**The**https://lh6.googleusercontent.com/maxSQjNri81SpQvPB9MH7-3ODyM0yxHol3eFKF-zZPCEhg6uqyt7IXek0oyEeXloz-r6C_PDI2XVjSy9vK6IonvdfEP9_pMER1toVO_fIzcOcPWVNKooAKT03ZplxJMK-MX4cSVJ **App**

*“Tech stuff fixed right. Right there. Right now.”*

*“Hail a Geek as easily as hailing a ride.”*

***Description of operation***

**CALL NOTES 5-21-2019**

* IF the scope changes, you’ll provide a separate quote for changes
* Payment processors: Paypal, Stripe mostly, and Braintree also
* Backup payment processor? Yes
* Credit card vaults? Stripe and Braintree, provides a credit card vault
  + Does any vault support both?
* Stripe and Braintree also support ACH payments
* Background check: they will share background check service Juumia (sp?)
* Google maps - add credit card to charge, they will send google estimated charges
* Maintenance: team available to update the app as needed, hourly charges, $15/hr
* They can update the app in advance based on published notes from Apple and Android
* Maintenance contract available
* **Scope document** - no charge - can share within a day
* Source code? We own and receive
* Timeline to develop? (Months)? Don’t want to guess, will be in scope document

***ITEMS TO ADD:***

* *Added 4-1-19: please quote additional cost of supporting 3 payment processors (rather than just one): Paypal, Braintree, and Swipe.*
* *Added 4-2-19: 2 new module to quote (below): MYSCHED*

**In this document:**

* **Admin portal fIelds / variables are referenced in blue. Anything in blue is a variable that should be modifiable in the admin portal**
* **Interaction variables, referenced in green, are modified in the course of the customer-geek interaction**
* **Modules are referenced in purple; modules are optional sections of code / optional parts of the project which might be added later; the project is divided up into modules for budgeting purposes. The app architecture must be designed to be ready for all modules.**
* **3rd party services / API’s are shown in brown**
* *Notes in italic are included mostly for business operation plans and may not be particularly relevant to the design of the app*

***Summary - what’s the point of this app?***

* ***“HAIL”*** *(engage the services of)* ***a GEEK*** *(computer tech)* ***as easy as hailing a ride.”***
* ***“Tech stuff fixed right. Right there. Right now.”***
* ***“IT Talent on demand.”***

***Quick description of the app for customers:*** *What if you could hail an onsite computer tech as easily as hailing a ride? Now you can, using The HAILaGEEK App. Like a ride hailing service, it starts up showing you a realtime map of where all the available geeks (computer technician contractors) are in the area  -- where they actually are right now. Choose what service you need, see the ones on the map that can do that, see what their hourly rate is, their customer rating and reviews, and how soon they can be there. You pay them by credit card and you approve their estimate for how long the work will take before they begin working. And if you don't like the work they did for you, you can request an automated peer resolution of their fee where 3 other technicians with similar skills will perform a peer review their work and decide a fair compromise on the charge.*

***Feature summary:***

* **Customer app** + **Geek app** work together to show customers available geeks, help customer choose a geek with exact real-time location, ratings, matching skills, accept customer credit card for payment, auto calculate the estimate, supervise the job, manage the work approval process, manage parts purchasing, manage customer approval for adjustments to final fee, calculate final fees, manage customer work approval process, and manage a full “peer review” arbitration for any incidents of customer dissatisfaction.
* **Admin portal** allows configuration of the app, company/app name change (not the name shown in the Apple app store / Google Play, but the name shown in the app), company/app logo changes, service type (skill) category changes, skill level label changes, allows admin staff to issue full or partial refunds / credits to customer cards, allows admin staff to correct wrong info on Geek or customer records, allows staff to initiate email or text based password reset processes, produces CSV exports for paying Geeks, generating 1099 and sales tax reports (see details below).

**ESSENTIAL ITEMS TO BE QUOTED:**

*Without these items, there is no HAILaGEEK. This is the bare minimum project.*

* **The HAILaGEEK customer-side and geek-side apps.** These are the core smartphone apps used by customers and geeks to do business with each other.
* **The HAILaGEEK app server** - coordinates the customer-side and geek-side apps.
* **The nowSearch core module.** This module is essentially a location storage server; a database + API that stores and updates the current location of ID’d entities. It is a very simple and multi-purpose module, but HAILaGEEK will use it to store the location of geeks.
* **The JFM** (job finish module) provides a very simple web interface that allows the customer, if their cell phone dies, to log on **after a job has already been started (even travel)**, and finish responding to job queries, including approving parts, being notified of a break, approving the job when it’s finished, etc. The JFM does not include any map or other complex features, it is the **bare minimum** web interface to allow the job to be completed and approved if the customer’s cell phone does. Note: The **JFM** is not needed if the **CWPM** (described below) is in place.

**HIGHLY DESIRABLE ITEMS TO BE QUOTED:**

*If at all possible, HAILaGEEK* ***will*** *also include these modules. Even if these desirable items not part of the original project, the architecture of the project must be designed in such a way to be ready for these items to be added on in the very near future or even mid-project.*

* **The ARAC module -** Automated Registration And Certification module. This module (built into the geek side app + appropriate support in the app server) takes the geek through the process of signing up to be a  HAILaGEEK geek.
* **The** **GWPM (geek web portal module)** adds the a rudimentary web portal for geeks to use HAILaGEEK without a smartphone, once they have started a job. They must have a smartphone to sign on as available and start a job, but the **GWPM** allows them to finish the job using the web module in case their smartphone dies after they’ve started the job.
* The **SUPPORT** module allows a geek on site to request support from other geeks, with billing support. An example scenario description is included below as part of the work process description.
* The **CUS-FINGER** module. Allows customers to sign on via their fingerprint, assuming this is supported by their smartphone.
* The **GEEK-FINGER** module. Allows geeks to sign on via their fingerprint, assuming this is supported by their smartphone.
* The **MYSCHED** module. Allows geek to set an auto-schedule for the app to sign them in as available, sign them out as not available. For example a geek might set an auto-schedule of M-F 8am (avail) to 5pm (not avail). This would cause the app mark them as BUSY UNTIL 8am every Sun thru Thur day at 5pm, which would cause them to show up as available every M-F at 8am.

**OPTIONAL ITEMS TO BE QUOTED:**

*These items are a part of HAILaGEEK’s ASAP future, and might even be part of the original project. Even if these optional items not part of the original project, architecture of the project must be designed in such a way to be ready for these items to be added on in the very near future.*

* **The** **CWPM (customer web portal module)** adds the a full web portal for customers to use HAILaGEEK without a smartphone (in case their smartphone is broken), but without the cool graphical map.
* **The** **CWPM-MAP** module adds the same cool graphical geek map the app has.
* **The MULTI-TECH** module adds the ability for a customer to request multiple geeks at the same time, for example, a company installing lots of new computers in a single day
* **The** **MCAM** **(multi-company addon module)** opens up HAILaGEEK to be used by IT or repair fleet companies, who get paid for their own service employees to go out on HAILaGEEK jobs.
* **The** **IAPAM (In-App-Advertising)** **module** - in addition to displaying geeks on the map, if this module is installed, the app also displays fixed-location repair shops, who register on the HAILaGEEK website and pay-per-click.
* **The NSP2 (nowSearch phase 2) module.**An addon to the nowSearch core module, the phase2 module expands the API with a few more features (described below in more detail) to be useful by more apps and web developers (not just us).
* **The NSP3 (nowSearch phase3 module.** An additional addon to the nowSearch module, the phase3 module fills out nowSearch into a website allowing the general public and contractors to locate each other, and contractors to register their ongoing realtime location with a simple app.
* **The** **AIM (additional-info module** adds an API to HAILaGEEK app server that provides an API for external 3rd party apps to retrieve formatted information about geeks, and, the ability for external links to carry a user into a HAILaGEEK web or app session.

**ITEMS WHICH WILL BE PROVIDED BY DYNAMIS:**

* **The certification webserver / website -** a website which will test the skills and check the references of geek applicants and report back to the HAILaGEEK app server concerning what skills the geek applicant qualified for, and skill levels

**The Geek Side App - Signup, REGISTRATION AND CERTIFICATION PROCESS (IF THE ARAC MODULE IS INSTALLED):**

**Note: IF the ARAC (AUTOMATED REGISTRATION AND CERTIFICATION MODULE) is not installed:**

* **The app simply passes the geeks to a signup page url on the certification webserver. The certification webserver / HAILaGEEK staff will collect all their info.**
* **geek records should then be able to be created manually in the admin portal (by HAILaGEEK staff) and all the fields in that record that are normally filled in by the ARAC module are still able to be typed in manually in the admin portal by HAILaGEEK staff**

**Step 1**

* To begin, the new geek applicant downloads and installs the app from the **Apple App Store** or **Google Play** like any other app.
* The app requires GPS location. If the customer declines, the app won’t go any further until GPS is allowed.
* If the geek’s location is outside the **current service area** (one of multiple areas defined in admin app), app displays a simple message indicating “You are outside our current service area. For more information, please visit **[web address]**” (clickable web address for this message also in admin portal)
* App shows **SCREEN 51** indicating this is the technician app.
* App shows **SCREEN 52**  the **list of skills** (maintained in the admin panel). “Click on the ones you are skilled in (you will be tested on the ones you select):” and shows clickable **list of skills**
* Geek clicks on the skills they think they qualify to provide
* App shows **SCREEN 53** an overview of the registration and certification process, informing geek that they must pass all the following before being approved to accept jobs:
  + - * + Cell # verification
        + Email verification
        + Driver's license / State ID verification\*

