ybar

Raising the bar on data collection

Architecture for mobile application for crowd sourced data collection over geographically distributed locations.

General Notes About Architecture and Development

- **Sever**:
  - REST API with oauth2
  - mySQL (structured data on workers, jobs) and Mongo (storing media)
  - Python/Django

- **Client**:
  - Web App or Native

- **General Dev.**:
  - **Documentation**
  - **Testing**
    - Create fake data in all the DB tables for testing
    - Unit tests
    - Continuous CI + Docker

Sample Workflow

Say an organization wants to measure noise pollution in Delhi. An admin (also a user) creates a job (see **DB Table jobs**). The admin decides that we need to collect data at 1,000 locations and that each worker should be allowed to collect data at as many as **10** locations, but no more. As part of `add_job`, using geo_sampling, the admin also uploads a spreadsheet with 1,000 rows with information about location and time. This data is then used to create 1,000 new tasks in **DB Table tasks**. (**Technically we will have to create more tasks than the data we need as some tasks will not get done.**) Workers only see jobs for which they are eligible (supports basic functionality for matching on location---city, country---and other features of workers, including eventually their reliability score.) Worker agrees to do a task (can do as many tasks as allowed --- here **10**). Worker reaches say location X at time Y as specified by task ID --- worker then clicks on the task ID and collect data and then clicks on submit data.) The data is uploaded when worker has wi-fi connectivity (this point is explained before). The administrator on clicking a `job_id` on list jobs page can look at status of all the tasks. Administrator on clicking `task_id` is able to see the uploaded data. And can hit approve or reject. Once

approved, points are added to the worker account. The worker can reimburse these points for money.

Signing up Workers

- **Sign up Page:**
    - **Get users to sign up right on the home page.**
    - Connect via oauth to **DB Table user**
    - **Supported API calls**
        - add_user

Workers

- **Profile Page**
    - Worker enters some basic data (see **DB Table user** for details)
    - Connects to **DB Table user** which carries worker profile plus username/password
        - Only user or db admin can access this
        - Connect via oauth to **DB Table user**
        - **Supported API calls** (each call sends **user_id**)
            - get_profile:
                - GET request. Returns selected fields from **DB Table user**
                - We can also show them average_rating for worker_id==worker_id from the **DB Table tasks**
            - Edit_profile
                - PUT request.
            - delete_profile

- **Username/Password Mgmt**
    - Username/password + password/login recovery page **and** application (email)

- **Register/Deregister Device**
    - people should be allowed to register multiple devices
    - register/deregister devices
        - **Supported API calls** (each call sends **user_id**)
            - register_device
            - deregister_device

- **List/Find Jobs**
    - List/find available (**eligible**) jobs and read some details on the jobs including time, description, pay, title. (Have a way for scrolling, only showing top 10 etc.)
        - **Eligibility** is based on worker characteristics stored in **DB Table user +** data entered by admin in **DB Table jobs**
    - Ideally a push notification whenever a new job is added to the db

    - Connect via oauth to **DB Table jobs**
    - **Supported API calls:**
        - list_jobs (send **user_id** which is used to determine eligibility, return selected columns from **DB Table jobs.** Only provide access to a particular view.)

- **Agree to do a task**
    - Button on the list_jobs page next to each job
    - If people agree and don't do the job, no penalty for now
    - People can work on multiple tasks at the same time
    - The agreement auto-expires after a set amount of time of not getting the data:
        - Defaults to the day the job end_date/end_time
        - If task has the field date and end time, the task auto-expires after that time passes
    - Connect via oauth to **DB Table tasks**
    - **Supported API calls**: (sends worker_id and agree/disagree)
        - accept_task (updates task_status to accept, inserts worker_id into worker_id field)

- **Accepted Task Status**
    - Lists all the jobs a worker has accepted
    - Status of the task (accepted, waiting for approval, approved, rejected --- see details of **DB Table tasks**)
    - Connect via oauth to **DB Table tasks**
    - **Supported API calls**: (sends worker_id)
        - list_accepted_tasks

- **Interact with task allocator if there is an issue**
    - Email/web form

- **Enable work**
    - jobs can include: merely providing access to sensor(s) for a fixed time period, going somewhere and doing something using one or more of the sensors, or filling data in a web form after interviewing people (or self), providing access to other data on the device for a fixed time period, for e.g., browser data or photos on the phone.
    - **Enable/disable access to various sensors**
        - Close to what you see when you open Skype or whatever --- allow for storing data under our application
    - Basic functionality to select files (images/sounds) whatever --- (only for active data collection jobs; for passive data collection, everything is uploaded) --- to upload.

