**Session 3 – Database and Statistics**

**Duration: 210 minutes**

# 1 Instructions

This document contains the special tasks for the 3rd session. We advise you to read this document carefully and to get an overview over the provided media like attached images, appendices and text.

The maximum time to complete as many tasks as possible is **210 minutes**. **Within this time** you have to read and understand all tasks and to store the results **into the folder** “***Session-3-YourName***” on the desktop of your computer.

You are allowed to use all components, source code and windows of the sessions before for this session and for the following sessions of the competition.

Please for every session create a file containing a short user guide and a technical description of how to configure and start up the application.

Make sure that you follow the provided style guide throughout all parts of the system.

Make sure that you provide appropriate validation and error messages throughout all parts of the system.

Make sure that all relevant buttons/links are working at the end of the session.

Make sure that you use appropriate naming conventions for all parts of the system as needed.

Make sure that your work is on the state of the art of Skill 09.

# 2 Contents

Session 3 of this Test Project consists of the following documentation/files:

1. AEC2024-Session3.pdf
2. Application Structure.pdf
3. Multiple input files in the assets/import folder.

# 3 Project

## 3.1 Introduction

In this session your task will be to create a Desktop application which helps organizers import their events and manage their bookings, on top of that calculate an event’s best pricing scheme.

## 3.2 The Task

Implement an import functionality to import and validate multiple types of data:

1. Import organizers from a *json* file.
2. Import locations from a *toml* file.
3. Import and validate events from a csv file.
4. Import and validate bookings from multiple sources, such as:
   1. A csv file.
   2. A morse coded file.
   3. A transposition ciphered file.

On top of that you must also implement a dashboard where multiple statistics can be displayed for each event and create an algorithm to predict future ticket purchases based on historical data and simulate ticket price changes.

# 4 Application Structure

The application consists of 3 windows:

1. Dashboard window.
2. Import window.
3. Event details window.

All these windows are connected using special buttons or functions respectively.

# 5 Tasks

All tasks should use the same database, provided to you in the ***sql***folder. You can find the database’s structure below:

A screenshot of a computer flowchart

Description automatically generated

## 5.1 Import window

**Quality tested** Programming, DB Access

**Work requested**

You have been provided with multiple types of import files, you must create a window, that allows the user to open a file from the filesystem and select which type of file they want to import. After the import has finished for the given file, the user must be provided with information about the process, in case the import encountered a validation error, the user must be provided with the line number and the row/object that failed (in case it was ciphered then, the plain text must be provided), in all validation cases, the process must proceed and import all other valid rows.

*Do not cache the provided files in the source code as a different file will be used for testing!*

A screenshot of a computer

Description automatically generated

## 5.1.1 Organizers

Organizers can be imported from the ***organizers.json***, no validation is required for this file.

## 5.1.2 Locations

Locations can be imported from the ***locations.toml***, no validation is required for this file.

## 5.1.3 Events

Locations can be imported from the ***events.csv***.

You must validate that the organizer and location exist for the event, also validate that the available tickets are not more than the maximum visitor count.

## 5.1.4 Bookings

Bookings can be imported from the ***bookings*** folder.

Bookings must be imported in order of the timestamp, if a user is missing it must be created.

If the bought tickets would exceed the available tickets, then it must be rejected.

There are three types of coding for the bookings:

1. Basic csv. **(.csv extension)**
2. Morse coded (you can find a morse table in the ***morse-table*** directory). **(.mc extension)**

* Each line is morse coded, each line is coded separately. Each morse code can be translated to a normal character. All characters should be considered lowercase after conversion. A space between two words is displayed via a slash **‘/’**.

1. Columnar Transposition ciphered. **(.tpc extension)**

* You are provided with a ciphering key, which is **EANB.**
* The key’s letters’ alphabetic order gives you the order of reading.
* You must then write the line in the order of the key.
* Then read the data from top to bottom.
* A space between two words is displayed via a slash **‘/’**.
* You may also find a more detailed description in **columnar-cipher.pdf.**

## 5.2 Dashboard window

**Quality tested** Programming, UI, DB Access

**Work requested**

You must create a dashboard which allows the user to see the details of an event group.

The following information must be provided for the user:

* Sum of total cost incurred by the creation of the events,
* Sum of total income from the events,
* The percentage increase/decrease between the cost and income,
* And information about the events, statistics, and tickets (find the required fields on the mocked UI).

A screenshot of a dashboard

Description automatically generatedA table of the events must be also created, here the user can see basic details about each ongoing and future events. Using a checkbox the user should be able to see all previous events as well.

The event details window is shown, when the user double-clicks on an event.

## 5.3 Event details window

**Quality tested** Programming, UI, DB Access

**Work requested**

In this window, you must calculate ticket demand, saleability and forecast ticket purchase.

An event’s base demand measures the predicted demand by the user, this corrects the calculated demand.

A tier’s demand is calculated from the ratio of the remaining and maximum available tickets for a given tier. Historical demand is calculated from all previous events’ ticket demands’ average (in the event group), for the same ticket tier. The two demands are averaged out (if historical is indetermined only the tier base demand is used) and finally corrected by the base demand for the event.

The price ratio for a given ticket tier is calculated by comparing the base price to the least expensive tier’s base price.

A tier’s saleability is calculated from price ratio calculated above at 30% and summing it up with the corrected demand at 70%.

The tier’s current price is calculated from the base price and the saleability factor.

The user should also be able to optimize the event’s pricing tiers calculated.

A table and diagram must be created, this component will forecast ticket purchases. It must display the already purchased tickets and the profit incurred by the event, after that a forecast should take place. All ticket tiers’ saleability must be calculated for the given state, then the system should “purchase” one from the highest saleability tier and repeat it until all tickets have been sold. For past events, the simulation doesn’t need to be executed. The table’s data can be exported into a .csv by the user. The diagram must display the current profit/loss of the event and the available tickets for each tier and the changes from previous purchases and the future changes predicted by the forecast system. The biggest loss and profit must also be displayed.

A screenshot of a report

Description automatically generated

## 5.6 Documentation / Notes

**Quality tested** Documentation

**Work requested**

* Please create a file containing a short user guide and a technical description of how to configure, how to get access to your database and start up the application.
* In this file you may also leave *notes* that could be useful for the evaluation. *Notes* in this document are not part of the evaluation process.