# Camp OAC - C Requirements Report

Zach Prenovost, Aidan Murphy, Cam Wilson, Karlen Speiser

## Software Description

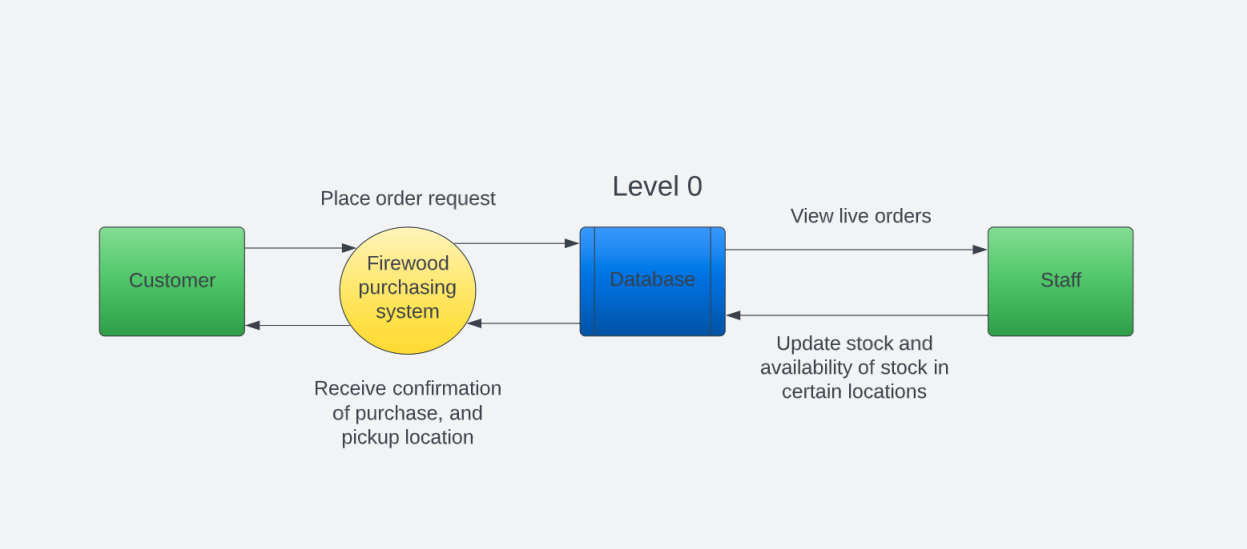
The product will be a website that allows customers to purchase firewood from Camp OAC and receive a pick-up location after the order has been processed. The firewood stock will be automatically updated when an order is placed. The objective is to automate the process which has been performed manually in the past. The website will have two buttons on the homepage, one of which leads to the firewood ordering page, while the other leads to an administrator portal. The firewood ordering page is where the customer fills in their contact information, the amount of firewood they would like, and the preferred pick-up location. Next, the page contents change to inquire about what type of payment method the customer would prefer, and options for Credit Card, E-transfer, and Cash are available. After this is done, and the purchase is confirmed the exact pick-up location will be given to them. If the customer selects cash/e-transfer payment there will be an additional step where staff contacts the customer to confirm the payment is processed. The admin/staff can view the order on the administrator portal, where they can approve cash/e-transfer orders, and prepare the order for pick-up.

There are two user groups for this software:

- ➔ Customers: Will navigate the website, and enter payment details into the system to purchase firewood, ideally, there is no login for customers, only entering contact details to receive the payment confirmation and pickup location.
- ➔ Admin/staff: Will need to login into the admin portal, staff will be able to manually adjust stock levels, and choose which locations will be selling firewood at any given time. Staff will also be able to approve e-transfer and cash orders.

