**Exam-Scheduler**

**Test Plan**

*This is a template to the project test plan,*

*It should be submitted as a Word file and as PDF*

**Submitted by**:

Roey Ashkenazy

Tal Gelber

Dor Bartov

Yosef Bader

Rephael Azoulay

Moisei Vainbaum

**Supervised by:**

Tanya Brokhman

**Table of Contents**

[1. Introduction 3](#_3znysh7)

[1.1 Purpose 3](#_2et92p0)

[1.2 Project overview 3](#_tyjcwt)

[2. Test strategy 3](#_3dy6vkm)

[2.1 Test objectives 3](#_1t3h5sf)

[2.2 Test assumptions 3](#_17dp8vu)

[2.3 Data approach 3](#_1y810tw)

[2.4 Scope and Levels of Testing 3](#_3rdcrjn)

[2.4.1 Functional Test 3](#_26in1rg)

[2.4.2 User Acceptance Test (UAT) 3](#_lnxbz9)

[3. Validation and Defect Management 4](#_35nkun2)

[4. TEST ENVIRONMENT 5](#_1ksv4uv)

[5. Test cases 5](#_44sinio)

[5.1 Function tests 5](#_2jxsxqh)

[5.1.1 MY\_first\_test 5](#_z337ya)

[5.2 UAT tests 5](#_3j2qqm3)

# 

# Introduction

## Purpose

This test plan describes the testing approach and overall framework that will drive the testing of the Exam Scheduler. The document introduces:

* Test Strategy: rules the test will be based on, including the givens of the project (e.g.: start / end dates, objectives, assumptions); description of the process to set up a valid test (e.g.: entry / exit criteria, creation of test cases, specific tasks to perform, scheduling, data strategy).
* Execution Strategy: describes how the test will be performed.

## Project overview

The exam schedule of the Computer Science faculty in the Technion is currently built manually. This means the undergraduate secretariat schedules important university-wide courses and then the faculty’s secretariat has to fit in their particular courses with as few conflicts as possible.

This project aims to help the secretary in charge of this task build an exam schedule for the faculty in a computerized manner. This will potentially save human resources, as well as optimize the schedule based on different configurations.

# Test strategy

## Test objectives

The objective of the test is to verify that the functionality of ExamScheduler works according to the specifications. The final product of the test is twofold:

* A production-ready software;
* A set of stable test scripts that can be reused for Functional test execution.

## Test assumptions

We assume the following:

* There are 3 possible exams scheduled in a single day - 9:00 AM, 13:00 PM and 17:30 PM.   
  On Friday there is only one possible exam - 9:00 AM. We will allow more than 3 tests a day.
* We will test the GUI by using manual user acceptance tests seeing as automatic GUI testing is very difficult.
* Our software will not take into account assigning rooms, and assume no restrictions in that regard.
* While some of the periods between exams may be shorter than requested, an acceptable schedule is possible.
* Since there is a minimum of 7 days between “Moed A” and “Moed B”, there is no relation between the schedule of each.
* The exams scheduled by Ulman are set and cannot be changed.

## Scope and Levels of Testing

### Functional Test

**PURPOSE**: Functional testing will be performed to check the functions of application. The functional testing is carried out by feeding the input and validates the output from the application.

**METHOD**: The test will be performed according to Functional scripts/Test procedures with a well-defined PASS/FAIL criteria.

### User Acceptance Test (UAT)

**PURPOSE**: this test allows the end users to complete one final review of the system prior to deployment.

**METHOD**: Will be performed manually by team members according to written test cases.

# Validation and Defect Management

It is the responsibility of the tester to open the defects, link them to the corresponding script, assign an initial severity and status,

It is the responsibility of the developer to retest after a fix is provided and close the defect.

