Global Problem-based Case - 2020-2
This case replaces all semester activity

The development platform **must be .NET Core 3.1**, this was the platform for tutorials and problem-based cases during the semester. The examination has 5 problems and must be delivered by email, ONLY ONE project must be delivered and, in the text of the email, you must explain what problems have been solved in the attached mail. Solutions for problems 1, 2, and 3 are mandatory for approval. Solutions 4 and 5 are for reaching additional grades. Only in the case that you have solved Problem 4, a second Project must be part of the delivered examination.

CASE: LOGISTICS

Several tracks arrive at different stores in the city at different times. The stores are three in the city and each one of these has limited space for storing suits (packs x 5), shoes (boxes of 10 pairs), and shirts (in boxes of 20 shirts). The stores and their capacities are:

Store	Shirt Boxes	Shoe Packs	Suits Packs
Downtown	120	160	90
Sporting Boulevard	240	190	150
Wschodzące Słońce Shopping	45	60	70

Obviously, each Store has a current stock (inventory) of the corresponding packages.

Problem 1 (Required for Approval). You must create a human-computer interface for entering an initial inventory of each product in each Store in the way of a spreadsheet (like the previous one) but for the current stock. In this interface, the maximum number of packs (capacity) must be reported for each case. If the entered number is greater than the capacity or less than zero then an alert must appear. You must use a Blazorserver template and the specified interface must appear when the first ítem in the vertical menu is selected.

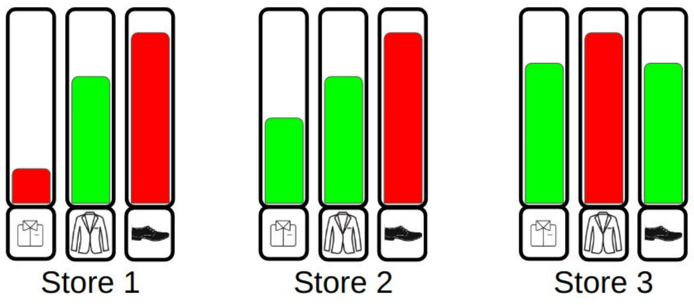
Items to review:.
P1.A. The layout includes all necessary fields for input (9)
P1.B. The layout includes the information of capacities
P1.C. The inputs admit only numbers
P1.D. The interface reports the corresponding warning messages
P1.E. The vertical menu has a proper name for the requested interface.

Problem 2 (Required for Approval). You must model a table for storing the capacity of the stores and the current stock for each type of package. Also, you must create the Class “StoreList”, for storing the complete List of objects of type Each object “store” allow access to the corresponding values (Capacity and Stock) for each type of product. The class StoreList must have the methods **get** for getting the data from the table and the method **update** for updating the stock of each product in the list. You must use the class MyConnection provided as part of Tutorial 5.

Items to review:

P2.A. There is a proper representation of the data model
P2.B. The database contains the exact representation of the model
P2.C. The previous user interface (P1) still works but now the button “Save” effectively puts the data in the database.
P2.D. The previous user interface (P1) gets the current values of stock and let these values as part of the input values.
P2.E. The class StoreList exists and also its methods get and update. These methods are used to save the data in the database and no others. These methods effectively use the class provided for Postgress connections s part of Tutorial 5.

Problem 3 (Required for Approval). In this part, you must create a new interface for reporting the status of stocks in a graphical way. The name of the option in the vertical menu must be “Status”. The produced SVG graphic must be like the next figure. Each group represents a store. You must write the name of the store. The bars represent the occupied percentage in each product (shirts, suits, and shoes). The color of each bar is red or green. When the percentage is less than 20% the bar must be red (risk of insufficient stock), when the percentage is greater than 80% the bar also must be red (risk of insufficient space), in all other cases the bar must be green. The value of each percentage must be shown on the top of each bar.



The way for implementing this is not free. You must develop this report by creating a Component (a tag) for reporting an individual store. The component must have two parameters, the name of the store and a string having the percentages separated by commas, for example <storestatus title=”Downtown” values=”18,65,82”/>. The previous interface must be a table having one row and three columns, in each cell, the previous tag must appear only one time.

Items to review:

P3.A. The component is well-defined: right name and number of parameters.
P3.B. The component displays each store according to the requirements (figure plus percentage in the top).
P3.C. The figure is created by using SVG (with the optional exception of icons).
P3.D. The figure shows the figure corresponding to current data in the database.

Problem 4. The computer in each truck should receive the status of the different stores in order to make a decision about what store must be the target store for this truck. The way of reporting the status is using a web service. The web service must have ONE option, having one parameter, called “type” which would have three values: “shoe”, “shirt”, or “suit”. The output should without parameters, which the output must have the format: [{“store”:”store 1”,”capability”:160,”stock”:22},{“store”:”store 2”,”capability”:190,”stock”:182}, {“store”:”store 3”,”capability”:60,”stock”:6}].

Items to review:

P4.A. The WS is well-defined, it accepts only the defined parameter and the output is a JSON having an array of JSON objects.
P4.B. The WS is well documented and the way of invoking it is established in a comment inside the web service.
P4.C. The WS data corresponds to the data in the database.

Problem 5.

In this case, a simulation is requested, and entering the data for the simulation is part of the problem. The first data requested is the consumption rate of shoes, shirts, and suits in each store, the rates are expressed in units (individual suits, shirts, and shoe pairs). Therefore a similar input to Problem 1 is required.

The second data required for the simulation is a list of the day programmed arrivals. This is a list of, at most 25 truck, each row in the list have the following information: (1) arriving time (only the time, in the way of 12:05, 15:40), (2) truck id, a short alphanumeric string, (3) numbers of suit packs in the truck, (4) number of shirt packs in the truck, and (5) number of shoe packs in the track, (6) target store (the name).

The simulation must consider the starting time to 00:00, and the last time to 23:59, the delay for arriving at any store is considered as 0. The delay for unloading is also 0, The simulation must run considering one simulation second as 15 minutes of real-time. The simulation must show the following information. If a store has non-satisficed demand then that demand is lost (client do not wait for next arrivals), however, trucks, wait for enough space in the store.

You must decide what classes you need to do, but, at least consider one queue of waiting for trucks in each store. Also, consider that there is no optimization for receiving trucks, i.e. if the first truck can be unloaded then it is unloaded, but is not, the following truck in the waiting queue is tested for unloading.

Items to review:

A). A graphic of the situation of the stores, like Problem 3, changes dynamically according to consumptions and arrivals. Additionally, there is a clock showing the current real-time of the simulation.
B.) The initial stock corresponds to the values in the database
C.) The interface for entering consumptions rates works ok (positive values) and can be updated for new simulations (no reset for a second simulation).
D.) The interface for entering truck arrivals works ok and can be updated for a new simulation (no reset for a second simulation)
E.) The simulation reports the total amount of wasted time by trucks waiting to be unloaded. Note that wasting time must be added. This is displayed as part of the simulation.
F.) The simulation reports the total number of non-sale shoe pairs, shirts, and suits because of stock breaks. These numbers are displayed as part of the simulation display.