**Satellite Shade Analysis**

**Design Document**

Prepared for: Seth Holland, Rooftop Solar

Prepared by: JBOP Solutions

* Taylor Born
* Ben Johnson
* Steffen Olson
* Rachel Pang

Version 1.10

Last Updated: February 5th, 2014

Introduction 1-3

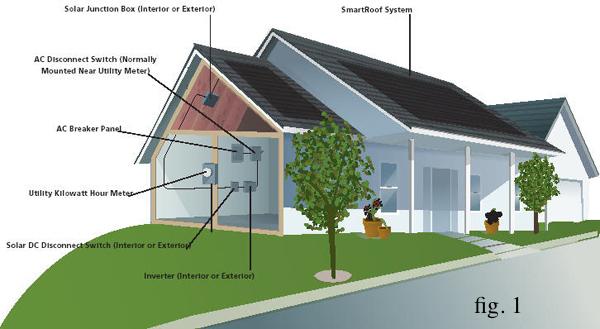
Architectural Overview 3-6

Modules and Interfaces 7-9

Implementation Plan 9-11

Conclusion 11

**Introduction**

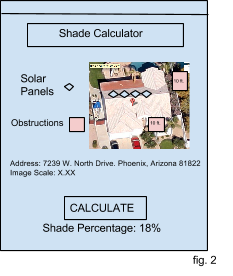
Power consumption has always been a necessity in todays society, and up until recently, people have relied solely on power plants to provide power to their homes. However, solar power has been a growing trend in providing clean, reliable energy for homes and businesses, and since its introduction in the 1980’s, it’s now more affordable and more accessible for everyday people to invest in.

Solar energy is generated through the use of a solar panel system that involves sunlight hitting the solar panels on the roof of the building, which directs the energy produced through a series of junction boxes and panels until it reaches the devices that require the electricity in the building, with the remainder flowing back to the power-grid (see fig. 1). Furthermore, with the introduction of Grid-Tie Solar, if the building receives more solar energy than needed, the extra energy produced can be sold back to their power company, reducing or even eliminating the adopter’s monthly electric bills.

Rooftop Solar is one of the solar paneling companies located in Flagstaff, Arizona that deals with the sales, marketing and installation of solar panels on buildings. On average, they receive around 30-40 clients a month that are potential adopters of solar energy, and give them the information and estimations needed to make a decision on it. They are just one of hundreds of the companies throughout the United States involved with the effort to make energy production greener.

There’s a lot of information to take into account when estimating the solar energy one would receive from paneling. There’s basic knowledge required such as the angle of the roof, as well as the amount of solar paneling desired. However, the hardest part of the estimate relies on obstructions around the building, such as trees and other buildings.

To estimate this shade, a device called a solar eye is used to take an upwards fish-eye lens photo of where the panel plans to be installed, and is done where every panel would be installed to capture any surrounding obstructions. This data is then run through an algorithm that uses sun positioning throughout the course of a year to estimate how much sunlight hits those panels (in a percentage) over the entire year. The amount of shade that hits the panels can make or break an investment of solar energy, with a bad estimate possibly costing an adopter electricity they need that they expected to have.

Our selected capstone project deals specifically with the inefficiencies of the current methods solar companies around the world use in order to get a shade analysis estimate of a client’s roof. The current process involves sending employees on the roof of clients’ buildings to do the analysis. This is not only a huge use of time, as it requires employees to go on site to every potential contract, but a safety hazard for employees, as they need to climb up on every roof and perform the solar eye analysis.

Our project will allow the company to perform these analyses through a web application. This application will take in an image from Google Maps, and allow the user to mark where obstructions are seen in the image, along with where the solar panels plan to be placed. The application will then use this information, with their relative heights and the image scale, and calculate how much percentage of shade will cover the panels over the course of a year (see fig. 2 for an example front-end).

Besides the calculation application, our project will also need to encompass a variety of extra features which includes a secure user system which allows for account creation, logging in, and a project creation system that includes an address, the scale of the image imported, and the locations of each of the user marked obstructions and solar panels in the images.

To keep track of all this information accurately, we plan to use databases implemented through Google App Engine; this will store information related to the current user logged in, as well as the information stored on each of the projects created. This project will all be contained on a website that we will create and will eventually be directed to on Rooftop Solar’s website for them to license out to other companies later on.

