CSC 444 Project
Teenage Services Market Place

The objective of this project is to design and implement a web-based application called TeenServ that offers a marketplace to hire local ***teenagers*** to provide specific services, such as babysitting, lawn-mowing, snow-shoveling, yard-cleanup, etc. Potential ***clients*** are adults who seek teenagers in their area who are willing to perform tasks for them. Teenagers (aged 13-19, inclusive) register on the site and specify the services they are willing to perform and at what price. The TeenServ application you will create should automatically connect clients and teenagers if there is a good match. A key requirement for your service is that the distance for the teenager to get from his/her house to the client be limited (say to 1km or so, perhaps depending on the age of the teenager).[1]

The TeenServ application you will create has similarities to many other offerings available today, including:

- TaskRabbit, Jiffy, Aladom, College Labor, Jobrunners, Handy, Porch, and many others --- your service should specifically target teenagers.
- Babysitter services, such as UrbanSitter, Sittercity, Sitter, CluckCluck, and Care.com, and Lawnmowing services, such as Mowdo, Eden, LawnGuru, MowSnowPros, and Yardly --- your service should be more general and support multiple types of services.
- Listing services, such as Craigslist, Close5, ClassifiedsByZIP, Flugpo, and Geebo --- your service should offer much more than just a bulletin board.

The TeenServ Website should:

- describe and market the service to the general public;
- allow teenagers to register and then create their own profile Web page offering their services and displaying their photo, endorsements, their star ratings, among other information;
- allow potential clients to register and then look for specific service offerings or request specific services;
- match clients with teenagers who can provide the service requested;
- bill and charge clients for the services performed by the teenagers, and pay teenagers for the services they performed;
- allow invited endorsers to provide endorsements;
- allow TeenServ administrators to manage the site (e.g. block certain users) and customer service representatives to help teenagers and clients that need assistance; and
- keep teenagers and customers engaged with the service.

[1] If you are overwhelmed just from reading this, perhaps you want to skim through the Recommendation section at the end of this document.

The end solution must be of commercial quality, offer a superb UX experience, and be reliable. The more vibrant and compelling the better. Some of the requirements will be provided in greater detail below. Others will be left vague, in which case you must establish requirements that you believe make the most sense. You may add other features that you believe will make your application more interesting and compelling. The server side of your solution must be implemented using Ruby on Rails. The user interface must be HTML based, but "responsive".

The solution is for a company called TeenServ (which is why we decided to call the service TeenServ).[2] TeenServ will make money as follows:

- it will charge 4% on credit card charges (while only having to pay 2%), and additionally
- charge 10% on each bill.

Users of service

The following types of users will access the TeenServ Web site.

1. **General public** – has access to the general TeenServ Web site for marketing info. The general public does **not** see any information regarding clients, teenagers, or specific job offers. However, general statistical information may be displayed (such as the number of teenagers and clients signed up in an area).

2. **Teenagers** – each sets up his/her own profile Web page (that is part of the TeenServ site) to market his/her abilities and offer his/her services along with times they are available. They apply for jobs and accept/decline job offers. They submit timesheets and manage their billing and payment processes.

3. **Clients** – review offered services and request services. They manage their billing and payment processes.

4. **TeenServ Customer Service Representatives** – have access to all client and teenager information so that they can assist them.

5. **TeenServ Admins** – have access to and can modify all client and teenager info. They can access the site as any client or teenager.

- **TeenServ Super Admin** – assigns roles and manages Admin and CSR permissions. A TeenServ Super Admin has full access to all databases via SQL interfaces.

All users, except for the general public, will need to authenticate themselves.

[2] You will have to come up with a better name!

Teen Services

The services supported by TeenServ must be suitable for execution by a teenager. Some services may be age restricted. TeenServ would be interested in supporting a subset of:

- babysitting
- yard work
  - lawn mowing
  - leaf raking
  - weeding
  - yard cleanup
- snow shoveling
- furniture moving
- simple cleaning tasks
- vacation services
  - mail pickup
  - pet feeding
  - lawn watering
- dog walking
- computer help (such as configurations, installation of programs, backups, disk cleanup, virus removal, etc.)
- tutoring
- reading (say, to and old lonely individual)

Some services might be reoccurring (e.g., leaf raking once a week). Other services might be triggered by events (e.g., snow shoveling whenever there is more than 2cm of snow on the ground).

Ratings, Reviews, and Endorsements

Whenever a teenager has completed a service, both the teenager and the client should be encouraged to rate the other using a star rating system. Ratings should be visible to each other, but only after there are a minimal number of them. (A single rating doesn't say very much.) While ratings should be visible to other users, a user should not be able to identify who gave them a particular rating. (You may also want to consider weighting more recent ratings more strongly.)

