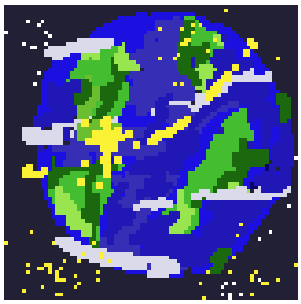
HelloWorldByMe

Final Report



April 30th, 2025

Project Sponsor:

Kevin Daily

Faculty Mentor(s):

Brian Donnelly, Savannah Chappus

Team Members:

Elizabeth Knight

Joey Banaszak

Jessica Maldonado Olivas

Samantha Madderom

Table of Contents

Introduction

After the COVID-19 pandemic, there has been an increasingly prominent issue in the city of Tucson, where there is not a lack of resources meant to alleviate homelessness situations, but a lack of coordination in providing these resources. Local service providers are known to work extremely hard, but this leads to navigators and service staff handling high caseloads while also managing fragmented information across multiple kinds of systems, vague updates across the city from other service providers, and memory. The overcongestion can lead to delays, inefficiencies, and allow clients to be overlooked by the current system. Our capstone project addresses this challenge by developing a secure, centralized platform that allows organizations to come together to support clients, track sessions, and coordinate services in real-time.

Our system was designed to be used by navigators, volunteers who interact with individuals in need, and to simplify the service provision process by providing an intuitive interface for organizing interactions, additional notes, and filtered searches. The software is meant to be deployed within a network of organizations that support men, women, and children facing a lack of shelter, food, mental health counseling, and substance abuse recovery, allowing for staff across the Valley to provide more responsive, coordinated care.

With the client's past experiences helping the Tucson community, he brought his new idea to us that involved a streamlined communication system that would better impact the individuals in need, service providers, and local governments. His idea introduced new needs such as improved data tracking, efficient interorganizational communication, and real-time client documentation. To meet these needs, we developed a full-stack solution that utilizes React, Node.js, PostgreSQL, AWS services, and Django microservices. Besides having reliability with technical aspects, our project demonstrates careful consideration of user experience, emphasizing visibility, usability, and efficiency that will align with the daily realities of nonprofit service work.

The WorldByMe Role-Based Access System doesn't just put necessary forms and contacts online, but provides a secure framework for increasing efficiency and consistency, supporting nonprofit service providers. Our work in the following sections will demonstrate how technology can help build trust with organizations, reduce burnout within the current workflows, and enhance the many missions of organizations serving our communities.

Process Overview

When developing our project, a collaborative and iterative process inspired by Agile principles was used. Regular weekly meetings were essential where we planned tasks, shared any progress that was made, and adjusted priorities based on any technical difficulties and client feedback. While we aimed to have a maintainable pace with our progress, we adapted our workflow when needed to accommodate any changes and feedback from our client.

Git and GitHub were implemented for version control, with a shared, open-source approach implemented using the GitHub Organization feature. Each component had its repository or branch while the team collaborated through pull requests and shared tasks. Our frontend tech stack consisted of React and the Leaflet JavaScript library, while the backend utilized Node.js, PostgreSQL, and Django. As more difficulties occurred concerning our architecture, we shifted from a monolithic approach to microservices, with our deployment being done through AWS.

For task management, Discord and our task reports were used, making sure that there was a shared space that tracked planned items, in-progress work, and completed tasks. As tasks were being distributed, each member of the team took on different responsibilities in our WorldByMe role-based system. Roles shifted slightly depending on availability, and who had the most expertise with certain tasks

- **Elizabeth** led the team, along with keeping the client in the loop with progress, and focused on the navigation system development. She also led the application integration process and integrated the navigation system with backend endpoints**.**
- **Joey** focused on the database schema and ensuring the database components were compatible. Additionally, he implemented role-based access control within the database.
- **Jessica** was in charge of the frontend development and specialized in the server-side logic, including authentication and organization interaction. She was also responsible for the GitHub repositories as a release manager.
- **Samantha** contributed to numerous kinds of documentation and developed the messaging system and its functionalities. She was also responsible for integrating the messaging system into the application.

In the early stages of our project, we made it a priority to organize our expectations and roles efficiently, with a Team Standards document. The document determined the expectations concerning communication, meeting attendance, documentation, tools used, reviews, and role(s) and responsibilities.

