Mapping of requirements to classes

Following are the tables of requirement to class mappings. Note that in some cases there are multiple candidates as classes for mapping, but I’ve chosen to showcase only a few of them in these listings for brevity’s sake. Ideally the classes listed in the tables are the good starting points to investigate, but they might not contain the whole flow for the requirement (especially when implementation reaches from client side to database through server and back).

|  |  |  |
| --- | --- | --- |
| **Basic Project Requirements** | | |
| Requirement No. | Classes on Symprap (Mobile) | Classes on Symprap (Server) |
| 1. | * LoginActivity * SymprapConnector * SymprapProxy * RegisterActivity * UserCreate * CurrentUser | * OAuth2SecurityConfiguration * User * SymprapUserDetails * UserController * UserService * UserRepository |
| 2. | * MainActivity * AnswerService * QuestionNotificationReceiver | - |
| 3. | * SymprapProxy * SymprapConnector * PublicSymprapProxy | * SymprapApplication * QuestionController |
| 4. | * LoginActivity * MainActivity * QuestionsActivity | - |
| 5. | * MainActivity * AboutActivity | - |
| 6. | * QuestionsTask | - |

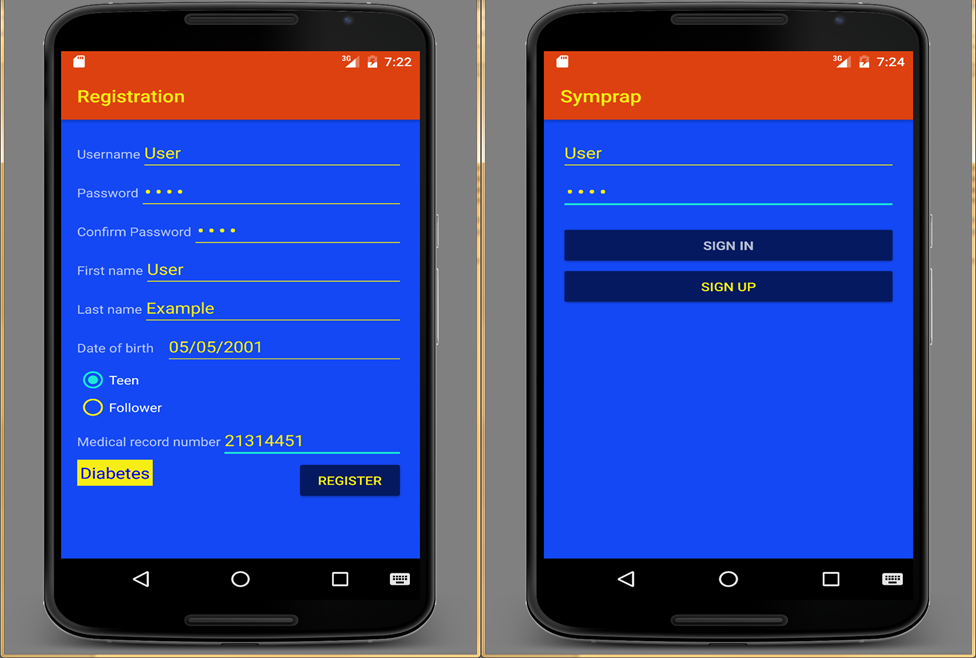
|  |  |  |
| --- | --- | --- |
| **Basic Functional Description and App Requirements for Got It** | | |
| Requirement No. | Classes on Symprap (Mobile) | Classes on Symprap (Server) |
| 1. | * User * UserRole | * User * UserRole |
| 2. | * ScheduleActivity * CustomTimeBasedAlarmManager * QuestionsNotificationReceiver | - |
| 3. | * QuestionsActivity * Question * Answer * AnswerService | * QuestionController * Question * AnswerController * Answer |
| 4. | * ReportsMainActivity * AnswerGet * ReportsUserSelectionActivity * TextReportActivity * BooleanReportActivity * BooleanPieChart * SingleLineChart | * AnswerController * AnswerGet |
| 5. | * User * UserRole * FollowerActivity | * User * UserRole * UserController |
| 6. | * QuestionsActivity * Answer | * Answer * AnswerController |
| 7. | * SymprapConnector * SymprapProxy * AnswerGet | * AnswerController * AuthenticationVerifier * OAuth2SecurityConfiguration * UserService * SymprapUserDetails |

Basic Project Requirements

Any potential Capstone project must support multiple users and should leverage services running remotely in the cloud. Each project's specification clearly outlines the app's intended high-level behavior, yet leaves substantial room for individual creativity. Students will therefore need to flesh out many important design and implementation details. Basic requirements for all Capstone MOOC project specifications include:

## Apps must support multiple users via individual user accounts. At least one user facing operation must be available only to authenticated users.

