Politecnico di Milano

Dipartimento di Elettronica Informazione e Bioingegneria

Poli-RISPOSTA Mobile App

Authors:
Rolando Brondolin
Danilo Ardagna

30th September, 2015

Contents

1 Introduction

Poli-RISPOSTA mobile app is an Android application developed to support the data gathering step of the Poli-RISPOSTA procedure performed after a flood event. Its aim is to digitalize the current paper based approach used by the Italian Civil Protection. The proposed system aims to provide a consistent approach for collecting and storing data and to develop a long-term application solution by providing an environment for analyzing, managing, accessing and reusing information, objects and data.

Despite the simplicity of the elements that cause floods, floods constitute a high risk to both rural and urban settlements. As more and more surface gets covered by urban settlement, grounds get less permeable resulting in more water unable to drip to the aquifer and submerging the surface. The main factor to push a growing risk of floods is the growing value of assets built on the lands.

The reconstruction process that follow such a tragic event need to be planned and tailored on an analysis of the damages to infrastructures and proprieties. Collecting data after floods is necessary not only to verify the request for reimbursements or budget allocations but also to gather knowledge about what factors and artifacts of settlements constitute risk in floods. While specific expertise is needed to assess damages to complex infrastructures, inspection of residential properties can be more repetitive and need a systematic methodology to address the multitude of sites to be inspected and amount of data to be gathered, organized and analysed.

2 Requirements analysis

Currently the Italian Civil Protection group is using *paper based methods* to collect data after a flood incident. In this manual system data is collected through interview and questioners. The data collectors go door to door to the victims' houses and ask various questions related to the flood event and the damages it caused and other relevant questions and record the answers on papers. The volunteers have to map the victim tale in the fields of the form, and this becomes a difficult task when the victim jumps from one topic to another without following the form structure. The solution we want to develop must address this behavior and try to figure out a fast way to move from one question to another, at least as fast as using paper.

This kind of data collection is in practice inefficient and lead to unreliable and uncompleted data. In order to solve this big issue, we tried to face the problem from a computer science and engineering perspective. We have to mitigate problems related to misplacement and misfiling of data and to avoid loss of documents, attachments and photos. We have also to reduce time loss for exchanging data between volunteers at different layers of the organization. These issues are solved by means of an infrastructure that takes into account data collection, storage and analysis. In other words, the Poli-RISPOSTA mobile app is a part of the solution and it is supported by a back-end Web portal.

In order to do so, we have to express some goals that our app wants to achieve. This goals are then expanded with the requirements analysis. The goals of our application are:

- Collect data in a fast and intuitive way
- Collect information that are easy to analyze afterwards
- Allow the search through questions
- Check at runtime the correctness of the data
- Let the user to jump from a question to another in a fast way
- Allow the users to make effective team collaboration
- Help users to efficiently collect data on the field by means of maps

The users of our application will be the volunteers of the Italian Civil Protection. They have to be registered on the system and they will be assigned to a team group, in which they will work on the field to collect data. We assume that the users have knowledgeable about the floods and related concepts, but

they do not have a deep knowledge of computer science and in particular of software. Our application must be as easy to use as possible, and must be intuitive to the user.

The final users of the application are only a subset of the stakeholders involved within Poli-RISPOSTA. The stakeholders are:

- Data collectors or volunteers, as said before
- The civil protection group, they are interested in an integrated solution able to manage coordination, data collection and analysis Poli-RISPOSTA mobile is just one of the tools needed by this stakeholder to perform his work
- Support and training staff, who is the stakeholder involved in the training phase of the volunteers
- Politecnico di Milano and Poli-RISPOSTA team, as the group involved in the development of the entire solution
- Victims, as people whose propriety suffered damages during or related to the flood event

2.1 Naming conventions and fundamental concepts

In order to understand the next sections of this document, it is important to define some concepts that stands behind this work. All this definitions are used to describe the functionalities of the system.

Building
A building is the target of the survey activity. The goal of a volunteer is to gather information of all the aspects of the building.

Housing unit
An housing unit is one part of the building, typically in houses or apartment blocks, it is a subject of the survey activity.

Annex
An annex can be a building in the same area of the original building, or a completely different one but related to the original building in some way. It is part of the survey activity.

Common part
A common part is a typical element in an apartment block and it is subject to damages as housing units or annexes. It is part of the survey activity.

Task
A task is a collection of forms defined afterwards by a volunteer on the server side of the system. A task is composed of different kind of forms, and for each one a maximum number of instances can be filled. Each task defines an assignment to be fulfilled by the volunteers during the day. Tasks can be created also by volunteers on the mobile app in case there is a building not include in other tasks.

Pre-filled element
A pre-filled element is a task or a form which contains a set of fields already answered. Usually tasks coming from the server are pre-filled in such a way that some information already known to the volunteers must not be filled from scratch. One of the most important pre-filled fields is the GPS position of a building, which allows the user to see that building on a map before reaching that building.

Form
The form is one of the most important element in our application. It is a model of the data and defines the structure of the data collection for a specific target (building, housing unit, annex or common part). Each time a user wants to fill a form, an instance is created starting from the model.

Form Instance
A form instance is an instance of the model. This structure holds all the answers and attachments of a form.

Section
A section is a structural subdivision of the form. In case of complex forms is extremely important to divide the questions in homogeneous sections allowing the user to move inside the form in a fast and efficient way. This is not only a logical subdivision, but affects also the graphical disposition of the questions inside a form.

Question
A question is another subdivision of the form. A question refers to a section, like a section refers to a form. A question is primarily a graphical subdivision of the form, allowing to show on the screen several fields at a time. It holds also references to the attachments.

Field
A field is the basic block of a from. For each field the user is required to provide an answer. A field can have only one data-type like numbers or strings or GPS coordinates. For each field a correctness and completeness control is active and the form engine handles the correct display of the fields at runtime depending on the field specifications.

Answer
An answer is the basic block of the form instance. Each answer refers to a specific field and contains the user input.

Attachment
An attachment can be a photo or a note. When the user is not able to express answers in the standard way, or when there are some inconsistencies in the answer, he can add a note or document the answers with a photo taken on the device or through an external camera.

Form search
In order to make more effective the filling process of a form instance, a form search is introduced. It searches through forms, sections, questions and fields in order to provide to the user the element needed without navigating in the form structure and wasting time. The search is performed considering only the words included within the form structure and not the answers.

Group
A group is a set of volunteers receiving tasks from the server side of the system. Generally it is composed by two or more volunteers collecting data on the field, one with the tablet, the other(s) with precision measure instruments.

Form Engine
The software component able to read the form model, to show at runtime the form structure, to save properly the data and to check the correctness and completeness of the user inputs. It is also able to show statistics on the form under completion and to show where there are missing or wrong data.

2.2 Functional requirements

In this section we list the requirements of the Poli-RISPOSTA app.

1. Login
   - Login user: Allow the user to log into the system to correctly perform data collection operations
   - Off-line operations: Allow the user to perform some operations even if the tablet is not connected
2. Map visualization

- Show the map: let the user to look at the map, scroll, zoom in and out, move to the Google maps app
- GPS localization: let the user to see his position in the map
- Survey area: let the user to see the area in which he has to perform his activity
- Task positioning: show all the tasks with GPS information on the map
- Task redirect: show basic information on tap on the Google maps pin and then after another tap redirect the user to the task page
- Team position: show on the map the last registered position of the other groups

3. ToDo list

   - List of tasks: Show the set of tasks the user has to perform, which are already stored on his/her tablet device
   - Task create: Let the user to create a new task from scratch when needed
   - Task remove: Let the user remove the selected task
   - Task send: Let the user to send a task to the server when the task is completed

4. Task status

   - List of forms: show the list of forms inside that task
   - Add/Delete forms in a task: let the user to modify the forms included in a task adding new forms or removing old ones
   - Show progress status: show to the user some qualitative information on the completeness of the various forms
   - Finalize form: allow the user to finalize a form and indicate whether it is suitable or not to be sent to the server

5. Edit form content

   - Show the form structure
   - Fill form: let the user to complete the form
   - Add Pictures: let the user to shoot a photo with the tablet or to choose one picture from the gallery
   - Add notes: let the user to add short notes for each question in the form
   - Support different answer data-types:
     - integer (positive, negative, standard)
     - float (positive, negative, standard)
     - string
     - gps
     - date and time
     - istat codes
     - measures
     - single selection
     - multiple selection
   - Achieve form flexibility: Let the user to duplicate questions and sections to express different information with the same form structure
   - Correctness: show if a question is filled in a correct way
   - Completeness: show if a form is fully filled in

- Not applicability: let the user to express whether a question is applicable or not (if the question applies to the specific case)

6. Search through the form
   - Search availability: let the user to search for questions inside Task status or inside the edit form content functionalities
   - Search through questions: the search must be performed on the form structure, not on the data gathered in order to help the user to find the correct question when necessary
   - Question redirect: let the user to be redirected to the correct question after search and after selection of a question
7. Download center
   - Show empty form to be downloaded
   - Show the forms currently inside the tablet
   - Show the task assigned to the group which holds the tablet
   - Form download: let the user to download the empty forms
   - Task download: let the user to download the task assigned to his group
8. Upload
   - Upload positions: The system must update the server on its position on a regular time basis
   - Upload task: The system, after user input, must send the selected task to the server including all its information

2.3 Non-functional requirements

In this section we list the non-functional requirements characterizing the Poli-RISPOSTA app.