Requirements

Our project utilized a semi-structured acquisition process that began with identifying the key challenges and needs of nonprofit service providers and local governments. With the help of our client, our meetings, additional research, and external stakeholder interviews enhanced our understanding of why a streamlined and intuitive system for service navigation and resource discovery is important. At the end of our acquisition process, the defined system requirements were categorized into functional and non-functional types.

Functional Requirements:

- **Intuitive user interface**

  The user interface was designed with a modular approach in mind, with a navigation bar with several tabs of components. The navigation bar was supposed to have a profile page, navigation section, messaging tab, and organization interaction tab(s).

- **Login/logout auditing**

  To uphold accountability among the many layers of our project, a login/logout auditing system was needed to track user logins and what actions were taken within the system.

- **User role management**

  A system for managing different user roles, ensuring that appropriate access is based on each user's role/status.

- **Organization creation**

  The ability to create and manage organizational contributors in an efficient manner that will allow many organizations to come together and meet the necessary contacts they are looking for when providing services for a client.

- **Navigation features**

  A system that will hold all of the tools that navigators, or volunteers working with service providers, who engage with the people in need, will utilize in navigating resources and client data.

- **Real-time database**

  A database is needed where information can be updated accurately and queried in real-time as users engage with the interface.

- **Messaging system**

An integrated messaging system that allows users to communicate via role, organization, and a specific user.

Non-Functional Requirements:

- **Password requirements**

  Security will be enhanced with specific password requirements, such as 8 characters minimum, one uppercase letter and one lowercase letter have to be present, and a symbol.

- **Scalability**

  The system should be able to handle multiple concurrent requests without performance degradation. Responses to the user requests should be around three seconds.

- **Security**

  Throughout the project, multiple layers of security will be present, with passwords being hashed when stored in the database, role-based access within the systems, and internal AWS security protocols. This requirement was especially important with the sensitive information being handled.

- **Ease of use**

  The user interface should be intuitive so that the time for training can be reduced and the learning curve can be minimized.

The requirements above ensure that the project provides a secure, scalable environment while utilizing the client's mission of unifying local governments and non-profit service providers. Throughout the capstone experience, we fulfilled most of these functional and nonfunctional requirements. Alignment within the realistic implementation and organized goals contributed to our ability to develop an effective and efficient solution that will enhance our client's real-world impact.