There is no limit to the users in the application. You can create them on the fly using the client (or admin tools). You can immediately login afterwards via ”Sign in” button on the client.



1 Registration and login activities

Most of the REST-operations to the server require that the user is authenticated. Only fetching list of diseases and registrations are available without authentication. Many of the operations are available only for the certain roles, for example, you need to have a user role TEEN in order to submit answers and add or remove followers. There are also many REST-methods that require role ADMIN, like creating or modifying diseases and questions.

## App implementations must comprise at least one instance of at least two of the following four fundamental Android components: Activity, BroadcastReceiver, Service, and ContentProvider.

There are over ten activities (RegisterActivity, LoginActivity, MainActivity, FollowerActivity, QuestionsActivity, ScheduleActivity, AboutActivity and various report activities), two services (AnswerService and QuestionService) and one receiver (QuestionsNotificationReceiver).

## Apps must interact with at least one remotely-hosted Java Spring-based service over the network via HTTP.

App interacts with a remotely-hosted Spring Boot service that uses Java 8. All communication with the server uses HTTPS.

## At runtime apps must allow users to navigate between at least three different user interface screens. For example, a hypothetical email reader app might have multiple screens, such as (1) a ListView showing all emails, (2) a detail View showing a single email, (3) a compose view for creating new emails, and (4) a Settings view for providing information about the user's email account.

There are a total of 12 different user interface screens as of writing this documentation.

## Apps must use at least one advanced capability or API from the following list covered in the MoCCA Specialization: multimedia capture, multimedia playback, touch gestures, sensors, or animation. Experienced students are welcome to use other advanced capabilities not covered in the specialization, such as BlueTooth or Wifi-Direct networking, push notifications, or search. Moreover, projects that are specified by commercial organizations may require the use of additional organization-specific APIs or features not covered in the MoCCA Specialization. In these cases, relevant instructional material will be provided by the specifying organization.

App uses animation capabilities of Android in two different activities (MainActivity and AboutActivity).

## Apps must support at least one operation that is performed off the UI Thread in one or more background Threads or a Thread pool.  Communication between background Threads and the UI Thread should be handled by one of Android concurrency frameworks, such as the HaMeR or AsyncTask framework.

App uses AsyncTask framework extensively during communications with server utilizing RetroFit. For example, there is an AsyncTask named QuestionsTask that fetches questions for diseases from the server and then starts a QuestionActivity.

Basic Functional Description and App Requirements for Got It

(Note that application in question is called Symprap instead of Got It)

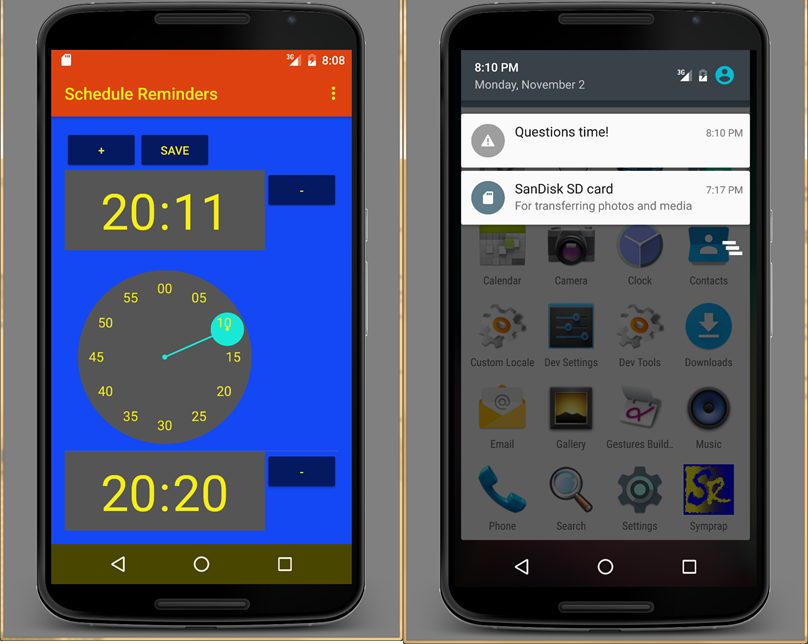
## The Teen is the primary user of the mobile app. A Teen is represented in the app by a unit of data containing the core set of identifying information about a diabetic adolescent, including (but not necessarily limited to) a first name, a last name, a date of birth, and a medical record number.