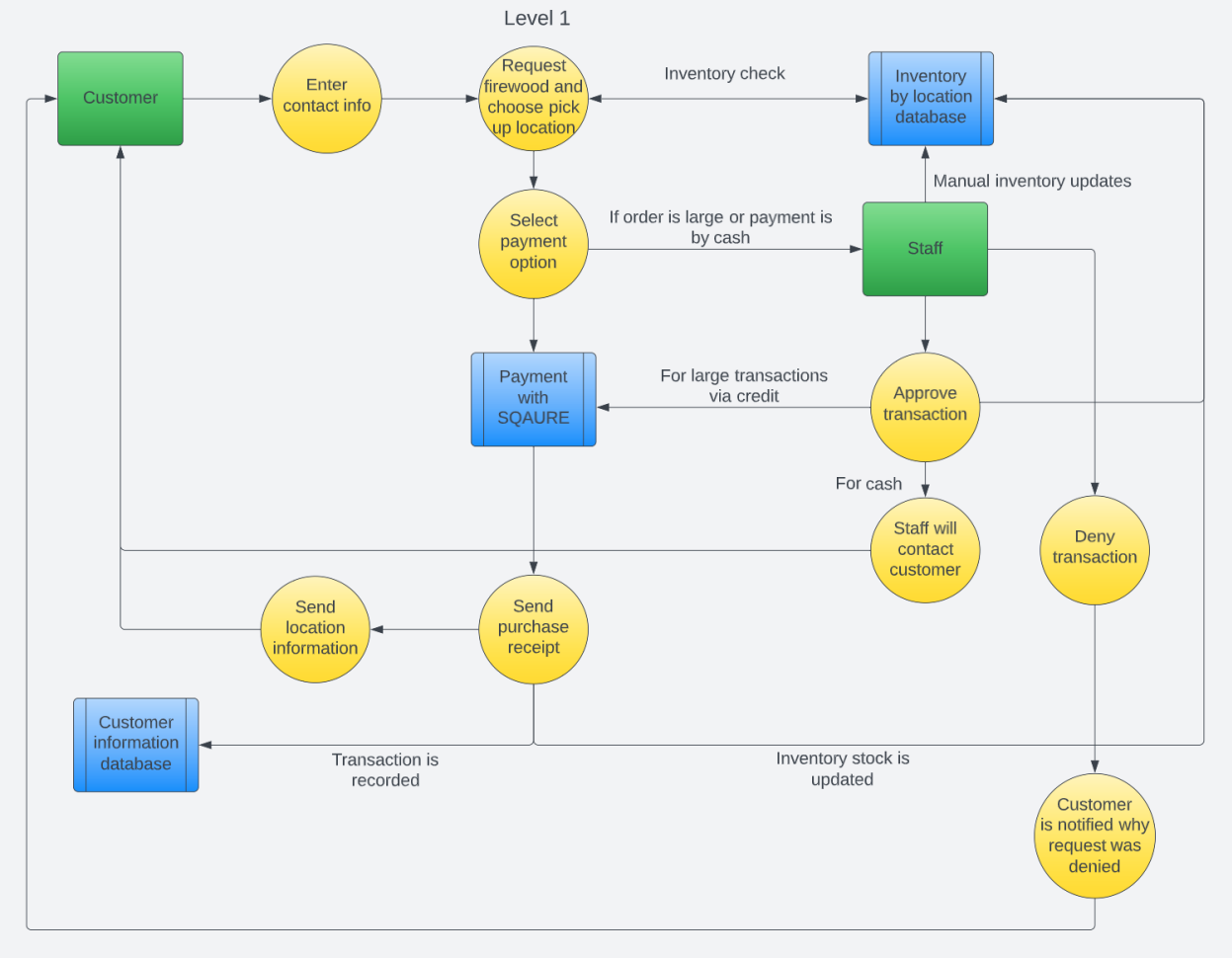
## System Architecture (DFD)

*Level 0 DFD*



The level 0 DFD depicts our system as a single process and the connections from the customer and administrators to the system. This process begins with the customer placing an order request with our firewood purchasing system. This request will be tracked through our database and staff will be able to view live orders. Staff will have permission to update stock and approve orders. When orders have been confirmed the customer will receive a receipt with pick-up location information attached.