Architecture and Implementation

Overview of the System

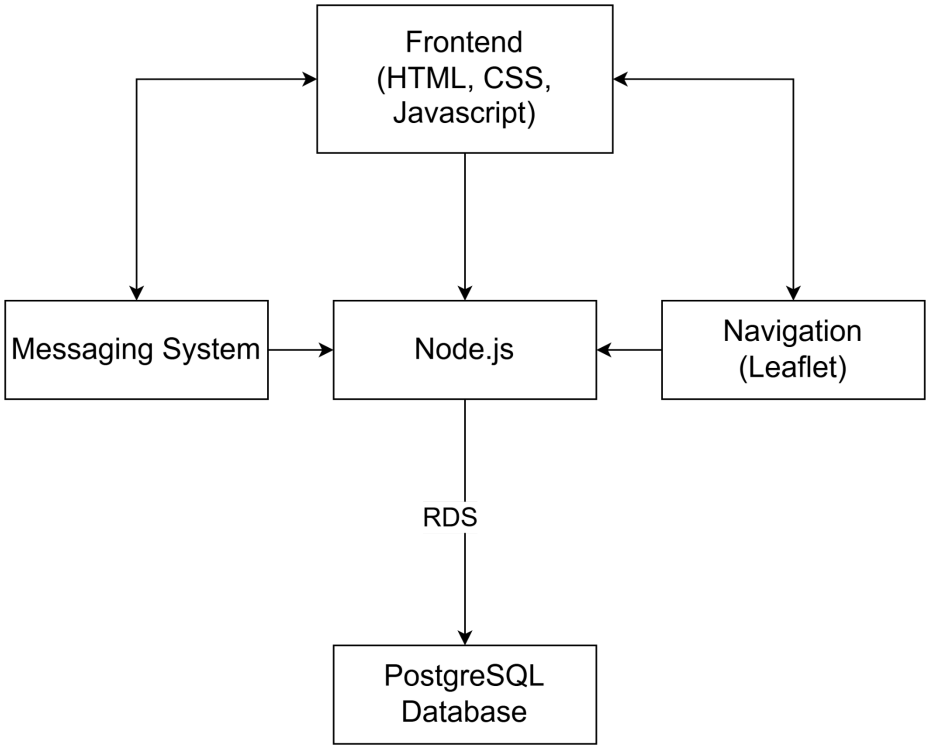


Figure 1. General architecture diagram

Database

For our solution, we needed a robust database that was not only secure but also fast. Due to the conditions to which this would be used, it was vital that it could return data fast because every second matters to our clients and users of the system. A couple of seconds could be the difference between someone wanting to change their ways to better themselves and backing out and staying in the cycle of life that they are currently in. The database is additionally going to need to be robust since most, if not all, of the individual components are connected to the database and rely on the reliability of fetching information quickly to be able to function properly. The different components will also share information through the database, so it was pivotal that we found a database that was not only reliable, but fast and secure.

Role-Based Access Control

Messaging System

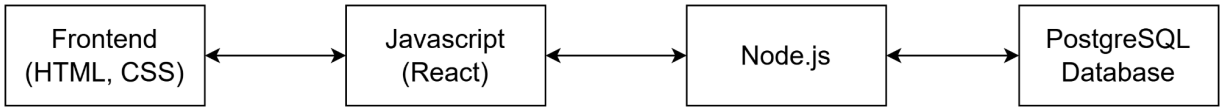


Figure 2. Architecture of Messaging System

Our solution features a messaging system that allows cross-organization communication. This means that organizations can communicate with each other directly without the need for specific contacts. Our system uses Node.js as the backend framework, and it works by storing messages in the database and retrieving them on the receiving end.

Because of our maintained role-based system, users have the ability to contact everyone with a given role in the system. You don't need the individual contact information of everyone you want to reach, and this ensures that no one is left out by accident. When sending a message to an individual or creating a custom group message, users can search for people by name or by role.

Navigation System

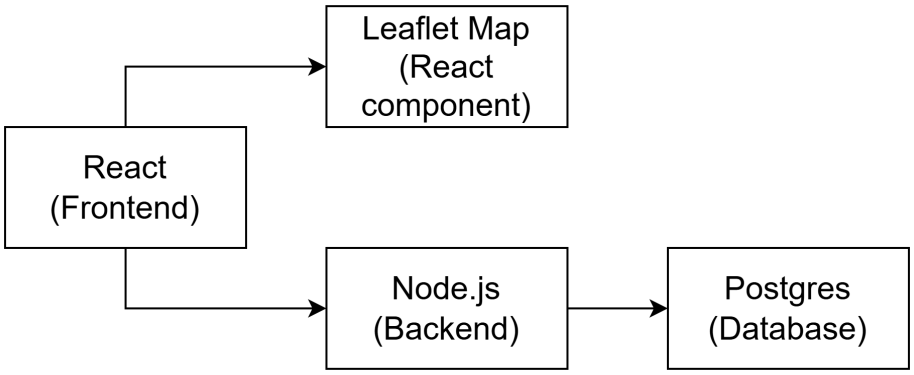


Figure 3: Architecture of Navigation System

Part of the newly developed application is the three-part navigation system, where the Navigation Form, Mapping Interface, and Service Matching Page are highlighted. These tools work together to help users called Navigators, trained volunteers who work with

service providers, in collecting client data, visualization of the data geographically, and matching clients with services based on their needs and wants.

All of the pages are rendered using React and rely on the Node.js backend architecture, which handles all the data flows between the display pages and the PostgreSQL database. Since the mapping interface utilizes an external mapping library called Leaflet, the React Leaflet components are embedded in the web pages directly. The navigation components are within separate pages but are unified in one set of tools for navigation, streamlining the Navigator role workflow. Altogether, the three forms are meant to demonstrate the different portions of the current workflow where volunteers called navigators can access and utilize them in one secure environment. The navigation components are modularized to allow for independent development and debugging, but share the same data, allowing for seamless integration.

User Interface

The goal for the user interface is to have the most modern and easy to use interface for the convenience of our users.

Implementation Details

Database

The database is hosted on an Amazon RDS instance through our Amazon EC2 server. It runs on a PostgreSQL engine, meaning that it is a relational database so that data can be stored and fetched much more efficiently than other options. The database is responsible for hosting all data regarding the project, ranging from user information to coordinates for the navigation system to message history for the messaging system. An extensive backup of the database does exist in the `DB_BACKUP.sql` file located in the `/backend/` directory. The database acts as a spine to the entire program, allowing different components to interact with each other while also providing necessary support to individual pieces of the project.