See picture 1. There is username and password information associated with the user as well.

## The Teen will receive a Reminder in the form of alarms or notifications at patient-adjustable times at least three times per day.

The Teen will receive a reminder in a form of notification. The Teen can schedule reminders without hard limits of any kind in Schedule Reminders user interface screen (see picture 2). The Teen can also answer questions whenever he or she so desires from the main screen of the application.

Notifications are implements utilizing SharedPreferences (where notification times are stored for editing purposes), BroadCastReceiver (that creates a notification that launches QuestionsActivity when accepted by user) and AlarmManager that a sends request to Receiver once per day per user created schedule.

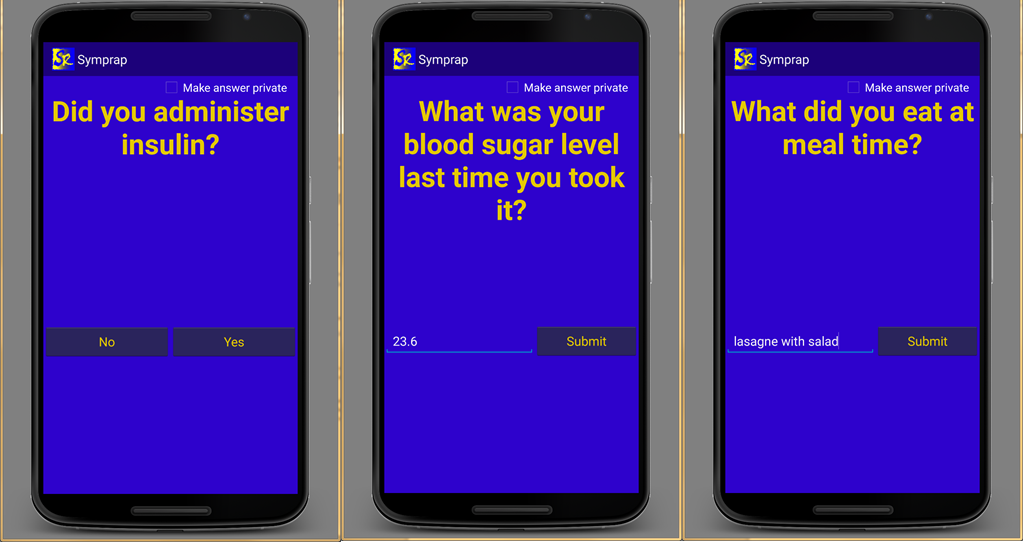


2 Schedule reminders activity and a notification that launches QuestionsActivity

## Once the Teen acknowledges a Reminder, the app will open for a Check-In. A Check-In is a unit of data associated with that Teen, a date, a time, and the user's responses to a set of Questions at that date and time (see the Implementation Considerations for some suggested questions).

The Teen will receive a list of questions (customizable by admin using admin tools in a browser) from the server. After answering each of those questions, application will post those answers to the server along with userid. Server sets the date and time to the answers when they are persisted into database.

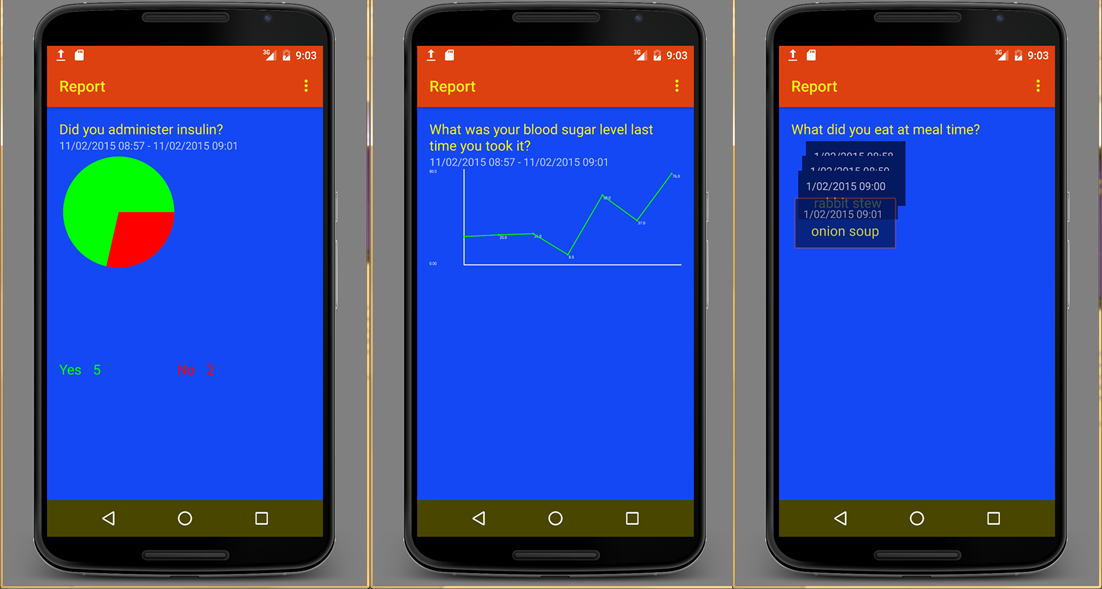
Note that there is no direct model for Check-In in this application. Instead a list of answers is submitted to server. On server side each Answer entity includes the following fields: id (long), Question (entity), answer (String), created (date), User (entity) and answerIsPrivate (boolean, defaults to false).