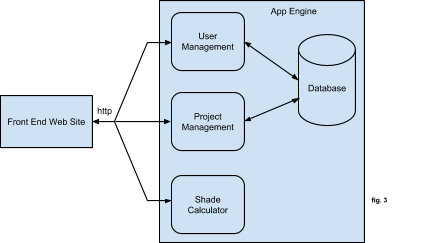
By creating this application, which will give an accurate estimate on how much shade will hit the panels throughout a year, solar companies will be able to give this information to potential clients/buyers to make an educated decision on if solar panel installation would be appropriate for their home. This would lead to a contract signing for the purchase and installation of solar panels, or save time in not having to send an employee on site to perform a shade estimate.

With future plans from Rooftop Solar to monetize and license out the application to other companies in the future, this application has the potential to save the time of solar companies around the globe in performing an accurate shade estimate, while also saving money and preventing accidental employee injury from climbing up on more rooftops.

In order to create an application that will achieve this successfully, a simple and effective architecture will need to be created, and our current plan for it will be explained in the next section.

**Architectural Overview**

Our approach to the architecture started with how the user was to go about obtaining the end result of the problem addressed by our solution. To reach this goal of the system, the system needs to have a concept of a project, reflecting what it means for a solar company employee to investigate a client’s site. This idea of a project needs to be associated with the user and stored for the user to access it again in the future. To mimic what is carried out when employees go on-site, the user will need to be able mark objects of interest and adjust variables according to the location and property at hand. Finally there is a basis and setup for making the desired calculation which is the end result that was sought after. Our diagram for this system can be seen in fig. 3.



### As seen in fig. 3, we broke the architecture down into five pieces corresponding to the different aspects of the system. The first part is the Front End Web Site which is where the application is accessed through and serves as a platform for user interaction in some of the other parts. The second part is the User Management which handles user sessions with login capabilities. The third part is the Project Management which is where the user may manage and edit existing projects or create new projects. The fourth part is the Shade Calculator which is where the processing occurs for calculations for a given project. The last part is the database where information for projects of users are stored.

### Front End Web Site

The Front End Web Site is as implied, the front end portion of our project. This will encompass functionalities of the web site seen by the user, and whatever process is initiated will be sent to the corresponding internal component that will handle the request, whether user or project management or information being sent to the shade calculator. For example, when the user visits the site they are prompted to login which is processed by the User Management component.

### User Management

User management will provide functionality regarding new users, logging into an existing user, and changing user details, such as a password. Modifications of users, or calls to check if the username and password match when logging in will call to the database and retrieve or change that information relating to that call. The result will then be sent back to the front end. For example: Once the user has authenticated their identity, a list of projects that they have saved is shown which brings us to the Project Management component.

### Project Management

Project management will provide functionality related to information that will be used within the shade calculator, including address, scale of images, location and relative heights of obstructions and solar panels as well as the Google Maps interface. Modifications or loadings of these projects will call to that specific user’s project database and the information relating to it will be used in the call, and a response will be sent back to the front end. For example: A list of projects belonging the user is displayed. The user has the option to create a new project or select an existing project, which case the user will be brought to the project manipulation screen where they may enter information like location and have access to the Google Maps interface for various markings and inputs pertaining to the different elements like obstructions mentioned above. The user has the ability to submit the project to be processed for determining the resulting estimate, this is done through the Shade Calculator component.

### Database

The back end of our system will encompass databases related to the user accounts, as well as the project information contained within them. This database uses entities and keys in order to have the ability for us to access various information related to a specific person’s account, including saved project listings with information related to each specific one, and account information such as name, password and company. This database will need to be accessed every time a new user is created, a user logs in, whenever a project listing needs to be shown, and whenever a project’s information needs to be loaded. These will also require security to ensure that they cannot be modified by an external source. For example: When the user first logs in, a list of projects that belongs to them needs to be pulled from the database. Once a project is selected, the specific parameters and data pertaining to that project must be retrieved from the database to be filled into fields, and used to generate the appropriate markups through the Google Maps interface. Finally, all this information for a project must be saved to the database when invoked by the user.

### 

### Shade Calculator

The Shade Calculator component will consist primarily of the algorithm to calculate the shade percentage for a given project; either loaded or created. The input will be the exact location of the property associated with the project (address or in longitude and latitude), as well as the information on the markups of obstructions and solar modules. The Shade Calculator component will then run this inputted data through the algorithm and output a breakdown of the shade percentage throughout the year. For example, a user might mark 2 trees as obstructions, and the output might show that a particular module has 75% sun access during june, 67% during december, and yearly percentage of 72%. We intend for this report to, at the very least, show a percentage for each month and a yearly total, with the possibility of giving a daily or even hourly breakdown if there is interest in such a feature.

