Production Run Reports

Design Document

sdmay23-06

JEDA Polymers

Dr. Stoytchev

Colton Carlson - Scrum Master / PDF Generation / Email Service

Hayden Havelka - Backend

Jay Arnold - Backend

Connor Linn - Frontend

Noah Meyer - Frontend

sdmay23-06@iastate.edu

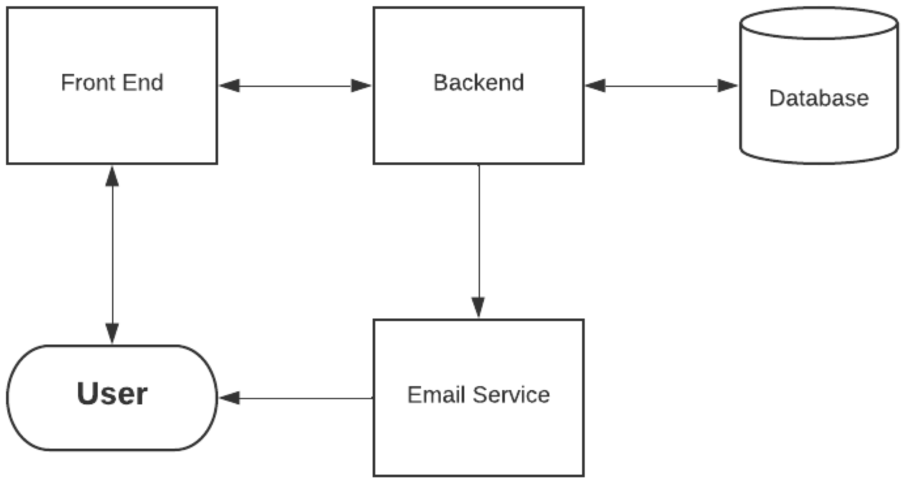
https://sdmay23-06.sd.ece.iastate.edu/

Table of Contents

1 Revised Project Design

1.1 Project Plan

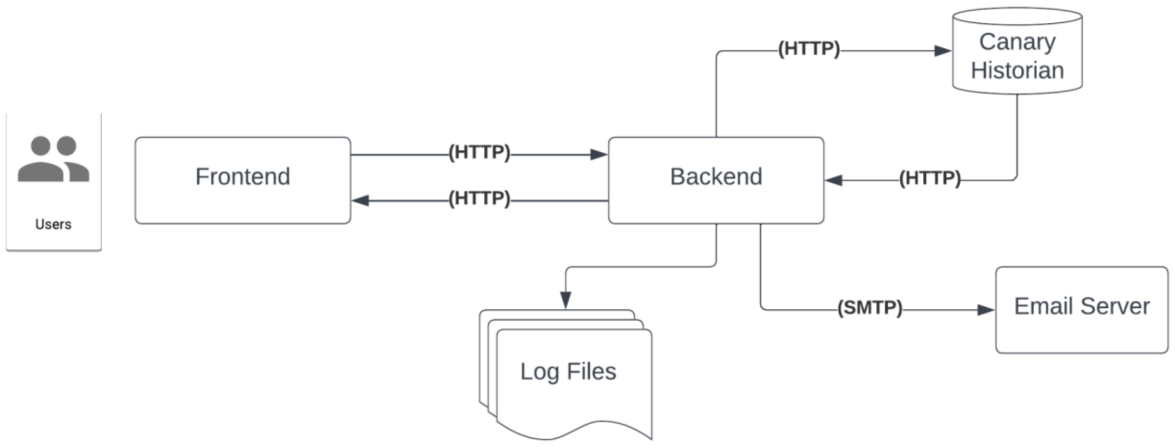
1.1.1 Design Overview



User = John/Any other user who wants a work order generated.
Email Service = REST API calls to send emails of generated PDFs to the user.
Front End = React website where users will be able to input work orders and be displayed results.
Backend = ASP.Net Core web API hosted on a Windows Server 2019.
Database = Canary Database storing all work order information.

Above is a very simplified version of our software architecture. The user will interact with our frontend which will make calls to our backend using our chosen APIs. In the backend a report will be generated and then formatted into a simple, easy-to-read PDF. The PDF will then be sent back to the frontend for the user to view, as well as emailed to the user if needed.