Clients should also be able to submit reviews of teenagers' services, and teenagers should be able to submit reviews of clients they have served. Reviews of teenagers' services should be accessible to clients wishing to request a service. Reviews of clients should be accessible to teenagers so they can decide whether to accept and offer or not.

A teenager just starting out will not have a star rating or any reviews. For this reason, they can ask others who know them to post endorsements that will be visible on their profile pages.

Matching Service

Teenagers offer services and clients request services. Your application will need to match services offered with services requested. How this matching occurs will be one of the more challenging aspects of your design. It can be done in several different ways. For example, a client can post a service they need with attendant details, and teenagers periodically review these posts, responding to those they wish to provide; that is, the teenagers are responsible for the matching. Alternatively, teenagers post services they offer along with the times they are available for the service, and clients select the teenager they wish to hire; that is, the clients are responsible for the matching. Yet another alternative might have the TeenServ service select an available teenager willing to do the service whenever a client posts a service they need; that is, TeenServ is responsible for the matching.

How the matching occurs is a key design element of your solution and has considerable consequences on (*i*) how teenagers and clients use the site and (*ii*) how engaging your solution is to all participants. As a result, TeenServ's business success will be greatly affected by your design choices. Some factors you may wish to take into account include:

- **speed**: quick matches provide certainty for all parties involved.
- **behavioral patterns**: teenagers may be more likely to continuously check to see if a job opening has become available rather than clients continuously checking to see if a teenager has become available.
- **preferences**: a client needing a babysitter for a Friday evening may wish to be able to prioritize babysitters that the client has hired previously and was happy with.
- **obsolete information**: a teenager may have posted they are generally available, say Fridays 5pm-11pm, which clients might wish to act on, but the teenager may have committed him/herself to something else on the particular Friday without updating their availability information.
- **attributes**: teenagers and clients get star ratings and reviews from each other; teenagers of a certain age may be more suitable for some tasks (e.g., older teenagers for furniture moving); etc.
- **stated age preferences**
- **hourly rate**
- **distance**: (perhaps based on age).

Your solution will need to deal with the possibility of race conditions (depending on your solution). For example, a teenager may apply for two jobs simultaneously so as to maximize his/her chances of getting a job, and two clients may try to offer their respective jobs to the same teenager at almost the same time, yet the teenager can accept only one. To address these sorts of race conditions, you will have to design some sort of communication protocol between the interested parties. This protocol should be as simple and efficient as possible. (You may choose to use a messaging service within your app, email, or text messaging for this type of communication.)

Timesheets, Billing, Transactions, and Payments

After a teenager has performed a service, the teenager must complete some form of a timesheet to indicated how long s/he worked and for which client. The client must then accept the timesheet. When this has occurred, then:

1. a transaction is created in a database that identifies
    - the teenager
    - the client
    - the type of work performed
    - when the work was performed
    - how many hours it involved
    - the hourly rate
    - the total amount
2. client is billed the total amount plus service fees, and their credit card is charged.
3. the teenager is credited with the total amount

Each client and each teenager should have access to the entire history of all their transactions. Credit card transactions that do not go through should be handled properly. Teenagers are not necessarily payed after each transaction; they have something like an account with TeenServ, but may request payment at any time. You will not be able to deposit money into a teenagers credit card account, but instead either must do a direct deposit into their bank account, send them an e-payment, or send them a check they can deposit somewhere.[3]

Your application should also be able to keep track of its own financials; i.e., how much money it has made each month.

One of the key design decisions you will have to make is how much teenagers are allowed to charge. There are a number of options with different tradeoffs, including:

- TeenServ sets the hourly rate for each service (perhaps depending on the age of the teenager[4]).
- Each teenager gets to set their hourly rate for each service.[5]
- Clients get to select the hourly rate they are willing to pay for a specific service they are looking for.[6]

Which option is chosen may affect the matching scheme used.

[3] You may want to consider using a payment scheme that specifically targets teenagers; e.g., KIK.

[4] For example, older teenagers are able to perform some tasks more efficiently and more reliably than younger teenagers.

[5] For example, a teenager with a high star rating may want to be able to charge more than teenagers with a lower rating.

[6] For example, a client that desperately needs a babysitter last minute may be willing to offer $30 an hour.

Web pages

Each Web page must have links to:

- an "About" page that describes the service and provides the history of TeenServ (to explain why TeenServ is so great).
- a page containing the terms of service
- a page describing the privacy policy
- a page with a site map
- a "Contact Us" page
- a page with a FAQ
- twitter
- facebook
- instagram

(the latter three for marketing purposes).