Defects will be categorized according to the following severity status:

|  |  |
| --- | --- |
| Severity | Impact |
| 1 (Critical) | * This bug is critical enough to crash the system, cause file corruption, or cause potential data loss * It causes an abnormal return to the operating system (crash or a system failure message appears). * It causes the application to hang and requires re-booting the system. |
| 2 (High) | * It causes a lack of vital program functionality with workaround. |
| 3 (Medium) | * This Bug will degrade the quality of the System. However there is an intelligent workaround for achieving the desired functionality - for example through another screen. * This bug prevents other areas of the product from being tested. However other areas can be independently tested. |
| 4 (Low) | * There is an insufficient or unclear error message, which has minimum impact on product use |
| 5 (Cosmetic) | * There is an insufficient or unclear error message that has no impact on product use. |

# TEST ENVIRONMENT

For LOGIC, most probably J-Unit.

# Test cases

## Function tests

### LOGIC tests

### 5.1.1.1 Loading input into data structures.

**Objective**: to make sure the user input and DB information is loaded successfully and correctly, into the algorithm’s data structures.

**Enter criteria**: the user has entered his inputs and the DB info has been loaded to memory.

**Exit criteria**: all data is inside the correct data structures sorted as defined.

**Defects categorization**:

* Critical– the data was loaded incorrectly.
* Critical – the load fails.
* Critical - the data was loaded successfully but without being sorted.
* Low – the load takes too long.

**Input**: small DB and simple user input.

**Test logic:**

1. loading DB and user input into memory.
2. making sure data structures exactly as expected.

### 5.1.1.2 Failing on unsolvable input

**Objective**: to make sure that once given an unsolvable input (restrictions are conflicting in some way), the algorithm will issue an error stating the problem isn’t solvable.

**Enter criteria**: the user has entered his inputs and the DB info has been loaded to memory. the input is unsolvable.

**Exit criteria**: the algorithm issues an error “unsolvable problem”

**Defects categorization**:

* Critical– the program crashes.
* Critical – the program outputs a solution (which isn’t correct, since the problem isn’t solvable)..
* High - no error is issued.

**Input**: small DB and simple user input with conflicting restrictions.

**Test logic:**

1. run the program on the given input.
2. make sure the program finishes, does not display an output, and issues an error.

### 5.1.1.3 Algorithm creates legal schedule from legal input

**Objective**: make sure that once given a legal solvable input, the algorithm will create a schedule that addresses all restrictions optimally, as we defined.

**Enter criteria**: A legal solvable input.

**Exit criteria**: The algorithm creates a schedule which obeys all restrictions, and is optimal.

**Defects categorization**:

* Critical– The program crashes or issues an error, not creating an output.
* Critical – The algorithm schedules a course more or less than once.
* High - The algorithm ignores some restrictions.
* Medium - program takes a long time to return.
* Low - the algorithm schedules many tests in the same day unnecessarily.

**Input**: A legal solvable large (in regards to the problem) input.

**Test logic:**

1. run the program on the given input.
2. make sure the program finishes and displays an output.
3. run across all courses, and make sure the schedule fulfills their restrictions.

### 5.1.1.4 Manual change after scheduling

**Objective**: Make sure a manual change of test date in a given finished schedule is possible iff it is legal.

**Enter criteria**: A legal finished schedule has been produced by our algorithm.

**Exit criteria**: An illegal change of test date returns an error. A legal change is successful.

**Defects categorization**:

* Critical– An illegal change is allowed..
* Critical – The program crashes or the data is corrupted while attempting the change.
* High - A legal change is not allowed.
* Medium - The change forces the program to run again.
* Medium - program takes a long time to return.

**Input**: A legal and an illegal change to a finished schedule

**Test logic:**

1. Run the program on a legal input. save output.
2. Attempt making an illegal change. Make sure this returns an error and does not change schedule.
3. Attempt making a legal change. Make sure this is successful and the schedule is updated accordingly.

### 5.1.2 Database tests

### 5.1.2.1 Database loading test

**Objective**: Ensure the database loading system works correctly.

**Enter criteria**: The user has a database of courses and constraints following the XML hierarchy we defined.

**Exit criteria**: The system successfully loaded all the data if the files were valids and the data was correctly. Otherwise, if the files are not valid or if there are contradictions inside the database, the loading should fail.