As for the algorithm for calculating the sun access, we believe we can treat the problem as a simple raycasting problem. Using spherical geometry, for a given latitude/longitude, we can calculate the position of the sun at any time of day every day of the year. We will ‘shoot’ a ray from various points on the solar modules towards the sun, and if it intersects with a tree, then that part of the module is in shade. Repeating this process for several times throughout the day, every day of a year, we believe we will be able to provide an accurate estimate of sun access.

### Architectural Style

This architecture primarily takes from a client-server architecture, with the front end website communicating with information stored on another server, in this case the Google App Engine. Information from our front end web site will be sent to the component programmed to handle that information and a response will be sent back and updated on the site. The architecture heavily uses communications with databases in order to receive its information and calculate its results.

## Module and Interface Descriptions

There are two primary modules that complete the role of the system for the user. The first module is the Accounts Module which is necessary to provide a means of controllable access to the application as well as a sense of persistency of users with the application. The second module is the Project Modification Module which is necessary for the task of manipulating the variable aspects of projects, giving the user the environment to do this.

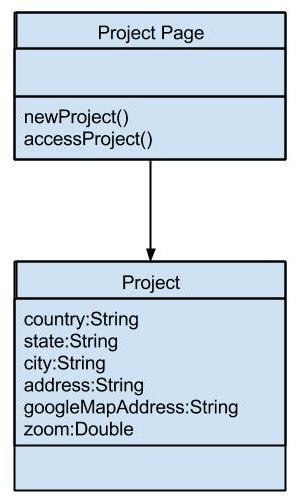
### Accounts Module

The Accounts Module, as the name suggests, deals with accounts. Accounts need to be created with credentials, namely a username and password. Google App Engine provides support for users. It gives three methods for authenticating users; Google Accounts, accounts on a particular Google Apps domain, and OpenID identifiers.

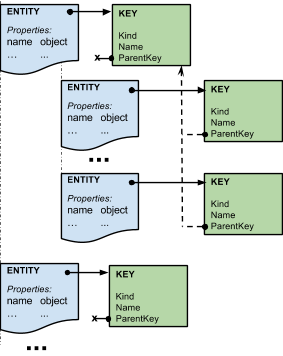
Support for OpenID identifiers is currently experimental and having our users sign up for a new Google Apps account would be a bit of a hassle for the user. The final option then, is to have users sign in with their own existing Google Account. The major caveat of this approach is that it allows anyone with a Google Account sign in to our application. To counteract this issue, we will generate a unique ‘token’ for each user that has access to the product (ie, they purchased a license).

This method will give us the ability to restrict access only to authenticated users, while giving all the benefits of Google’s accounts system. The module will work in conjunction with the project module to ensure that users can create and edit projects.

### Project Modification Module

The project modification module is the bulk of the front end of the application. This is the part of the application where the user will be able to view and modify a project. The user will input data using the embedded Google Maps and some simple html forms. Then the user can save the project and request a calculation based on the input parameters. Both of these operations will be handled with AJAX calls, that will construct a JSON object with the necessary data to send to the server. The public API for the server will be discussed in the specification for the server-side modules.

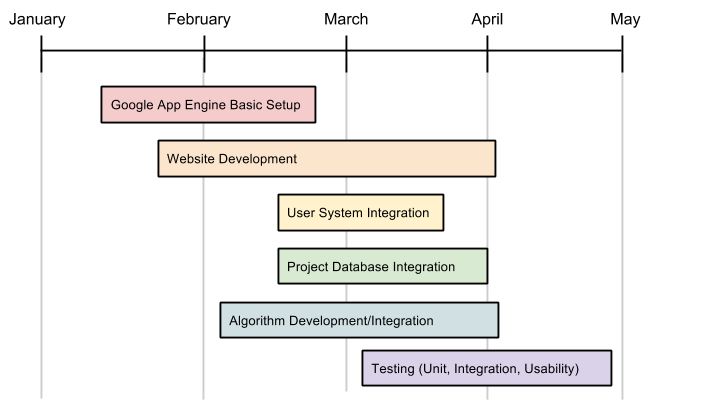
The shade calculation module is responsible for accepting the user inputted data and calculating the shade percentage of a given project. It has a single public API call exposed through the App Engine webapp2 framework. This will consist of a simple POST request directed to /api/calculate with all the necessary JSON data (‘necessary’ data to be defined later). Once the calculations have been performed, the module will write the calculated value(s) back to the client, which will be responsible for displaying the information.