1. Usability
   - The system must be able to guarantee fast responsiveness on different devices, adapting to their screen size. Due to the particular scenario, the project addresses only tablets, due to their flexibility on managing many information in the same view.
   - The UI must be clear and visible in different weather conditions
   - The UI must be responsive enough to let the user to be at least as productive as working with paper based forms
2. Reliability
   - The system must be reliable during its work
   - The system must be able to deal with exceptions or to difficult working conditions like lack of network signal or GPS signal
   - The system must guarantee the persistence of all the data filled in by the user and has to save information as soon as possible in a secure way
3. Security
   - The system must store the information collected in a secure way
   - The server must be able to acquire information using authenticated sessions
   - The server must be secured from unexpected and unauthorized accesses
4. Performance and Energy efficiency

- The system must operate for several hours.
- The mobile app must work without complex UI effects and without consuming too much battery
- The mobile app must be responsive and perform its internal processes in a fast and transparent way
- The mobile app must reduce memory usage and sensor usage as much as possible

5. Maintainability
   - The architecture of the application must be as general as possible
   - The algorithm used to generate the views must be general in such a way that every kind of form specified in a correct way can be processed and displayed

2.4 Use cases

In this section we will focus on the use cases that realize the goals and the functional requirements reported in Dection 2. This are the most important use cases implemented by the mobile application. We left out the login use case because it is very simple.

1. Map interaction
2. Task redirect
3. List of tasks
4. Create new task
5. Remove task
6. Send task
7. Show task
8. Add form
9. Delete form instance
10. Finalize/de-finalize form instance
11. Show form instance
12. Input data into form
13. Add picture
14. Delete picture
15. Add note
16. Delete note
17. Replicate question/section
18. Delete question/section
19. Reset question/section
20. Search for a field
21. Download form model
22. Download task
23. Delete form model from tablet

2.4.1 Use case: map interaction

Description a user interacts with the map

Pre-conditions the user must be logged in, the device must be connected

Post-conditions the map is visualized with the correct information

Use case flow

1. the user opens the app
2. the user clicks on "MAP" tab
3. the system shows the map centred in the user GPS position
4. the system shows the area in which the user have to work
5. the system shows the points in which are located the buildings to be surveyed and already surveyed
6. the system shows the points in which the other groups are located

Exceptions

1. the map is not displayed due to lack of internet connection
2. the information about area and group position are not displayed if the user is not logged-in or if the server is unavailable

2.4.2 Use case: task redirect

Description the user is redirected to the selected task from the map view

Pre-conditions teh user is logged-in and app showing the map view, internet connection working, task present on the system with a registered gps position

Post-conditions the user views the task status interface

Use case flow

1. the user clicks on a red Google map pin
2. the system shows some information about the form with that gps coordinates
3. the user clicks on the dialog
4. the system redirects to the task status interface

Exceptions

1. the map is not displayed due to lack of internet connection

2.4.3 Use case: list of tasks

Description the user can look at the list of tasks currently on the mobile device

Pre-conditions the user is logged-in

Post-conditions the system shows to the user the list of tasks

Use case flow

1. the user opens the app
2. the system shows the list of tasks currently on the mobile device

Exceptions none

2.4.4 Use case: create new task

Description the user can create a new task if none of the tasks assigned to him cover a building

Pre-conditions the user is logged-in

Post-conditions the new task is created

Use case flow

1. the user opens the app
2. the system shows the list of tasks currently on the mobile device
3. the user clicks on the *new task* button
4. the system shows the new task interface
5. the user types in name and description of the task
6. the user selects the forms to be added to the task
7. the user types in the maximum number of instances for each form
8. the user clicks on the *generate* button
9. the system creates the new task
10. the system shows the task status interface

Exceptions

1. the user does not type the name or the description
2. the user does not select any form

2.4.5 Use case: remove task

Description the user removes a task form the mobile app, deleting all the related information

Pre-conditions a task is present on the mobile device

Post-conditions the task is no longer inside the system

Use case flow

1. the user opens the app
2. the user long-clicks on a task
3. the system shows the *ToDo options* dialog
4. the user clicks on the *Remove taks* list item
5. the system delete permanently the task

Exceptions

none

2.4.6 Use case: send task

Description the user sends the task to the server

Pre-conditions user logged-in, one task present in the mobile device

Post-conditions the task is sent and can not be updated

Use case flow

1. the user opens the app

2. the user long-clicks on a task
3. the systems shows the *ToDo options* dialog
4. the user clicks on *send task* list item
5. the system connects with Google drive
6. the system checks the completeness of the task
7. the system sends the task to the server
8. the system prompts a success message

Exceptions

1. the app is not registered on Google drive, the system shows the login procedure
2. the task is not complete, the user is asked to decide to send it anyway or not
3. the server does not accept the task, the system prompts to the user an error message
4. the task does not contain any form, the system shows an error message
5. the task contains forms not finalized, the system shows an error message

2.4.7 Use case: show task

Description the system shows the task status interface to the user

Pre-conditions a task must be stored inside the mobile device

Post-conditions the system shows the tasks status

Use case flow

1. the user opens the app
2. the system shows the list of tasks
3. the user clicks on a task
4. the system opens the task status interface

Exceptions none

2.4.8 Use case: add form

Description the user adds a new form from the one allowed by the task and show it in the system

Pre-conditions there must be at least one form model inside the mobile app and at least one task

Post-conditions the system adds a new form and shows it to the user, the user clicks *back button* or *cancel* and the form is not added

Use case flow

1. the user opens the app
2. the system shows the list of tasks
3. the user clicks on one task
4. the system shows the task status interface
5. the user clicks on the *new form* button
6. the system shows a dialog interface with the forms that the user can add to the task
7. the user chooses one form and clicks on it
8. the system creates the new form
9. the system shows the new empty form

Exceptions

1. the task is already sent, the system does not allow to add a new form
2. the instances of the selected form reached the maximum amount fro the task, the system does not allow to add the new selected form

2.4.9 Use case: delete form instance

Description the user deletes a form inside a task

Pre-conditions the system has at least one task with at least one form instance

Post-conditions the selected form instance is deleted from the system

Use case flow

1. the user opens the app
2. the system shows the list of tasks
3. the user clicks on one task
4. the system shows the task status interface
5. the user long-clicks on a form instance
6. the system shows the *Task status options* dialog
7. the user clicks on the *delete* list element
8. the system deletes the selected form instance

Exceptions none

2.4.10 Use case: finalize/de-finalize form

Description the user finalizes a form, preparing it to the *send task* functionality

Pre-conditions at least one task present in the system, at least one form present in a task

Post-conditions the form is finalized and ready to be sent or de-finalized and modifiable

Use case flow

1. the user opens the app
2. the system shows the list of tasks
3. the user selects a task and clicks on it
4. the system shows the task status interface
5. the user long-clicks on a form instance
6. the system shows the *Task status options* dialog
7. the user selects the finalize/de-finalize option
8. the system finalize/de-finalize the form

Exceptions

none

2.4.11 Use case: show form instance

Description The user selects a form instance and the system shows the related content

Pre-conditions at least one task in the system, at least one form in the system

Post-conditions the user can look and edit the form content

Use case flow

1. the user opens the app
2. the system shows the list of tasks
3. the user selects a task and clicks on it
4. the system shows the task status interface
5. the user clicks on a form instance
6. the system shows the form edit interface

Exceptions

none

2.4.12 Use case: input data into form

Description the user can add new data in the fields of the form

Pre-conditions use case *show form instance*

Post-conditions the new content is saved in the DB storage

Use case flow

1. after the opening of the form edit interface, the system shows the form status
2. the user clicks on the list of sections
3. the system shows the list of questions belonging to that section
4. the user clicks on a question
5. the system shows the list of fields
6. the user type into a field some data
7. the user goes back or changes question
8. the system saves the edited content

Exceptions

1. the input is not valid, the system will not accept it

2.4.13 Use case: add picture

Description the user can add one or more pictures to a question

Pre-conditions use case *show form instance*

Post-conditions the image is added

Use case flow

1. the user clicks on a section

2. the system shows the list of questions
3. the use selects a question
4. the system shows the list of fields with the answers if already done before
5. the user clicks on the *add picture* button
6. the system shows the *pictures* dialog
7. the user chooses from which source import the picture, if camera or local storage
8. the system shows the related interface
9. the user chooses one picture or shoot a new one
10. the system adds the image to the question

Exceptions

1. the user cancel the operation in any of these steps, the image is not added to the question

2.4.14 Use case: remove picture

Description the user removes a picture from a question

Pre-conditions use case *add picture*

Post-conditions the image is no longer associated with the question

Use case flow

1. the user clicks on the image
2. the system shows the entire image with buttons *delete* and *cancel*
3. the user clicks on *delete* button
4. the system hide the dialog and deletes the image from the question