IF the geek applicant did not specify any skills in the previous step that qualify for #ES# waiver, also add this text:

\*(**$6** charge, refunded after first **$80** of paid work)

* + - * + Skills tests
        + Criminal background check\*

IF the geek applicant did not specify any skills in the previous step that qualify for #ES# waiver, also add this text:

\*(**$26** charge, refunded after first **$280** of paid work)

* + - * Note at bottom of list if they DID qualify for #ES# Waiver:
        + “ \*Note: there will be a **$32** additional charge to complete your registration if your skills tests do not qualify you to provide at least one of the following skills: [followed by list of skills they selected that qualify for #ES# waiver]
  + Note: “Early signups” are the first **#ES#** signups in a geographic area with a particular skill; **#ES#** is a variable in the admin panel for each skill. (The initial value of **#ES#** will be **10** for each skill, meaning the first **10** applicants to show up with that skill will have their fees waived, as long as they pass the skills test in at least one skill where **#ES#** justifies a waiver).
* Geek chooses to create an account or not ; if not, app returns to start
* App shows **SCREENS 54, 55, 56:** If creating the account, geek
  + - * + types their **email address**,
        + a **password**,
        + fingerprint (if the **GEEK-FINGER** module is installed)
        + their **name**,
        + their **cell #**,
        + their **mailing address**, and
        + uploads a **picture** **of themself**,
        + uploads a **picture** **of their driver’s license**,
* It asks the geek applicant “are you a **registered business**? (As a registered business you would receive a commission discount, i.e. we would subtract a lower commission from your earnings).”
* Note: **Commission rate is a field stored in each geek’s record (editable via the admin portal on each geek record), and the admin portal has two overall default fields for  #default commission rate charged to individual# and #default commission rate charged to a company#.** These **defaults** are automatically copied to the **commission rate** field in new geek records. *We will begin with these values: Individual is charged 20% but a registered business is only charged 15%. (This is (1) because officially registered sole proprietors and corporations are considered a much lower risk and (2) because we want to provide a big-enough incentive for geeks to register themselves as businesses, so that they will be on the local sales tax authority’s radar, so that they will be less likely to entice customers to circumvent our app for the purpose of avoiding sales tax).*
* *Administrative note: see* [*https://services.amazon.com/selling-services/home.html*](https://services.amazon.com/selling-services/home.html) *to compare how fees with Selling Services on Amazon works, see* [*https://www.moneyunder30.com/driving-for-uber-or-lyft*](https://www.moneyunder30.com/driving-for-uber-or-lyft) *for how fees incurred with Driving for Uber and Lyft work.*
* If they indicate they are registered as a business, several more fields open up:
  + upload **picture of proof of registered DBA, corporation, or other**. (e.g. local county clerk stamped DBA registration, texas state corporation registration letter, etc.)
  + upload **company logo** (if they have one),
  + **Date their registration was recorded as approved by a local, state or federal govt entity** (with a note “you must have been in business at least 2 months to qualify for the discounted commission rate”)
  + Copy of their **business liability insurance document** if they have it (notifies them they receive another 5% discount if they have this)
* **NOTE:** we should be able to add additional form fields in the admin portal for additional info to be collected on the geek signup form in the admin portal, even though the results of these added fields will simply be stored in the DB and not used by the overall app process until later.
* If at least one of the geek's indicated **skills** was one we have fewer than **#ES#** geek's already in **that area** for, they are informed “Identity and background check fees will be waived for now because you qualify as an early signup in at least one skill category.”  The app server updates a **“waiver” field** on the geek’s record with a list of the **skill categories** that qualify the geek for a identity check fee waiver based on the **#ES#** variables for those **skill categories.** (We will make use of the info in that field in step 4).

**Step 2:**

* The app will text the geek applicant at their **cell #** to verify cell #
* The app will email the geek applicant at their **email address** to verify email address
* Geek applicant is presented with a series of terms of service agreements. At the bottom of each agreement is a place where the geek applicant must type their name and click “I agree” to proceed.
  + These terms of service agreements are stored in the Admin Panel. The admin panel should allow Terms of Service agreements to be added and modified at any time and keep track of which ones have been agreed to by each geek.
  + Any time there is a change to a Terms of Service agreement, that Term of Service agreement should be re-displayed the next time the app is accessed (except NOT while a job is open) and required to be agreed to before the account can continue to be used.
* The app will request a **credit card** from the geek (if fees are not being waived due to **#ES#** variable). “Your card will be charged **$6** for identity verification through **EVIDENTID** or **CheckR.”**
* AFTER cell# verified, email verified, card charged, the app will submit the relevant info from the signup to **EVIDENTID** or **CheckR** (a background check service) via their API for an identity check. (We don’t do a background check yet, as that is more expensive).
* **EVIDENTID** or **CheckR** will come back with a **pass or fail.** A fail means admin review is needed. A pass means the technician is cleared to work as far as identity check goes, and continues to the next step, being passed to our server for further certification (see below - Step 3).
* With an identity **fail**, the app should simply say to the applicant "**your application is being reviewed.** " (note: that text should be stored in a field in the admin portal). and store the information received from **EVIDENTID** or **CheckR** in the **ID CHECK RESULT** field (a large text field) on the geek’s record. A section on the admin portal should allow us to view signups that **failed** identity check via **EVIDENTID** or **CheckR**. On the list, we should be able to manually change to a **PASS** if we choose, then the process resumes as if it was a pass (i.e. passing the info on to our server for the next step).

**Step 3:**

* **(When ID check passed).** Geek is sent an **email** from the app server, **containing a certification URL** link **which includes a key code,** to sign on to the **certification website** at the **certification URL** to be certified. (App notifies geek to look for the email, and offers to re-send it if needed).
  + - NOTE: The **key code** is generated by the **HAILaGEEK app server**. It is 20 alphanumeric characters long, containing upper and lower case letters and numbers,  and is unique.
    - The **key code**, geek’s **full name** and geek’s **email address** are reported to the **certification webserver** via a **back-end** **new signup reporting URL** when the user passes the ID check, for example:
      * http://www.**5ki115.com**/backend/register.htm?**name**=John\_Smith&**email**=john.smith@gmail.com&**keycode**=FhJv7fvYaSDjf8ay7s76
    - When the **key code** is registered with the certification server, certification

server responds with a “**return code**” - another unique, 20 character combination of letters and numbers. The **return code** is stored in the geek’s record on the **HAILaGEEK app server** and will be used to authenticate updates to the geek’s record such as:

* + - * + **Skills** the geek is certified in and a **skill level** for each skill
    - *Note: Dynamis (my company) will provide the* ***certification server*** *where the new Geek applicant is certified in their skills by various means, and also fills out a more comprehensive application including:* 
      * + *References*
        + *industry certifications (a list)*
        + *Payment preference (Paypal or ACH) and bank/paypal info*
    - Once the new Geek is skill-certified, reference-certified, etc. the **certification server** notifies the HAILaGEEK app server via a back-end URL that the geek is certified in certain categories and ready for background check. For example, the **certification server** might make this http request to the app server to notify the **app server** of the certification results:
      * http://www.HAILaGEEK.app/backend/certify.htm?email=john.smith@gmail.com&keycode=[FhJv7fvYaSDjf8ay7s76](http://www.mysite/backend/register.htm?name=John_Smith&email=john.smith@gmail.com&keycode=FhJv7fvYaSDjf8ay7s76)&returncode=X98iJ67DTqTr87HjkNyM&**percent**=xx&**certifiedcats**=1.5-2.1-5.3-6.3-8.4
    - Note: the **certifiedcats** variable passes a list of categories the Geek is certified in + a period + their skill level in each category, separated by dashes. These correspond to categories and skill levels listed in the admin portal, in order they are show in the admin portal. For example, in the example request above, 1.5-2.1-5.3-6.3-8.4 means the Geek is certified in category 1 with skill level 5, category 2 with skill level 1, category 5 with skill level 3, category 6 with skill level 3, and category 8 with skill level 4.
    - Note: the **percent** variable passes a commission percent that the geek is charged of each job performed. For example if it’s a 10 that means the geek is charged a  10% commission of each job they perform (The geek is paid 90%, HAILaGEEK gets the other 10%). This percentage is used in calculating estimates and what the customer is charged per hour, based on what hourly rate the geek enters they wish to make that day.
    - Note: the **certification server** should be able to update the Geek’s record on the app server any time by making the HTTP request above.