**Defects categorization**:

* Critical– The system loaded a bad database (containing mistakes or having invalid XML structure), or the system failed to load a valid database.
* Critical – The data loaded by the system doesn’t match the data written in the XML files.
* Low – Loading the data takes a long time (more than a few seconds).

**Input**: A valid database, and several “bad” databases: a database having missing files, a database which XML structure doesn’t follow the expected structure, a database containing duplicate courses/study programs, a database containing invalid data (negative course ID).

**Test logic:**

1. Load the valid database - the loading should be successful.
2. Check that all the data in the loaded database matches the XML files.
3. Load each one of the bad databases and ensure the loading fails and returns the correct error message.

### 5.1.2.2 Database creation test

**Objective**: Ensure that the user can create a new semester database, without having to manually add relevant data from previous semesters.

**Enter criteria**: The user has database for previous semesters.

**Exit criteria**: The user created a new database, containing all the courses from previous, newest semester.

**Defects categorization**:

* Critical– The system created a new semester database containing incorrect data or where data from previous semesters are missing.
* Medium - The system copied data from an older database, that can potentially be outdated.
* Low – The process takes a long time (more than a few seconds).

**Input**: Valid databases for several semesters.

**Test logic:**

1. Create a new database and check that all the courses and study programs in the previous newest semester are also in the new semester.

### 5.1.2.3 Database manipulation test

**Objective**: Ensure that the user can add/remove courses, constraints and schedules.

**Enter criteria**: The user loaded a database.

**Exit criteria**: The user was able to add/edit/remove courses, study programs, constraints and schedules successfully.

**Defects categorization**:

* Critical– The user failed to edit the database as needed.
* Critical - The user managed to edit the database although the edit makes the database invalid (create a duplicate course, schedule two courses to the same date and hour).

**Input**: A valid database.

**Test logic:**

1. Load the database.
2. Execute each of the possible legal manipulations on the database: add or remove a course, define the semester at which the course is taken according to the mumletset, schedule and unschedule course exams, define some constraints... All of the manipulations should be successful.
3. Execute illegal manipulation and get the expected exception : Add a duplicate course or study program, schedule a course to a taken time slot, input negative course ID. All of the manipulations should fail.

### 5.1.2.3 Database serialization test

**Objective**: Ensure that the databases are saved correctly into XML files.

**Enter criteria**: The user has a database loaded in the system.

**Exit criteria**: The user has successfully saved the database.

**Defects categorization**:

* Critical– The saved files are incorrects: one the file is corrupted, some of the data is missing, or not properly written into XML files that follow the expected hierarchy.
* Medium - The name of the database is incorrect (but its content is still valid).

**Input**: A valid database.

**Test logic:**

1. Load the database, and perform some manipulations on it.
2. Execute the serialization process.
3. Reload the database and check that it stayed identical to the previous database before serialization.

## 5.2 UAT tests

**5.2.1 Login Test**

**Objective**: to make sure the login screen works correctly.

**Enter criteria**: the user opened the program and is facing the initial login screen.

**Exit criteria**: The user managed to successfully log in and move to the next screen.

**Defects categorization**:

* Critical– the user cannot enter his details in the screen/the user details are **correct** but login is unsuccessful/the user details are **incorrect** but login is successful.
* Low – it is not clear if the user was able to login after entering details or not.
* Cosmetic – the input boxes do not fit well in the screen/the screen is unintuitive.

**Input**: the user’s UG username and password.

**Test logic:**

Using a dummy account details as input for the test we should test if the login is successful.

We should make sure incorrect details can never log into the system.

