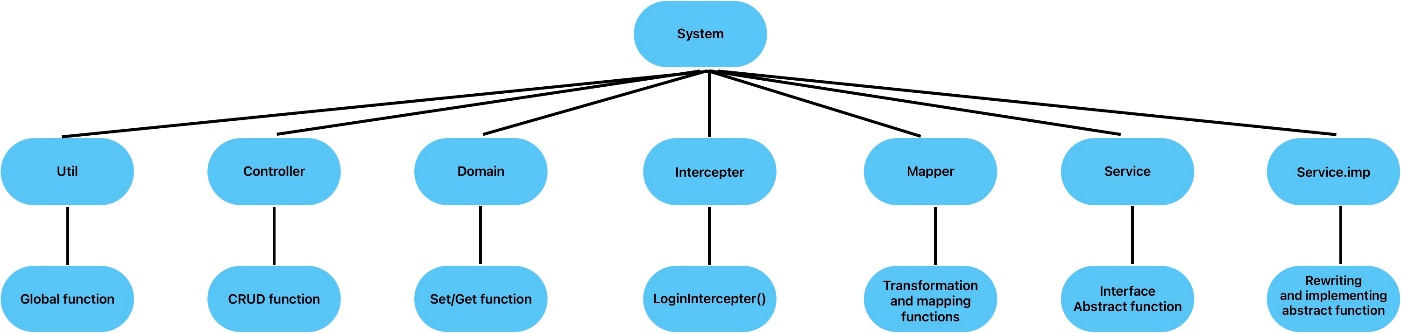
* **Purpose**: To test whether the system has set a time limit for validation code.
* **Process description**: When the validation code has not been refreshed for a long time during the login process, the system still shows successful login after inputting the unrefreshed validation code.
* **Result**: The system does not set a time limit for validation code.

2.2.5 Password secondary validation test

* **Purpose**: To test whether the system performs secondary verification of password change.
* **Process description**: When the original password is intentionally wrong when changing the password, the system will continue to execute the next command.

**Result**: The system does not check the password change twice.

**2.3 Development Test**



* + 1. **Divide the system into components**

Determine system functionality: Understand the overall function and requirements of the system. Identify the main features, tasks or responsibilities that the system needs to fulfill, such as recording students' academic records, storage of personal information for modification, etc.

Analyze Dependencies: Identify the dependencies and correlations between different parts of the system. Determine which functions are closely related and can be combined, e.g. service has an interface class and implementation class relationship with service.Impl.

Define Component Boundaries: Define boundaries for each component based on the identified functionality and dependencies. A component should encapsulate a set of related functions and have well-defined interfaces to interact with other components.

Improve cohesion: Aim for high cohesion within each component, i.e., the functionality within a component should be tightly related and focused on a specific purpose. This helps to ensure that each component has clear responsibilities and is not overburdened with extraneous tasks, such as the system treating each function as a separate component.

Managing component interactions: Determine how components interact with each other, ensuring that interactions are clear and follow good architectural principles. For example, the service has an interface class and implementation class relationship with service.Impl.

Consider reusability: Look for opportunities to create reusable components that can be used in distinct parts of the system or in future projects, e.g. the Util component is used in distinct parts of the system.

**2.3.2 Divide the components into units**

Identify Generic Functionality: Carefully review the functionality and logic in the component to identify generic functionality that is likely to be reused across multiple components, for example, parsing, mapping, and transforming data is a generic functionality that is used by all to convert instructions to database operations.

Extract Generic Functionality: Extract the identified generic functionality from the component into separate utilities. These utilities should be reusable, independent of the particular component, and have well-defined inputs and outputs, e.g., a set/get function in a Domain that returns or changes the properties of an object.

Define interfaces: Define clear interfaces for utilities so that other components can use them. The interface should clearly specify the input parameters and return values of the utility, as well as possible exceptions, such as getting the controller's commands after the service component will be implemented by service.impl.

Encapsulate utilities: Encapsulate utilities as separate modules or classes, e.g. the 'Mapper' component contains the data conversion functions for each Object.

Manage dependencies: Ensure that the dependencies between the modified utility and other components are managed correctly. If the Modify Grades utility depends on other components or libraries, ensure that these dependencies are properly declared and handled.

Reuse and Maintenance: Reuse utilities throughout the system and ensure that they are maintained and updated in a timely manner. For example, when a data conversion feature extension or fix is required, only the data conversion utility needs to be modified without affecting other components.