DTutor Design Document

Travis Peters, Haider Syed, Arvind Kumaran

May 17, 2014

Contents

1 Overview

1.1 Application Description

DTutor is a specialty smartphone app designed **by** students **for** students. Our primary goal in creating this application is to help students of all disciplines get the help they need, set/monitor goals, connect with peers, and explore the vast educational resources that Dartmouth College has to offer.

1.2 Project Scope

Due to the limited time left in the course, we intend to specfically address the following application components/features:

- DTutor profile
  - Fill out Dartmouth's Tutor/Tutee application.
  - Define contact info for meeting with tutors/tutees or other students for a study group.
- Connect with students to form study groups
- Tutor browser
  - Request to be tutored by a tutor you know
  - Get matched with a tutor by us
- Reserve a space for your study group
- Set and monitor goals
  - Additionally, share goals with family, friends, coaches, professors, advisors, etc.
- Learn about Dartmouth educational resources
  - Connect with Tutor Clearninghouse staff
  - Schedule academic advising
  - …
- …

2 Packages & Classes Overview

2.1 Main Package

2.2 Data Package

Member

The member class is a data carrier object that holds personal information about the user of the DTutor *app.*

Application

Super class that provides generic/universal components for Dartmouth Tutor Clearinghouse applications.

TutorApplication extends Application

Subclass of Application that provides specific components for Dartmouth Tutor Clearinghouse Tutor application.

TuteeApplication extends Application

Subclass of Application that provides specific components for Dartmouth Tutor Clearinghouse Tutee application.

StudyGroupLeaderApplication extends Application

Subclass of Application that provides specific components for Dartmouth Tutor Clearinghouse Study Group Leader application.

DatabaseHelper

…

2.3 Utils Package

3 User Interface

3.1 Main Activity

3.1.1 Home Fragment

3.1.2 Connect Fragment

3.1.3 Profile Fragment

4 Database Implementation

4.1 Data Structures

A primary component of our application is the data that we are handling. In order to build an app that would be consistent with the infrastructure that the Tutor Clearinghouse already has in place to manage their Tutoring system, we need various data carrying classes to collect/store data that they require. The following are our primary data carrier classes:

```
public class Member {

    // Required fields
    private long mId;
    private String mStudentId;
    private String mFirstName;
    private String mLastName;
    private List<String> mRoles;
    private int mClassYear;
    private String mEmail;

    // Optional fields
```

```
    private List<Course> mCourses;
    private String mAddress;
    private String mPhoneNumber;
    private String mGender;
    private String mEthnicity;
    private Comments mComments;

    private LatLng mLocation;
}

public class Application {

    private long mId;
    private String mAppType;
    private List<Course> mCourses;
    private int mMaxStudents;
    private String mAttrTutorType;
    private String mAttrAthlete;
    private String mGreekAffiliation;
    private String mVisaStatus;
    private Calendar mVisaStartDate;
    private Calendar mVisaEndDate;
    private Boolean mEmploymentFormsSubmitted;
    private String mHeardAboutUs;
    private String mSpecialComments;

}
```

Note: *The majority of the class implementation has been excluded for the sake of brevity, but we have already defined getters/setters (where necessary), various helper methods for working with the objects, and a series of validation methods to ensure data is accurate/complete*