**Step 4:**

* The HAILaGEEK **app server** checks the “**waiver**” field on this geek record to see if there was a waiver registered for this geek and if so, what skill categories that was based on at that time.
  + If the geek didn’t qualify at least a level 2 in at least one of those skill categories, the app pops up “Unfortunately, none of your certified skills qualified as an early signup to validate a fee waiver, so we will need to charge you the standard fees for identity and background check.” (this language is stored / editable in the admin portal). The app then requires a credit card for both the identity check (**$6**) and the background check (**$26**) to proceed.
* The **app server** will submit the relevant info from the signup to **EVIDENTID** or **CheckR**  for a  background check) via the **EVIDENTID** or **CheckR** API.
  + Note: any time the app server attempts to communicate with **EVIDENTID**’s API or the certification server’s API, and does not receive a response, it should store the failed request and some type of process marker in a queue, so that it can be tried again at a retry interval.
  + For example: **EVIDENTID** server goes down for 3 hours. During that time, several geeks were trying to sign up. Some of their signups were at the identity check part of the process when **EVIDENTID**’s server went down; others were at the background check part of the process. The app server should (1) notify the geek their signup has been queued, and to check back into the app later, (2) put these geek’s signup processes into the queue so that those calls can be retried later, and (3) retry them later and then allow the geek to continue to process when the API process has once again been successful.
* **EVIDENTID** or **CheckR** will come back with a pass or fail. A fail means admin review is needed. A pass means the technician is **cleared to work** in the **skills** they have been certified in.
* Geek must be **cleared to work** before being allowed to work.
* With a fail, the app should simply say to the applicant "your application is being reviewed" and store the information on the app server. A section of the admin portal should allow us to view geeks with failed background checks. On the list, we should be able to manually change to a **PASS** if we choose, then it will act as if it was a pass (i.e. the geek can begin work in **skills** they have been certified in)

**CUSTOMER side app - SIGNUP PROCESS**

* The customer downloads and installs the app from the **Apple App Store** or **Google Play** like any other app.
* The app requires GPS location. If the customer declines, the app won’t go any further until GPS is allowed.
* If the customer’s location is not within some area on the list of **current service areas** (one of multiple areas defined in admin app), app displays a simple **message** indicating “You are outside our current service area. For more information, please visit [web address]” (clickable [web address] URL for this message is also modifiable in admin portal)
* When launching the app the very first time after installation, the app shows ***”HAILaGEEK is really easy to use! Swipe left for an easy 30-second tour.”*** With ***“Watch video instead”*** and ***“skip tour”*** buttons. The pop up window has the familiar dots at the bottom and “*(swipe left to continue tour)*.” As they swipe left, the remaining screens of the tour are shown. The **tour screens** are all static picture screens which can be updated in the admin panel.  The **tour screens** can be zoomed in and out by using the familiar two finger expand and contract motions.
* When the app is first launched (after the tour screens if first time), the customer is shown **SCREEN 1,** a ride-hailing-like screen showing all online geeks in the area, but it is not clickable yet; instead, two buttons appear in the middle which read **“Sign up” and “Sign In.”** The app retrieve and display the list of geeks that are within the displayed radius. (The app should use some combination of the **app server,** **Glympse, Google Maps, MapBox,** and/or **nowSearch** to achieve this). **Geeks are shown on the map as icons which were uploaded to the app server: the available geek icon and the geek on job icon.** *The preliminary icons used will be G's with propeller caps. Geeks on jobs will be shown with red https://lh4.googleusercontent.com/g4N1nDQQ5Xr7bG0CZjtLCslk5t7Zh3IPSRuocfg-du7PpjjJn2TAcd1MIMRCKIdjFMw11guRqWJgiM5_rf-oIX-Hlgla92UOPZ1Z4QPaAVB5So2jdxXA1G7YwdXMffNYd5BqPsBS’s; available geeks as green https://lh6.googleusercontent.com/-ONRhehIKLZXxZ-9byHGh2byxxD3f9urRMRHAw97Mv_s_jz0VeSjf_NXQkA2UIPei8RIWBNeNCeS1RubAy0zvY5EKZqBF-kvhCy-1TvsZBJILN4q0CRANL4IsZCgm6INpa1TEW26‘s.* **When the geek is moving, the animated version of the icon is invoked.**
* *For example, with our preliminary icons, you would see a green non animated https://lh6.googleusercontent.com/-ONRhehIKLZXxZ-9byHGh2byxxD3f9urRMRHAw97Mv_s_jz0VeSjf_NXQkA2UIPei8RIWBNeNCeS1RubAy0zvY5EKZqBF-kvhCy-1TvsZBJILN4q0CRANL4IsZCgm6INpa1TEW26 where an available geek is nonmoving; a red non animated https://lh4.googleusercontent.com/g4N1nDQQ5Xr7bG0CZjtLCslk5t7Zh3IPSRuocfg-du7PpjjJn2TAcd1MIMRCKIdjFMw11guRqWJgiM5_rf-oIX-Hlgla92UOPZ1Z4QPaAVB5So2jdxXA1G7YwdXMffNYd5BqPsBS where a working geek is nonmoving; a green animated https://lh6.googleusercontent.com/f-wdOzGdJ1FoA9g4XUJj__F2cW5kTPetuv2JG0dwfXhkt52xxjcsFd0FF7_CbsgqjQaTYQX7Wwvs8ajEzko7j4Upo-OhUc70m5pTSCkC4hSO3qbCQ3t82bQC61-aIOJdZ2sXkjeRwhere an available geek is driving somewhere; an animated red https://lh5.googleusercontent.com/-07Vn4twhpoEK6djq0eg75pvy0MiRSPNAbro-lyegqoJeeO34xtK2qYK8QhueWlVr-8Cygcrh9cclzYUfx5rUmaq5BR6ZaMdtRBGt33CUzwIapkl6vufRTqjD7tdhPvGeQGbIwug where a geek on a job is driving on a parts trip or on the way to the job.*
  + If there are more than 4 **available** geeks within the zoomed in (on screen) area, **geeks on jobs** are not shown. Geeks are moving in real time; if a geek is driving somewhere, you see their icon moving down the road. (If there are no Geeks in the area, two simulated busy (red) Geeks are shown at two locations registered in the admin portal. There is a **toll free number message** (editable in admin portal) shown at the top of the app to call for support, if the day/time is within the **7-day support schedule**. The **7-day support schedule** is a schedule stored on the app server.
* Every screen on the app has a **screen #**, shown at the top left in a **gray** **square.** Every popup window in the app has a **popup #**, show at the top left in a **blue circle.**  These numbers in grey squares / blue circles are used to quickly identify what screen the customer is viewing in the app if they call in for support. These screen #’s and popup window #’s only appear in the customer app.
* When signing up **(SCREEN 2)**, customer provides:
  + Name
  + Cell # (authenticated via sending an SMS)
  + Email address (authenticated by sending an email)
    - If email account is not yet verified when customer attempts to request their first Geek, customer is notified that all information will be sent by text instead of email until an email account is verified (and they are encouraged to verify an email account).

\*\*\*\* An admin portal select box setting defines whether credit card info is requested **(a) at signup** (during this signup process)    - or - **(b) at first service call** the first time the customer actually attempts to *contact* a geek for service.

*Business note: we will start off with option (b), do not request cc info until first time customer requests to contact geek for service. This will allow the “where’s G” promotion to run with participants finding G’s using the app.*

\*\*\*\* Credit card info is authenticated using standard CC authentication methods, and must be stored in a 3rd party credit card vault, not in our system

* + A password for their account.
  + If the **CUS-FINGER** module is installed, the customer if offered the option of signing in with their finger print rather than having to enter their password.
* Customer is presented with a series of terms of service agreements. At the bottom of each agreement is a place where the customer must type their name and click “I agree” to proceed.
  + These terms of service agreements are stored in the Admin Panel. The admin panel should allow Terms of Service agreements to be added and modified at any time and keep track of which ones have been agreed to by each customer.
  + Any time there is a change to a Terms of Service agreement, that Term of Service agreement should be re-displayed the next time the app is accessed (except NOT while a job is open) and required to be agreed to before the account can continue to be used.

