Requirements Document

Prepared for COSC 499

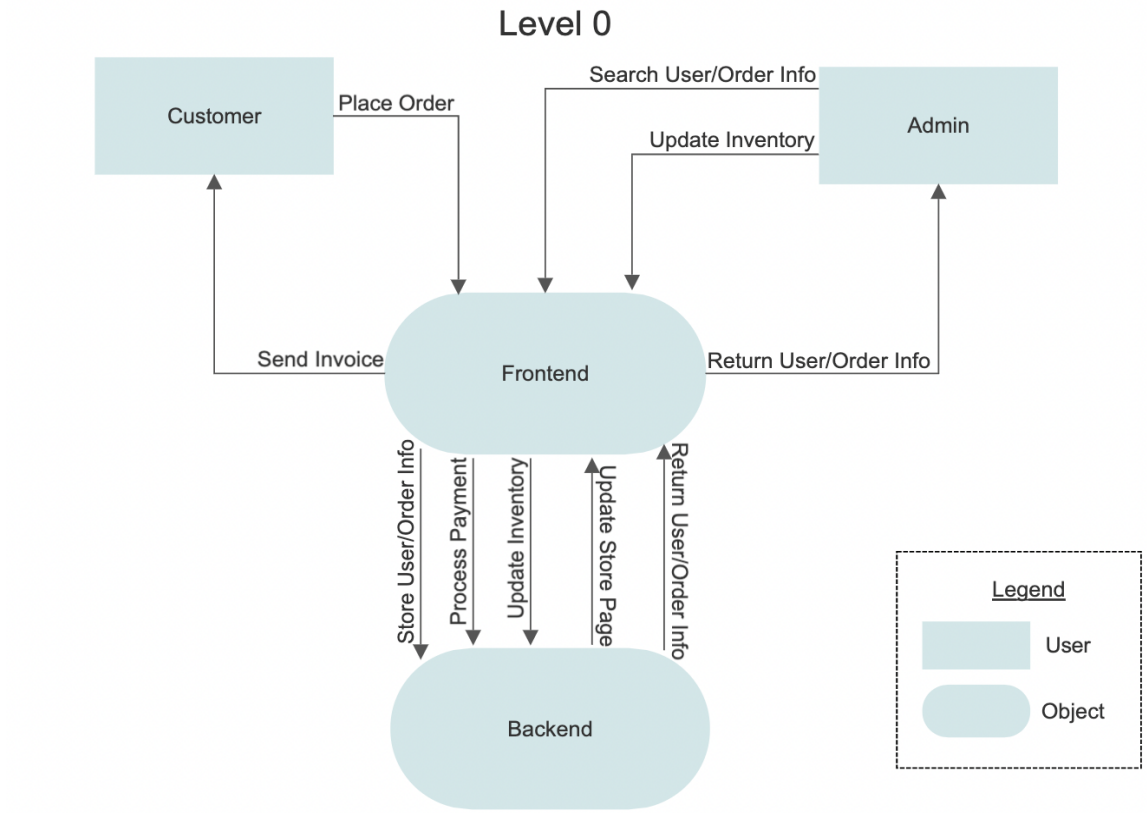
Created by Camp-OAC: Group B

High-Level Description

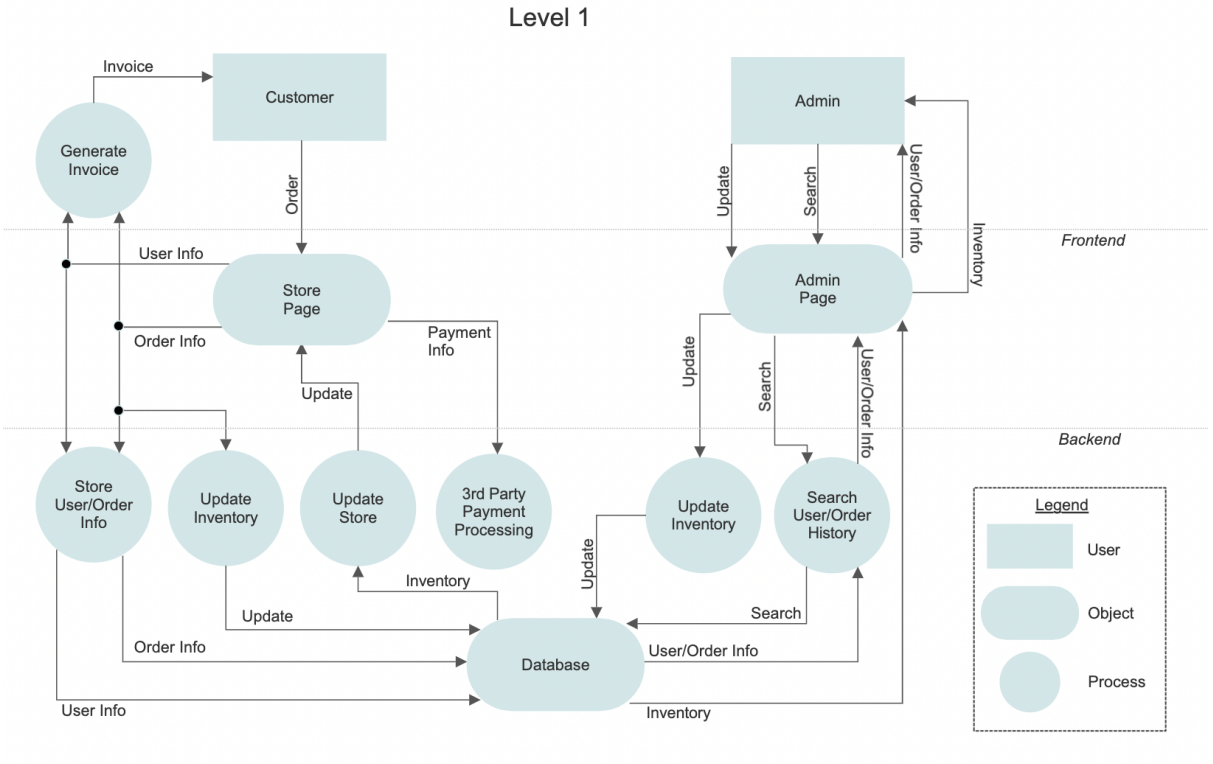
We are building a mobile-friendly website to automate the process of obtaining campfire wood for the general public. As a regular user (not an administrator) the website will consist of choosing a pickup location (from a list of available locations maintained by our database) and the desired number of bags of firewood. Then, they will automatically receive an invoice by email that is produced based on pricing and quantity ordered. The website then makes payment OR retains an invoice for cash payment upon pick up. The receipt is generated automatically. Users groups include local residents as well as tourists, most people who need a smaller quantity of wood. We anticipate that for larger orders people would be more likely to turn to a bulk supplier.

Administrative users, including designated CampOAC staff / Rotary club members in charge of the fundraising campaign, can access the website differently. In the admin view, they can access and manipulate certain aspects of the database, including order history, inventory amounts, and pick-up locations.

DFD



The level 0 DFD shows the most simplified dataflow version through the website. From this level, we can see that users will place an order on the site, and the website will then generate and send them an invoice. Admin will be able to access an admin page through which they can view order histories or view and update the inventory across locations.



This level 1 DFD provides further detail to the level 0 diagram. We can see that the front end consists of a store page and an admin page, while the back end includes a database for user and order info and inventory and a third-party payment processor. When a user places an order, the site will generate and email them an invoice, the order info is stored in the database, the payment processor securely handles the payment info, and the inventory is updated. If inventory runs out at any location, the site will update to not display that location for following orders. Through the admin page, the admin can display order histories from the user database and view and update the inventory database. If an admin restocks a location that had run out, the site will update to display this location again.

Functional requirements

The user (customer) must be able to place an order by selecting a number of bags of firewood and a location for pickup. The number of bags users can input will be capped at the available inventory at a given location. Users will not be able to select locations with no remaining inventory. The system must store inventory for each location and update inventory automatically with each order. Admins must be able to view inventory and update inventory manually. When an order is placed, the system must automatically generate an invoice and email it to the customer. The system must store order history and user information. Admins must be able to view user info and order histories, and admins must be able to sort orders by the customer.

Non-functional requirements

The URL link must work on mobile and desktop. Page visuals and usability must function on mobile or desktop. The website must support the most popular browsers (i.e. Chrome, Safari, Firefox, Explorer). Site visuals must adhere to client organization (Rotary Kelowna) graphic design standards. The website must support multiple concurrent users. The system must use Square credit card processing. Customer credit card information must be encrypted and should not be stored. User info must be encrypted, and only admins should be able to view user info. Admins will require a username and password to log in.