When a teenager or client is logged in, his/her name should be visible on the Webpage, and there should be a logout button. When a visitor is not logged in, there should be a "My Account Login" button/link clearly visible near the top.

All access to these Web pages should via https, and any first visit via http should be automatically redirected to https. Secure sessions must be used for those teenagers and clients after they have logged in.

Data Collection

The following is a subset of the data that needs to be collected and stored persistently and securely:

- Teenager:
    - name
    - home address
    - email address
    - home phone
    - cell phone
    - age
    - profile picture
    - bank account info for direct deposit[7]
    - services offered and, for each service, the times available
    - billing and payment history
    - rating data
    - reviews

[7] Generally, it is not a good idea to store credit card and payment information in your own database for security reasons. It is better to use the facilities of a payment processor (e.g., Stripe, Paypal, etc.) for this.

- Client
    - name
    - home address
    - email address
    - home phone
    - cell phone
    - optional profile picture
    - credit card info[2]
    - billing and payment history
    - rating data

General Software Requirements

- Ruby-on-Rails for all backend matter.
- HTML5-based responsive website interface for all front-end matter.
- High usability standards, given that many who will use the service will be computer challenged, old-fashioned, and often not very smart. Context aware help info; extensive input error checking; helpful error messages; etc. is important.
- Locale friendly and multi-lingual capable.
- Well-tested and reliable.
- Online info and payment processing must be secure.
- The Website has to run out of Heroku, a cloud service.
- GitHub must be used for managing all source code; the Master branch must always be runnable, and no other branches allowed. The (email-)id used to access GigHub must be a UofT email address.
- Each module/service must have externalizable RESTful API
- Userids must always be email addresses. Standard "forgot password" functionality required.
- All captured data must be stored in a database; all displayed info must be obtained from that database.
- Any persistent data that is modified must be logged in a change log, identifying who made what changes.

Recommendations

This document lists a lot of required functionality. Worse, many features required for a practical solution are not even listed --- you will have to figure those out on your own. Finally, putting a complete solution together requires a lot of work, and a complete solution is not fully specified here so you will have to develop one yourself.

Do not panic. Remember that this is a typical situation when creating a software solution. You may not be able to deliver all of what was requested here, yet still may be able to create a compelling and usable solution. Or you may actually be able to deliver much more than what was requested.

Here are some recommendations for being successful:

- Before starting to spend too much time with complex implementations, consider and play with multiple high-level designs.

- Start small and simple, and then iterate by adding functionality/features in very small chunks. For example, start just by creating a simple model (i.e., database) for the teenager, and provide functionality to access that data. It doesn't have to be complete, because you can always add to the database schema later on. Then add functionality to add new teenagers via the Web, storing the gathered info in the same database. Then do the same for clients. You are going to need this functionality no matter what.

  Then start thinking about how you wish to design your matching service. Implement simple versions of it (without bells and whistles initially) until you settle on the approach you will use. You will want to do this before you do too much other work, because the design option you select will affect many of the Web pages you will have to design and implement.

  Leave to later many of the complex parts of your solution, such as authentication, payments, timesheets, etc.

- After you have a skeleton solution with some basic functionality in place, order the functionality you wish to implement thereafter carefully so that the service is as usable as possible and appears complete (even if it is not). Prioritize the order of functionality and features you tackle, design and implement: first pick those features that TeenServ is asking for that add the most value and lead to what looks like a complete, usable solution.

- Different team members should be advocates for the teenagers, the clients, and the admins, respectively, when considering different design choices.

- Completing this project requires expertise in many different areas. We recommend that you split up your team so that a team member is responsible for being an in-depth

expert for an area who then can educate the other team members. Some areas of expertise are:
  - Heruko
  - GitHub
  - Database technology and Migrations
  - Security
  - Session management
  - Responsive Web designs and templates
  - HTML best practices
  - CSS and CSS best practices
  - Testing harnesses
  - Payment Processing
  - Web servers
  - Internationalization
  - etc.

- Putting everything together is a lot of work. We recommend that you partition the content and testing that must be provided and assign each part to a member of your team.

- Feel free to use 3rd party software and 3rd party services (as long as they are free!).[8]

- Remember that team members may be re-assigned to a different group at any time, or may drop out of the course, so:
  - pushing software updates to the master branch daily is critical;
  - all software written should be understood by multiple team members (regular code reviews or "pair-programming" are a good way to achieve this); and
  - there should be more than one expert for each subject for which expertise is required.
- Always keep in mind that requirements may change at any time (as in real life).

More recommendations will be forthcoming throughout the term…

[8] If you absolutely must use commercial 3rd party software/services, then you must create the software that emulates said software/services.