* **[R1] The user must be logged into the system to access application features.**

As explained in the IITP part, there is dependency between *Travel* and *Appointment Manager* and *Account Manager*: the account managing infrastructure is created first and travel/appointment data are acquired from the database.

* **[R2] The user must be able to choose the option of creating a new appointment.**

*Appointment Manager* implements the functionality *createAppointment().* See section 4.3: *Create new appointment.*

* **[R3] The user must be able to choose the option of editing a selected appointment.**

*Appointment Manager* implements the functionality *editAppointment().* See section 4.3: *Edit existing appointment.*

* **[R4] The user must be able to choose the option of deleting a selected appointment.**

*Appointment Manager* implements the functionality *deleteAppointment().*

* **[R5] The system must be able to provide the user with an overview of his calendar and the user**

**must be able to view all appointments fixed in a certain period.**

The system gets information about all the created appointment from the database and display them to with the interfaces described in the *User Interfaces* section of this document.

* **[R6] The user must be able to select a chosen day from the overview of his calendar.**

The system keeps track of the different dates (see Class Diagram: *Date* class)and offers the *daily calendar* option of display. See section 4.2: *Calendar views.*

* **[R7] The user must be able to select a specific appointment in his calendar.**

Account Manager loads the information about the different appointments (see Class Diagram: *Appointment* class). See section 4.4: *Editing existing appointment* to see the correspondent user interfaces.

* **[R8] The system must ask the user to provide all information needed for the creation of a new**

**appointment, such as place and time of start and overall duration.**

This requirement is covered by the *createAppointment()* functionality of *Appointment Manager*. See section 4.3: *Create new appointment.*

* **[R9] The system must check if the information provided by the user are correct.**

Functionalities covered during phases of creation/editing. See Sequence diagrams on section 5.2 of RASD document.

* **[R10] The system must check if an appointment overlaps with other events and must eventually**

**notify it to the user.**

See section 3.2.3: *Check overlap* algorithm.

* **[R11] The system must give the user access to all details of a selected appointment and the user**

**must be allowed to edit the information needed.**

Account Manager loads the information about the different appointments (see Class Diagram: *Appointment* class) and display them to the user. It also implements the functionality *editAppointment().* See section 4.4: *Editing existing appointment* to see user interfaces.

* **[R12] The user must be able to set advanced information for a created appointment.**

See *advancedOptions()* in section 4.3: *Create new appointment.*

* **[R13] The user must be able to set an appointment as flexible, specifying the interval of time.**

Functionality of *Appointment Manager*. See *advancedOptions()* in section 4.3: *Create new appointment* and the *Flexible Appointment* class in the Class Diagram.

* **[R14] The user must be able to set an appointment as repeatable, specifying the desired days.**

Functionality of *Appointment Manager*. See *advancedOptions()* in section 4.3: *Create new appointment.*

* **[R15] The system must schedule any flexible or repeatable appointment in the correct way, avoiding overlapping with other appointments.**

The system is provided with algorithm for overlapping checking and flexible appointments scheduling. See section 3.2 for further info.

* **[R16] The appointment intended to be modified must have been previously successfully created**

**and not already deleted.**

The process of editing appointments is made by *Appointment Manager* and works only with existing appointments in the database. A process is stored in the database only at the creation and removed during the deletion. See section 2.4: R*untime view*.

* **[R17] The user must confirm the creation of the new appointment.**

*createAppointment()* functionality of *Appointment Manager*. See *confirmCreation()* in section 4.3: *Create new appointment.*

* **[R18] The user must confirm any appointment modification.**

*editAppointment()* functionality of *Appointment Manager*. See *saveModification()* in section 4.4: *Create new appointment.*

* **[R19] The system must save the user modifications in memory and the calendar must be updated.**

*Appointment Manager* interacts with the DBMS: after any modification, the database is updated. See section 2.4: R*untime view*.

* **[R20] The system must remove a deleted appointment from the memory and delete every alert**

**related to it.**

*Appointment Manager* interacts with the DBMS: after any modification, the database is updated.

This functionality is implemented by *deleteAppointment()*.

* **[R21] The user must be able to switch between different possible calendar, such as daily calendar, weekly calendar and monthly calendar.**

See section 4.2: *Calendar view*.

* **[R22] The system must be able to provide information about the scheduled travels for a chosen**

**day, showing the transport means and the estimated time required from each travel.**

User can recall the function viewDailySchedule() from his application, and Travel Manager takes care of provide all the information in detail. See section 2.4: R*untime view*.

* **[R23] The system must choose the best option between the possible travel alternatives according**

**to the preferences expressed in the user profile settings and the information about external weather.**

This requirement is covered by the *computeTravel()* and *loadPreferences()* functionality of *Travel Manager*. See section 3.2.1: *Compute travel* for further info on the algorithm.