*Level 1 DFD*




The level 1 DFD shows the main functions of our system in a more in-depth form. Customers will input contact information, a quantity of firewood, and select their preferred pick-up location. There will then be an inventory check before orders are approved to ensure quantity is available. Next, the customer can choose their payment option: if they want to pay with debit or credit, their transaction will be processed using Square. If the case is the customer wants to pay by cash/e-transfer then this will require staff verification, staff will have to contact the customer to arrange the payment. Once payment has been approved the customer will receive a purchase receipt with location information attached. We will then log the transaction information in our customer information database.

## Milestones and Functional Requirements

*Milestone 2 Peer testing I:* Customers will be able to input all necessary order information. Customers will be able to view the payment options available. Also, we will complete the front-end development for the website as well as the generation of receipts.

*Milestone 3 Peer Testing II:* Our database will be implemented, allowing for inventory checks to take place and for automatic stock updates. The database will also store order transaction information. Payment with SQUARE will be integrated.

*Milestone 4 Finished Product:* The administrator portal will be created allowing staff to log in and manage orders, stock, and payments on our website. Notifications will be created to notify vendors when there are cash/ e-transfer payments.

## Non-Functional Requirements

- ➔ Usability: With our user group having such a wide range, the web application needs to be easy to learn and use. We will do this using React.js, optimizing UX by minimizing actions needed and increasing efficiency using the frameworks' functions.

- ➔ Scalability: Using MongoDB's NoSQL style database, the database will be very flexible and horizontally scalable. This will smoother changes if they are needed in the future.

- ➔ Error-handling: UI will be programmed such that it is difficult to make invalid actions. The order of operations should be clear to the user and error prevention will be implemented.

## Tech Stack

The client was flexible in the choice of tech stack. We decided to look into three different tech stacks to help us achieve our goals: MEAN, LAMP, and MERN.

| Stack | Database | Middleware | FrontEnd | BackEnd |
|---|---|---|---|---|
| MEAN | MongoDB | ExpressJS | Angular | Node.js |
| LAMP | MySQL | Apache | Linux | PHP |
| MERN | MongoDB | ExpressJS | React | Node.js |

Since our goal is to build a responsive and agile web application, we decided to focus our research on the MEAN and MERN stacks. Below is the information and the frameworks/libraries used in each stack. We included a few CSS toolkits in case it is needed for future use.

***Front-End:***

- JavaScript
- CSS
- HTML

Possible JavaScript frameworks:

Angular

Pros:

- ➔ Two-way data-binding: Data binding is straightforward in Angular. It ensures that any changes made to the view/presentation(HTML) layer immediately update the project data model.
- ➔ Open source: available for anyone to incorporate into a project.
- ➔ Easy to test: Allows for simplified end-to-end testing, simplifying tests and debugging.
- ➔ Improved server performance: reduces the burden on server CPUs because AngularJS supports caching and other useful processes.

Cons:

- ➔ JavaScript support mandatory: Users would need a JavaScript-enabled device to access the website.
- ➔ Difficult to learn: AngularJS has minimal documentation, with complex scopes that may prove challenging to understand.
- ➔ Large size: Angular's file size is much larger than React or Vue.

React

Pros:

- ➔ Can create cross-platform applications: Can easily create web apps, can also use React-Native which is used to develop apps for iOS and/or Android devices in React using the same syntax.
- ➔ Open source and popular:  Free to use and has a large user base, which is great for troubleshooting problems.
- ➔ Reusable components: Let developers reuse pieces of UI in any project and combine them into a larger UI system.
- ➔ Backward compatibility: React API remains unchanged after updates. Allowing for prolonged functionality of React without requiring much code maintenance.