Exceptions

none

2.4.15 Use case: add note

Description the user add a note to a question

Pre-conditions use case *show form instance*

Post-conditions a note is added to the question

Use case flow

1. the user selects a section
2. the system shows the list of questions related to that section
3. the user chooses a question
4. the system shows the list of fields
5. the user clicks on the *add note* button
6. the system adds a note
7. the user types the new note

Exceptions

none

2.4.16 Use case: delete note

Description the user deletes a previously created note

Pre-conditions use case *add note*

Post-conditions the note is removed

Use case flow

1. the system shows the list of fields
2. the user clicks on the *delete note* button
3. the system deletes the note from the question

Exceptions

none

2.4.17 Use case: replicate question/section

Description the user decides to replicate a question or a section to the form instance

Pre-conditions use case *show form instance*

Post-conditions the question/section is repeated

Use case flow

1. the user long-click on a section/question
2. the system shows the *edit form options* dialog
3. the user clicks on *repeat section/question*
4. the system replicates the selected question/section

Exceptions

1. the selected section/question is not repeatable, the system shows an error message

2.4.18 Use case: delete question/section

Description the user deletes a question/section from the form instance

Pre-conditions use case *show form instance*

Post-conditions the question/section is deleted

Use case flow

1. the user long-clicks on the selected question/section
2. the system shows the *edit form options* dialog
3. the user clicks on *delete question/section*
4. the system deletes the selected element

Exceptions

1. the selected question/section is the last one of that kind of section/question, the system shows an error message

2.4.19 Use case: reset question/section

Description the user decides to reset a question/section

Pre-conditions use case *show form instance*

Post-conditions the selected question/section has only empty answers, all the answers related are deleted

Use case flow

1. the user long-clicks on the selected question/section
2. the system shows the *edit form options* dialog
3. the user clicks on *reset question/section*
4. the system deletes the answers of that section/question

Exceptions

none

2.4.20 Use case: search for a field

Description the user searches for a field inside a form. This use case is valid also for question/section searches

Pre-conditions at least one task inside the system, at least one form inside a task, task status interface opened or form edit interface opened

Post-conditions the user can modify the searched field

Use case flow

1. the user clicks on the magnifier icon
2. the user types the search keywords
3. the user clicks *enter* on the device keyboard
4. the system shows the search interface
5. the user selects the element to search (field, question, section)
6. the user selects the form in which the system has to search (if coming from the task status interface)
7. the user clicks on one search result
8. the system shows the related item on the edit form interface

Exceptions

1. the search does not show any result, the user must go back and retype

2.4.21 Use case: download form model

Description the system downloads an new form model, needed to perform the tasks

Pre-conditions user logged-in, internet connection available

Post-conditions the new form model is downloaded and can be used inside the system

Use case flow

1. the user opens the app
2. the system shows the list of tasks

3. the user clicks on the *downloads* tab
4. the system shows the list of form models inside the tablet, the form models in the server and the tasks assigned to the group
5. the user long-clicks on a *remote* form model
6. the system shows the *download options* dialog
7. the user clicks on *download form* list element
8. the system downloads the selected form model

Exceptions

1. lack of internet connection, the downloads tab shows only the local form models
2. the form model is already available locally, the system shows an error message

2.4.22 Use case: download task

Description the user asks the system to download a task from the server

Pre-conditions user logged-in, internet connection available

Post-conditions the task is downloaded and can be completed

Use case flow

1. the user opens the app
2. the system shows the list of tasks
3. the user clicks on the *downloads* tab
4. the system shows the list of form models inside the tablet, the form models in the server and the tasks assigned to the group
5. the user long-clicks on a *remote* task
6. the system shows the *download options* dialog
7. the user clicks on *download task* list element
8. the system downloads the selected task

Exceptions

1. lack of internet connection, the downloads tab shows only the local form models
2. the task is already available locally, the system shows an error message

2.4.23 Use case: delete form model from tablet

Description the user deletes a form model from the system. This cause the deletion of all the tasks requiring that form model

Pre-conditions at least a form model inside the system

Post-conditions the selected form model is deleted from the system

Use case flow

1. the user opens the app
2. the system shows the list of tasks
3. the user clicks on the *downloads* tab
4. the system shows the list of form models inside the tablet, the form models in the server and the tasks assigned to the group

5. the user long-clicks on a *local* task
6. the user clicks on *delete local form* list element
7. the system removes the selected form model

Exceptions

none

3 Design

In order to achieve the requirements specified in section 2, we need to address a complex design able to take into account several aspects of the system. In the next sections we will present the architecture of the system (developed as a client-server application), the database, the user interface and the software architecture of the mobile application. In this Section we will describe in detail only the mobile application design, while the server side of the system is not part of this work. For completeness, we will briefly describe only some of the key aspects of the server in section 3.1, the one that affects the behaviour of the mobile application.

3.1 System architecture

The system architecture can be classified as the classical 3-tier client-server architecture. On the server side the system has the storage layer, on which all the final information are saved. Moreover there is an application layer useful to control communication between client and server. The application layer is also present, with a more complex architecture on the client side. The client is also responsible for the presentation of the data, which are stored locally for the time needed to perform the required operations. In Figure 1 we can see a simple overview of the entire system.

The server side of the system provides the API required for forms download, the tasks and information about survey area and groups on the field. It allows also to upload the completed tasks and information about the current position.

External services like Google maps and Google drive are used respectively to get access from the tablet to map services and remote storage services for picture uploads.

On the client side, starting from the information coming from the server, the user can perform different operations. In Figure 1 we have listed some macro-functionalities offered by the app. The user can view the map and get information on his survey area while looking at other team positions. He has also the possibility, after downloading the assigned tasks, to see where he has to go in order to fulfill the assignments.

The user can download the tasks assigned to his group, and after that, he has the possibility to access to the task management functionality. In this view the user can add new forms or modify the pre-filled ones. When the user selects one form, the form engine processes the selected form model and opens the form editor, in which it is possible to fill the required questions. If the user does not find a question or a field, he can search for it through the search functionality.

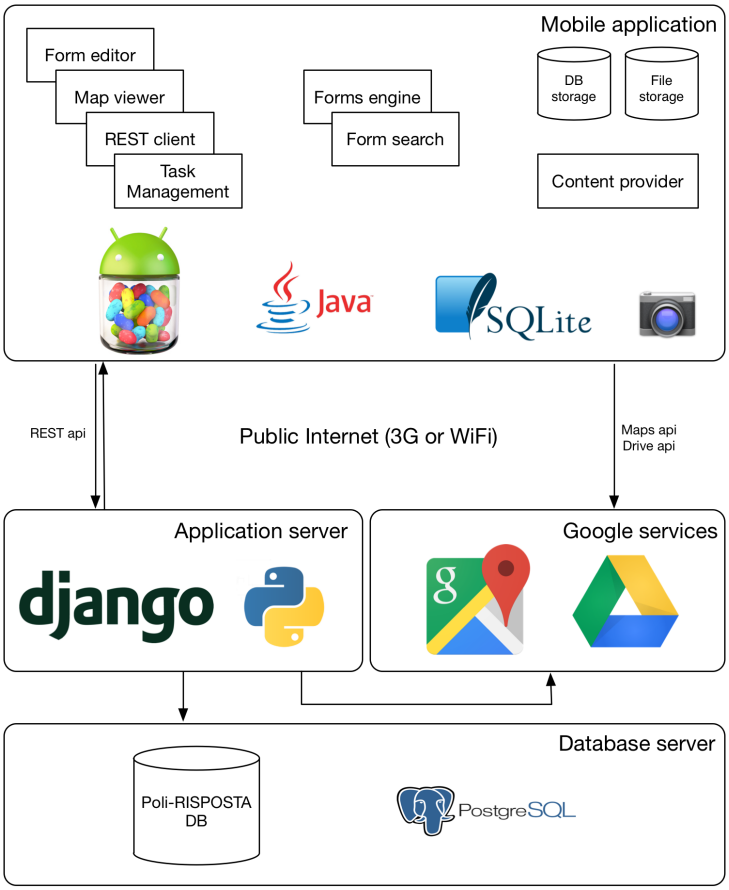

Figure 1: System architecture

3.2 Data design

In this section we want to explore the data model behind the mobile application. There are few guidelines for this design that we decided to follow in order to achieve our goals and requirements. We want to handle forms in the most general way, in the sense that the application is able to persist data into this structure independently w.r.t. the form used. Most of the solutions currently available on the market (like Open Data Kit[1]) use the database only to store data and not form models. In order to do so they create one table for each form model inside the system. This approach works in case of data analysis with forms with fixed data structure, but it does not fit to a mobile application which focuses only on data collection and not on data analysis. Moreover, with our database schema, is no longer needed to delete the old data or the entire table in case of update of the form structure. In Section 3.2.1 we will focus on the upper part of Figure 2, looking at the tables that store the form models. In Section 3.2.2 we will move on the bottom part of Figure 2, looking at the tables that store the instance data. Finally, in Section 3.2.3 we will see the entities storing task information.

3.2.1 Form model entities

As form model entities we intend the entities Form, Section, Question, Resource, Field, Choice. This entities stores and represents the form model. Each time a form must be rendered on the screen, the application looks at this table.