* **[R24] The user must be able to select a specific travel in his daily schedule.**

See *selectTravel()* in section 4.6: *View daily schedule and travel/movement details.*

* **[R25] The system must provide detailed information about the travels selected by the user, such**

**as the trace route on the map and the weather conditions.**

This requirement is covered by the *computeTravel()* functionality of *Travel Manager*. More specifically, see section 2.4 for runtime view and section 3.2.1: *Compute travel* for further info on the algorithm.

* **[R26] The system must provide the user with an overview of the possible travel alternatives for the chosen travel, specifying all details for each one.**

*Travel Manager* functionalities. see section 3.2.6: *Check travel alternative* for further info on the algorithm and *viewAlternatives()* in section 4.6: *View daily schedule and travel/movement details* for user interfaces.

* **[R27] The user must be able to filter the travel alternatives furnished.**

*Travel Manager* functionalities. see section 3.2.6: *Check travel alternative* for further info on the algorithm.

* **[R28] The user must be able to choose a favorite travel option different from the displayed default one.**

*Travel Manager* functionalities. See section 3.2.5: *Check travel alternatives* for further info on the algorithm.

* **[R29] The user must be able to select a specific movement in a travel.**

See *selectMovement()* in section 4.6: *View daily schedule and travel/movement details.*

* **[R30] The system must provide detailed information about the movements selected by the user,**

**such as the specific trace route on the map and the price of the ticket.**

*Travel Manager* functionalities. All information are provided by the external APIs and collected by the system as explained in section 2.4 and 3.2.1. See section 4.6: *View daily schedule and travel/movement details* for user interfaces.

* **[R31] The user must be able to choose an alternative transport mean for a selected movement, if**

**there are any.**

*Travel Manager* functionalities. See section 3.2.6: *Check movement alternative* for further info on the algorithm.

* **[R32] The system must update the daily schedule according to the travel option chosen by the user and the user must be able to see the new updated schedule.**

*Travel Manager* functionality. The user can choose a travel alternative (see section 3.2.6: *Check travel alternative* for info on the algorithm) and *Travel Manager* updates info.

* **[R33] The system must give to the user the possibility of buying the ticket for the selected travel.**

This requirement is covered by *Ticket Manager* functionalities. See section 4.5: *Component interfaces* for full functionalities overview.

* **[R34] The system must save a copy of the bought tickets.**

Tickets manager interacts with DMBS to store tickets data after the purchase. See section 2.4 for runtime views.

* **[R35] The user must be able to access to a ticket page from the home page.**

Bought tickets are saved in the database during the purchase process (see section 2.4 for runtime view) and the user can access them by clicking on “My tickets) on the side panel menu. See section 4.7: *Buy and view tickets* for user interfaces.

* **[R36] The system must provide a list of all the bought tickets and the user must be able to select**

**and view a specific one.**

This requirement is covered by the *viewTickets()* functionality of *Travel Manager.* See section 4.7: *Buy and view tickets* for user interfaces.

* **[R37] The user must be able to access the preferences panel of his account.**

*Account Manager* functionality. See section 4.5: *Manage preferences* for user interfaces.

* **[R38] The system must give the user the possibility of setting various preferences, such as owned**

**and preferred travel means, address of Home and other general travel preferences.**

*Account Manager* functionality. See section 4.5: *Manage preferences* for user interfaces.

* **[R39] The user must be able to edit the provided preferences when needed.**

The preferences panel is always accessible from the home page (See section 4.5: *Manage preferences* for user interfaces). *Account Manager* interacts with DBMS, so any modification will be update in the database.

* **[R40] The system must give the user the possibility of adding an alert to an appointment while it is being created or modified.**

*Functionality of Appointment Manager*. See section 5.2.3 and 5.2.4 or RASD document: *Appointment creation* and *Appointment editing* for info on Sequence diagrams and sections 4.3/4.4 of this document for user interfaces

* **[R41] The user must be able to choose a desired interval of time for the warning alert.**

*Functionality of Appointment Manager.* See section 5.2.6 of RASD document: *Alert editing* for info on Sequence diagrams and sections 4.3/4.4 of this document for user interfaces.

* **[R42] The user must confirm the alert creation and the system must save the insertion in the memory.**

*Appointment manager* interacts with DBMS to store information in the system memory. See section 5.2.6 of RASD document: *Alert editing* for info on Sequence diagrams and sections 4.3/4.4 of this document for user interfaces.

* **[R43] The user must be able to modify or remove the inserted alert when needed.**

*Appointment manager* functionalities are always accessible by the user through the correspondent interfaces.

* **[R44] In case of any alert modification made by the user, the user must confirm the modification**

**and the system must save all changes.**

*Appointment manager* interacts with DBMS and updates data after any modification. See section 5.2.6 of RASD document: *Alert editing* for info on Sequence diagrams and sections 4.3/4.4 of this document for user interfaces.