**REGULAR WORK PROCESS (for registered customers and registered geeks)**

|  |  |
| --- | --- |
| **CUSTOMER SIDE** | **GEEK SIDE** |
| * When the customer launches the app, if they are not logged in, the app shows them **SCREEN 1** (same screen as pre-signup), and the app has them log in to their account. * The “**customer message of the day**” pops up, if a message is configured in the admin portal for that day of the week. *For example, for Sundays, we will configure a message: “It’s REV day! -- Rest / Emergency Volunteer day. If you have a genuine computer emergency, you might find a geek volunteering to help you free of charge. (You can tip though).”* * If the customer tries to log in at a location more than **120** miles away from the location where they last requested service or signed up, the app requires them to re-authenticate; to enter a code received via a text or email. (This is not needed if the If the **CUS-FINGER** module is installed and customer is on their fingerprint-enabled device). * Once customer is logged in, the app shows the customer **SCREEN 3,** a ride-hailing-like screen showing all online geeks in the area, but it is not clickable yet; instead, a **popup (15)** appears in the middle with the question “Where do you need a geek?” and two buttons which read **“HAILaGEEK here”** and **“Different address.”**  The customer’s location is a flashing blue dot in the middle of the screen. * If customer clicks **“Send a geek to another address”** a window **POPUP (16)** pops up requesting the address specifics including a “who to meet on site” line and multi line “how to find them” field. Once that address is identified, the map changes view to the area around that address. * **Geeks are shown as G's with propeller caps. Geeks on jobs are shown with red**   **https://lh4.googleusercontent.com/g4N1nDQQ5Xr7bG0CZjtLCslk5t7Zh3IPSRuocfg-du7PpjjJn2TAcd1MIMRCKIdjFMw11guRqWJgiM5_rf-oIX-Hlgla92UOPZ1Z4QPaAVB5So2jdxXA1G7YwdXMffNYd5BqPsBS’s; available geeks as green https://lh6.googleusercontent.com/-ONRhehIKLZXxZ-9byHGh2byxxD3f9urRMRHAw97Mv_s_jz0VeSjf_NXQkA2UIPei8RIWBNeNCeS1RubAy0zvY5EKZqBF-kvhCy-1TvsZBJILN4q0CRANL4IsZCgm6INpa1TEW26‘s. *When the geek moves, the propeller spins.*** **Geeks who are on jobs with “take jobs after this one” disabled are not shown at all.** If the customer has a credit on their account (see admin portal “customer accounts” feature below), “$**xxx** credit” appears at the top.  Another small button says “refer a friend, get $**5**”   * If the customer clicks “refer a friend” a popup windows explains that when their referred friend spends or earns $**50**+ on HAILaGEEK, we’ll credit the referring account for $**5**. * ***QUESTION for developers: what is the most efficient 3rd party API or any method of providing the “map of geeks” functionality? Glympse? Google Maps? Build our own? For example, Glympse has an excellent API for sharing serviceperson location with customers - should we use that?*** | * Whenever the Geek side app is launched, it shows (1) the Geek’s current credit balance (unpaid money earned) and (2) the date and amount of the last payment. * The “**geek message of the day**” pops up, if a message is configured in the admin portal for that day of the week. *For example, for Sundays, we will configure a message: “It’s R/EV day! -- Rest / Emergency Volunteer day. Your work and travel rates will be set to 0. The people you help can tip you if they like (but not through the app). People you help for free tend to give a 5 star rating … ”* * The geek has a “refer a friend, get $5” button. If geek clicks it, a pop up window reads **“Refer a friend and when they spend or earn $50 on the app, we’ll credit you $5.”** * If a geek wants to appear on the map but appear as on a job, the geek clicks “**appear on map as busy**” and enters a time (hours+mins or actual time) when they will be available. (Geek also has the opportunity to modify their “**work rate of the day**” and “**travel rate of the day**.”) **Appear on map as busy** causes the geek to appear on the map as if they were on a job. (The geek cannot receive messages, but viewers can see their rates and other info). * When a Geek is available to work, geeks signs on to the app as “**appear on map as available**” and has the opportunity to modify their “**work rate of the day**” and “**travel rate of the day**.” They can be entered in terms of what the customer is charged, or what the geek is paid; the app automatically calculates and fills in the other #’s as these values are changed. * If the geek has never entered a rate before, the default rates are entered, from the admin panel. *We will start with travel @ $48/hr and labor @ $93/hr.* * These are temporarily set to **$0** if it’s Sunday, but the app server still remembers the geek’s previous workday rates so it can restore them the next workday. * (Geek can enter **per minute** or **per hour** rate; if you enter per minute it calculates/autofills **per hour** and vice versa). (It auto fills with the previous rate of the day or if he’s brand new, a “suggested” rate of the day based on his skills). *These are set to $0 if it’s Sunday, but the app server still remembers the geek’s previous rates so it can restore them the next workday.* * For example, say a geek has been assigned 15% commission rate. The geek enters $102/hr as their hourly rate. The app shows them **in gray** that after commission, they will receive **$88.69/hr.** |
| * **POPUP 4:** Then it asks them “what type of Geek do you need?” The list of **visible categories** (maintained in the admin app) is shown, and they can select multiple … (the following list is just an example): * *General Home Computer Support* * *General Business IT support* * *Wifi / Networking / Security* * *Virus Removal* * *Smartphone repair* * *PC repair* * *Mac repair* * *VoIP phone support* * *Audio & Video* * *Smart Home* * *Cisco device configuration* * *Server hardware repair* * *Windows Server support* * *Linux support* * NOTE: The names of all the categories listed above is just a sample list. It must be able to be edited (and each category marked visible /  non-visible) via the admin portal at any time and that list in the admin portal should automatically be used in the app, i.e. **the app should *always* either pull its list of categories from the app server / admin portal list** **or update it once per day.** * Also note: for each category, there is a **“STFM” (“service type flag message”)** flag which can be checked or unchecked (and the accompanying message text).  This flag affects whether a special message appears while the geek is on his way. (see below). * Once the Customer selects what type of work they need, the app asks the Customer to enter a brief job description, such as “something took over my pc and I need to get to my files!” or “my home theater sound isn’t working” then click “Show me the geeks!”  **(popup 5)** * **SCREEN 6:** The non relevant geeks disappear from the map and now only Geeks matching all those categories of expertise appear. **There are also now three tabs at the top: map view, list view,** and **favorites.**  If the customer chooses List View, the map disappears and a simple list of geeks appears with distance in miles and estimated travel time, and the List View screen also has a Favorites button. If the customer chooses Favorites (on either view), only geeks the Customer has previously chosen as Favorites appear. * If there is not one Geek who matches all the multiple categories selected, a note pops up “not one Geek in the area had all those skills. We’ve matched your request to the ones with the most skills on your list. Click each match for more details.” At this point the app shows all matches that meet at least part of the criteria,attempting to show worst case at least 1 match that matches at least one of the categories, and showing up to 10 partial matches that match the most criteria and are available soonest.  If no geeks matched any categories selected, no geeks are shown and a popup message **POPUP (9)** appears “no geeks are currently available to match that list of categories. Please check back again later, or request an appointment at a later time that works for you.” (A clickable link that opens up the web browser to an **appointment form at a URL** specified in the admin panel, passing the email address, name and phone number of the customer to the form). |  Geek can choose “**appear on map as busy**” or “**appear on map as available**.” If geek is set to “**appear on map**” their location is reported to **nowSearch** every time it changes by **50** feet (variable can be changed in admin panel. *Higher number saves bandwidth*).   * While waiting for queries, Geek is shown a map of where customers have opened the customer app any time in the past 2 hours (red dots) and where they are opening it / actively using it now (green dots), until they choose and begin working with a geek (then their dot goes back to red). *(The purpose of this is so that the geek can travel to a different part of town if there are more people opening the app in that part of town).* It also shows how many potential customers have viewed the Geek’s profile today and this month. |
|  **IF OPTIONAL IAPAM (IN-APP ADVERTISING MODULE) IS INSTALLED:** Fixed location repair shops and roaming vendors (from **NSP3**) are also shown, as blue “shacks" on the map. If customer clicks on one of these it shows “fixed location repair shop”, name of shop, and skills, etc. and “click for contact info and driving directions.” If they click for that info, shop is charged a “contact per-click **fee**” which is stored in the database / admin panel for each IAPAM shop. |  |
| * The customer can click each match on the map, then read **POPUP (8),** which contains the following details about that Geek:   + When he is available (e.g. “right now”, “in about 30 mins”, “in about 3 hours”). If the geek is on a job, the time estimate is based on the estimated current job completion time entered by the geek.   + The Geek’s picture (and his company logo if he uploaded one)   + The Geek’s current customer rating (1 to 5 stars including partial https://lh6.googleusercontent.com/gA_fBEnbbCK2bjvNlbHETVjvBBGgJLaP7MCFsslT8Jn5XFKuOWh5ag_gFdCFvZLwrOKk_6klMhU9G57Ip8dWZ-Hp0GrUCir1UPOVC006Zr_IcHHk6uT1UrYwyESKrJtKQ0llTG5xstars) and # of jobs completed.   + What his skills are, with **skill levels** listed in parenthesis beside (for numeric) or underneath (for text) each skill (e.g. “4 / 5 ”, “uncertified” , “master”)     - Skills levels are 1 through 5.     - If the skill level is not *labeled* in the admin portal, the simply shows a *number* (1-5),   e.g. “ 4 / 5 “.   * + - These **skill levels** can be labeled in the admin portal. For example, in the admin portal, there is a category “Cisco configuration” listed, and skill levels 1-5 for “Cisco configuration” are defined in the admin portal as:       * Uncertified but capable       * CCT / equiv       * CCNA / equiv       * CCNP / equiv       * CCIE+     - As with the list of categories, the skill levels must also always be editable in the admin portal and the app must pull the descriptions of the skills from the admin portal list.   + What his charge is by the hour (also auto-computes and shows the by the min version, if the “show by the minute” option is set in the admin panel), including HAILaGEEK’s commission.   + Skill levels in the geek’s certified categories (show 2, then “view all”) to expand to more, so that “Customer reviews” is hopefully visible within the first screen of info without scrolling down)   + Customer reviews (probably have to scroll down to view all these) * The customer can then choose that Geek using the “Pick [name] ! “ button or choose “Back to list …” to go back to the map to view others. * If the geek is on a job (or set to appear busy) when the customer chooses “Pick [name]!”, a POPUP (25) appears notifying customer that the geek is on a job, but if they choose to leave the app on that screen to wait for that geek they will have priority over any other requests that geek might receive. In other words, they are next in line.   + Note: this causes the geek to not show up on the map for any other customers, and if a subsequent customer whose screen already had this geek shown clicks “Pick [name!”, they are shown POPUP (26) letting them know the “next in line” spot has already been taken. * Then when the geek is done with their current job or “set to appear busy” period, the app sends a notification text and/or email to the customer, and plays a notification sound on the customer app, and then immediately proceeds with the next step. This allows the customer to stay “on hold” / “in line” for this geek as long as they want. * When / if the customer chooses a Geek, the app shows **POPUP (10)** “a few quick questions” asks these questions, and adds a summary of the answers to the job description:   + - Do you have another estimate for the work you need done? (if customer selects yes, show three other questions):       * Is the quote for carry-in or on site service? (carry-in / on site / ship-in)       * What is the amount of the other quote? (amount)       * Do you think the price quoted is fair? (yes/no/no comment)   + NOTES: 1. None of these questions have to be answered to proceed. 2. These questions are only asked once. 3. The answers to the questions are added to the **job description** when answered, and they remain part of the **job description.**  For example, the text added to the **job description** might be: “I have another quote for carry-in repair for $80, seems unfair. “ * If customer is waiting for a geek to finish their current job (next in line), and the geek changes their estimated time to complete job, the customer is notified via a popup and sound. |  Each time a customer views the geeks profile, a small (silent) window appears at the the bottom for a few seconds, and again if they read the actual reviews on that geek; the window shows a scrolling log of when a customer viewed the geek's profile and/or read their reviews. E.G. "a customer just viewed your profile" and/or "a customer just read your reviews." This helps the geek understand (a little bit) if customers keep reviewing their profile and reviews and then not choosing them. It also helps them understand customers are actually seeing they are there. |
| * **POPUP (11)**Customer is notified that the Geek is being offered the job and will decide to take it or not within **XX** seconds, (a countdown timer is shown) and that the Geek may message the customer with questions about the job. * If Geek initiates a message session, the message **POPUP (13)** is also added  to the customer screen. * (Note: If a message session begins, this pauses the **XX**-second timer during the message session). * If the Geek rejects the job, the customer is notified and is then directed back to the map. If the Geek doesn’t respond within **XX** seconds, the customer is given the choice to either wait another **XX** seconds for a response or go back and find another. | * When a customer chooses the Geek, the Geek receives a notification along with job description and vague map location and has **XX** seconds to approve or deny the job being offered. Geek can also initiate a message conversation with the potential customer at this point, to clarify any questions. The **XX** second timer resets to **XX** when geek or customer types in the chat window; the chat session ends automatically if no key is pressed for over **XX** seconds.  (**XX** is a number specified in the admin panel. *When the app is first created,* ***XX*** *will start with a value of 60).* |
|  If the geek suggests this be a remote job (by pressing his remote job button), we go to **POPUP (12)** “**[Geek-Name]** has requested to begin the work now without coming on site.” with customer buttons for approve or deny, but we also keep the chat session open. |  If, during the course of the conversation, the geek suggests this be a **remote job** and geek and customer agree that remote support / diagnosis / repair (without travel, for now) is what is needed, the geek can click “**remote job**” button.  This skips the travel phase and jumps straight to the **job estimate** phase (below). |
| * When the geek is ready to accept the job and begin travel and clicks the start travel button, **POPUP (17) appears** and customer can approve or deny for travel to start. Meanwhile, the conversation can continue over the still visible message **POPUP (13).** |  When/if the geek is ready to accept the job as a onsite job and begin travel, geek clicks the **accept job, request to start travel** button. Geek is notified the customer is being asked to approve for travel to begin. A message from “Sys” is added to the message conversation saying “Address will be sent when trip approved.” Only geek sees this message. |
| **Travel phase**   * When/if the Geek and Customer agree for travel to begin, the app server attempts to put a $30 hold on the customer’s card (or otherwise verify it is still valid, not blocking charges, etc). If there is a problem, customer is notified and is taken to a screen to update their credit card info. If the customer takes more than **10** minutes to update their card info, then after completing updating their card, they are taken back to the screen to choose a geek again. * After card is verified, the customer is notified the Geek is on the way and the app shows **screen [7],** including the Geek’s position on the map and **POPUP (14)** with the estimated time of arrival.   + ***QUESTION for developers: what is the most efficient method / API of providing this “watch the geek come to you” functionality? MapBox? Glympse? Google Maps? Build our own? Glympse and MapBox have an excellent API for sharing serviceperson location with customers - should we use that?***   + ***Will it save money to develop this app without the NOWsearch phase 1 module?*** * **Screen [7]** includes a new indicator at top right displaying the calculated job charge. It updates from the server once every **UU** minutes. | **Travel phase**   * When/if Geek and Customer agree for travel to begin, geek is notified that the server is checking the customer’s credit card before travel begins. If there is a problem with the customer’s credit card, the geek is informed that the customer is updating their credit card information. The geek is asked to hold for up to **10** minutes while this is completed (with a countdown timer). (I.E., the geek cannot just quickly back out and be available for other jobs). * After card is verified, Geek receives the customer’s actual full address and begins driving to the customer location. (The app opens Google Maps or Waze for driving directions, and also allows the geek to copy the address (street address or GPS coordinates) to the clipboard. Note: the address passed to Google Maps or Waze should be GPS coordinates if a precise address listing was not able to be identified by the app). * The app server begins computing the **travel charge** at the **travel rate.** (The customer can see the geek on the map, driving to their location). When the geek is moving, the propeller spins. |
|  Initially, the message section disappears at this point because it is not safe to message the geek during travel. |  Initially, the message section disappears at this point because it is not safe to message the geek during travel. |
|  There is a button “Send urgent message to GEEK” in **POPUP (14)** which the customer can press.   * If customer presses this button, a new **POPUP (18)** message pops up “**[geek name]** is already en route to your location. You can send an urgent message to the **[geek name]**, but please be patient while the **[geek name]** finds a safe place to pull over to safely read and respond to your message.”  Then an **OK** and a “**it can wait (recommended)**” button. * If customer presses **OK** button then the messaging **POPUP (13)** reappears and messaging can resume, but **POPUP (18)** remains on screen. If at any time the customer presses “it can wait” the messaging **POPUP (13)** is removed from the screen. | * If the customer actually sends a message to the geek during this phase, notification sound plays from the geek’s app and a notification window (if possible with OS restrictions) appears  on the geek’s phone: “Please pull over to a safe location when possible to receive a urgent message from the customer.” (This same message also appears as popup on HAILaGEEK geek-side app itself). * If this occurs, app beeps at a regular **urgent geek notification interval** until geek location stops changing or geek presses “safely pulled over” button on notification window or switches to the HAILaGEEK app (where it is on top). popup window and presses OK. * Once safely pulled over is indicated (one way or another e.g. lack of movement, e.g. button “Passenger Responding” ), geek can participate in messaging session. |
|  If there is a **STFDR “service type flag dialog record”** in the admin portal for this service type, the **STFDR** (entered for this service type in the portal) pops up in the **POPUP (19)** format. The **STFDR** consists of:   * An **STFDR name** * A short message * A list of choices to select * A place for the customer to type their name IF they are choosing the first choice on the list * A submit button    The customer is required to make a choice, type their name, and click “Submit.”  *For example, for the service “Virus Removal”, the* ***STFDR message*** *entered in the admin portal, will be: “Your data is the most valuable part of your device. Protecting you from data loss is very important to us. Please indicate whether you would like****[$geek\_name]*** *to make a backup copy of your data.” The choices entered in the admin panel for this* ***STFDR*** *would be: (1) I decline to have* ***[$geek\_name]*** *make**a backup, or (2) I need* ***[$geek\_name]*** *to make one to protect my data.”*   There is a “skip for now” button which allows the customer to answer this later. *This gives the customer a chance to think about it while the geek is on the way, even though labor can’t begin until a choice is made.* |  |