Cons:

- ➔ Mixed views by default: Unlike AngularJS there isn't a separate component logic and view. Instead, the component contains a function which returns JSX. (syntax extension)
- ➔ Based on 3rd party libraries:  React itself doesn't have many base tools, it relies on 3rd party libraries which can sometimes cause version compatibility issues.

Vue.js

Pros:

- ➔ Very lightweight: The framework itself is very small and simple, improving user experience and search engine optimization (if needed).

- ➔ Virtual DOM: by using a virtual Document Object Model, Vue optimizes application performance by reducing DOM update times.
- ➔ Two-way data binding: Similar to AngularJS, Vue uses two-way data binding to enable data exchange between the DOM view and data model.
- ➔ Compatibility with mobile devices: Vue can be used for building apps for iOS and Android using NativeScript, Ionic, or other UI frameworks.
- ➔ Compatible with Electron framework for desktop development.

Cons:

- ➔ Limited resources: Not as many available libraries to work with compared to AngularJS.
- ➔ Lack of support for large-scale projects: Vue still has a relatively small dev team and community, so it's better to use it in smaller projects.
- ➔ Not fully stable: 90% of API remains the same after updates.

Possible HTML and CSS toolkits:

<u>Bootstrap</u>

Pros:

- ➔ Consistent framework and the majority of browsers: Bootstrap remains open source and is compatible with the most commonly used browsers.
- ➔ Responsive structures and styles: Responsive design lets websites ‘adapt’ to different screen sizes without compromising usability and UX.
- ➔ Accelerated testing: Bootstrap allows developers to easily test prototypes without having to deal with compatibility issues.
- ➔ Massive ecosystem: Offers the most in terms of layouts and UI elements, also has a large and active community.

Cons:

- ➔ Can require many style overrides: Can cause redundancy in CSS elements causing performance loss.
- ➔ Require extra customization: Many websites using bootstrap with heavy customization look the same.

<u>Foundation</u>

Pros:

- ➔ Better customization: Foundation offers a better customization function that allows most projects using Foundation to have a unique look.
- ➔ Great browser support: Like Bootstrap, Foundation is compatible with Chrome, Safari, Firefox, and Opera. Foundation supports newer versions of Internet Explorer than Bootstrap.
- ➔ Flexible grid system: Foundation offers more grid-related features when it comes to managing a grid system.

Cons:

- ➔ Learning curve: It will take more time for team members to familiarize themselves with the framework.

- ➔ Lack of support: Due to Bootstraps overwhelming usage, its support on Twitter and for troubleshooting in general is much better than Foundation's.

Bulma
Pros:
- ➔ Easy to use: Bulma's modular design and great customization options make it a commonly used toolkit.
- ➔ Easy to learn: Bulma is designed to be easily navigable, to make the learning process as quick as possible.
- ➔ Browser compatibility: Websites designed using Bulma are compatible with most major browsers, making it great for testing.
- ➔ Framework is lightweight.

Cons:
- ➔ New toolkit: Bulma's final version is still yet to release, so its community and documentation may not be as comprehensive as its competition.
- ➔ CSS Only: Bulma does not include inbuilt JavaScript or jQuery like Bootstrap does.

## ***Back-End:***

We chose JavaScript for the backend, Why?
Because we are already using it for frontend and can use it for backend. JavaScript has the ability to be employed on the frontend and backend. By using JavaScript on both ends, the overall flexibility and efficiency of the project can be increased.

JavaScript Frameworks:

Node.js
Is not a framework, but a runtime environment. Needs a framework.
Pros:
- ➔ Open source: Free and available to anyone who needs an environment to run JavaScript.
- ➔ Fast-processing: Uses Chrome's V8 JavaScript engine, resulting in very fast execution times.
- ➔ Event-based model: Due to node.js's asynchronous, non-blocking, single-threaded nature, node.js is great for web apps that have constant updates.

