**Part A: Application Description - Web Front-End (Option 3a)**

Many people have packed schedules, and therefore limited time and energy to learn new things. But what if you could learn something new while waiting at the subway station or lunch line? That is where our app comes in: we aim to provide users with short, educational videos on a topic of their choice.

We would like to populate our database using the YouTube API. Specifically, the user will **query** our app with a search string related to their topic of interest (ex. “cats”). Our app will then query YouTube and return the 10-15 most highly rated short videos (duration < 4 min) matching the user’s query.

Besides querying for videos, the user can also **rate** (give 1-5 star rating) and **review** (give a short, written description of how helpful they found the video) each of the videos returned by their query.

While the results of each query will not be saved in our database (and is not counted in the Videos entity), the user has the option of creating **lists** of videos that they like and adding video bookmarks to these lists. Any video that has been listed/bookmarked, rated, and/or reviewed will be stored in our database, and any user can view all of the ratings and reviews left for a video.

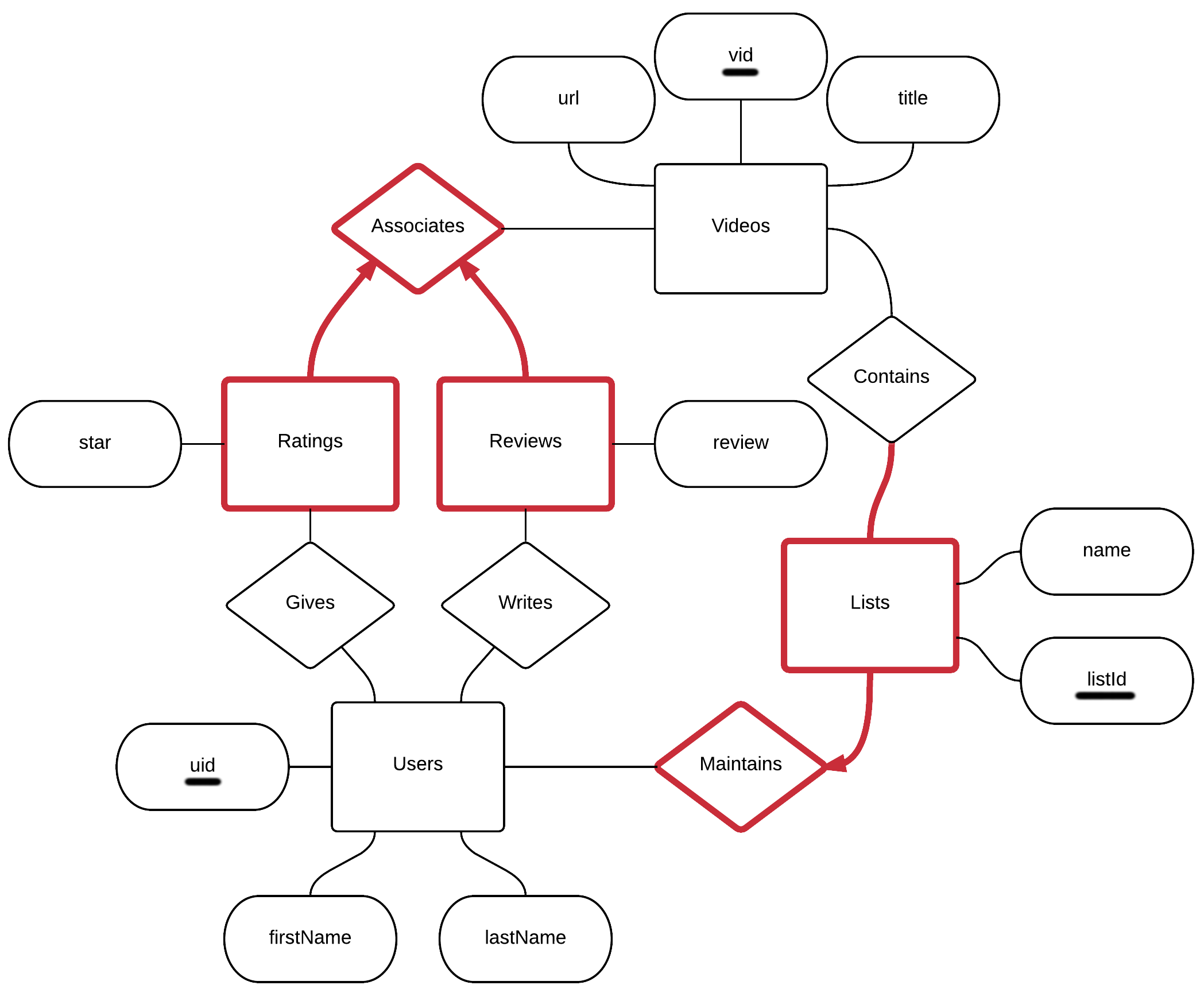
One of our contingency plans covers the case where the YouTube API proves difficult to use (See Part D), and the other covers the case where one of us drops the class.