Form

The Form entity stores the information related to a form, like id, name, text name and description. Name is a unique string that identifies the form, text name and description are respectively the name and the description showed by the application in the list of forms.

Section

The Section entity stores the information related to a single section within a form. One section is connected to only a form, while a form can contain more than one section. As attributes, the section entity has id, name, textname, description, required, repeat. The first four attributes are similar to the ones in the form entity. Required is a flag that says that the section is required to be completed by the user, repeat instead, is a flag that says if a section can be repeated more than one time.

Question

The Question entity stores the information related to a single question within a form. One question is connected to a single section, while a section can contain more questions. Moreover, for each question there can be more resources attached. In this version of the application, we can manage only images, but in a future version we may add audio or video resources. The attributes of the entity are the same as the one presented for the section.

Resource

The Resource entity stores the information related to an image or audio or video within a form. In this verison of the app we support only pictures, but we may add more type of resources in the future. Each resource is related only to a question, while a question may have more then one resource attached. The attributes of the entity are: id, name, path, type. Name is a unique string identifier, path is the identifier of the resource on the local storage and type holds information about the type of the resource, like image, audio and so on.

Field

The Field entity stores the information related to a single field of the form. Each field is related to only one question, while a question may have more then one field. The attributes of the entity are: id, name, textname, type, required, repeat, cond, condition_statement, op1, op2. We want to focus on the type attribute. It stores the information related to the type of field, and this affects the visualization of the field in the UI and the way in which the answers are stored in the DB.

[1] https://opendatakit.org

The data-types are: string, integer (positive, negative or standard), float (positive, negative or standard), gps, date, date time, istat code, measure, single selection or multiple selection. Other interesting attributes are cond, condition statement, op1 and op2. They are the attributes used to express conditioning between fields.

Choice

The Choice entity stores the choices for fields of type single selection or multiple selection. One choice is related to one field, while a field can have more than one choice. The attributes are: id, text, other. The attribute other is a flag used to say whether or not that choice is a other choice. In the user interface the user can fill the other choice with the data he/she wants if there is not a choice that fits that data.

3.2.2 Instance entities

As instance entities we intend the entities Form_instance, Section_instance, Question_instance, Note, Answer, Resource_instance.

Form_instance

The Form_instance entity stores the information related to the instances created by the user or downloaded as pre-filled by the server. Each form instance refers only to one Form entity, while a Form entity can have more than one Form_instance.

Section_instance

The Section_instance entity stores the information related to the sections created within a form_instance. Each section_instance refers to only one form_instance, while a form_instance can have more than one section_instance. A section_instance refers to only one section, while a section can have more section_instances. The entity contains the attributes id, index. The index attributes is used as weak key in order to address and store replicated sections.

Question_instance

The Question_instance entity stores the information related to the questions generated within a form_instance. Each question_instance refers to only one section_instance, while a section_instance may have more question_instances. Each question_instance refers to only one question in the model, while a question may have several question_instances. The attributes are id and index. As for section_instance, the index attribute is used as weak key in order to store repeated question_instance belonging to the same form_instance and the same form model.

Resource_instance

The Resource_instance entity stores the attachments for the specific form_instance. An attachment is related to only a question_instance, while a question_instance may have more attachments. The attributes are id, type, path. Type is used to express whether a resource_instance is an image or an audio tape. Actually the application supports only images as attachment.

Note

The Note entity stores the notes written by the user during the survey activity. A note refers only to one question_instance, while a question_instance may have more than one note. The attributes are id and text.

Answer

The answer entity stores the answers prompted by the user during the survey activity. Each answer is related to only one question_instance and to only one field. A field can have more answers related and also one question_instance may have more than one answer. The attributes are id, prefilled, applicable, type, filled, answer_string, answer_integer, answer_float. The prefilled attributes says if the answer was pre-filled and thus downloaded from the server. The applicable and the filled attributes says respectively if the answer (and so the related field) is meaningful and if the answer was completed. The type attribute has the same rules as the one in the field table. The answer_

attributes are used to store the answer given by the user. Depending from the type attribute, they are used in different ways.

3.2.3 Task entities

As Task entities we intend the entities responsible to store the information related to the tasks and their specification, like Task and Task_module.

Task

The Task entity stores the information related to the task assigned to a group. Each time a user downloads a task from the server, its basic information are stored in this entity and in Task_module. Each task can contain more than one form_instance, while one form_instance is related to only one task. Each task can have more than one task_module, while one task_module is related to only one task. The attributes are id, name, description, group_id, generated, user_defined, sent. The attributes name description identifies the task from the user point of view. The attribute group_id refers to the group which the task is assigned. Generated is a flag that says if the task is already generated in terms of form and form_instances. User_defined is a flag that states if the task comes from the server or if the user has created it. The sent flag is used to store whether the task was sent back to the server or not.

Task_module

The task_module entity stores the information related to the elements that must compose a task. Each task_module is related to only one task and specifies which form model is related to the task. A task can have more than one task_module and one form_instance can be related to more than one task_module. The attributes are id and n_instances. This attribute states the maximum number of the selected form_instance can be placed inside a task. If the value is 0, there is no limit on the number.