Cons:
- ➔ Single threaded: Since node is single threaded, CPU heavy tasks tend to decrease performance.(Multi-threading it remains experimental)
- ➔ Callbacks: Node.js uses callbacks when executing multiple tasks, which can become complex affecting performance if not managed properly.

Express.js
Pros:
- ➔ Easy to learn: Only uses JavaScript, which is necessary for the front end anyway.

- ➔ Supports Node.js.
- ➔ Supported by Google V8 engine: High performance using a new(er) engine from Google.
- ➔ Supports caching: Allows webpages to load even faster for people who have visited before.

Cons:

- ➔ Can have callback issues.

Web servers:

- ➔ Will figure it out at a later date. When testing becomes applicable. We will run a local server for testing, and look into web server options when we reach that point in the project.

### ***Databases:***

SQL vs NoSQL

SQL: Structured Query Language

- ➔ Used for relational databases like MySQL
- ➔ Very strict requirements for data (Clear schema)
- ➔ All records must follow this Schema
- ➔ Uses relations like one-one, one-to-many, etc.

Data is stored across multiple tables that are connected with relations and queried with SQL

NoSQL: Different than SQL

MongoDB

- ➔ Collections instead of tables
- ➔ Documents instead of rows
- ➔ Documents don't have a strict schema (different structure in each one)
- ➔ The advantage is super flexible in case of shifting requirements
- ➔ No relations (relational data needs to be merged manually)
- ➔ Collections should contain all necessary data for querying, no joins possible
- ➔ Duplicate data would be an issue, as would need to write same info multiple times
- ➔ Collections can be merged

We need to decide what tables/fields we need. Whether we can keep it in one collection or multiple*

- ➔ If one: then MongoDB may be best
- ➔ If multiple: then SQL could be better
- ➔ If requirements change or we want flexibility in what is stored. Mongo is the best.
- ➔ If requirements are fairly predictable SQL would be fine

Scaling: Harder for SQL, way easier for NoSQL. Mongo could have some better longevity after we have left if the database continues to grow.

MongoDB EXAMPLE:
How would this work with our data?
1. Orders collection
   - ➔ Customer info;
   - ➔ Order id;
   - ➔ Location id;
   - ➔ Quantity ordered;
   - ➔ Etc.

2. Location collection
   - ➔ Stock of wood;
   - ➔ Location id;

3. Customer collection: (optional / likely not recommended)
   - ➔ When an order is placed check if customerID exists and if NULL then ADD
   - ➔ Could be redundant if possible to query by customer id from orders and remove duplicates

When a user makes an order we query a location id from the location collection that has the stock of wood available, then we update location id with new stock. We also separately update the Orders Collection and add a new order with the order information.

### *Summary:*

After considering all the options above, the MERN stack proves to be the most suitable tech stack. The MERN and MEAN stack both offer advantages when it comes to building a robust web application. When comparing these stacks to the LAMP stack, both offer highly responsive web applications that are much better at dealing with smaller amounts of non-relational data. Furthermore, both MEAN and MERN allow us to use JavaScript, an extremely powerful web scripting language with a variety of open-source libraries. The differences between MEAN and MERN are slim, mainly concerning the Front-End library. Between Angular and React, we decided React was the appropriate library, mainly due to the virtual DOM(Document Object Model) to speed up the performance, and reusable software components for smoother development, it also has less of a learning curve compared to Angular with a fast-growing community.

*Therefore we will be using MERN: MongoDB, ExpressJS, ReactJS, and Node.js.*

### *Testing:*

Testing will consist of:
- ➔ unit testing
- ➔ regression testing
- ➔ integration testing (where appropriate)

All unit tests should be written before and after development on the relevant feature, and retained throughout the project's development.

We plan to use the JEST framework for unit testing, this is a framework developed by Facebook and is widely used, making it easier to resolve issues and create tests. It is easy to download and install, and easy to learn to use. Click (https://jestjs.io/docs/getting-started) for basic info on how to use jest to test JavaScript functions. It also makes the test-all testing procedure easy to perform.

