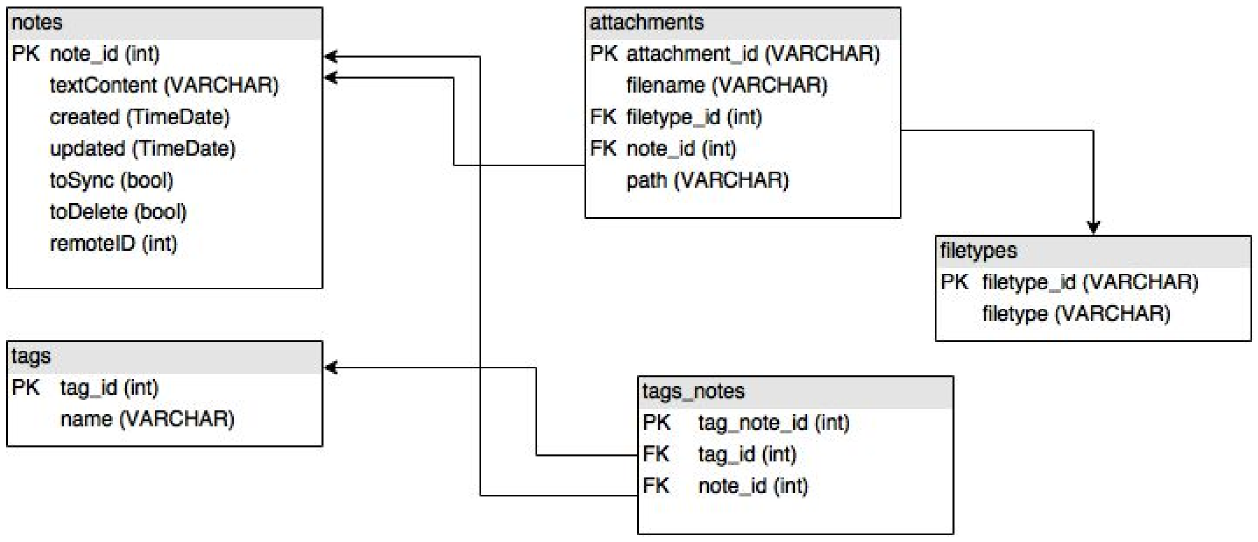
Internal Database



The internal database is an SQLite database consisting of five tables. The primary table is the notes table. This table holds the primary information about a note. This kind of information includes the content of the note, when it was created, and optionally when it was updated.

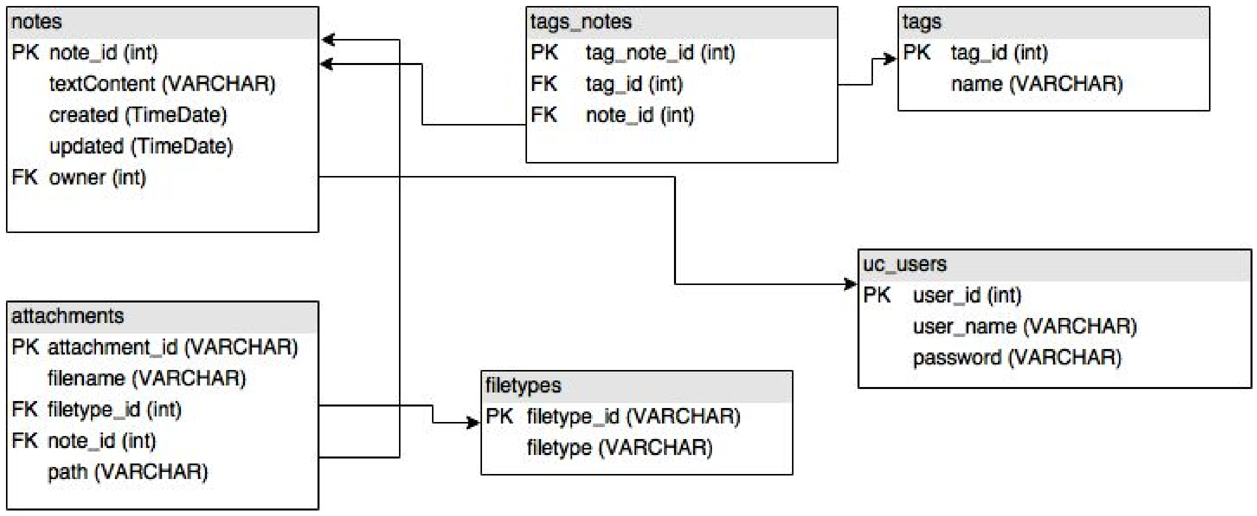
In addition to primary information about the note, the notes table also contains information pertaining to its sync status. There are three fields to handle this information. The toSync field is set to true when there have been local changes to the note that need to be synced to the server. The toDelete field signifies whether or not that sync up needs to delete a note remotely. The remoteID field is used to specify which note on the server needs to be updated or deleted, as the local and remote IDs may not be the same. If there is no remoteID set, then the note is new and only stored locally, meaning the note needs to be created on the server.