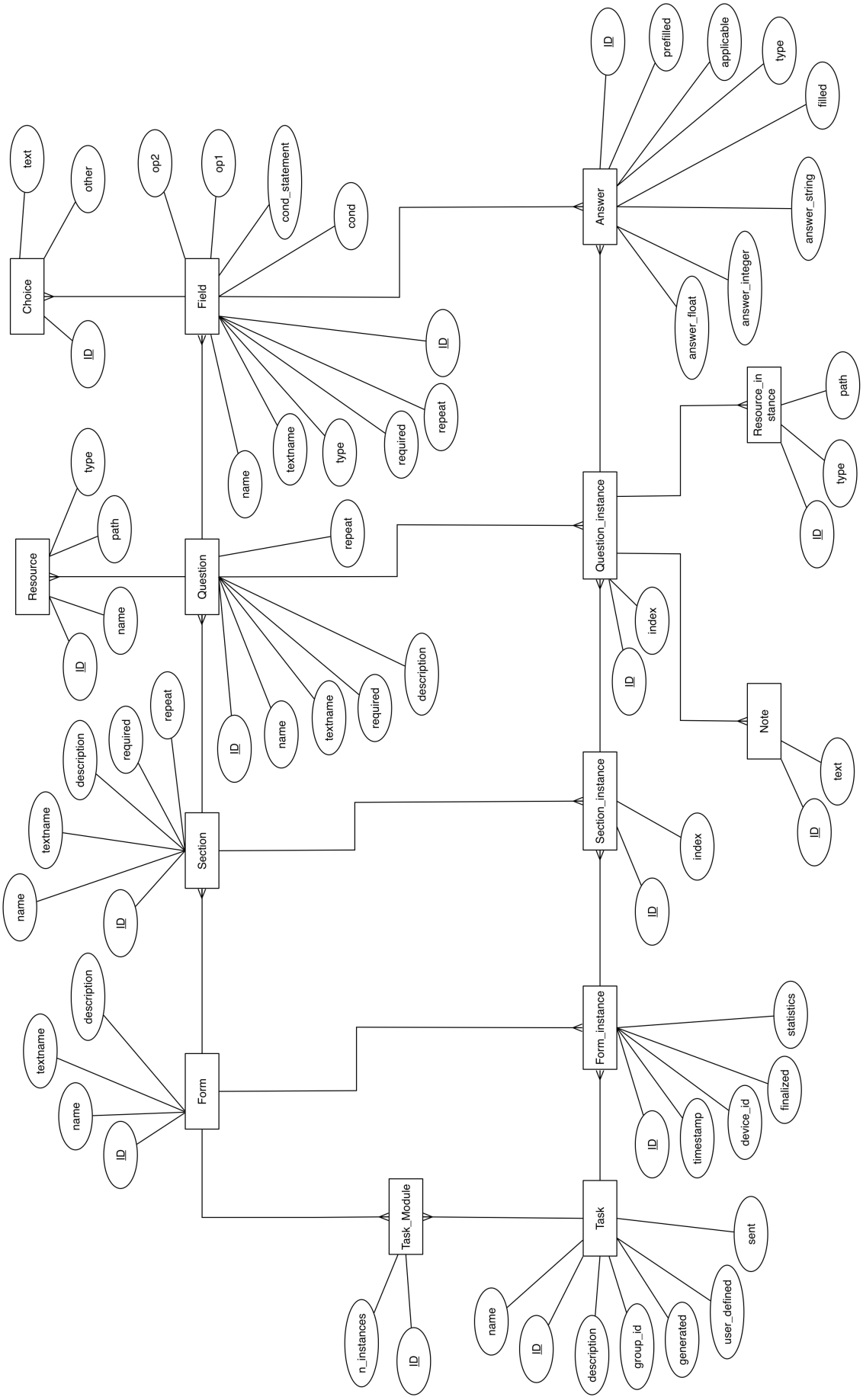

Figure 2: Data architecture

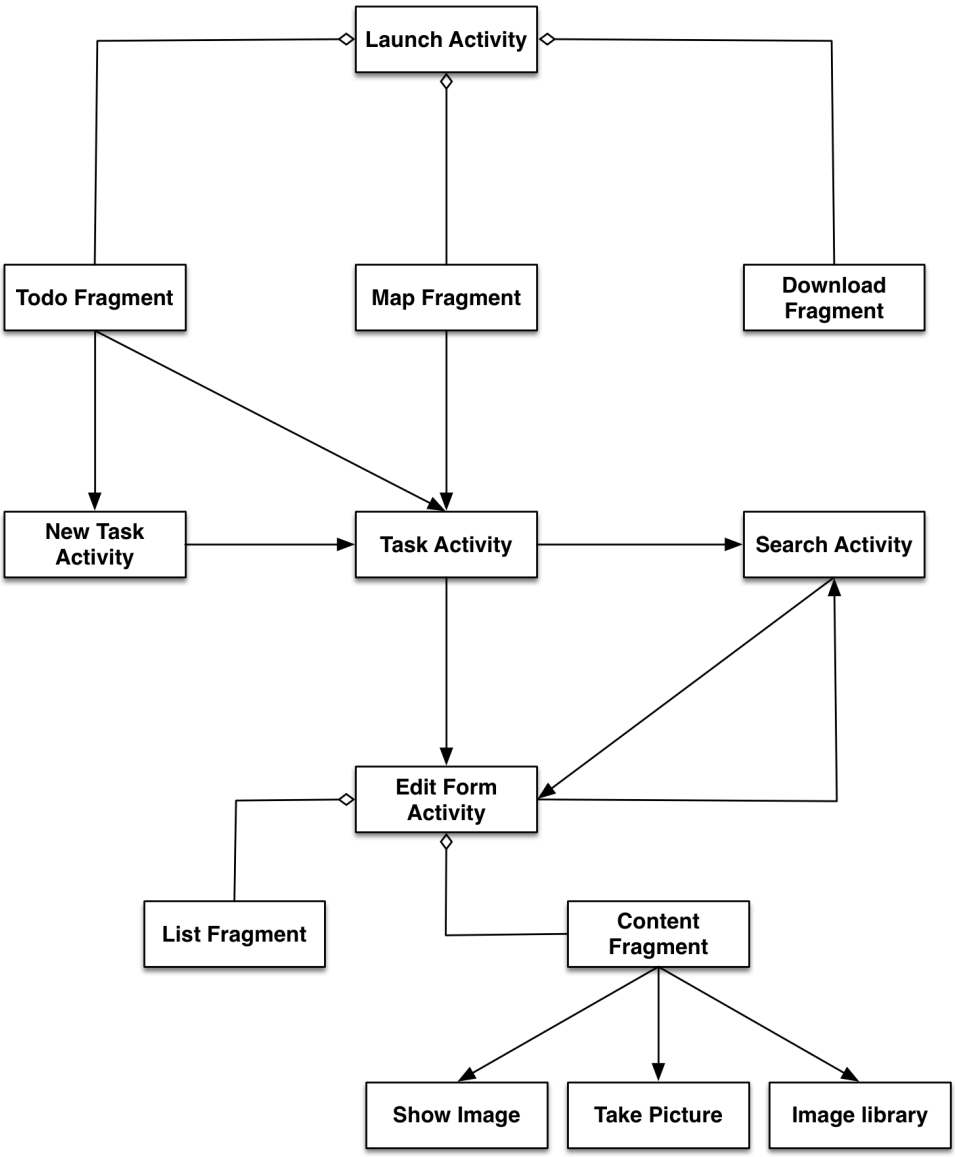

Figure 3: UI architecture

3.3 User interface

In this section we explore the user interface able to provide the functionalities intorduced in Section 2. The Android OS provides some facilities in order to develop the user interface, starting from activities and fragments. In Figure 3 the activities and fragments composing the app are listed. A fragment is a portion of the user interface that can be changed at runtime, helping the developer to build a smart UI with dynamic components. The most complex fragment is the ContentFragment, which belongs to the Edit Form Activity. His goal is to show the fields inside a selected question, reading the contents from the database and showing the UI elements depending on the form structure.

In the same graph, activities and fragments are connected by arrows, which shows the intents used to move from one activity to another. The navigation is from the top activity to the bottom one, and it is always possible to go one level up using the Android back button.

3.3.1 Launch activity

The launch activity is the first activity displayed by the application after launch. It is composed by 3 fragments: ToDo, Map and Downloads. The first fragment allows the user to control the local tasks, the second helps the user with Geo-localization and group coordination, while the third one is the one deputed to communicate with the server. The fragments are organized as horizontal scrolling tabs. On the top left corner there is a wheel icon, which opens the settings view. The user can insert the login data with the address of the server.

ToDo Fragment

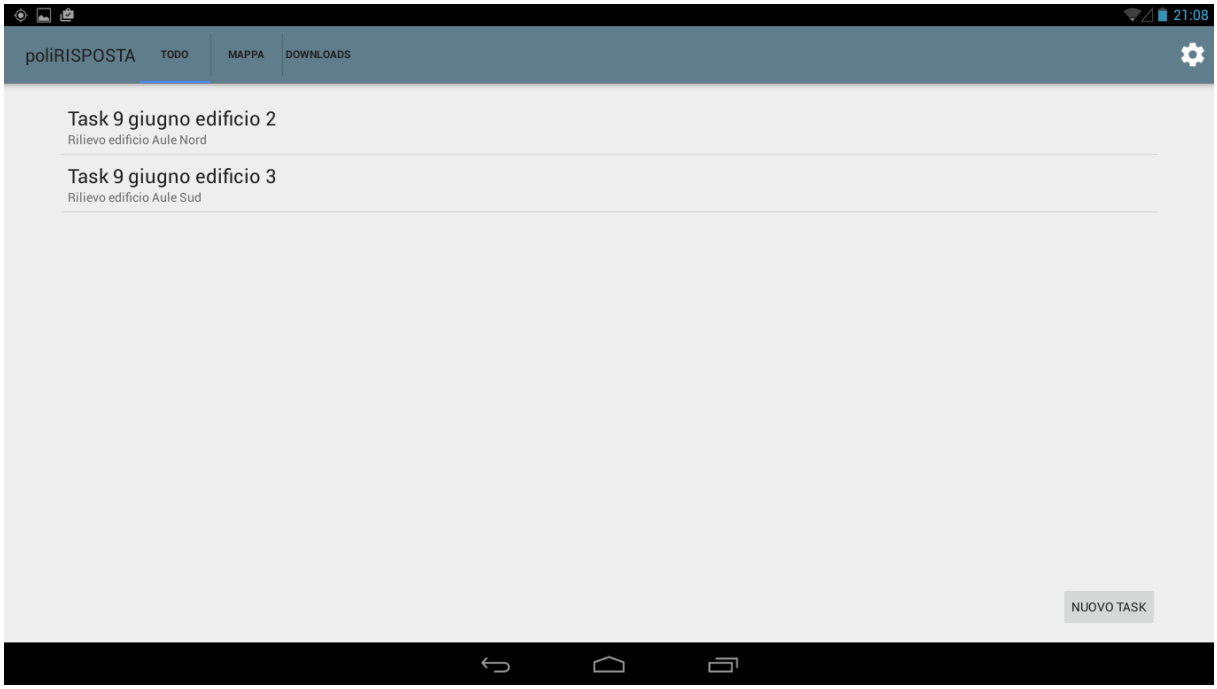

Figure 4: ToDo fragment

The ToDo fragment (Figure 4) is the landing view of the application. Each time the user opens the Poli-RISPOSTA mobile app, the system shows this view. From the ToDo fragment the user can select a task present in the system by clicking on the chosen element in the list; decide to create a new task; delete a task or upload it to the server. In this latter case the user has to long-click the selected element until a context menu appears. The system check if all the fields are answered and if not asks for a send confirmation.

