|  |  |  |  |
| --- | --- | --- | --- |
| **Module** | **XPs** | **Requirements**  Remember: all the requirements of a module  must be fulfilled in order to get its XPs.  Module A is a mandatory module. | **Implemented**  Mark the implemented modules with "X" in this column in order to be able to get the corresponding XPs. |
| **A** | 1 | * The submission has been done as a .zip package. When unzipped into a directory, this directory should work directly as the project directory. (This means that you should zip the relevant content of your project directory into a zip file, not add your project directory into a zip file so that when unzipped into a directory, there is your project directory within this directory.) * This instructions document has been filled out properly (concerning the *Implemented* column) and included in the submitted .zip package (on the root level among other files). * The application has at least three separate "views" with different content, and the user is able to keep switching between the views easily using a menu (a dropdown menu or a menu bar), tabs, or buttons within the user interface of the application. | x |
| B | 3 | * The application has an "Info" (or "About") menu option or button, via which the user can get information on the application. At least these pieces of information must be available via this functionality:   + The name of the author   + Short instructions for using the application. **If instructions are unnecessary – i.e., you are sure that your UI is extremely easy to use and understandable intuitively – this state of matters must be mentioned explicitly, then. (The info can contain, e.g., the statement "The UI hints help the user to figure out how to use the application," "Using the application should be intuitive and straightforward," or something else suitable.)**   + If all the (possible) pictures and other such content used in your application are originally created by you yourself, tell it here explicitly. Otherwise, mention the licenses/permissions under which you use content from external sources. Make sure that the possible license conditions actually allow such use and you fulfil all the related requirements. (The application code itself must, naturally, be written by you.)   + How many working hours, approximately, did you spend working with this programming assignment? (The number does not affect the grading, but this is asked in order to understand the work load so that adjustments can be made in the future, if necessary. Therefore, honest answers are appreciated.)     - This is just an estimate, but it must be expressed as **one integer number** of hours spent. (I.e., **NOT** as some interval like 30–40 hours or some estimate expressed in days or working weeks or anything else than hours.)   + What did you consider as the most difficult/tedious feature to implement?   (Note: One of the views required by Module A may, for instance, be an info page presenting the necessary information required by this module, but you are free to implement this otherwise, also.) | x |
| C | 5 | * The application can communicate with a simple local backend server\*, and by using an API offered by it initialize the application:when the application starts (the page is loaded), it must automatically (i.e., without any input by or interaction with the user) fetch from the back end information about *tasks*. Especially, for each *task* existing in the system, the application must get at least its   + **name** (a string, e.g., “code”, “fix the computer”, or “feed the cat”) and   + the related ***tags***. A task always has at least one textual *tag*, but can have several. (Tags (e.g., “school”, “animal care”, and “cool”) may be used to categorize and describe *tasks* and attach context to them.) * Based on the fetched data, the application must show corresponding *task elements*, implemented as React components, within its UI. There must be one *task element* for each *task*, and the name and the *tags* of a *task* must be included in the visual representation of the corresponding *task element*.   \* You are not to implement this yourself for this assignment. This backend server functionality is obtained by using *json-server* npm package. You can assume that the person checking your work will run the following command in your project directory (root level) prior to starting the frontend:  npx json-server -H localhost -p 3010 -w ./db.json  So, use the port 3010 for the API calls, and add a proper file named as *db.json* to your project root to be used as the “database.” | x |
| D | 4 | * Via the application UI, the user must be able to...   + freely edit the *task* names shown in the *task elements,*   + add *tags* to and remove them from *task elements* arbitrarily (still respecting the rule that each *task element* must have at least one *tag*),   + create new *tags* (by defining arbitrary strings to be used as such),   + add new *task elements* to be shown within the UI (defining their names and the associated *tags*), and   + remove task elements one by one in an arbitrary order. | x |
| E | 4 | * The changes made by the user via the application UI (concerning *task elements*) are stored using suitable API calls to the back end so that the application can fetch the changed data successfully on start (see Module C). The application can store information automatically and continuously (when changes are made) or only when the user explicitly asks for it via the UI (e.g., pushes a Save button). | x |
| F | 3 | * The user can, via the application UI, filter *task elements* based on their tags. I.e., the user must be able to limit the selection of task elements to be shown so that only the *task elements* with the desired (arbitrary) *tags* are shown and the other *task elements* are hidden. (The user can select several *tags*, and only those *task elements* having all of those tags are shown.) The user must also be able to see all the task elements regardless to their tags. |  |