4.2 Database Table Schema

Arguably the most important data that is stored in this app pertains to the offical applications that are filled out by any student that wishes to be tutored, tutor, lead a study group, etc. Below is the database schema for recoding any/all tutoring applications that members submit. Local to the device, a user may have one or even a few applications, however, the power of our system will come into storing all of these records remotely, at which point we will be able to automate the process of tutor/tutee pairing.

```
private static final String CREATE_TABLE_MEMBERS =
                    "CREATE TABLE IF NOT EXISTS " + TABLE_MEMBERS
                        + " ("
                        + COLUMN_ID
                        + " INTEGER PRIMARY KEY AUTOINCREMENT, "
                        + COLUMN_STUDENT_ID
```

```
                        + " TEXT NOT NULL, "
                        + COLUMN_FIRSTNAME
                        + " TEXT NOT NULL, "
                        + COLUMN_LASTNAME
                        + " TEXT NOT NULL, "
                        + COLUMN_ROLES
                        + " TEXT NOT NULL, "
                        + COLUMN_CLASS_YEAR
                        + " INTEGER NOT NULL, "
                        + COLUMN_EMAIL
                        + " TEXT NOT NULL, "
                        + COLUMN_MEMBER_COURSES
                        + " TEXT NOT NULL, "
                        + COLUMN_PHONE
                        + " TEXT NOT NULL, "
                        + COLUMN_ADDRESS
                        + " TEXT NOT NULL, "
                        + COLUMN_GENDER
                        + " TEXT NOT NULL, "
                        + COLUMN_ETHNICITY
                        + " TEXT NOT NULL "
                        + COLUMN_COMMENTS
                        + " TEXT NOT NULL "
                        + ");";

private static final String CREATE_TABLE_APPLICATIONS =
                    "CREATE TABLE IF NOT EXISTS " + TABLE_APPLICATIONS
                        + " ("
                        + COLUMN_ID
                        + " INTEGER PRIMARY KEY AUTOINCREMENT, "
                        + COLUMN_APP_TYPE
                        + " INTEGER NOT NULL, "
                        + COLUMN_APP_COURSES
                        + " TEXT NOT NULL, "
                        + COLUMN_MAX_STUDENTS
                        + " INTEGER NOT NULL, "
                        + COLUMN_TUTOR_TYPE
                        + " TEXT NOT NULL, "
                        + COLUMN_ATHLETE
                        + " TEXT NOT NULL, "
                        + COLUMN_GREEK
                        + " TEXT NOT NULL, "
                        + COLUMN_VISA_STATUS
                        + " TEXT NOT NULL, "
                        + COLUMN_VISA_START
                        + " DATETIME NOT NULL, "
                        + COLUMN_VISA_END
                        + " DATETIME NOT NULL, "
```

```
                        + COLUMN_EMPLOYMENT_FORM_SUBMITTED
                        + " INTEGER NOT NULL, "
                        + COLUMN_HEARD_ABOUT_US
                        + " TEXT, "
                        + COLUMN_COMMENTS
                        + " TEXT "
                        + COLUMN_LOCATION
                        + " BLOB "
                        + ");";

private static final String CREATE_TABLE_MEMBERS_APPLICATIONS =
                    "CREATE TABLE IF NOT EXISTS " + TABLE_MEMBERS_APPLICATIONS
                        + " ("
                        + COLUMN_ID
                        + " INTEGER PRIMARY KEY AUTOINCREMENT, "
                        + COLUMN_MEMBER_ID
                        + " INTEGER NOT NULL, "
                        + COLUMN_APPLICATION_ID
                        + " INTEGER NOT NULL "
                        + ");";
```

The autoincremented *_id* is used as the primary key, though the *student_id* is also a unique identifier and may be used if something warrants it (such as remote storage in something like a Google App Engine Datastore).

4.3 Implement Database Operations

```
// Database Methods: Setup/Configuring & Handling

public synchronized static DatabaseHelper getInstance(Context context) {
    if(singleton == null) {
        singleton = new DatabaseHelper(context.getApplicationContext());
    }
    return(singleton);
}

private DatabaseHelper(Context context) {
    super(context, DATABASE_NAME, null, DATABASE_VERSION);
}

@Override
public void onCreate(SQLiteDatabase database) {
    database.execSQL(CREATE_TABLE_MEMBERS);
    database.execSQL(CREATE_TABLE_APPLICATIONS);
    database.execSQL(CREATE_TABLE_MEMBERS_APPLICATIONS);
}

@Override
```

```
public void onUpgrade(SQLiteDatabase database, int oldVersion, int newVersion) {}

// Database Methods: Members

public long insertMember(Member member) {}

public void updateMember(Member member) {}

public void deleteMember(long rowId) {}

public Member fetchMember(long rowId) {}

public ArrayList<Member> fetchAllMembers() {}

public ArrayList<Application> fetchMemberApplications(long rowId) {}

public void deleteMemberApplication(Member member, long rowId) {}

private static Member cursorToMember(Cursor cursor) {}

// Database Methods: Applications

public long insertApplication(Application app) {}

public long updateApplication(Application app) {}

public void deleteApplication(long rowId) {}

public Application fetchApplication(long rowId) {}

public ArrayList<Application> fetchAllApplications() {}

private static Application cursorToApplication(Cursor cursor) {}

// Database Methods: Members-Applications

private long createMemberApplicationBind(long memberId, long appId) {}

private long deleteMemberApplicationBind(long memberId, long appId) {}
```

5 Service Implementation

5.1 Service 1

5.2 Servce 2

6 [Other] Noteable Technical Details/Algorithms

- One of the features we hope to provide in this app is the ability to automate the process of matching tutors/tutees and study group members with study group leaders. While a lot of the procedure can be scripted, there are often "personal touches" that go into finalizing any matches. For this reason we propose to implement a sort of approval system or recommendation system by which authorized staff can review the system's suggested matchings and have the final say in whether or not a particular matching will be finalized.