Map Fragment

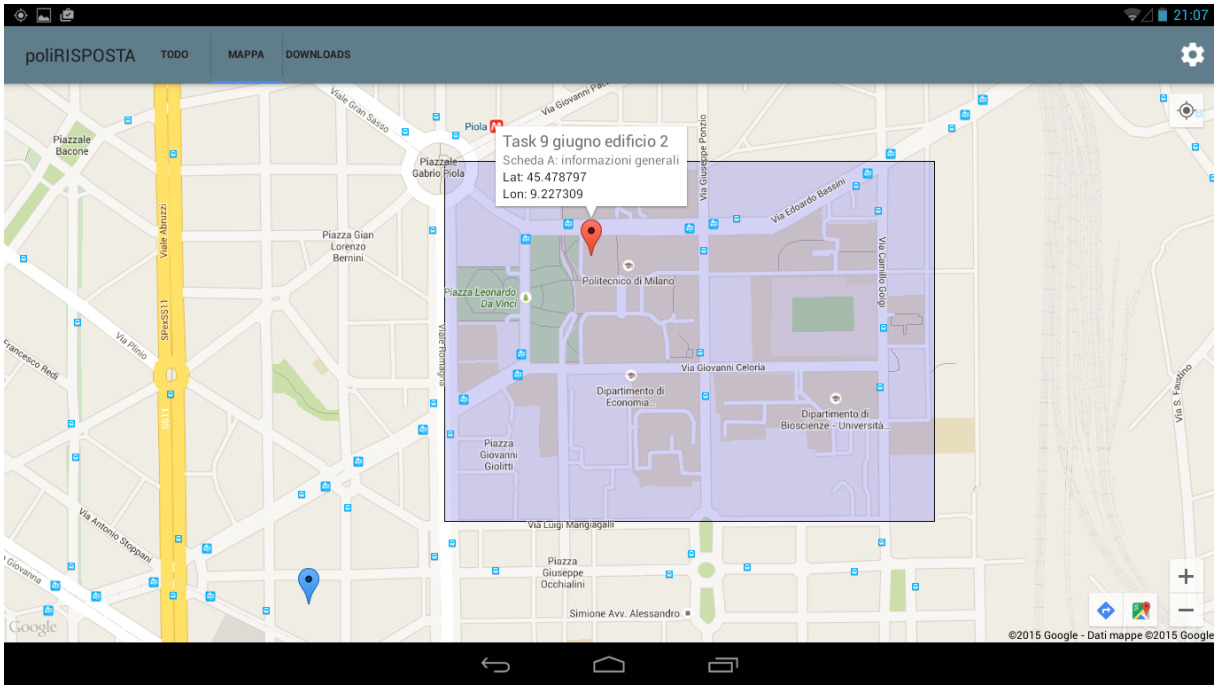

Figure 5: Map fragment

The map fragment (Figure 5) is the second view listed in the action bar. It displays the map, localized to the users position. It is responsible to show the survey area, in which the user has to complete the tasks. It has also to show the gps position of each group and of each task inside the system. For each Google maps pin, the map can show a brief description of the element. If the element is a form instance, the user can click on the description and the system will redirect the user to the related task.

Each time the map is loaded, the app asks to the server for updated data of positions and area. Within this fragment, the system triggers periodically a position update, which is sent to the server to inform the other groups about the users position.

Each gps position inside the map can be passed to the Google maps application in order to leverage its navigation facilities. This can be done after selection of a pin, clicking to the bottom right buttons of the map fragment.

Download Fragment

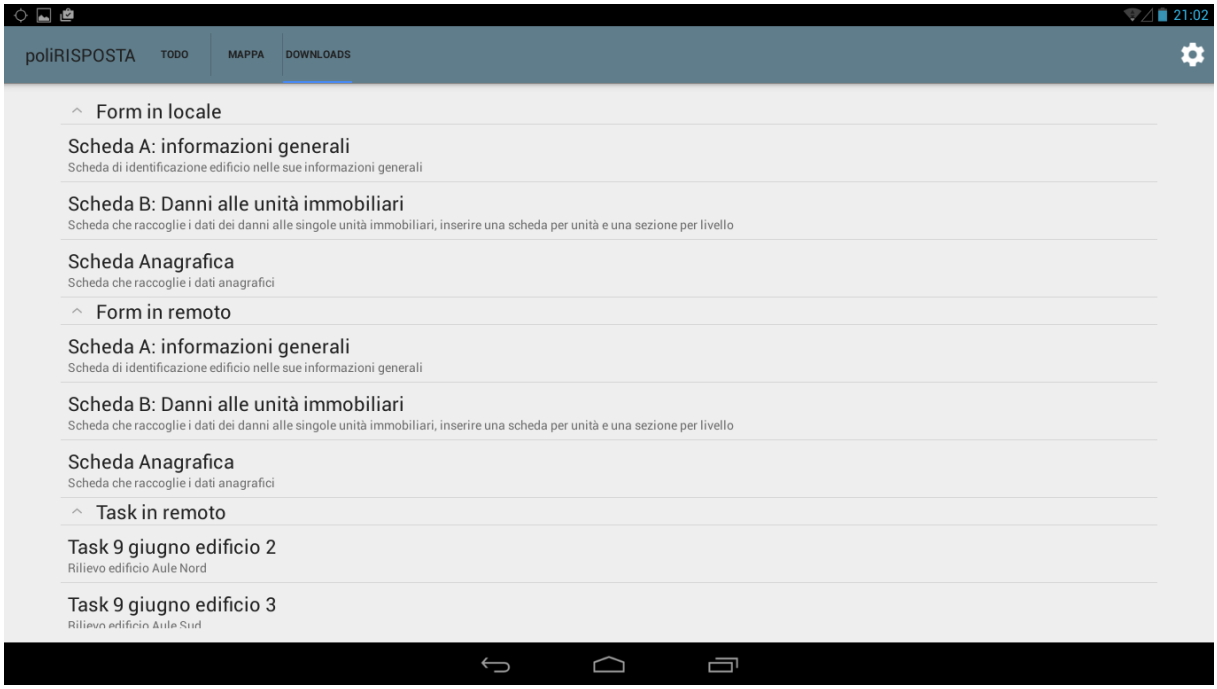

Figure 6: Download fragment

The download fragment (Figure 6) is the fragment responsible of all the incoming communication with the server side of the Poli-RISPOSTA infrastructure. It contains a list of different items, starting from the list of local form models, the list of remote form models, and finally the list of remote tasks. A long-click on each element shows the options menu, and after that it is possible to download the selected element. The local form models list, instead, allows the user to delete a form model from the system (this action will remove all the tasks referencing that form model).

3.3.2 New Task Activity

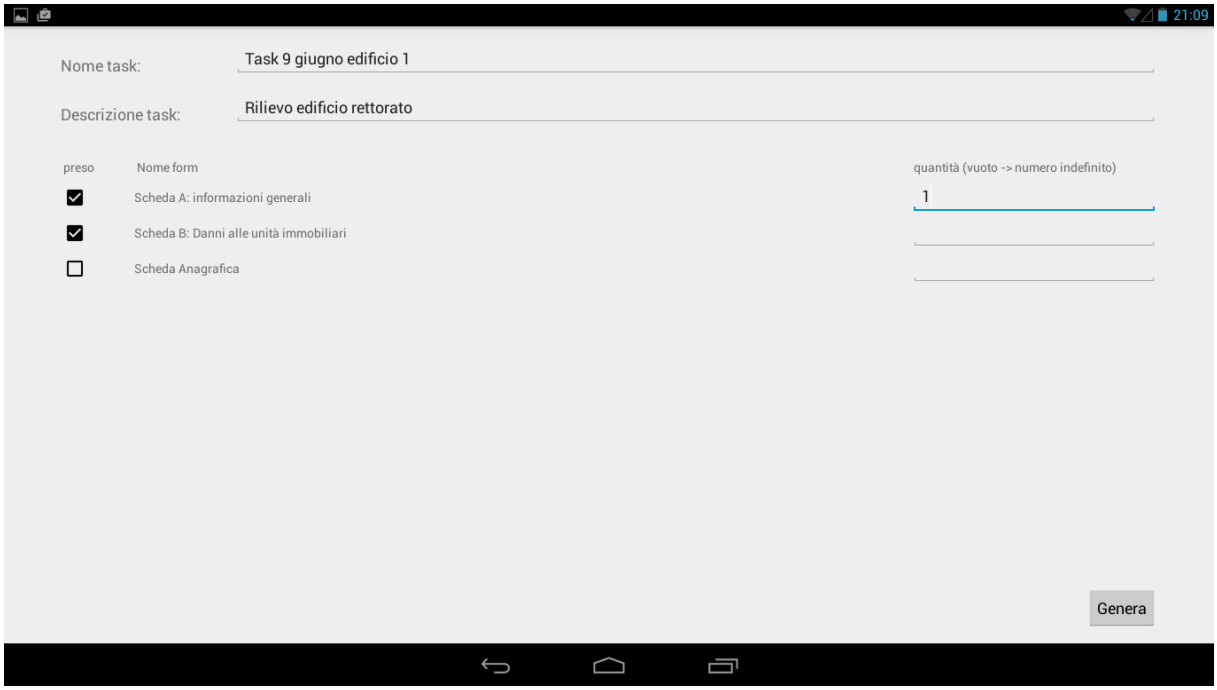

Figure 7: New task activity

The new task activity (Figure 7) allows the user to add a new task into the system. The user has to choose the name and a brief description of the task. Then the user has to choose the form models that can be used in the task and their maximum number. If the number field is left empty, the system assumes that there is no limit for that form. After the creation process, the user can click on the *generate* button, which generates the task and redirects to it.