| * Note: the job state is held in the app server. That way if the customer’s phone crashes or has to reboot or something, when the customer logs back in or switches to a web browser, the job state is automatically loaded from the server and resumes where it left off. * If a customer logs on to their account on another device, the session is resumed on that device and the old device is logged out of the account. |  Note: the job state is held in the app server. That way if the Geek’s phone crashes or has to reboot or something, when the geek logs back in the job state is automatically loaded from the server and resumes where it left off.   If a geek logs on to their account on another device, the session is resumed on that device and the old device is logged out of the account. |
| * During travel, if the geek presses the button “message customer” the **message popup (13)** reappears on the customer’s app and a short sound is made each time a message is received from the geek in travel mode. |  During travel, the geek has a button “message customer” In case the geek wants to stop and ask the customer a question during travel. The geek cannot press the button while in motion unless they also press a “Passenger Messaging”. If the geek is in motion and tries to press the button, the app pops up “please find a safe place to pull over before messaging” with a [passenger messaging] button to bypass this safeguard. |
|  When the Geek arrives at the site, the customer is notified to be looking for the Geek (picture is shown) and has the opportunity to message with the Geek over the app. | * When Geek arrives, he clicks “I have arrived” The app reminds the Geek that video recording his work **is recommended** for the Geek’s safety and protection, and can assist in resolving any future disputes about the quality of the work or the identity of the customer. (The geek can then start their own smartphone’s video recording app if they want - their choice). |
|  | * The geek receives a report concerning the customer’s answers to any **STFDR**’s. (This way, for example, the geek knows if they need to run a backup or not). * If the customer clicked “skip for now” on any **STFDR**’s, the geek is informed that the customer will need to make a selection on the **STFDR** before work can begin. |
|  When the Geek meets the customer, they talk about the job.   If the geek resubmits any **STFDR** to the customer, it pops up on the customer’s screen. |  The Geek locates the customer at the job site, they meet and briefly discuss the job.   The Geek can click the **Resend Terms** button to resubmit any **STFDR’s** to the customer. |
| The **job estimate** phase   * When the Geek has entered his estimate of how long the job will take (just a quick hours and minutes estimate) the app notifies the customer **(SCREEN 8)** of estimated time and charges for the customer’s approval. If the Geek selected “FIXED PRICE” the Customer app notes “Fixed Price” and the price on the screen for the customer’s approval. Sales tax is also estimated and shown. * ***Note:*** *the* ***FIXED PRICE OPTION*** *CAN BE TURNED ON OR OFF IN THE ADMIN PANEL.* | The **job estimate** phase   * The Geek enters Geek’s ***estimate*** of how long the job will take in hours and minutes. App automatically shows Geek the total amount customer will be charged AND total Geek will be paid (different #’s due to our commission) on screen while Geek enters these numbers. * Geek also has option to check an option box marked “FIRM FLAT FEE QUOTE” and enter a fixed price (firm, flat fee quote) for the job as total customer will pay OR total Geek will be paid, and app automatically updates hours, minutes, customer-pay, and/or Geek-paid figures on screen based on entry. For example, if the geek’s hourly rate to the customer is $100/hr, and the geek enters a price of $50 as a fixed price for the job, then the app estimates that the job will take 30 minutes.However, the geek can modify the estimate of how long the job will take. For example, the geek might enter a fixed price of $50, but still estimate the job will take the geek a full hour. This is the Geek’s choice. * *Geek also indicates whether he will continue taking other jobs after this one. (The system uses this information to estimate if/when Geek will be available again for other jobs).* |
| * If the customer wants the Geek to change the estimate, the Geek makes the estimate and the customer app notifies the customer of the changed figures. Geek can makes changes several times.  If the customer finally disapproves the Geek’s estimate, the customer still pays for the Geek’s travel time and the interaction is completed. | * If the Geek decides not to accept the job at this point, he can cancel. Geek will still get paid for the trip if there was a trip, and customer will still get charged for the trip. (Customer can also decide not to use this Geek at this point; customer will still pay the Geek’s travel time). |
| * Note: if a customer disapproves jobs and/or begins onsite or offsite discussions with Geeks more than twice AND more than 30% of times in any given period (for example more than 3 times in a 100 day period), that customer is automatically demoted to a 1-star rating until the customer’s job approval ratio improves.   + Also, that customer is not shown any matching Geeks within 50 miles of their area for the same amount of time   + All customers which are on this type of “hold” status are listed on a “on hold” page in the admin panel where their “hold” status can be reset. This also allows an admin to contact them to talk about their experiences. | * Once the geek has been contacted about or called out to at least 3 jobs, if he or his customer has cancelled more than 3% of the time (e.g. more than 3 times in the last 100 days or since the geek first signed up) the geek is hidden from the map of available geeks until the ratio goes down.  (As time passes, the ratio will go down, because the ratio is based on calendar days (it is not based on job offer events). So say for example the geek cancels 4 times in a period of 100 days. That is a ratio of 4% so he gets hidden from the map of available geeks. Time passes, and 35 days go by. Now he has only cancelled 4 times in 135 days, which is equal to 2.98%. 2.98% is under 3%, so the geek can now appear on the map again).   + All geeks which are on this type of “hold” status are listed on a “on hold” page in the admin panel where their “hold” status can be reset. This also allows an admin to contact them to talk about their experiences. |
|  |  |
| * If the customer approves the Geek doing the job, the customer clicks “approve work.” Note: if the customer’s credit card was not verified for travel (e.g. there was no travel), then card is verified using the process described above at the beginning of the travel phase. * If there are any unanswered **STFDR**’s, they reappear on the customer’s screen (in sequence). Below each is another small window **POPUP (20)**, “you must submit the agreement before work can begin.” Once the customer has successfully clicked “approve work”, the Geek can start working when geek is ready. | * When customer has approved the work (and if customer’s credit card has been verified), the geek’s “begin work” button appears. * When Geek is ready to begin work, he clicks the flashing “begin work” button and begins the work. There is a small message under the button “work timer will not start until “begin work” is pressed.” |
|  At this point the app service begins computing the customer’s charge based on the “work rate” which the Geek indicated.   The customer app **(SCREEN 9)** regularly (once every minute or so?) refreshes the view of the actual total charges and time, including sales tax. |  At this point the app service begins computing the customer’s charge based on the “work rate” which the Geek indicated. |
|  | * Any time during the job, the geek can checkmark or un-check mark the “continue taking jobs after this one” option. * Any time during the job, the geek can click the “Notes” button and add notes about the job and/or the customer’s setup. * The Notes screen warns the geek to \*never\* enter any passwords, pins, or private information, and encourages the geek to enter other information which will be helpful to future geeks assisting this customer. |
|  If the geek requests to view notes from previous jobs for this customer, the app asks the customer if it’s OK for the geek to view the previous notes **POPUP (21).**If the app window is not on top, the app makes a recurring sound at regular intervals until the question is answered.   The customer still has the opportunity to interactively message the Geek through the app. |  On the Notes, screen, the geek can click “request to view previous jobs notes for this customer”. If he does, the app asks the customer if it’s OK for the geek to view the previous notes (and informs the geek the customer is being asked). If the customer approves, the previous notes are then available to the geek. *These notes might include information about the network, etc. but should never include passwords, private info or pins.* |
|  When / if the Geek takes a break, **the customer is notified (SCREEN 10) that work and charges have stopped,** how long the break will be for and what for and when to expect the Geek back to resume work.   The customer still has the opportunity to interactively message the Geek through the app.   * The app notifies the customer **(SCREEN 11)** when the geek resumes work. * IF the geek chooses to charge the customer for work done so far, the app notifies the customer **(SCREEN 12)** when customer’s card will be charged for “charges so far” because geek is beginning a break more than 24 hours long.  (If the customer has a credit balance on their account, this credit will be used first). The customer has the option to decline charge. If customer chooses “declines charge”, geek is notified.  If customer chooses “approve charge”, the customer’s card is charged and the “decline” and “approve” buttons disappear. | * During the job, the Geek can “take a break” to eat lunch, take a personal phone call, etc. by clicking a “go on break” button and indicating how long he anticipates the break will take and/or what day and time he will resume this job. The app should advise the Geek to CONTACT the customer and discuss this before proceeding. (Breaks should be noted in the job log for admin viewing later). * While on break, the geek can update his estimate date/time of when he will return to the job. (Some jobs can take several days, so, a “break” might be an all-night or even all-weekend break). * Customer ***cannot*** see geek’s location while geek is on break. * **IF** there was a previous travel session to this job, the Geek clicks “**Return Trip**” when he is ready to travel back to the job site. The app doesn’t charge the customer for this time, but does notify the customer the geek is en route, and allows the customer to see the geek traveling on the map. * The Geek clicks “**END BREAK / RESUME WORK**” when he is back at the job site and ready to resume work. * Note: if the break length is set for more than 24 hours, the **app asks the geek** if it should should charge the customer for the whole job so far, letting the geek know the customer will be notified of the charge, and recommending the geek talk with the customer before choosing this option. |
|  | * **IF THE OPTIONAL “SUPPORT” MODULE IS INSTALLED:**During the job, the Geek can click the “support” button and type a description of the tech support needed along with selecting a category. This initiates a process whereby 3 geeks with that skill the job is related to, but a higher skill level, (any available on the system) are offered (one by one) the opportunity to provide support at whatever their current hourly rate of the day is. Potential assisting geek must agree within 1 minute, or it moves on to the next one. If one of them agrees, they estimate how long the support will take based on the support need described. the customer is notified of and asked to approve the estimated additional charge, the original geek is placed in “break” mode, and a sub-job is added to the current job at the supporting geek’s rate. The original geek can end support **at any time** and go off break back onto active work (and the supporting geek is notified their support has ended). During support, a chat session ensues between the supporting geek and the original geek. The geeks can share their phone #’s or other contact info, if they wish, for a voice call, remote support session, or whatever they want to do to accomplish the support. The sub-job is added to the total for the job. |
|  Customer is notified when geek has entered a “reduce time” entry; the customer is notified about how much money is saved by the reduce time entry, rather than actual time. .  **POPUP (22)** |  Any time during the job, a geek can press  “**reduce-time**” button to enter a reduction in labor time charged, for time spent on non-job-related phone calls or texts (such as helping another client who called them). .   Note: (The difference between this and a regular break is that on a regular break, the exact length of the break is not known, whereas on a reduce-time break, since it is entered after it happened, the exact length ***is*** known). These will be shown on the customer’s receipt and deducted from the end charge. They should also be entered into the job log as “reduce-time breaks” for admin viewing later. |
|  Customer is notified **(SCREEN 13)** when (if)  geek enters a list of parts to be purchased. Customer is asked to approve or reject the needed parts. Customer’s card is charged immediately for the  parts (only the parts) when they approve the parts *(to ensure funding for the parts).* |  The Geek can hit a **“PARTS”** button to enter a list of needed parts.   As long as the job has actually been started, the parts trip option is available whether or not there was a trip to the job site yet. Sometimes the geek just knows what parts are needed, perhaps by a remote support session.   The geek is shown a screen “choose a parts vendor” (1) Other (2) Any HAILaGEEK approved parts vendor (they are shown the phone # to call the vendor and get a price on the parts needed). (3) midnightparts.com.   If they choose midnightparts.com the app opens a browser window to **midnightparts.com?ordernum=$ordernum&key=$key&geekemail=$email** where $ordernum is a unique serialized numeric order # generated by the app server and $key is the next key in a “order placement key file” which will be provided by midnightparts.com, and geekemail is the email address of the geek using the app. Geek selects needed parts. Midnightparts.com then posts the selected parts list to HAILaGEEK app server (along with ordernum, key, and geekemail variables).  HAILaGEEK geek-side app displays list to customer and informs geek customer is being shown list of parts to approve.   If the parts vendor doesn’t have the needed parts, the geek can click a “back to list” button to choose another parts vendor.   If the geek chooses “other” as the parts vendor, the geek is notified they will need to work out a separate payment arrangement with the customer, because only HAILaGEEK approved parts vendors can supply parts that are charged through the app. At this point they can go back and choose another parts vendor if they wish.   Once the parts needed have been located and priced, they geek can enter the parts and prices into fields in the HAILaGEEK app and it will ask the customer to approve the charge.   When the customer approves the parts charge, a PO is emailed to that parts vendor (at the email address managed for that vendor in the admin portal). At the top of the PO is a message in red: “Please do NOT ever fill any orders from HAILaGEEK without first going to VERIFYPO.COM and typing in the PO # to verify that it is legitimate. Please notify us immediately at 1-844-4DA-GEEK” if you ever receive a PO without this message.” When the parts vendor goes to VERIFYPO.COM they are actually taken to a special page on the HAILaGEEK app web server where they can simply type in the PO number and are shown the PO.   IF the parts vendor is **midnightparts.com**, the HAILaGEEK app server makes a HTTP GET to **midnightparts.com/pmtapi/payment.nws?order=ordernum=$ordernum&amt=$amt&key=$key&geekemail=$email** (where $amt is the amount the customer approved to be paid, $email is the email of the geek operating the app, and $key is the next key in a “payment key file” which will be provided by midnightparts.com).  The app displays on the screen the result of the HTTP GET (which will usually be *“Payment registered. An email is being sent with parts pickup instructions.”* but might be an error, for example, if the geek email that opened the order doesn’t match the geek email that registered the payment. |
| * Customer can watch the geek on the parts trip - since they are being charged for it. | * When Geek is ready to begin parts trip, they click “START PARTS TRIP” and the travel time begins accruing again (instead of their labor time). |
|  |  |
| * If, during a work phase, the geek is working by remote and decides it is necessary to resume work onsite, the app asks customer for permission ( **POPUP (24)** ) for geek to travel to site, and if customer approves, then this job goes back to the **travel phase.** |  If, during a work phase, the geek is working by remote and decides it is necessary to resume work onsite, the geek can hit “request travel to customer site” link. App asks customer for permission for geek to travel to site, and then this job goes back to the **travel phase.** |
|  | * *When the geek gets within 10 minutes of running out of the geek’s estimated time to complete the job, an alert should pop up (and sound) asking if he needs to update his estimated time and allow him to do that. If he does, it notifies both his current customer and any customer who is currently waiting on him. (NOTE: if he already has one customer waiting on him, he does not show up at all for any additional potential customers).* |
| * When the Geek clicks that the job is completed, the customer is notified the job is completed and sees the total charge plus **sales tax**. *Note: sales tax will start at a simple 8.25%. Later we will upgrade the app and server to support different sales tax for different zip codes.* * Customer can click a link to view the job notes if they wish. * An “invoice” (including sales tax) is sent to the customer when the job is marked completed by the geek | * When the job is completed, the Geek clicks “job seems completed” and the geek is advised to get the customer to examine the work before marking the job completed by clicking the “Confirm - job completed.” * The geek enters any final notes about the job and the customer’s setup on a screen that reminds the Geek the customer will be able to read the notes.   + The screen also reminds the Geek NOT to enter any gate codes, alarm codes or other sensitive information in the notes. * The geek is informed job has been submitted for customer payment approval. * The geek can choose “appear as available”, “appear as busy”, or “go offline” (options vary depending on whether the geek was marked “take jobs after this one”) * The app server sends a separate email privately to the Geek, with a link where the Geek can later rate the customer and enter notes (not seen by customer, visible only to admins) about the job and customer. |
| * The customer now has the opportunity to “approve” “disapprove” or “defer approval” on the work within **15** **minutes.**  **(Screen 21).** If they do not approve or disapprove within **15** **minutes, the** “defer approval” option is automatically chosen. | * The Geek is notified immediately via email regarding changes in approved, disapproved, or deferred status. |
|  Privately, the app server sends an email and/or text to the customer offering them an opportunity (via web portal) to:   * **rate** and **review** the geek (publicly viewable) * enter **private notes** (simple text) concerning the Geek (which appear next time this customer looks at this geek on the map), * “**favorite**” the geek if the customer chooses * (The list of al geeks used, last time each geek was used, private notes, and “**favorite**” status of each geek is stored in each customer’s record).    The email includes “**job-done addon message of the day**” (stored in admin portal). *For example, on Sundays, the addon message will be “if your geek helped you free of charge today, it would be super nice of you to give them a 5 star rating.”*   If they chose (or the **15** min timer expired and app defaulted to) “defer approval” they are also offered the opportunity to approve or disapprove the work. If they do not approve or disapprove within **24** hours, the work is marked approved (and they are notified this will occur). |  The Geek is notified immediately via email regarding changes in approved, disapproved, or deferred status. |
|  The customer is charged when **(A)** the work is marked completed by the Geek **AND** **(B)** the customer approves the work, OR after **24** hours if the customer defers approval but does not specifically disapprove the work.   * A reminder email (and a reminder text if between 10am and 6pm) is sent every 8 hours while a job is awaiting approval    A “receipt” is also sent to the customer when the job is charged. If the customer has a credit balance on their account (see admin feature “customer accounts” below), that credit amount is used before the customer’s card is charged for whatever amount remains to be charged after the credit (if any). |  |
| * **If the customer disapproves the work, charging is deferred, and the customer is notified that charging is deferred and can choose to work it out with the geek or purchase a peer review of the job. (See peer review process below).** | If customer purchases  a peer review, geek is notified by email that a peer review is taking place. |
| * When the job is **finalized** (including any peer review resolutions, etc) a back end server http request is sent to **HAILaGEEK.net** with job info including customer geek rating, amount of charge, zip code. (Allow us to take unique actions such as offer a coupon code for a good review on google, etc). |  |