**Our entities:** Users (people who use our app), Lists containing bookmarked videos, Ratings (a 1-5 star rating that Users can optionally give to a video), Reviews (short written feedback that Users can optionally give to a video), and Videos (either Rated, Reviewed, or Listed by at least one User)

**Our relationships:** Gives (Users Give Ratings), Writes (Users Write Reviews), Associates (Ratings and Reviews Associate with Videos), Maintains (Subscribers Maintain Lists), Contains (Lists Contain Videos)

(any 3 actions – put it in the DB)

**Part B: E/R Diagram**

****

**Part C: SQL Schema**

CREATE TABLE Users (

uid int NOT NULL, /\* list primary keys first \*/

firstName char(20),

lastName char(20),

PRIMARY KEY (uid)

);

CREATE TABLE Videos (

vid int NOT NULL,

url text NOT NULL,

title text NOT NULL,

PRIMARY KEY (vid)

);

/\* Relationships don't necessarily need a table \*/

CREATE TABLE UserRatings (

uid int NOT NULL,

vid int NOT NULL,

star int,

CHECK (

star = 1 or star = 2 or star = 3 or start = 4 or star = 5

)

PRIMARY KEY (uid, vid),

FOREIGN KEY (uid) REFERENCES Users(uid),

FOREIGN KEY (vid) REFERENCES Videos(vid)

ON DELETE CASCADE

);

CREATE TABLE UserReviews (

uid int NOT NULL,

vid int NOT NULL,

review char(140),

PRIMARY KEY (uid, vid),

FOREIGN KEY (uid) REFERENCES Users(uid),

FOREIGN KEY (vid) REFERENCES Videos(vid)

ON DELETE CASCADE

);

CREATE TABLE Lists (

listId int NOT NULL, /\* weak entities can have primary keys \*/

uid int NOT NULL,

name char(20) NOT NULL,

PRIMARY KEY (listId),

FOREIGN KEY (uid) REFERENCES Users(uid)

ON DELETE CASCADE

);

CREATE TABLE ListVideos ( /\* this is one record in the list \*/

listId int NOT NULL,

vid int NOT NULL,

PRIMARY KEY (listId, vid),

FOREIGN KEY (listId) REFERENCES Lists(listId)