3.3.3 Task activity

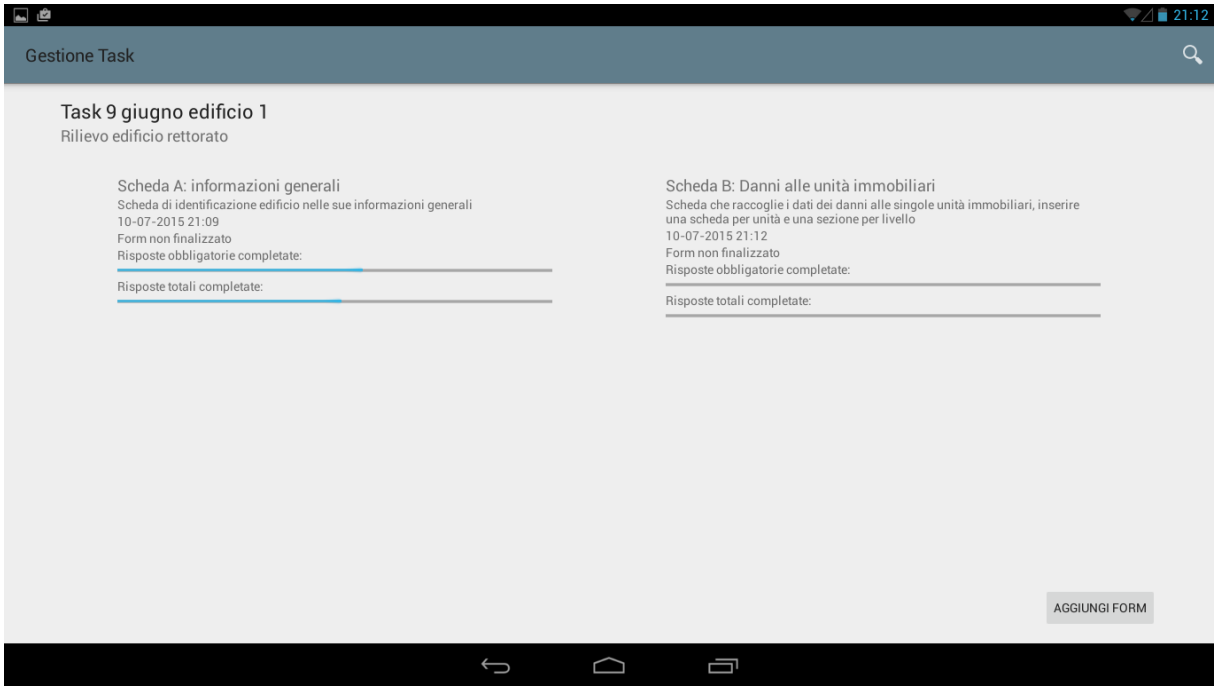

Figure 8: Task activity

The task activity (Figure 8) allows the user to manage a task. The activity shows a grid of the form instances already present inside the task. For each form instance, there are some information, like name and description of the form, the date and the status in terms of answered questions. The user can add a new form instance by clicking on the *add form* button. The system shows a context menu from which the user can choose between the form models attached to the task. If the user clicks on one of the elements in the list, the system creates and opens a new form instance. This form is also added to the list of form instances in the activity of Figure 8.

Each form must be finalized in order to be sent inside a task. If a task does not contain form instances or if they are not finalized, the form cannot be sent back to the server. To finalize a form the user has to long-click on the form and select the proper choice in the context menu. The menu shows also the possibility to delete a form instance in a task.

With the magnifier icon on the top left corner of the screen, the user can search through the sections, questions and fields of all the forms present in the task.

3.3.4 Edit Form activity

The edit form activity is the heart of this project. It allows to store the information of the forms in the database and to actually perform the data collection. It is composed by two fragments: the list fragment, which holds the sections and the questions of the form and the content fragment, which contains the fields, the images and the notes. At the beginning the activity shows the home fragment with statistics about the fulfillment of the form.

In the action bar there is also the magnifier icon, which calls the search activity, but this time focusing only on the form currently opened.

Edit form activity - Home

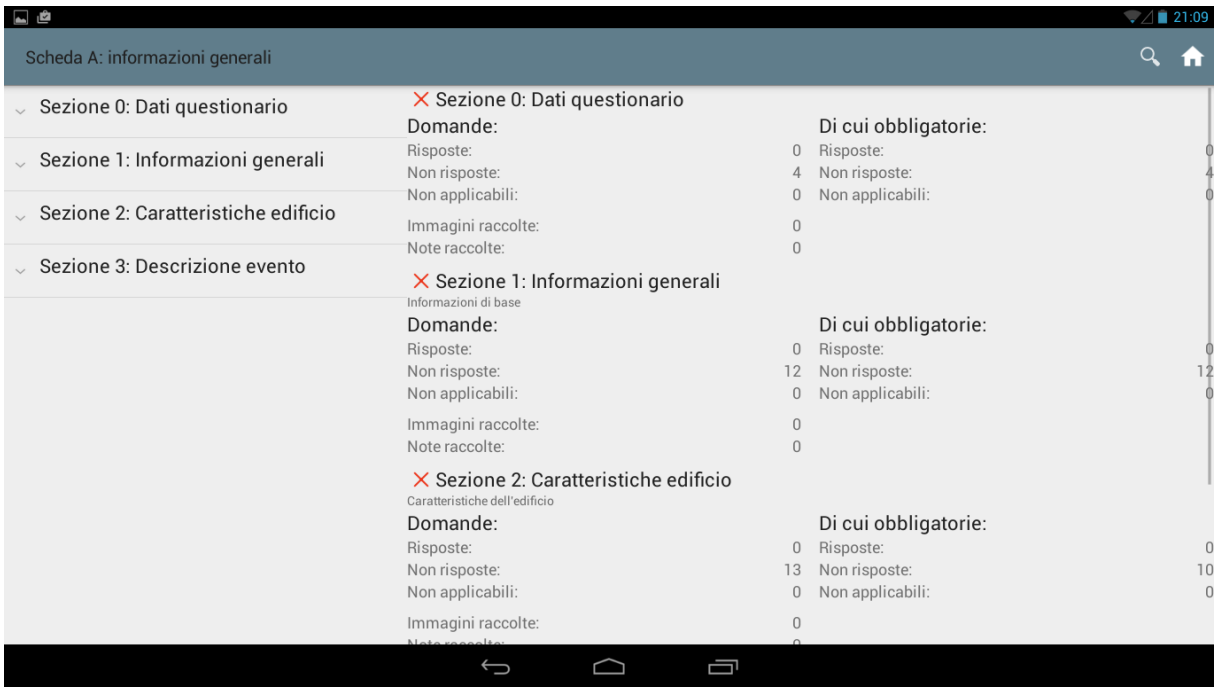

Figure 9: Edit form activity - Home

In this first fragment (Figure 9), which can be called with the home button in the left side of the action bar, all the statistics about the form are listed. For each section, the view shows the number of filled, not filled and not applicable answers given by the user. There are also the same kind of statistics for the mandatory fields. For each section the app computes the number of pictures and the number of notes collected.

Edit form activity - Editing

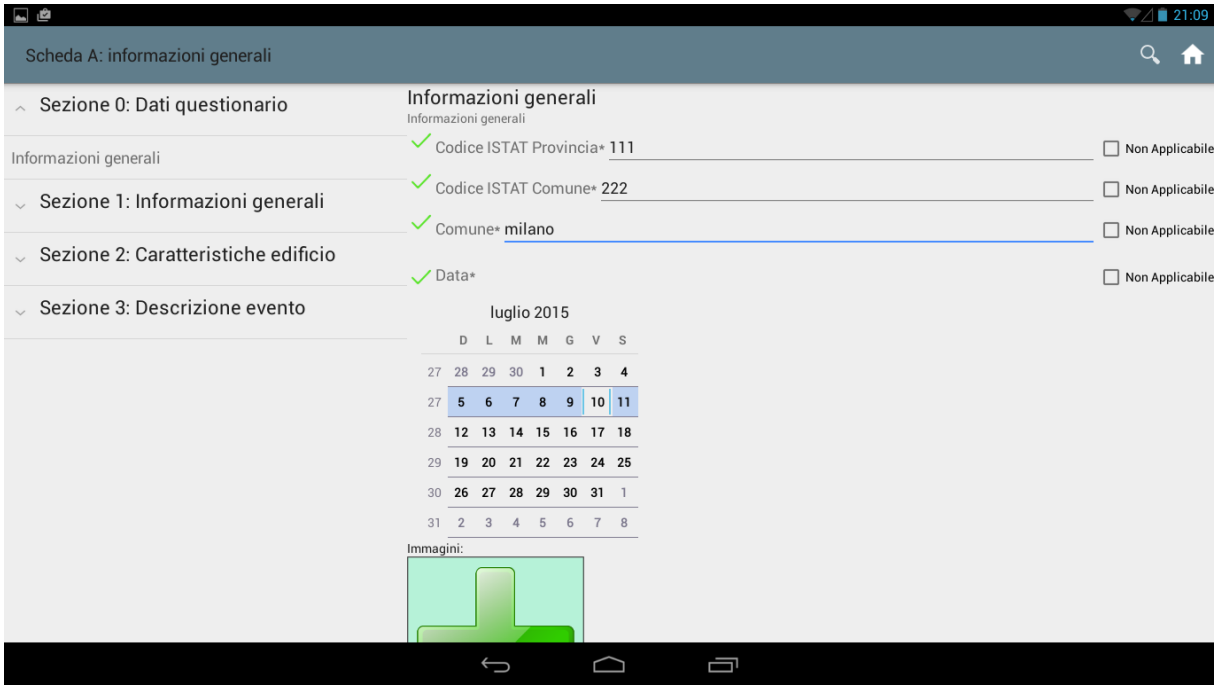

Figure 10: Edit form activity - Editing

Figure 10 shows an example of a question inside a form. The fragment shows all the fields belonging to a question with the related answers. For each field the system checks at runtime the correctness of the input and highlights it with a green tick if it is correct. If not, the system shows a red cross. If the field is not applicable, the system shows a yellow exclamation mark. Instead, the completeness check is spread through this activity, the task activity and the ToDo fragment, in which in the sending phase the system performs the final completeness check.