|  |
| --- |
| **PEER REVIEW PROCESS**   * The customer is charged **$20** for the peer review. * 3 other peer geeks (local peers with the same or similar skill set, with at least a history of 3 successful jobs) are chosen “at random” (preferring those with more skill set and history) by the HAILaGEEK app server. * App service sends **job id,** **contact info for these three peers** and a **unique** **review key** to **feeresolution.com server** to handle peer review process. * App service gives the three peers read-only access to the job history on their web portal. The peers can log on to the HAILaGEEK app web server and see the job history:   + All message conversations including Initial conversation before travel   + Time travel started   + Time travel ended   + Time work started, breaks, parts trips   + Parts purchased and total amounts   + Time work ended   + Job notes from geek   + Total of all fees and parts |
| * After 72 hours (approx) the **feeresolution.com** **server** responds to the HAILaGEEK app server with the **job id**, **review key** and a **charge amount** via a URL, the customer is charged and the Geek is credited.   + If the customer’s credit card is declined, admin is notified by email, geek is notified by email, Geek’s credit for the job is put on hold, and admin intervenes manually.   + If the reviewers decided the customer should have been charged the $20 for the peer review, the $20 is added to the amount paid to the geek.   + If the reviewers decided the customer should NOT have been charged the $20 for the peer review, the $20 charge is refunded to the customer’s card. |