**5.2.2 Semester Picking Test**

**Objective**: to make sure the semester picking screen works correctly.

**Enter criteria**: the user opened the program and managed to log into the system.

**Exit criteria**: The user managed to successfully pick a wanted semester and move to the next screen.

**Defects categorization**:

* Critical– the user picked a valid semester yet he cannot continue to the next screen.
* High -the user managed to continue to the next screen without picking a year/semester/both.
* Low – the previous screen button does not work/it is not clear if the user was able to choose a semester or not.
* Cosmetic – the input boxes do not fit well in the screen/the screen is unintuitive.

**Input**: A wanted year and semester (Semester A/Semester B/Summer).

**Test logic:**

1. Make sure you can go to the previous screen with the ‘previou screen’ button.
2. Make sure pressing the next button displays an informative error message that tells the user he did not enter his wanted year and semester.
3. Choose a year from the list and make sure there is an error that tells the user a semester has to be picked.
4. Choose a semester and make sure the user can go to the next screen.

**5.2.3.1 Calendars Screen Test (adding a constraint)**

**Objective**: to make sure you can add a constraint

**Enter criteria**: the user opened the program and managed to log into the system, and selected a valid semester

**Exit criteria**: The user managed to successfully add a constraint

**Defects categorization**:

* Critical– 1. the user pressed ‘+’ button and the constraint window did not appear

2. the user picked a valid constraint yet it did not add it in the DB

* High - the user added a valid constraint but it did not appear in the calendar GUI
* Cosmetic – the input boxes in the constraint window do not fit well in the screen/the screen is unintuitive.

**Input**: (in constraint window) selected course out of a list of courses

**Test logic:**

1. Make sure you can press the ‘+’ button in each day in the calendar, and the constraint window will appear
2. Make sure when adding a constraint it will appear in the calendar GUI and DB

**5.2.3.2 Calendars Screen Test (adding/removing a course)**

**Objective**: to make sure you can add/remove a course

**Enter criteria**: the user opened the program and managed to log into the system, and selected a valid semester

**Exit criteria**: The user managed to successfully add/remove a course

**Defects categorization**:

* Critical– 1. the user pressed ‘add/remove course’ button and the course window did not

apear

2. the user picked a valid course yet it did not add/remove it in the DB

* High - the user added/removed a valid course but it did not appear in the course list GUI
* Cosmetic – the input boxes in the add/remove course window do not fit well in the screen/the screen is unintuitive.

**Input**: (in adding/removing course window) selected course out of a list of courses

**Test logic:**

1. Make sure you can press the ‘add/remove course’ button in calendar screen, and the add/remove course window will appear
2. Make sure when adding/removing a course it will appear in the course list GUI and DB

**5.2.3.3 Calendars Screen Test (Schedule button)**

**Objective**: to make ‘Schedule’ button will run the algorithm and result will appear in the GUI

**Enter criteria**: the user opened the program and managed to log into the system, selected a valid semester and pressed ‘Schedule’ button.

**Exit criteria**: Valid sheduling algorithm result appears in the Calendars.

**Defects categorization**:

* Critical– the user pressed ‘Schedule’ button and the result appear
* High - the user pressed ‘Schedule’ the result appears but its conflicting the constraints
* Cosmetic –scheduled courses do not fit well in the calendars.

**Input**: None

**Test logic:**

1. Make sure you can press the ‘Schedule’ button in calendar screen, and the right scheduling result will appear in the calendars.
2. Make sure the scheduling result in the algorithm is identical to what appears on the GUI
3. Make sure the study route picking in correct
4. Make sure the end result do not conflict according to all constraints

**5.2.4 LOGIC tests**

**5.2.4.1 results to output correctness**

**Objective**: Make sure the results created by the algorithm in the memory matches the result given in the output file

**Enter criteria**: The program is given a solvable input and runs successfully.

**Exit criteria**: The results stored in the ram (while debugging) match the results in the output file.

**Defects categorization**:

* High - There is a mismatch between memory and file.

**Input**: Small solvable input.

**Test logic:**

1. Run the algorithm on given input in a debugger.
2. at the end, before clearing memory, make sure the result in memory matches the result on file.