For regression testing we will be using the test-all procedure: we will run all previous tests whenever integrating a new feature. The scale of this project is small, so this is an achievable and sensible regression testing method. Again jest allows the running of multiple tests in isolated environments, making testing efficient and fast.

Integration testing will ensure the database and website communicate as intended, for stock checks and staff login.

**Group Evaluation Questions:**

1. Will there be a map of all the locations for pickup? Is there a plan for delivery services?

- *No, we will not have a map of the precise pick-up locations because some of these locations are personal addresses and should not be available to anyone who visits the website. We want to conceal the pick-up location until payment has been confirmed. There will be no delivery service.*

2. How do you plan to make sure your transactions are secure and there's no data loss? If the feature of multiple transactions is supported, how many transactions can be made at a time?

- *The security of our debit/ credit transactions should be very secure since we are using SQUARE. No credit card information will be stored by us, square handles all of that data. As for data storage and collection, we will have in-depth testing with MongoDB to ensure transaction records are properly stored. Multiple transactions in one order will not be supported. The quantity of wood will be specified upon order and will need to be picked up at the same location.*

3. Will there be a login for repeat customers to prevent frustration from continually inputting the same information?

- *Customers will not be required or asked to create an account. They are only required to input an email, each time. However, returning customers' info will be stored in the database to ensure priority goes to them. There is only one input, email, and it is only entered every time someone needs firewood, so this should not cause any negative UX.*

4. Will you be mocking data to ensure frontend functionality works until the database is integrated in milestone 3? What do you mean by a flexible database? Please elaborate

- *Yes, we will test our database using mock data to ensure functionality across the entire project. Our database will not be fully functional until milestone three, but we will still be able to test functionally on the database before milestone three because there will be minimal integration with the back and front end. MongoDB is a non-relational document database, which allows for the storage of unstructured data. Compared to a structured database, MongoDB allows for very simple storage of varied data types making it very flexible.*

5. Will there be a feature implemented that reminds the user of their pick-up time and location when the appointment is coming up soon?

- *This is a great idea and something we can consider if extra time permits. We expect customers to pick up firewood within a day or two of their purchase, but a text/ email reminder feature would be useful.*

6. How are you going to keep it low maintenance?

- *We are keeping the website simple, it uses one major language (JavaScript) for the majority of the code. The administrator portal will include a UI that allows the client to change what they need to change on the site which may include the price of wood, qty of wood, access to customer information etc… Our frameworks have backward compatibility so updates should not break the code. And should the admins of the website need to maintain it, the developers they hire will only need to know JavaScript and how to work with MongoDB.*

7. How will refunding processes occur?

- *The staff will be responsible for processing refunds correctly, they will need to find the purchase record and select refund to correct the stock in the database. For square purchases, the cost of the firewood will be given back to the customer, Square has built-in functionality for this. For cash purchases, staff will need to sort that out directly with the customer, and the same for e-transfers*

8. How are you hosting the app? What are the tech stack and hosting costs?

- *The tech stack is open-source and free to develop. Costs will be related primarily to web hosting. We will host the app locally(docker) until milestone 3 after which we will pay for a server for the website to be hosted on. The selection of which hosting service to use will come at a later time when more of our project details are actualized. We do not have a budget set but aim to keep the costs under $20 a month when non-local hosting begins.*

9. Nice description of what the idea is and what the Idea looks like. Will there be any protection against fraudulent requests? What should the user see on their end, any example designs? How will your website be hosted? Is there some provider that will be chosen? Good overall timeline, providing what goals you have in the future. Do you think that there is room to expand once you have completed your project? Great description of tech stack.

- *We do need to consider security measures to provide protection from basic attacks on the website. For testing, we will use docker to host the website, and later on, we will decide on a permanent hosting option. The website design leaves space for further expansion on the website, but it is not in our plans to go beyond the scope the client has asked for; we have a lot of work ahead of us.*