Role-Based Access Control

Messaging System

The messaging system is built using Node.js as the backend framework and JavaScript, HTML, and CSS for the frontend. All code related to the messaging system is located in the `Messages.js` file under the `frontend/src/` directory. It uses several routes to make queries to the database. These queries are specified in the `backend/` directory in a file called `server.js`. These routes are responsible for querying the database to both add and retrieve messages. There are different routes for individual messages and group messages, as those are stored differently in our database. There are also routes for checking which users have which roles. This is to support the role-based messaging and role-based searching functionalities.

Navigation System

Navigator Form

Located in `frontend/src/NavForm.js,` the navigator form component allows navigators to view client profiles, add new notes from recent interactions, and record services offered during a session. Within the source code, `useState` and `useEffect` are used to render and control different page states and interact with backend endpoints such as `GET/clients`, `POST/locations`, `POST/notes`, and `GET/notes`. Client records are stored in the `clients` table in our database, with any session notes being saved in an overlapped table called `notes`. The time and date of any sessions done between a navigator and a client are also stored in the `notes` table. However, for any of the client data to be available to the user, authentication using the generated `JSON Web Token (JWT)` is required and is checked when the Navigator form section is clicked on.

Mapping Interface

The mapping interface was implemented in `frontend/src/Mapping.js` and stands out from the rest of the navigation forms due to its integration with the React Leaflet library. Location data of clients is retrieved from the PostgreSQL database utilizing the Node.js backend. Each piece of location data is then rendered into interactive map markers that display the client's name, date, and time of interaction. Utilizing Leaflet's open source mapping capabilities allows for the map to be zoomed in and out, panned, and allows for polylines to be drawn to demonstrate the relationship between two or more locations. Coordinates are held in the `locations` table, where each location is linked to a client by their ID in the `clients` table.

Service Matching Page

Utilized in `frontend/src/ServiceMatch.js`, the service matching component starts off by requiring the user or navigator to fill out the form demographic information, such as age group, gender, and needed/wanted levels of help for a specific client. From there, the database is queried for all the services available in the `organization_services` table and is filtered by the demographic data given in the input fields. The `ServiceMatch.js` file contains the service matching logic, where it is confirmed that the organizations' restrictions in the `restrictions` table match certain categories that a client falls under. The defining highlight of this feature is the ability for service providers to collaborate by offering a comprehensive list of services. With that functionality in mind, navigators or other users can explore options based on the client's needs and submit a referral, which is logged in a table called `referrals`.

User Interface

The user interface has a very simple but modern design that allows for navigation across the site to be explanatory and easy to understand. The main goal was convenience and ease of use for the user. The best technology for that implementation included React for dynamic navigation and Bootstrap for the clean look with CSS integration. Since it was a web application, HTML and CSS were used to create the skeleton and basic styling for the web pages. This allowed the pages to carry the same structure and design for a clean and consistent look all around. React allowed us to break the website into a more component-based architecture, which allowed for a faster and efficient build for all the pages.

To make the pages interactive on top of the skeleton and style elements, JavaScript was needed. This ensured that our website not only had the style but also the silent side logic and rendering to make the website work in its intended manner.

Conclusion

The original architecture used Django, but as the project evolved, it became clear that Django's synchronous request/response model couldn't meet the dependency of real-time notifications and updates. A more event-driven system was needed to handle live updates and notifications efficiently, which was something Django struggled with even when extended. Additionally, Django's tight Python dependency became increasingly more difficult to work with as the project shifted toward a full-stack JavaScript architecture. Moving to Node.js allowed us to unify the stack, reduce context switching, and implement a smoother, faster system optimized for real-time interactions.

We also planned to implement media attachments in the messaging system and utilize WebSockets for more optimized results. However, due to time constraints and the lack of websockets, we had to opt for a more traditional messaging architecture where direct queries to our PostgreSQL database from the frontend were required. Additionally, prioritization of media file support was set as a stretch goal after a meeting with our client consisted of feedback that basic messaging communication was needed for MVP deployment. Overall, our architectural decisions were influenced by our client's feedback and real-world constraints, which resulted in a developed solution that emphasizes a modular approach and practical use focus than we originally anticipated,

Testing

- For this section, discuss your testing activities.
- This should include your overall testing strategy and the kinds of tests you ran to validate your implementation.
- Your aim here is to give the reader an honest and solid overview of how the delivered software has been tested.
- Note that source-code tests are just one aspect of testing. Integration and usability tests should also be used to make sure that the functionality needed has actually been implemented in a way that users can access.
- This section should also describe the results of your testing and any changes you made to your design or source code in response to testing results.