**Administrative Website**

1. Ability to edit the list of all the skill (service type) categories (numbered), along with whether each category is a category currently visible on the app, whether a service type flag message (STFM) is enabled, the text of the service type flag message (STFM), and the skill level labels for each category. For example:
   * + General computer support: Visible [y/n] STFM [y/n] :\_\_*(stfm message)*\_\_\_\_\_\_
       - 1-2 yrs experience
       - 2-3 yrs experience
       - 4-5 yrs experience
       - 6-10 yrs experience
       - 11+ yrs experience
     + Cisco configuration : Visible [y/n] STFM [y/n] :\_\_*(stfm message)*\_\_\_\_\_\_
       - Uncertified but capable
       - CCT / equiv
       - CCNA / equiv
       - CCNP / equiv
       - CCIE+
     + Microsoft Server : Visible [y/n] STFM [y/n] :\_\_*(stfm message)*\_\_\_\_\_\_
       - Uncertified but capable
       - MTA or equiv
       - MCSA or equiv
       - MCSD or equiv
       - MCSE+ or equiv

Etc . etc. etc.

Note: if no “label” (such as 1-2 yrs experience, etc.) is entered in the admin app, the customer app will simply show the Geek as having a “Level x” skill level (Level 1, Level 3, etc)

Note: once a category is created, it is assigned a record #; this number and the category itself are permanent, although the name can be changed, and whether or not it is currently visible in the app can also be changed.