1.1.2 Detailed Design and Visuals



1.2 Project Evolution Since CPRE 491

1.2.1 Frontend

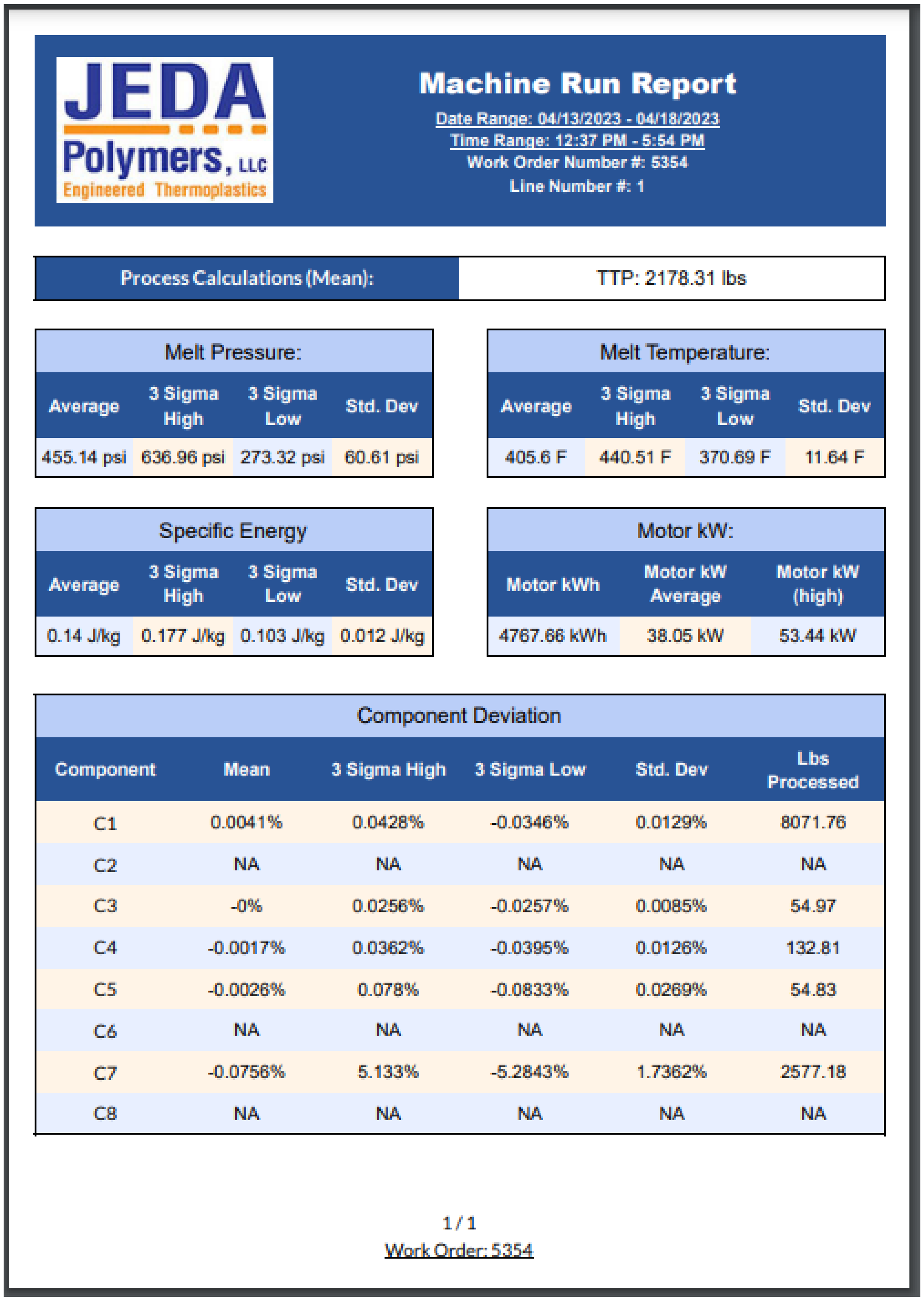
Our frontend was written using React. And it connects to the backend using our REST API. There are several pages: Log in, Request Data, Results Page, and New Account/Forgot Password pages. These have a responsive UI that works across all browsers as well as operating systems. The use of React hooks helps make this process clean and concise.

1.2.2 Backend

The backend was written in C# and ASP.NET Core 6.0. This is where the REST APIs are programmed along with a few other subcomponents. The first subcomponent is PDF generation. To generate a PDF, QuestPDF is being used. QuestPDF is a modern .NET library for PDF document generation. The next subcomponent is working with a Canary database. The database is configured on our client's server. We get the data from the Canary database to generate a PDF. Once a PDF is generated, it will be sent to the front end to be viewed by the user. Another subcomponent is the log file. The log file will have start and end times for report generation. The last subcomponent is an email API. If a user on the frontend wants a generated PDF sent to emails, then the email API will send the PDF to those emails. All of this is configured to run on Windows Server 2019 as our client requested.

1.2.3 PDF

Below is our finalized PDF template using mock data as a demonstration.



JEDA Polymers, LLC
Engineered Thermoplastics

Machine Run Report

Date Range: 04/13/2023 - 04/18/2023
Time Range: 12:37 PM - 5:54 PM
Work Order Number #: 5354
Line Number #: 1

Process Calculations (Mean):	TTP: 2178.31 lbs

Melt Pressure:

Average	3 Sigma High	3 Sigma Low	Std. Dev
455.14 psi	636.96 psi	273.32 psi	60.61 psi

Melt Temperature:

Average	3 Sigma High	3 Sigma Low	Std. Dev
405.6 F	440.51 F	370.69 F	11.64 F

Specific Energy

Average	3 Sigma High	3 Sigma Low	Std. Dev
0.14 J/kg	0.177 J/kg	0.103 J/kg	0.012 J/kg

Motor kW:

Motor kWh	Motor kW Average	Motor kW (high)
4767.66 kWh	38.05 kW	53.44 kW

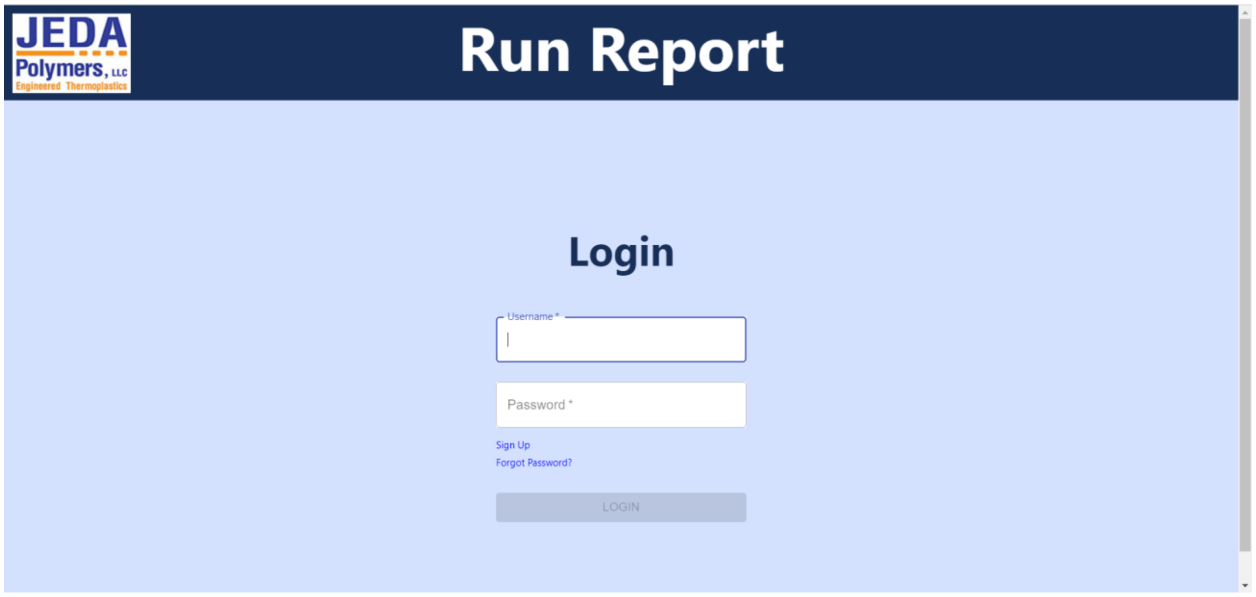
Component Deviation

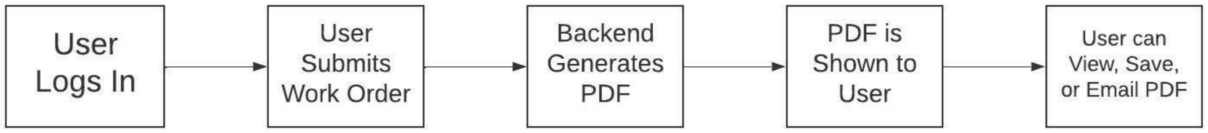
Component	Mean	3 Sigma High	3 Sigma Low	Std. Dev	Lbs Processed
C1	0.0041%	0.0428%	-0.0346%	0.0129%	8071.76
C2	NA	NA	NA	NA	NA
C3	-0%	0.0256%	-0.0257%	0.0085%	54.97
C4	-0.0017%	0.0362%	-0.0395%	0.0126%	132.81
C5	-0.0026%	0.078%	-0.0833%	0.0269%	54.83
C6	NA	NA	NA	NA	NA
C7	-0.0756%	5.133%	-5.2843%	1.7362%	2577.18
C8	NA	NA	NA	NA	NA

1 / 1
Work Order: 5354

1.2.4 Functionality

This project is designed to work on Chrome, Firefox, Safari, Bravo, and Edge web browsers. Let's go over a scenario with a user and real world context. Our client wants to generate a production report to show to a customer. The user can load up their browser of choice, and type in the web address for the application. Once the user is loaded into the application, they will be greeted with a login page. A screenshot of the login screen is shown below. After logging in, the user can generate a report with a work order number or a line number. Once one of those is submitted, a PDF production report will be generated. The PDF can be viewed from the website, downloaded or sent to specified emails.





This timeline above shows the intended real world use of the project.

1.3 Summary of Requirements

1.3.1 Functional Reqs

- When a request is made the system shall produce a new report
- When an email request is made, the system shall email the most recently made report to the specified email address

- Where the line number is included, the system shall only look in the line specified for the work order.
- When the time turns 11:59 PM CST the system shall email a report for every work order that finished during the previous 24 hours.
- If a work order does not exist, then the system shall notify the user.
- When a report is produced, then any locally obtained data shall be deleted.
- When a report is requested, add date, time, and work order number to the log file
- When a report is finalized, add date, time, and work order number to the log file
- When a report is emailed, add date, time, and work order number to the log file
- If a report fails to be generated, then add date, time, work order number, and error to the log file, then retry.
- If a report fails to be generated 5 times in a row, display the error message on the front end for the user to see.