Overall Testing Strategy

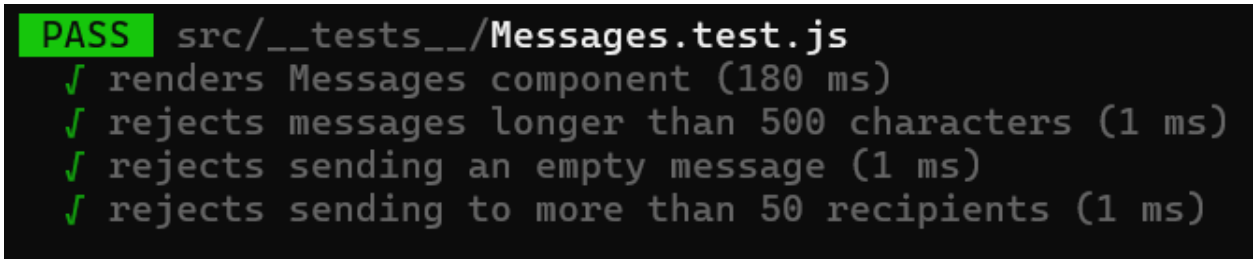
To ensure the quality and integrity of our product, we are conducting extensive testing of our code. Planning to reach 100% code coverage through unit tests, integration testing, and user testing. To complete our unit testing we are using pytest for our python aspects and jest for the javascript. We chose Selenium be used to conduct integration testing for our front end to ensuring that everything is working as intended, while fixing any problems with the code as they came up. Lastly, we had a small base of users help us tweak the alpha version of our product and we will be giving our client a beta version so that real

navigators will be able to test out our software and give feedback to us. We are planning to have this all completed in the coming weeks, which leads us to our schedule.

Individual component testing

Messaging Testing

The messaging system was tested primarily using Jest unit tests. These tests ensured that the input constraints we put on the software were being enforced by the program. All input constraints were validated by the unit testing.



Furthermore, our messaging system was user tested by our mentor, as well as several other of our peers, who provided valuable feedback. While we weren't able to implement all of their suggestions due to time constraints and restrictions related to client requirements, their input helped us evaluate our design and plan for future improvements.

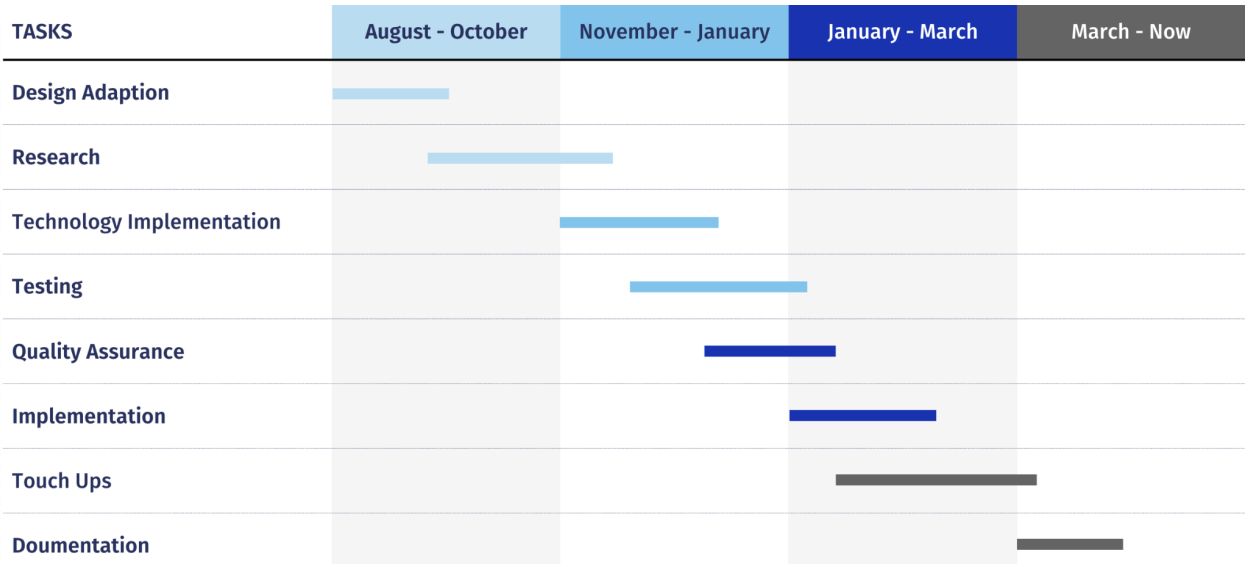
Integration testing for the messaging was mostly performed manually. The backend logic and UI were developed at the same time for testing purposes, and since the page is separate from any other page, there was not much need for intensive integration testing in regards to other elements of the system. When moving from local development to the server, there was a slight updated needed in the way the routes were set up due to a discrepancy between how the test environment was set up with docker and how the main server is set up to run. No functional changes were needed at this time.