Google App Engine has a database service, called DatastoreService. It is a hierarchal structure similar to a file system. The primary component is the Entity, which has a property storage similar to a hash map (String names map to Objects). Each Entity is distinguished by a unique Key. Keys and Entities are closely tied together. Keys are designated via two Strings (a kind and a name) and a parent Key. The kind can be compared to file type. A null parent Key would mean that it will be located at the root level of the database. A non null parent key would place the Entity at the level “under” the parent Entity (the Entity associated with the parent Key). Key names must be unique among other Keys of the same kind and parent. Keys are created through the KeyFactory. To query the database, a Query is created with a kind (String) and a parent Key. A Query allows us to obtain a list of Entities of the specified kind, found at a place within the database according to the specified parent Key. Queries also can have sorting and filtering features applied to them (i.e. according to Entity property fields).

One approach to storing user projects is to have Entities stored at the root level, one per user, of kind “Account” and name reflecting users’ name. Projects will be represented as Entities with associated Keys of kind “Project” and name reflecting the project’s name, as well as a parent Key of kind “Account” with name associated with the current user. Projects then are created and accessed by first having the Key associated with the “Account” Entity (given the current user). This will be our parent Key, which then is used to create another Key with kind “Project” and name according to the project’s name. This new Key is then used to retrieve the project Entity that will have properties according to the project.

## Implementation Plan

In order to make progress and complete this project on schedule, we have a simplified implementation plan to follow to help us stay on schedule and finish up with the functionality of the project by the beginning of May. This will include features such as the user database, with the project database integration, the front end website, and the shade calculator itself. A graphic representation of this schedule will be as follows:



**Google App Engine Basic Setup**

The implementation of our project will be done in several different sections, each relating to a different piece of functionality in our system. Since the Google App Engine is a major part of our back end, in keeping track of the user’s account information as well as their projects, fully understanding and implementing basic prototypes to ensuring this works as we intend it to is a must.

**Website Development**

Since users will access this application through a website, the front end development will be done throughout the semester, beginning with basic integration with the Google App Engine. As more of the interfaces are developed, they will be integrated and accessed through the website. Even once all the functionality is done, there is still the need for an appealing user interface that looks professional and is easy to understand.

**User System Integration and Project Database Integration**

With a better understanding of our back end, Google App Engine, at this point we will be ready to integrate a user system that will be able to easily integrate with our website, and be able to interact with other components implemented in the future. This will include account creation, logging in and out, and account detail modifications. Once this system is fully in place, project database integration can start, in allowing for each user to create new projects, delete projects and edit existing projects.

**User System Security**

With the desire of Rooftop Solar to monetize this application in the future, there needs to be sufficient security and protection of the user’s data and information in order to ensure the product and company retain their reliability. Therefore, as work is being done on other components, there will need to be sufficient implementation and testing of the security of the system. This not only includes database security, but application security as well. This is to ensure that the results stay accurate, and so the application is not prone to being stolen.

**Algorithm Development/Integration**

Since the majority of our project relates to the development of a web application that will take in data from Google Maps, and run it through an algorithm to calculate a shade percentage, the development of this will also run until the end of the semester, beginning with taking in an address from Google Maps and scaling the image, marking obstructions and solar panels, then finally ensuring the correct data inputted will be sent to the algorithm correctly to produce a correct result. This development requires a lot of error checking for false input as well as for accuracy so this section will likely require the most testing as well as the most work.

By following this current implementation plan, we hope to integrate our components in an order that allows for testing each component as they are developed, and allows for bug fixing and test plans to be made as they are needed.

## Conclusion

Currently in the solar industry, there is a severe inefficiency in the amount of work is being done on solar estimations, as workers are being sent out to countless rooftops of potential contracts to make shade estimations, taking not only a good amount of company time, but company resources and money as well.

With our design of a web application that will have the ability to make accurate shade estimations without sending people on site, while also saving their company accounts, and project information, this application has the potential to be an efficient way to give customers a quick estimate of just how much sunlight will hit their roof over the course of a year, speeding up the formation of contracts as well as general efficiency in the company.

By following the implementation plan relating to creating an simple yet professional front-end website, a secure user and project database with an accurate shade calculator, and accurate integration with the Google App Engine, the project will be a viable alternative for any company authorized to use it to make accurate solar shade estimates without wasting potentially important and needed company time and resources, and making their contracts with customers more efficiently.