1. Ability to see disapproved jobs, and to modify outcomes during that 24 hr period after the peer reviewers have had their 48 hrs to deliberate.
2. On demand, creates a CSV export with a report   of completed jobs credited to each Geek which will be used to pay Geeks (monthly) and/or print / submit 1099’s using a different app (annually). Report should include Geek name, email address, a unique key for each geek (something we can use to find the geek in the HAILaGEEK geek database), and total of all amounts earned between the two dates. Sales tax report should just include all jobs between two dates summarized and totaled by zip code (show fields: name of customer, customer cell #, and total of job). (Having the total for each zip code allows us to pay the sales tax due in each zip code). This report also includes a checkbox: “mark all reported amounts as paid” which causes the Geek app to show that date as the date of last payment, the Geek’s credit amount as the amount of last payment, and reduces the current credit amount in the Geek’s account by that amount.
   * **For example:** the report is run on October 5, for the dates of September 1 through September 30. Geek Bob did $500 worth of work in September. When the report is run, Geek Bob has a credit amount of $567.23 in his account. The Admin is issuing payments for September that day, so the Admin selects “Mark all reported amount as paid.” The admin portal reduces the Geek’s credit by $500, leaving a credit of $67.23 in Geek Bob’s account. The next time Geek Bob runs the app, he sees the “last payment date” as October 5, “last payment amount” as $500, and “current credit amount” as $67.23.
3. Add: On demand, creates a CSV report showing all the zip codes where jobs were performed between two dates, and the total amount of the jobs in each zip code. This allows HAILaGEEK to pay the correct sales tax on all jobs. The HAILaGEEK portal does not need to compute sales tax, only show a total for all jobs performed in each zip code.
4. Allows us (HAILaGEEK co) to specify our commission **for each geek.**
5. (for security and IT purposes) Allows us to edit the list of certification server IP addresses that the certification servers will make HTTP requests from (for app servers to receive updates to Geek records from certification servers, such as indicating a geek has been approved for skills and their levels). App servers should only accept updates from these listed IP addresses.
6. (for security and IT purposes) Allows us to edit the list of certification server IP addresses that the app servers should make HTTP requests to (for informing us of new Geek signups with their keycodes). certification servers will only listen for requests at these IP addresses.
7. **IF THE MCAM (MULTI-COMPANY ADDON MODULE)** (please quote separately) **IS INSTALLED:** Ability of a IT company to create a “company account" where they are able to enter and set attributes for their Geeks (skills, levels, etc) and see history and revenue totals for their Geeks. This creates logins & passwords for their Geeks, and all payments go to a single company account.  **Note: Regarding comission % rates for company accounts: using the admin portal, we (superadmin) set the commission % rate for the whole company (it will usually be a 10% commission they pay to us) and and then that commission % rate is applied to all the geeks working under that company.**
8. (If the **iAPAM (in-app-advertising module)** is installed) ability of **fixed-location repair shops** to create an account, register their location and skills, and see their HAILaGEEK map listing revenue. (Having done this, they show up as “shacks” on the HAILaGEEK customer app if skills match, and their location is also registered on **NOWsearch**).
9. **Optional CWPM (customer web portal module)** (please quote separately) **to allow customers create an account, add a credit card,  log on with their account and perform rudimentary non graphical tasks: cancel a request, approve a job, request a geek (text listings only, including how far from their address, skills, rating, and how soon available, and change their password. This is in case their phone dies and they need to hail a geek to fix it.  Optional CWPM-MAP addon integrated web map: graphical clickable map similar to google maps** (please separately quote additional cost for implementing this graphical clickable map).
10. **Optional GWPM (gech web portal module)** (please quote separately) **to allow techs to log on with their account and perform rudimentary non graphical tasks: everything they can do with the app once a job is started, but with no mapping / navigation and no initial job acceptance features. This is in case their phone dies mid job, they can still complete the job.**
11. **Ability of admin to edit the 7 day schedule of toll free customer phone support availability. (The 1-844 support # is shown at the top of the app if the day / time is within these support availability hours).**
12. **Ability to search for (by geek ID or customer ID) and review every job that was done. The job log should include geek location at initiation of travel time; customer location at initiation of travel time; customer contact info, geek contact info, time and location where geek indicated geek was on site; time and location when geek indicated job was beginning; time and location when geek took any breaks; time and location when geek resumed from any breaks; time and location job was completed; time and location of customer when they approved, disapproved, or deferred approval for the job, customer notes, geek notes.**
13. **Ability to edit / modify customer accounts, including adding a “credit” to the customer account. A “credit” is a positive balance they can use for a future job. Also ability to change name, ph #, email, and what geeks are “favorites.”**
14. **Other Admin Portal modifiable settings (admin variables) (app settings are offered to app, app downloads them once per day on startup):**
    * **Company Name** *[default: HAILaGEEK]* **(used on web portals and apps)**
    * **Company Logo (upload: is used on website and web portals)**
    * **Geek App splash pic: (upload: is centered on Geek App splash screen)**
    * **Geek App logo: (upload: used in various places on app)**
    * **Geek App background color:** *[default: F7DD00]* **(anywhere a background exists)**
    * **Customer App splash pic: (upload: is centered on Geek App splash screen)**
    * **Customer App logo: (upload: used in various places on app)**
    * **Customer App background color:** *[default: F7DD00]* **(anywhere a background exists)**
    * **Customer app moving available geek animation file (upload)**
    * **Customer app stationary available geek icon (upload)**
    * **Customer app moving busy geek animation file (upload)**
    * **Customer app moving busy geek icon file (upload)**
15. **If the optional additional-info module** (please quote separately) **is installed, the app server has an API that is able to respond to requests from external entities such as NOWsearch with an html response including a formatted rectangular table with various info about an ID (e.g. whose location is stored in NOWsearch), e.g. information about a geek being pointed to e.g. by a web user in NSP3 (NOWsearch phase 3).**
    * **If this optional AIM (additional-info module)** is installed, it’s necessary to be able to specify what information is included in the additional-info response:
      + **Additional-Info content adjustment for external apps calling additional-info:**
        - * **URL+vars: what URL and variables does the external link provided link to**
          * **Logo icon file (upload)**
          * **Checkbox: include available now / when info**
          * **Checkbox: include name**
          * **Checkbox: include business name**
          * **Checkbox: include business logo**
          * **Checkbox: include geek’s Website URL**
          * **Checkbox: include geek’s Google URL**
          * **Checkbox: include geek’s Yelp URL**
          * **Checkbox: include geek’s URL4**
16. **Ability to edit geek accounts, including all geek info (name, company, email, ph #), plus ability to add and remove reviews of this geek**

**The nowSearch core module - created as part of HAILaGEEK app project**

**(Please quote the cost of this module separately)**

* **The nowSearch module** is part of the HAILaGEEK project, but might be used by other projects and entities as well.
* The **nowSearch** module is a server with the following simple capabilities:
  + Receive device location records, hold them in a mysql database short-term, and log all submissions to a second mysql database long-term. Each device record includes these fields:
    - Device ID
    - Latitude
    - Longitude
    - Record expiration date/time
    - Additional-Info URL
  + Replace a device location record when a new device location record is submitted with the same Device ID as a previous record
  + Respond to **list** queries from authorized entities with a list (XML) of device locations within a radius of provided location  (same fields)
  + Respond to **single** queries for the last reported location of a single device ID
  + Delete device location records which have **expired** from the short-term database.
  + Respond to **expire** request by immediately deleting a device location record on request from the an authorized entity
  + **Simple admin panel** allowing the **accounts list** to be edited
* **Sample scenario:**
  + Server X posts to nowSearch: [www.nowsearch.net?id=asfSUDF7878ashJDH&lat=29.7017444&long=-95.4096019&exp=201908011332](http://www.nowsearch.net?id=asefSUFIDF7878asdfhJDHFJ&lat=29.7017444&long=-95.4096019&exp=201908011232) which stores a new record with ID [asfSUDF7878ashJDH](http://www.nowsearch.net?id=asefSUFIDF7878asdfhJDHFJ&lat=29.7017444&long=-95.4096019&exp=201908011232), lat [29.7017444](http://www.nowsearch.net?id=asefSUFIDF7878asdfhJDHFJ&lat=29.7017444&long=-95.4096019&exp=201908011232), long [-95.4096019](http://www.nowsearch.net?id=asefSUFIDF7878asdfhJDHFJ&lat=29.7017444&long=-95.4096019&exp=201908011232) and an expiration date/time of 2019-08-01@13:32.
  + 5 minutes later, Server X posts to nowSearch: [www.nowsearch.net?id=asfSUDF7878ashJDH&lat=29.7017444&long=-95.4096432&exp=201908011161](http://www.nowsearch.net?id=asefSUFIDF7878asdfhJDHFJ&lat=29.7017444&long=-95.4096019&exp=201908011232) which replaces the previous record with ID [asfSUDF7878ashJDH](http://www.nowsearch.net?id=asefSUFIDF7878asdfhJDHFJ&lat=29.7017444&long=-95.4096019&exp=201908011232), with the new latitude, longitude and expiration date in the short-term database, and logs the new submission in the long-term database (without replacing anything in the long-term database).
  + 10 minutes later, server Y requests an XML list of devices within a certain radius. nowSearch responds with the requested list.
  + 2 minutes later single query requests location of a particular ID. Service responds with the location.
  + 5 minutes later a delete request (target went offline) so item is deleted before its registered expiration

**NSP2 (nowSearch phase 2 module) adds the following capabilities:**

**(please quote separately)**

* + Respond to **map** queries (from any entity) with a google map of clickable device locations within a radius of a location, with a date and time and “total matches” below the google map. When you point to the devices a small rectangle window pops up w/contents from the Additional-Info URL
    - * For example: the Additional-Info URL might be: [www.HAILaGEEK.com/additional-info.php?deviceid=asefjiasd78asdf](http://www.hailageek.com/additional-info.php?deviceid=asefjiasd78asdf)
        + and the response from that URL might include a simple html table with the HAILaGEEK logo, “**Computer technician skilled in Virus Removal, PC repair, General Home Computer Support**” and a clickable link with the words “**Available now on HAILaGEEK.com**”
        + So that response (table, logo, text, link) would be displayed in the rectangle
  + Maintain a database of **app accounts** with up to 5 IP addresses per account approved to submit **XML list queries** and/or **map queries.** App accounts are used by other apps which are licensed to submit information to the nowSearch server.

**NSP3 (nowSearch phase 3 module) adds the following capabilities:**

**(please quote separately)**

* ability of **roaming repair entities** to create an account on nowSearch website, and register their skills. (Having done this, they show up as “repair trucks” on the HAILaGEEK customer app if skills match, and are also registered on **NOWsearch** when they launch the **NOWsearch contractor locator app**).
* **NOWsearch contractor locator app:** a separate app which allows a contractor to log into their registered account and repeatedly updates the contractor’s location onto the nowsearch server  as long as they are logged in, and actively expires them from nowSearch when they log out

**QUOTES MUST INCLUDE:**

* Registration of the apps (or assistance registering until completed) on Apple App Store and Google Play, and the information, instructions, and access required to quickly modify the name of the app in both locations if needed due to any unanticipated legal trademark issues
* Installation of the server side of the apps (or assistance registering until completed) on Amazon Web Services, Google Servers, or Microsoft Azure, as well as complete administrative access to this installation, and information about how to manually scale up the size of this hosting service as needed to accomodate extremely fast growth.
* Installation of the server side of the apps on Dynamis’ Windows or Ubuntu Linux Servers, as a standby backup installation, as well as assistance with a procedure to migrate the server side of the apps (database & other data) from AWS/Google/Azure to these standby servers as needed.