- **Storing Application Data on User Machine**

    - Worker can work on multiple jobs at the same time. Store data for each task under job_id/task_id/
        - Partition and securely store the data on the worker machine.
    - Delete data on the machine as soon as it is successfully uploaded
    - If it is browsing data that say a "worker" is sharing, we copy over the history and then destroy the copy of the history

- **Upload data to server**
    - Clicks upload next to task_id on list accepted tasks page
        - Check if there is a wi-fi connection or ask user whether ok with doing it now or when wi-fi is there.
        - Connect via oauth to **DB Table tasks**
        - **Supported API calls**:
            - upload_data
                - uploads all the task data in JSON format
                    - All uploads have worker_id, device_id, job_id, task_id as fields
                    - Secure transfer
                    - POST/PUT requests. sends a job_id and task_id. server puts in the job_id/task_id folder.
                - on success, updates task status in **DB Table tasks** to waiting for approval
- **Payment**
    - When admin approves the job:
        - Field status on **DB Table jobs changes to done**
        - Insert points (using job_id from **DB Table job**) in **DB Table pay**
            - Triggers PUSH notification
            - Notification on app. Shows worker has points
        - Reimburse points:
            - Worker clicks on reimburse button and that triggers an email with appropriate paytm link.
            - Also executes a query on **DB Table pay** that deducts points from the table.
            - Currently only allows a worker to reimburse all the points they have accumulated. We can also put in a minimum limit before reimbursement can be triggered. So an additional query that checks whether points > min.
                - **Supported API calls**:
                    - reimburse (sends **user_id**)

Admin

- **List, find, add, edit, delete jobs**

- **List all jobs**
  - Find (by keyword, date), scroll, functionality
  - Connect via oauth to **DB Table jobs**
  - **Supported API calls:**
    - SAME AS FIND/LIST JOBS FOR WORKERS BUT WITHOUT ELIGIBILITY REQUIREMENTS
    - list_jobs (sends **user_id** which is used to determine eligibility, return selected columns from **DB Table jobs.** Only provide access to a particular view.)
- **Edit/delete jobs**
  - On the list_task page, buttons for edit/delete
  - Connect via oauth to **DB Table jobs**
  - **Supported API calls:**
    - edit_job
    - delete_job
- **Add jobs**
  - Admin interface has an add task button
  - Each job will have one/more tasks (**task = unit of work**) associated with it
    - If location/times need to be specified, a csv with following fields: lat, long, time (date, start_time, end_time) uploaded by admin. csv should have as many rows as n_rows
  - Connect via oauth to **DB Table jobs and tasks**
  - **Supported API calls:**
    - add_job
      - triggers insert to table jobs
      - triggers insert to table tasks

- **Task monitoring**
  - From List Jobs page, clicking on a job takes the admin to page listing tasks associated with that job. On clicking on a task, the admin is able to see the data uploaded.
  - Connect via oauth to **DB Table jobs and tasks**
    - **Supported API calls:**
      - list_tasks

- **Approve/reject jobs**
  - On list_tasks page, each task also has button to approve/reject task
  - Connect via oauth to **DB Table tasks**
  - **Supported API calls:**
    - Up_or_down (insert approve/reject into task_status)

- **Rate the worker on the task**
  - On list_tasks page, each task also has button to rate worker on a 5 point scale

- Connect via oauth to **DB Table tasks**
  - **Supported API calls:**
    - rate_worker (inserts rating into worker_rating)

DB Tables

- **DB Table user**
  - user_id, name, sex, email, phone, address, age, paytm, username, passwd (encrypted), auto detection of device type + whatever information you can gather on the device, device_ids, worker_or_admin

- **DB Table pay**
  - user_id, job_id, historical_points, current_points

- **DB Table job**
  - job_id (string; key; mandate 10 chars), title (varchar; don't limit size), description (varchar; no limit size), pay_per_task, n_tasks (numeric), n_tasks_per_worker_allowed (numeric), devices_allowed (varchar; one of the following), location_restictions, start_date (date and time), end_date (date and time), what_sensors (varchar; one of the following), client_name (varchar; no limit on size), additional comments, kind_of_job (one of the following)

- **DB Table tasks**
  - Be aware that a person can do multiple tasks associated with one job (decided by attribute n_tasks_per_worker in table job)
  - Each job has multiple tasks (as many tasks as data one wants to collect)
    - task_id
    - job_id
    - user_id
    - task_status (accepted, waiting for approval, approved, rejected)
    - latitude (if no file provided with this information during job creation, defaults to DB Table job start_date and time)
    - longitude (if no file provided with this information during job creation, defaults to DB Table job start_date and time)
    - start_time (if no file provided with this information during job creation, defaults to DB Table job start_date and time)
    - end_time (if no file provided with this information during job creation, defaults to DB Table job start_date and time)
    - worker_id
    - worker_rating