1.3.2 Non Functional Reqs

- The system shall produce a report in 15 seconds. ***Constraint***
- The system shall produce a PDF report based on the template decided on. ***Constraint***
- The system shall produce a PDF that contains the machine data from the requested work order.
- The system shall use company colors. ***Constraint***
- The system shall be able to support new lines/components.
- The systems frontend shall use React/Node.js libraries ***Constraint***
- The systems frontend shall be accessible from Mac and Windows. ***Constraint***
- The system shall run on Safari, Chrome, Firefox, Opera, Edge. ***Constraint***
- The system's backend shall run on a Windows 2019 Server. ***Constraint***
- The system shall not include null/zero values in the report.
- The system shall use FTP to send the report to the front end. ***Constraint***
- The system shall use the internal email system to send emails.
- The system shall use IMAP as the protocol for sending emails. ***Constraint***
- The system shall use OPC UA as the protocol for communicating with Canary ***Constraint***
    - This allows canary and backend communication to be OS independent.
- The system shall have access to the work machine's data via the Canary Database.

1.4 Standards & Practices Used

- IEEE 26515-2018 - Agile Development Cycle
    - Developmental cycle we will use. Allows us to be flexible with the customer and have clear constant deliverables.

- IEEE 829 - Software Test Documentation
    - Allows us to easily document our tests, easily shows what test does what and explains expected results.
- IEEE P1363 - Public Key Cryptography
    - Allows us to ensure that only authenticated users will request the generated reports.

1.5 Engineering Constraints

- The system shall produce a report in 15 seconds. ***Constraint***
- The system shall produce a PDF report based on the template decided on. ***Constraint***
- The system shall use company colors. ***Constraint***
- The systems frontend shall use React/Node.js libraries ***Constraint***
- The systems frontend shall be accessible from Mac and Windows. ***Constraint***
- The system shall run on Safari, Chrome, Firefox, Opera, Edge. ***Constraint***
- The system’s backend shall run on a Windows 2019 Server. ***Constraint***
- The system shall use FTP to send the report to the front end. ***Constraint***
- The system shall use IMAP as the protocol for sending emails. ***Constraint***
- The system shall use OPC UA as the protocol for communicating with Canary ***Constraint***

1.6 Security Concerns and Countermeasures

The client did not see the need to implement a feature to log in to the web application. However, after some communication back and forth with them, we were able to convince them that while it is not a wanted feature, it would be necessary to keep their data and system safe.

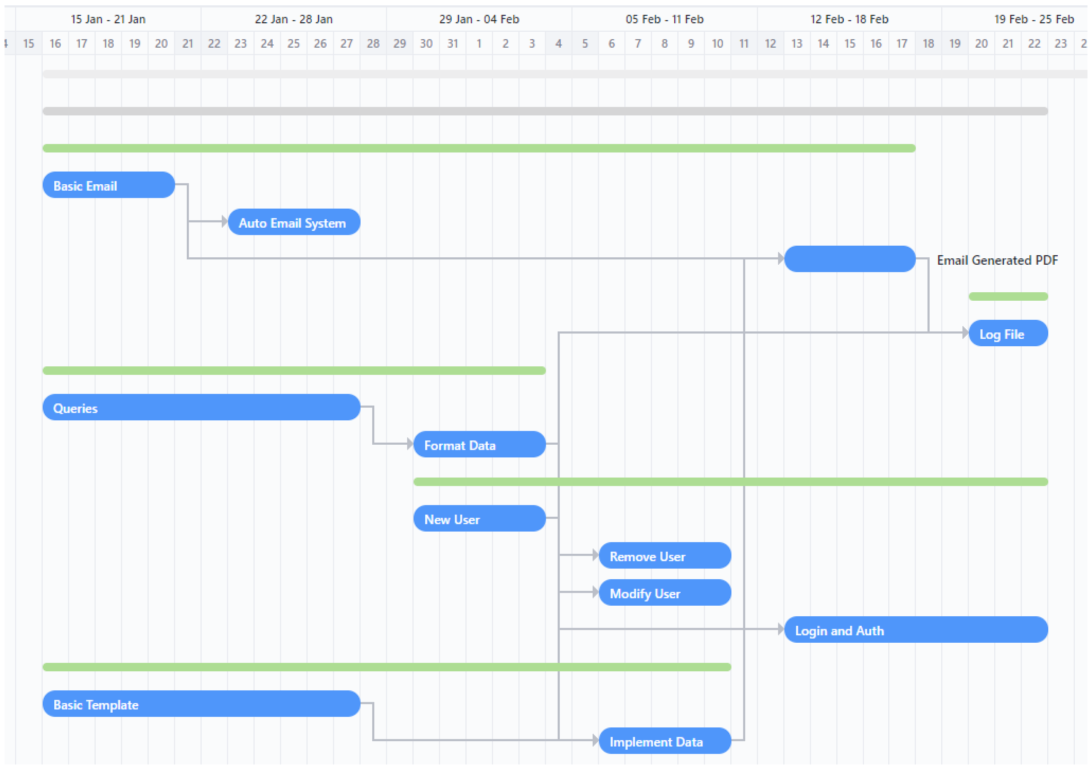
- Unauthorized users accessing restricted web pages
    - Only users who are logged in are able to access restricted pages.
- Malicious data injection into the client’s database
    - Users do not have direct access to the database and are only able to pull information from the database.