The other tables contain additional information about the notes. The tags table contains a list of text tags (or hashtags) that have been applied to notes. Since multiple tags can be associated with a single note and a single note can be associated with a single tag, there is a tags_notes table to manage the relations between the two.

With this design, a note may have zero or more attachments of multiple types. The attachments table holds information about these attachments like the filename, a foreign key to the note, the file path, and a foreign key to a filetypes table. The attachment data is not stored in the database, however. The attachments are stored in the file system, and a path to the file is stored in the database tables. This assures a faster load time.

The filetypes table serves two purposes. First, the filetypes table prevents the need to store filetype multiple times in the attachments table, but also serves as a quick source to find all supported filetypes. For example, if a user wanted to upload a file, the filetypes table could be quickly queried to see if the filetype exists in the table, signifying that the app could accept and store the filetype. This is a much simpler method than hard-coding file type checks into both the server and app-side code.

External Database



The external database is built on MySQL. The structure of the database is very similar to that of the internal database. In addition to the notes table found on the internal database, the notes table has a foreign key field, “owner” that points to the table of users.

The uc_users table is a table containing information about users, such as their username and hashed passwords. This table is only used for authentication purposes, ensuring users can only access notes that belong to them.

Technical Features

The app accesses many technical capabilities of Android devices, such as the following:

- **Android Camera**: For capturing photos and videos to attach to notes. The camera will open should a user decide to capture a new photo.
- **Android Device File System**: This will be used for storing all attachments to notes prior to uploading to the server.

- **Android Audio Recording**: For recording short audio notes to attach to notes.
- **PHP & MySQL**: For storing and accessing data on the server. Server usage will be for backup or sync across multiple devices.
- **UserCake**: A PHP user authentication library for validating a user's login credentials in a secure manner. UserCake uses a salted hashing algorithm to ensure the security of passwords stored in the database. Even should someone gain unauthorized access to the database, the person would only have access to the processed password, not the raw password known to the user. This protects the users' other accounts, should they used similar or identical passwords on other services.
- **Volley**: A library published by Google for simplifying HTTP requests between the app and server.

Web Services

The web services used to access ThrowNote features are based off PHP with a RESTful interface. A RESTful API is typically accessed through HTTP methods. The method of HTTP request defines what action the user is trying to make. For example, an HTTP GET request indicates that the user is trying to access, not modify data, an HTTP POST request is typically used to add or update data, and an HTTP DELETE request signifies the user wants to remove data.

The URL of the service indicates what type of data (or resources) the user is trying to access or modify. Table 1 shows the available API endpoints to the app, and what each endpoint is designed for

/api/v1/notes	Accesses the collection of all notes - HTTP POST: add or update a note to the list of notes
/api/v1/notes/<id>	Accesses a single note - HTTP GET: gets information about a single note with an id of <id> - HTTP POST: updates a note with an id of <id> - HTTP DELETE: deletes a note with an id of <id>
/api/v1/notes/<id>/file	Accesses the files attached to a note

	- HTTP GET: returns a note’s attachments - HTTP POST: adds or updates a note’s attachment - HTTP DELETE: removes any files associated with a note
/users	Accesses the collection of all users - HTTP POST: adds a user to the database of users
/users/<id>/notes	Accesses the notes of a single user - HTTP GET: returns a JSON array of notes
/users/<username>	Access a user’s authentication methods - HTTP POST: used to verify username and password. Returns a user’s ID if successful

Table 1: RESTful API Documentation

Like most RESTful web services, all ThrowNote web services return JSON objects for processing within the app. For more information on how RESTful APIs are designed and used, visit https://en.wikipedia.org/wiki/Representational_state_transfer.