3 QuestionsActivity shown for different types of questions

## Feedback is the mechanism by which Check-In data is summarized and provided to the user in a meaningful way. A Teen is able to monitor their Feedback data that is updated at some appropriate interval (e.g., when a Check-In is completed, daily, weekly, or when requested by Followers). The Feedback data can be viewed graphically on the mobile device.

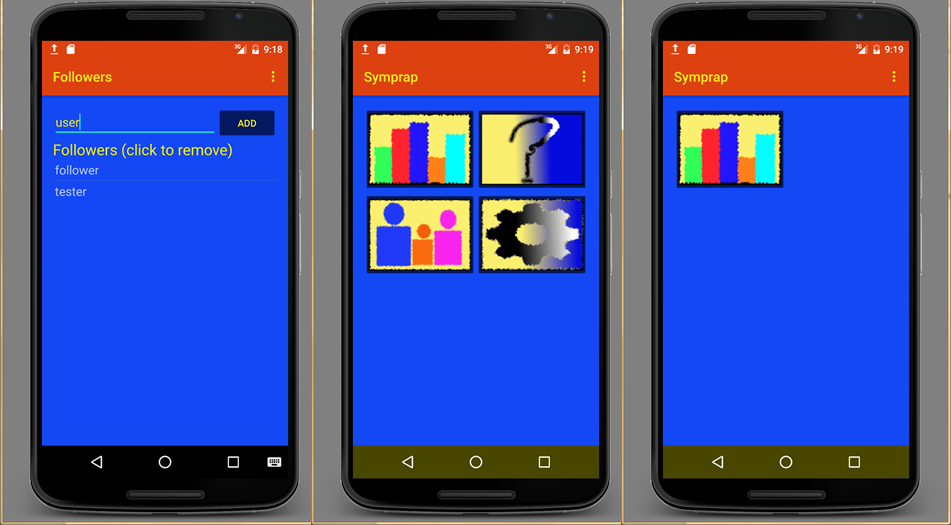
The Teen can view reports of his or her answers anytime he or she wishes. They are fetched from the server on-demand. There are three different types of answers: Boolean for simple yes/no questions, Double for numeric answers and text for strings. A pie chart is provided for Boolean, single line chart for numeric and stackview for text answers (see picture 4).



4 From left to right: piechart (for yes/no answers), single line chart (for numeric answers) and stackview (for answers with text)

## A Follower is a different type of user (e.g., a parent, clinician, friend, etc.) who does not the ability to perform Check-Ins, but who can receive Check-In data shared from one or more Teens. A Teen can be a Follower for other Teens.

A Follower/Teen can view reports of the Teen users they follow. A Teen user can add and remove Followers if they know Follower’s username. Note that majority of the UI is hidden from the Follower. They only have access to the reports functionality after they log in (see picture 5).



5 From left to right: removing and adding followers, main screen of the application and the same screen for the Follower

## The Teen can choose what part(s) of their data to share with one or more Followers.

The Teen can select upon answering which answers should be visible only for the Teen himself/herself (see picture 3).

## Teen data should only be disseminated to authorized/authenticated Followers and accessed over HTTPS to enhance privacy and security of the data.

As of writing this, all communication with the server is over HTTPS utilizing unsecure certification, OAuth2 on majority of the requests and user passwords are stored hashed and salted with bcrypt on the database. Only Teen and his or her followers can access the answer data from the server.