2 Implementation

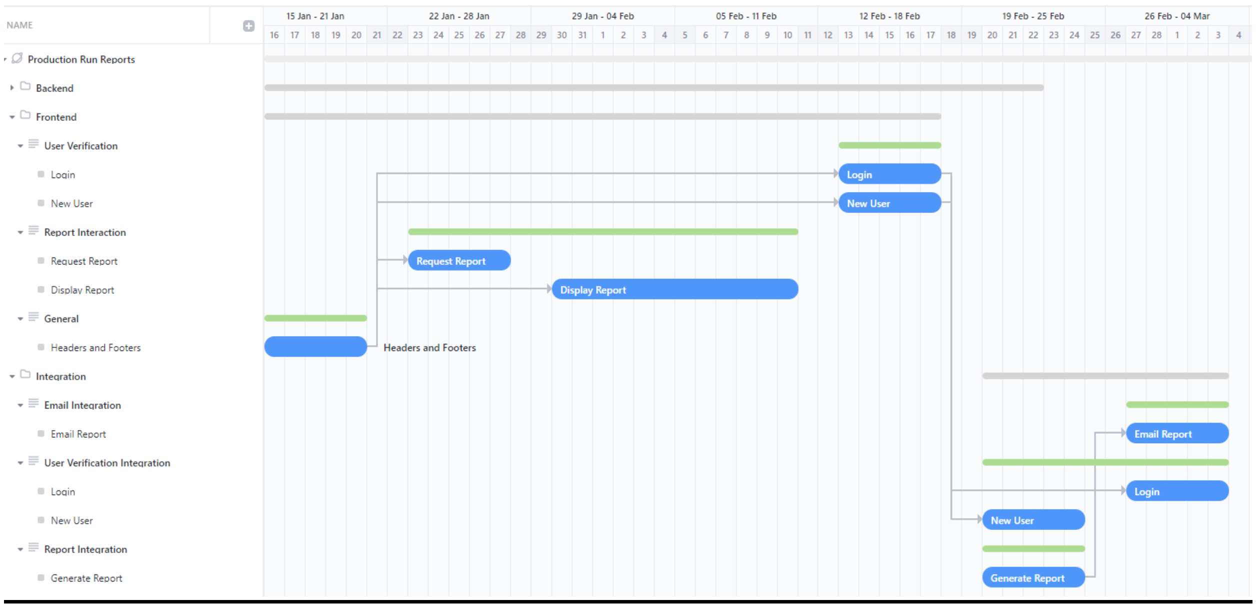
2.1 Timeline

Last semester we had planned to follow the Gantt chart below. In the end, we were a week behind schedule but were able to confidently meet all requirements. Back-end and front-end worked in parallel to each other. After both teams finished developing the functionality, testing began on the front-end and the back-end separately before integrating and implementing the code onto the server. This way we were able to find out bugs and improve the code before bringing more variables into the setup. Once we knew the front-end and back-end communicated flawlessly, we implemented the code onto the server. This way we knew that any new issues were related to the server and not to the code. Once we confirmed everything worked, we began the rest of the testing.