| G | 4 | * Handling *tags* has been implemented so that when adding a *tag* for a *task element*, the user is not always required to write the *tag* text, but it is possible to select among the existing *tags* that have already been created before. * The user can rearrange *task elements* within the UI. (The idea is that the mutual visual presentation order of the elements can be changed arbitrarily; e.g., automatic layout changes based on the size of the screen are thus not enough to fulfil this requirement.) |  |
| H | 5 | * Each visible *task element* must always (without any extra operations by the user) show clearly whether the task is *active* at the moment or not. Being *active* means that time used for the respective *task* is currently recorded – the idea is that someone is just performing the activities related to this task, and the overall time used for this task keeps increasing until the task becomes *inactive* (i.e., performing the related activity is stopped). * If a *task* is *inactive* (i.e., time is not counted for it), the user can easily (using a functionality offered by the corresponding *task element*) activate it (make it *active*, i.e., start/continue counting time for it). Correspondingly, an *active* task can easily be inactivated (made *inactive*, i.e., pause/stop counting time for it). * The application stores, using the API provided by json-server – see Module C – the starting/continuing and pausing/stopping times so that when the application is started the next time, counting time continues automatically for the *active tasks* (the *tasks* that have been activated without inactivating them after that) exactly as it would continue if the application had been running whole the time. (In other words, time of being *active* (*active time*) for the *active tasks* must effectively be recorded also when the application is not running.) * Several (unlimited number of) *tasks* can be *active* simultaneously. |  |
| I | 4 | * The user can specify and freely modify an arbitrary *observation interval* from a start moment (starting date and time) to an end moment (ending date and time). By default, this interval covers the current day from its beginning to the current time. Minute resolution or more precise one (e.g., second resolution) must be used – dealing with, e.g., full hours only is not a sufficient precision. Let us call *tasks* that have been active at some point during the *observation interval* *tasks* *of interest* and tags associated with these *tasks* *tags of interest*. * The user can easily see a summary about the *total* *active times* (sums of durations of individual activity periods) separately for each *task of interest* and each *tag of interest* within the *observation interval*. In order to keep the view clear and uncluttered, the other *tasks* or *tags* than *tasks of interest* and *tags of interest* are not to be shown in the summary. (Their total active times within the *observation interval* are known to be zero, anyway.) |  |
| J | 4 | * The user can specify and freely modify an arbitrary *task* *details interval* from a start moment (starting date and time) to an end moment (ending date and time). * The user can select any *task* and see a list of all the distinct time intervals when that particular *task* has been *active* during the *task details interval*. These intervals of activity must be listed in chronological order. (If the *task* under observation has been *active* in the beginning of the *task details interval*, the starting moment of the first listed interval must be the starting moment of the *task details interval*. Similarly, if the *task* has been active in the end of the *task details interval*, the ending moment of the last listed interval must be the ending moment of the *task details interval*.) * The user can modify the starting and ending times of the listed intervals and save the changes using the API offered by json-server (see Module C). Activity intervals can also be added and removed on the list. Any special error handling is not required, but if the changes make sense (the intervals remain distinct etc.), the application keeps working in a sensible way and uses the modified values correctly. |  |
| K | 4 | * The user can select a *task*, specify an arbitrary time interval (*daily activity chart interval*), and see a bar chart depicting daily activity times concerning the selected *task* (daily sums of durations of time intervals when the task has been active). There should be a bar per day, and only activity taking place during *daily activity chart interval* counts – other activity is not taken into account nor are the days outside *daily activity chart interval* included in the chart. |  |
| L | 3 | * The user can access an "Options" or "Settings" view/dialog, in which the user can at least   + change the theme of the application (must clearly affect at least the UI colors) and   + change the application into an alternative mode in which only one *task* can be active at a time (activating a task inactivates the other tasks) and back into the default mode.   The settings must be stored via the API provided by json-server (see Module C). When you submit your work, the default mode (in which several *tasks* can be active) must be used, but after that, the application must always start with the most recently set configuration.) |  |
| M | 3 | * The user interface is responsive and supports different screen resolutions. One must be able to use the application in a small browser window of width of only 500 px, but if wider screen in available, the extra space must be used sensibly. Merely scaling the elements is not considered to be "responsiveness," but the layout must change reasonably based on the screen size (e.g., by adjusting the orientations/locations of elements that are shown and/or hiding nonessential information). |  |
| N | 3 | * The application feels usable and accessible. When using the application with a PC, all the features can be effectively used with keyboard only (without a mouse or other such pointing device). The font sizes are suitable, and the color contrasts meet the WCAG AA requirements (see, e.g., <https://developer.mozilla.org/en-US/docs/Web/Accessibility/Understanding_WCAG/Perceivable/Color_contrast>). (It is a good idea to take advantage of the accessibility tools of a browser when implementing your application and checking the accessibility.) |  |