Milestones

Milestone 1: Submitting requirements report (this document).

Milestone 2 (Peer Testing 1): Mobile-friendly site is hosted, functional backend, but not used for anything other than proof of concept (proof that the frontend is able to communicate with the backend) at this stage. Design at this point will be subject to change.

Milestone 3 (Peer Testing 2): The design of the site is fully fleshed out based on graphic standards provided by Rotary International. Full back-end integration, including a fully-fledged database schema. Purchase form and process implemented (no actual payment/order fulfilment)

Final deliverable: All previous milestone goals, integrating Square payment processing and invoice generation fully-fledged.

Tech Stack

Because the client had no specific preference, we gave ourselves a few different options for the tech stack. After our research, we decided to use the MERN stack (MongoDB, Express, React, NodeJS). We made this decision for multiple reasons. Firstly, because this stack allows us to use primarily javascript for the majority of development, this will help us unify all of our development flows under one language, and javascript is great for readability and prototyping. Secondly, MongoDB's NoSQL database allows us and the client to quickly understand what is happening in our database at any time, compared to a relational or SQL-based alternative where this would be a difficult task (for example: quickly prototyping database returns). Using express and nodeJS to serve the front end and communicate with the database will ensure that we have very low overhead as far as performance is concerned. This

is important when using our hosting solution. DigitalOcean, as the hardware is not precisely ‘powerful,’ requires us to squeeze every bit we can get out of our web server. Finally, with React on the front end, we can easily make the website mobile-friendly, taking a mobile-first approach to our design. Not to mention React makes it easy to add animations and functionality, which would require further overhead in other stack solutions.

We considered using two other options: MEAN and LAMP. The MEAN stack is the same as the stack we decided on, with the only difference being switching React for Angular. Angular is a front-end framework with a heavier bundle size than react. We opted for MERN because React is a javascript library, meaning it has a more negligible performance overhead and gives us a smaller overall package footprint. LAMP (Linux, Apache, MySQL, PHP/Python/Perl) and its variants were also an option. We chose not to use any of these because we feel they are too robust for our needs. In order to make prototypes of features/designs faster, we chose to use something that primarily uses javascript, which we can quickly program, test, and deploy. Alternatively, with the LAMP stack, if we wanted to add a new feature, our team would have to learn advanced HTML/CSS and be fluent in PHP for interactivity - not to mention being able to create secure and efficient SQL statements confidentially. While the LAMP stack is very good at creating robust and flexible web applications, we feel that when we consider development time, server upkeep, and security profiling, the cost outweighs the benefits.

Testing

The stack we are using gives us lots of flexibility for testing based on the availability of testing libraries for JS. We plan on using Jest as our testing framework. On the front end, we can use chrome's developer tools and run regression tests with different browsers (including

mobile browsers) to ensure nothing breaks, and if something does, we can isolate that incident. On the backend, we can utilize unit tests to ensure certain aspects of our API work. We will use postman, a REST client, to ensure our API is secure and cannot be spoofed by hackers who would steal information.

We will be using GitHub actions to automate our continuous integration. However, we are subject to change depending on what our codebase looks like at certain milestones.

Feedback / Questions

Q. Since this website involves selling and picking up products, it is essential for users to know if there is stock available. What is the delay time if there is one? How do you plan to update the stock in real time?

A. We will use asynchronous calls to our database to update real-time stock. At the moment of release to the public, the stock will be based on what is available - from that moment onwards, all orders will reduce stock. Admins can go into the database anytime from admin control to make changes as necessary.

Q. They only mention credit payments. Will debit payment be available as well?

A. As our client has requested to use Square, which only handles credit cards, that will be our primary focus. However, we will examine the feasibility of implementing Interac e-transfer support, time permitting.

Q. Will firebase not be better than mongo DB? It probably may speed up the dev process as using a non-relational database will handle concurrent use of the data points internally

A. Firebase would be a better option if we only used more of their features. Having plenty of things on the toolchain is great. We could handle payment processing, the

database, and hosting on the same platform. To fit our client's needs, we will not be using any more of the features that Firebase provides.

All other questions were considered and accounted for in the final draft of this document.