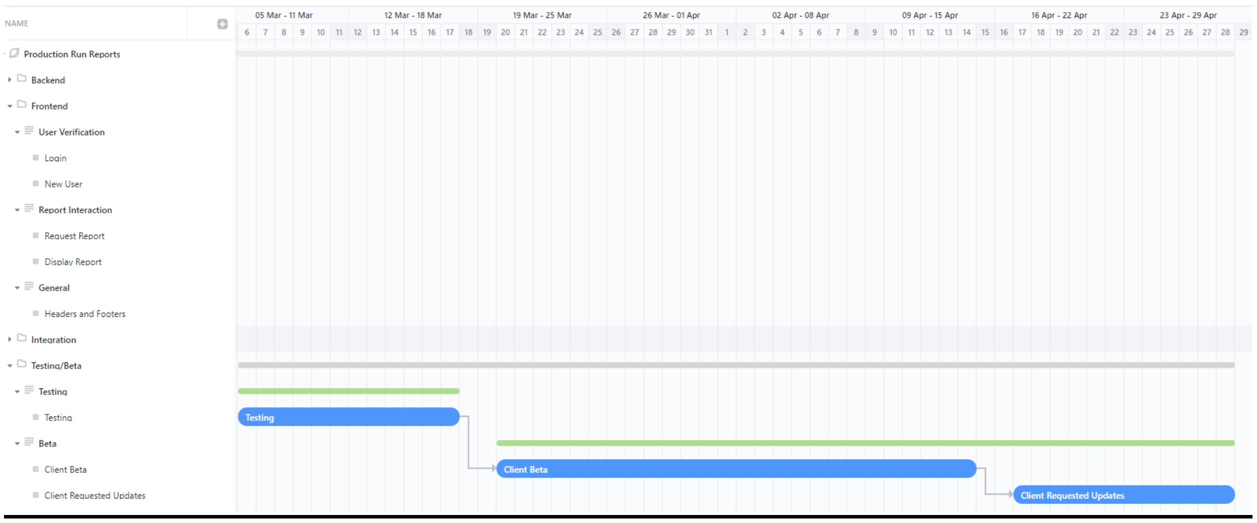
2.1.1 Backend



2.1.2 Frontend & Integration



2.1.3 Testing, Beta, & Updates



2.2 Technologies Used

ASP.net Core 6.0 Web API is used as our backend service. Since it is in C#, it runs natively on windows 2019 servers, which is what our client hosts the server on. ASP.Net is used by several companies in the real world so third party documentation/tutorials are plentiful while maintaining similarities to Spring Boot, a framework we used in SE 309. ASP.Net also has native and easy ways to port our microservices over to docker containers so in unlikely event the client decides to switch from one operating system to another all we would need to do is switch to using docker without losing any of our progress.

QuestPDP is an open source .NET PDF generation library. We like it because of its extensive documentation, including several examples, making learning how to use the framework easier than its competitors. Because the library runs natively in C#, we do not have to worry about installing third party packages or other programming languages and it creates an easier experience for our client.

Canary Database is the current database JEDA polymers is using to store the data from their production lines. Because this was given to us we did not get a say in choosing a database to use. However, Canary has full documentation online which made integrating it into our project relatively easy.

React is a user-friendly framework for creating web-applications. This was an easy choice to use on our frontend because several of us already had experience using React. Compared to other languages, it keeps up in speed and makes working on different browsers and operating systems seamless.

GitLab is a DevOps platform that helps automate builds, verification, and integration of code. It will be used as our repo to hold all the code. It will have a build pipeline, issues, and merge requests. Other similar platforms are Azure DevOps, Jenkins, Google Cloud, and more. Gitlab is an industry standard. It is an all in one platform compared to Jenkins which doesn’t have release, packaging, monitoring, and configuration services.

2.3 Roadblocks

We ran into a few roadblocks throughout the development of the project. Namely these were the most difficult to deal with however our team was able to problem solve and work around them with the help from our advisor and our client.

- ASP.Net Core planned obsoletion in June 2023.
    - Our team member, Colton, spent 40 hours over the course of one week changing all of the ASP.Net Core dependencies and usages over to ASP.Net API calls in order to deliver our client a project that would not stop working after we graduate.
- Remoting into Canary Database on a MAC device.
- Integrating the front and back end of the project evolved into a larger task than originally planned for.

3 Testing

3.1 Testing Process

Our team built the product in a test-driven development environment. Test-drive development is a style of programming which includes three tightly interwoven activities: coding, testing, and design. The benefits of doing this is that it usually reduces the amount of defect rates.

Test-driven development follows a set of rules to work as efficiently as possible. The first rule is to write a single unit test that describes an aspect of the program. Next, run the test. This will obviously fail because the code base doesn't have the feature yet. After the first test, write the minimal and simplest amount of code to make the test pass. After the second test passes, refactor the code so that every line, method, and class has a distinct purpose, idea, or responsibility. The refactored code should also have a minimum number of components, and it should have no duplication. Once the refactored code is complete, keep repeating until the program is finished with all of the features.

The team believes this is the best way to develop the product. The set of rules will keep development in progress without running into poorly written code that can't pass tests. The rules make sure that every single line of code has a purpose. TDD is a great style of programming for this project.

3.1.1 Unit Testing

3.1.1.1 Frontend

Unit testing for the frontend consisted of utilizing Jest which is a viral library that allows developers to conduct unit tests on React. By using Jest's watch mode, we were able to generate snapshots of our code which is then compared to previous snapshots. If the snapshots differ, we are able to trace where in our code the tests failed, or produced unintended results. These snapshots were vital when conducting unit tests because they helped us identify areas of our frontend where components changed where they should not have.

The units that we tested were the user login page, generate report page, the generated report page, and all of the user management pages. These components must all perform correctly in order for our product to work. By ensuring that these were thoroughly unit tested, we were able to deliver a product that is up to the standards of the team and client.

An example of a unit test that we ran would be creating a snapshot of the generated report, then generating the same report again. By comparing the snapshots, we can verify that the components display the correct data and are working as intended.

3.1.1.2 Backend

Unit testing for the backend will be pretty simple. ASP.NET has the ability to create unit tests for controller actions. Since we will most likely only have one controller, we will only have to create one test controller. The test controller will check to see if the data is correct and properly formatted. To test the log file, we will just feed mock data to the log file. If the data being sent and the data showing up in the log file match, then we know the log file is complete.

An example of a unit test would be creating a test controller for the Work Order controller. This will consist of making a Work Order test object with specific data. Then we can do a check to see if the API call of the Work Order test object is equal to the expectation.