In any question the user can attach a photo allowing further analysis after the data collection phase. It is also possible to add notes.

Within the edit form activity, if the form model allows it, the user can duplicate a section or a question. This is useful when there are several homogeneous information, and we want to collect all without limiting their number.

The user may want to delete a section or a question. This can be done only if the selected section or question was previously duplicated. For each form instance there must be at least one section or question instance related to each question or section in the form module.

In order to delete the answers contained in a certain question or section, the user can reset them. After that the answers becomes empty. All this functionalities are accessible long-clicking the element on which the action has to be performed in the list fragment.

3.3.5 Search Activity

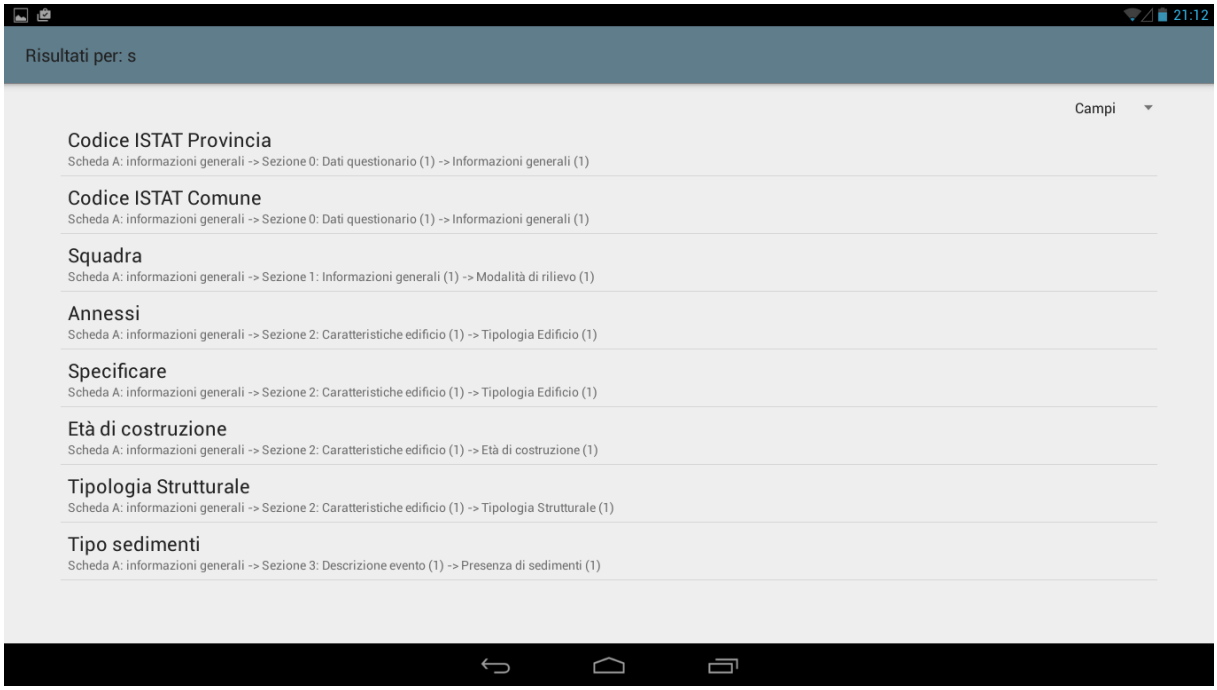

Figure 11: Search Activity

The search activity (Figure 11) is another important view in the application. It realizes one of the most important needs of the stakeholders, because allows the user to search through the form model in order to find the correct question or section or field for the circumstance. The structure of a form does not follow a flow because the surveyed tell the flooding event in a non-linear way w.r.t. the form structure. The volunteer needs to move from one question to another in a fast way, and the search functionality comes into play when the user does not know the exact position or name of the needed question.

The user must type the search keywords in the task activity or in the edit form activity. The system redirects the user to the search activity, and allow the user to filter the results for question, section or field. If the search is launched from the task activity, the interface shows also a drop-down menu to choose the form on which to show the results.

If the user clicks on one of the results listed in the view, the system opens the edit form activity with the selected item.

3.4 Software architecture

The mobile application is split into different packages, typically one for each activity. Each package collects the view code and the methods responsible to manage and store in the model the data coming from the view. The model is in a separate package, with model classes, database definition and content provider. The packages are:

- launch.activity, which collects the activity, the fragments and the async task that manages the server connection
- task.activity, which collects the classes related to task activity and new task activity and the adapter for the task list view
- form.activity, which contains the activity, the related fragments, all the async tasks which manages the database interaction and the listeners for the view events, like the request for a position update
- search.activity, which contains the activity, the adapters for the list view and the async task responsible to the form search
- model, which contains all the classes containing the data coming from the database, the database definition and the related content provider

In the next sections we will focus on some design decisions taken during the design and development phase.

3.4.1 Model classes

In order to manage the data coming from the database, we defined a set of model classes. For the answers we defined an abstract class, extended depending on the data-type of the answer.

- Form
- FormInstance
- Section
- SectionInstance
- Question
- QuestionInstance
- Field
- DataType (enum)
- Answer (abstract)
  - DateAnswer
  - FloatAnswer
  - GpsAnswer
  - IntegerAnswer
  - IstatAnswer
  - MeasureAnswer
  - NegativeFloatAnswer
  - NegativeIntegerAnswer
  - PositiveFloatAnswer
  - PositiveIntegerAnswer

  - Select1Answer
  - SelectAnswer
  - StringAnswer
- Task
- Note
- Resource
- ResourceType (enum)

3.4.2 Form engine

The form engine is the part of the application able to take the user input and to manage it, starting from the storage in the database, moving to the correctness and completeness check and the statistics computation. Given the event based paradigm of the Android applications, the form engine starts each computation after an input event.

When the user selects a form, the form engine loads it from the database and shows it in the view. When a question is selected, the form engine reads the content of the question from the form model and creates the right interface to display dynamically the field views. Each field is generated starting from a view XML description that is customized depending on the field type and text. The views are inflated in the content fragment, which contains the name, the description of the question, the list of fields, grouped in a linear layout, the list of images and the list of notes.

When the user answers to a question, the system stores the answer and check if the content of the answer is correct for the data type. If the answer is not applicable, the content is saved in any case, but is not further considered. When the user leaves the question to move to another or closes the activity, the system computes the new statistics related to the form instance.

When is needed and if the form model allows, the user can replicate a question or a section. In this case the system looks at the form model and replicates the selected item creating a new instance of it. This allows a flexible data collection w.r.t. paper based one, in which the number of eventual replicas is fixed.

This approach allows to specify a form model in terms of rows in a database, and the mobile app is in charge of rendering the model in the proper way. The administrator of the system is only in charge to define the form model in terms of form structure and type of the fields. He/she can specify if a question or a section is mandatory and/or repeatable.

Other approaches to the form modeling, like the one involving XForms, stores the answers for each form in a single huge table, with hundreds of columns. The form model defines the table attributes, while mapping the questions to that attributes without taking into account the structure of the survey. Our approach tries to manage the structure from a logical and graphical point of view. This means that the form specification takes into account a different way to show information, which is more flexible for our case study. Moreover, with our approach, when the form structure changes, the data inside the database are not affected by the change in the structure.

3.4.3 Async tasks

The Android OS runs on resources with limited performance an power consumption. In order to maintain the UI responsive in each condition, the most time consuming operations must be done in background. Moreover, the networking operations must be done in background, due to OS limitations. Given this environment, we decided to move all the complex database operations and the networking activity on several async tasks, a particular implementation of the thread library. With this classes, the UI is constantly updated with the information coming from background threads. Moreover, the code under the onPreExecute and onPostExecute methods are executed in the UI thread, allowing to warn the user that something is happening. External APIs, like Google maps and Google drive are asynchronous by

design, for the first one we use it in an asynchronous way, while the second is used synchronously inside a background thread.

The async tasks we created are:

- SetupDB, which opens the form model and the form instance the user wants to modify
- RepeatQuestionInstance, which creates the duplicated question instance in the database
- RepeatSectionInstance, which creates the duplicated section instance in the database
- SaveAsyncTask, which saves the answers when the user changes question or moves away from the edit form activity
- CheckTaskCompletion, which looks at the form instances inside a task and checks if the forms are completed
- FormDownload, which connects to the server and downloads the form models
- SendTask, which connects to the server and uploads the selected task
- TaskDownload, which downloads the information related to a task assigned to the user
- NetworkAsyncTask, which performs the other network activities, like login, download of positions and areas and so on
- UploadPosition, which uploads to the server each time period the updated position of the user
- SearchAsyncTask, which searches through the database for the form model elements

4 Conclusions

This technical report provides an overview of the Poli-RISPOSTA mobile app developed to support the data gathering step of the Poli-RISPOSTA project. The app is available as open source at `https://brseeker@bitbucket.org/brseeker/polirispostamobile.git` under MIT license[2].

[2] `https://opensource.org/licenses/MIT`