ON DELETE CASCADE,

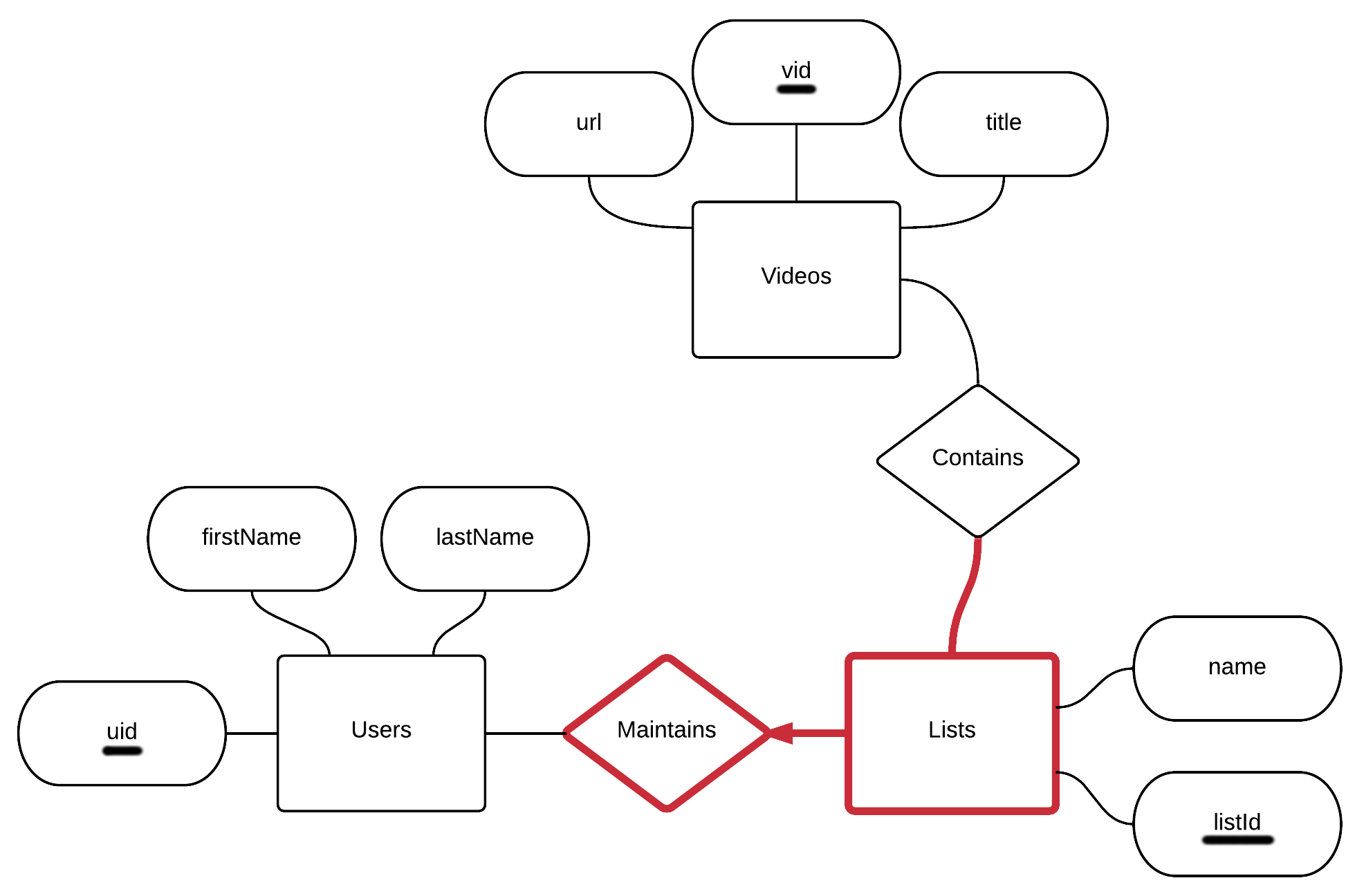
FOREIGN KEY (vid) REFERENCES Videos(vid)

ON DELETE CASCADE

);

**Part D: Contingency Plans**

Backup Plan 1: If one or us is eaten by a bear, the survivor will build the app using the simplified ER diagram below. The app will be the same as previously described, without the user having the ability to rate and review videos. Below is the contingency ER:



Below is the contingency SQL:

CREATE TABLE Users (

uid int NOT NULL, /\* list primary keys first \*/

firstName char(20),

lastName char(20),

PRIMARY KEY (uid)

);

CREATE TABLE Videos (

vid int NOT NULL,

url text NOT NULL,

title text NOT NULL,

PRIMARY KEY (vid)

);

/\* Relationships don't necesarily need a table \*/

CREATE TABLE Lists (

listId int NOT NULL, /\* weak entities can have primary keys \*/

uid int NOT NULL,

name char(20) NOT NULL,

PRIMARY KEY (listId),

FOREIGN KEY (uid) REFERENCES Users(uid)

ON DELETE CASCADE

);

CREATE TABLE Lists\_Videos ( /\* this is one record in the list \*/

listId int NOT NULL,

vid int NOT NULL,

PRIMARY KEY (listId, vid),

FOREIGN KEY (listId) REFERENCES Lists(listId)

ON DELETE CASCADE,

FOREIGN KEY (vid) REFERENCES Videos(vid)

ON DELETE CASCADE

);

Backup Plan 2: If the YouTube API doesn’t want to work with us, then we will allow users to populate the database by entering the url, title, and tags of the videos they wish to add to their list. For queries, instead of returning videos from YouTube, we will send the user’s queries to a small database of videos that we created ourselves (maybe with 500 videos to begin with). They will be able to add videos to this smaller database (extra functionality), in addition to maintaining lists.