3.1.2 Interface Testing

For interface testing there are numerous different routes we could take, user testing is going to be our first main form of interface testing. The plan moving forward will be to have both a mix of engineers and people who have never coded before.These groups will comb through the website to test every link and endpoint available. The mixed demographic is important here because engineers are nice for checking edge cases and things that will most likely break the site. Meanwhile, having a non-engineering perspective can be nice for testing the usability of the site and receiving input on how the common user feels about the interface.

Additionally, any formal testing will be completed using a framework called LambdaTest. Lambda test makes automated GUI testing easy and fast. This may not end up being necessary, but additional testing never hurts. With the majority of our code we will test on a regular basis to avoid any foundational errors. However, the majority of the testing will take place after the interface is already made due to the nature of interface testing.

3.1.3 Integration Testing

For integration testing we used Robot Framework, which is a python library that makes automated test cases easy to run with easy to interpret output. One of our group members is also very familiar with RF, and that reduced the learning curve.

We found our critical integration pathways by looking at our software architecture as well as our object structure. A key integration test we performed was linking the front end request button to the back endPDF calls.

List of some of our critical integration parts:

(Front End) Request PDF with (Back End) PDF Generation

Pulling Data and PDF Generation

(Front End) Email PDF with (Back End) Email

PDF Generation with Log File

A sample test case for PDF generation and log file integration (written in robot framework style code):

```
PDF Gen Log File Test

        Generate PDF        ${sWorkOrder}

        @{sLogFile}=                Read Log File

        RUN KEYWORD IF  ‘${sLogFile}’ not in @{sLogFile}    Fail    ‘Work    Order
not found in file’
```

3.1.4 System Testing

Since system testing involves the entire scope of the system we performed “end to end testing”. We did this by testing different inputs on various parts of the system to make sure that they yield the desired output. For example, a test input would be given to the front end of the system while a desired output would be tested on the back end of the system. Since multiple tests have already been done in order to make sure that subsystem outputs are correct, this allowed us to check to make sure that an input can travel through the entire system without throwing any errors or producing an incorrect output. While the majority of our tests were written to test individual components, we still ran end-to-end tests to make sure that the system works correctly.

We used the versatile framework “Robot Framework” as mentioned in previous sections. For more information on how we integrated Robot Framework see the integration testing portion of the document (Section 3.1.3). For more information on how we received and used the output of Robot Framework see the results portion of the document (Section 3.2).

3.1.5 Regression Testing

JEDA Polymers doesn’t have any current report generation for us to add on/modify.

3.1.6 Acceptance Testing

This portion of testing consisted of two things. The first form of demonstration was going through the requirements list. Each functional and nonfunctional requirement was reviewed to see if it is met. The last form of demonstration was to show the client a “beta” version of the product. This ensured that the product has all of the requirements desired by the client. Although most of the product is already done, we might be able to change some small things that the client desires. For the most part it will be demonstrating to the client that the product meets the requirements list.

3.1.7 Security Testing (if applicable)

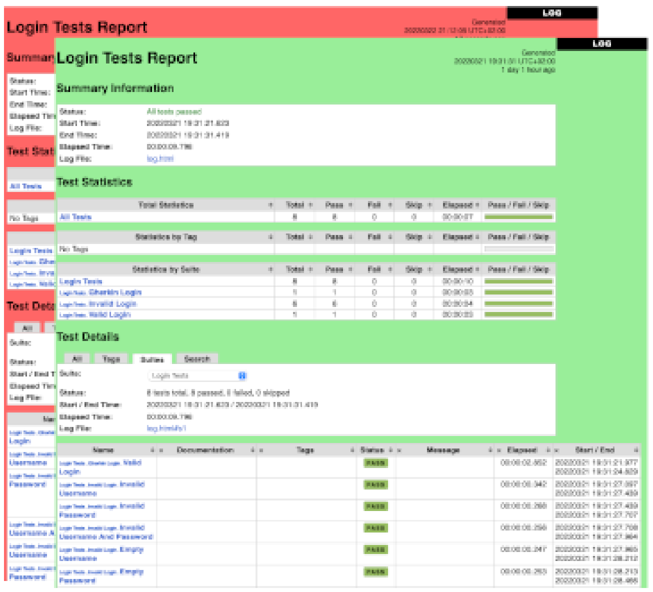
Security testing is not applicable for this project.

3.1.8 User Testing

We conducted user testing by having a mix of engineering and non-engineering students at Iowa State University complete 3 different tasks in the application. First, make a new account, secondly, reset their password, and finally, enter in a work order number to generate, download, and email a report. Additionally we performed a beta test with the client and they had three work weeks to use the application.

3.2 Testing Results

Testing has ensured compliance with our requirements as all of our requirements are testable. Our results from Robot Framework are automatically formatted into a report, log, and output file, below is a picture from Robot Framework’s website where they display a test result. We can see that it breaks down test cases into steps, time for each step and case, inputs and outputs, as well as reasons for failures. This makes it very easy to interpret the results, and when we want to show our client our testing results it will not be complicated for them to understand.