User Interface Testing

The test for the user interface had both function and style specific test where real test users measured the functionality of the design for the websites and gave their honest feedback to see if anything was not measuring up to the intended satisfaction level and standards.

Project Timeline

- Now that we know what your team produced, give a quick review of the overall project timeline.
- You could do this as a Gantt chart...or as a sequential list of milestones.
- In either case, briefly walk through the schedule you present, describing the key phases as well as any interesting factoids (how work was divided up in the team, who led on what, etc.) related to individual phases.
- Be sure to say where you are at this moment in the schedule you have outlined.



Future Work

Future work

- Sending file attachments through the messaging

- Adding text and email messaging options
- Messaging using websockets
- Dynamic roles, allowing organizations to create their own roles with defined permissions
- Organizing roles by organization (like you could find all the navigators associated with a certain organization)

Conclusion

- As always, every document needs a conclusion that wraps up the discussion. Start by briefly reviewing the context and motivations of the project, your client's business workflow and what's wrong with it.
- Then state what you have built that addresses these needs, pausing to bullet out a few of the best specific features.
- Wrap up with discussing the impact your project solution will have (a) for your client, e.g., time-saved, accuracy improved, throughput increased... whatever.
- Put some specific numbers to it if you can. Then talk about (b) broader impacts (if any) that your product might have, i.e., other contexts or clients that might be able to use it, potential for broader markets, and so on.
- Close with some reflective comments on your team, the project process, and the Capstone class

Glossary

If you use any terms that have special meaning (domain-specific terminology, for example), lay out the definitions here.

Appendix A: Development Environment and Toolchain (absolutely required)

Remember how uninformed you were on practical mechanics of how to actually develop code in your chosen environment when you started? How long it took you to figure out your toolchain and development cycle? One of the key pieces of "organizational knowledge" for a project is exactly this: how should a new team member configure their development machine, and what is the process leading from code to production of a runnable product? This is what you want to explain in this appendix. Write it as a "how-to" for setting up your machine:

- **Hardware**: Start with an overview of your environment: what platform(s) did your team develop on (Linux, Mac, etc.)? Give a rough overview of the tech specs (processor, memory) of the machines. Comment on whether you feel there are any minimum hardware requirements for the effective development of your software, beyond "a decent machine".
- **Toolchain**: Next introduce and discuss all the software tools you tools used: development environment/editor + plug-ins that were useful/critical, backend databases, other supportive tools/packages (e.g. package managers, etc.) that you installed to make life easier, or that are required for other pieces of the software chain to work right. For each one, name it, explain briefly what it does in general, and then why/how it was helpful/needed for this project.
- **Setup**: Now discuss how to set up your environment. This should be a step-by-step guide: install this, install that, place this in that directory, etc. You don't have to discuss in detail how to install individual packages (instructions for that are presumably on the site for that package) but do point out any special settings or configurations that should be made in installing it that are relevant to this project. Just imagine yourself walking a new team member through setting up their machine.
- **Production Cycle**: At this point, your instructions should have allowed a newbie to completely set up his/her machine for action. Now let's get down to work: explain the production cycle, i.e., walk through how the edit-compile-deploy process works. It might be helpful here to focus it around a specific example:

explain how one might change, for instance, the text that appears in some obvious dialogue in the application...then what you'd do to build and produce a "new version" of your app. To avoid missing small but critical steps, it is useful to just do a little edit yourself pay attention, and write down each step in the process...