LOG

Login Tests Report

Summary Information

Status:	All tests passed
Start Time:	20220321 19:31:21.823
End Time:	20220321 19:31:31.419
Elapsed Time:	00:00:09.796
Log File:	log.html

Test Statistics

Total Statistics	Total	Pass	Fail	Skip	Elapsed	Pass / Fail / Skip
All Tests	8	8	0	0	00:00:07	

Statistics by Tag	Total	Pass	Fail	Skip	Elapsed	Pass / Fail / Skip
No Tags						

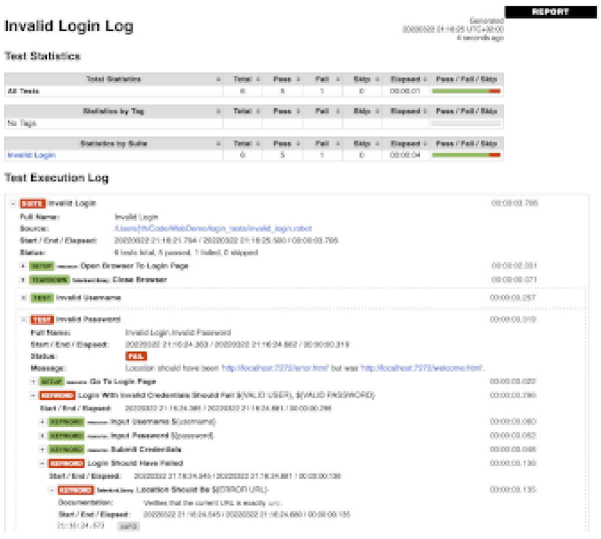
Statistics by Suite	Total	Pass	Fail	Skip	Elapsed	Pass / Fail / Skip
Login Tests	8	8	0	0	00:00:10	
Login Tests . Gherkin Login	1	1	0	0	00:00:03	
Login Tests . Invalid Login	6	6	0	0	00:00:04	
Login Tests . Valid Login	1	1	0	0	00:00:03	

Test Details

All | Tags | Suites | Search

Suite:	Login Tests
Status:	8 tests total, 8 passed, 0 failed, 0 skipped
Start / End Time:	20220321 19:31:21.823 / 20220321 19:31:31.419
Elapsed Time:	00:00:09.796
Log File:	log.html#s1

Name	Documentation	Tags	Status	Message	Elapsed	Start / End
Login Tests . Gherkin Login . Valid Login			PASS		00:00:02.652	20220321 19:31:21.977 20220321 19:31:24.629
Login Tests . Invalid Login . Invalid Username			PASS		00:00:00.342	20220321 19:31:27.097 20220321 19:31:27.439
Login Tests . Invalid Login . Invalid Password			PASS		00:00:00.268	20220321 19:31:27.439 20220321 19:31:27.707
Login Tests . Invalid Login . Invalid Username And Password			PASS		00:00:00.256	20220321 19:31:27.708 20220321 19:31:27.964
Login Tests . Invalid Login . Empty Username			PASS		00:00:00.247	20220321 19:31:27.965 20220321 19:31:28.212
Login Tests . Invalid Login . Empty Password			PASS		00:00:00.253	20220321 19:31:28.213 20220321 19:31:28.466


REPORT

Invalid Login Log

Test Statistics

Total Statistics	Total	Pass	Fail	Skip	Elapsed	Pass / Fail / Skip
All Tests	6	5	1	0	00:00:01	

Statistics by Tag	Total	Pass	Fail	Skip	Elapsed	Pass / Fail / Skip
No Tags						

Statistics by Suite	Total	Pass	Fail	Skip	Elapsed	Pass / Fail / Skip
Invalid Login	6	5	1	0	00:00:04	

Test Execution Log

```
- SUITE Invalid Login                                                        00:00:03.786
  Full Name:            Invalid Login
  Source:               /Users/[illegible]/Code/WebDemo/login_tests/invalid_login.robot
  Start / End / Elapsed: 20220322 21:16:21.794 / 20220322 21:16:25.580 / 00:00:03.786
  Status:               6 tests total, 5 passed, 1 failed, 0 skipped
  + SETUP Open Browser To Login Page                                         00:00:02.331
  + TEARDOWN Close Browser                                                   00:00:00.071
  + TEST Invalid Username                                                    00:00:00.257
  - TEST Invalid Password                                                    00:00:00.319
    Full Name:            Invalid Login.Invalid Password
    Start / End / Elapsed: 20220322 21:16:24.363 / 20220322 21:16:24.682 / 00:00:00.319
    Status:               FAIL
    Message:              Location should have been 'http://localhost:7272/error.html' but was 'http://localhost:7272/welcome.html'.
    + SETUP Go To Login Page                                                 00:00:00.022
    - KEYWORD Login With Invalid Credentials Should Fail ${VALID USER}, ${VALID PASSWORD}   00:00:00.296
      Start / End / Elapsed: 20220322 21:16:24.385 / 20220322 21:16:24.681 / 00:00:00.296
      + KEYWORD Input Username ${username}                                   00:00:00.060
      + KEYWORD Input Password ${password}                                   00:00:00.052
      + KEYWORD Submit Credentials                                           00:00:00.048
      - KEYWORD Login Should Have Failed                                     00:00:00.136
        Start / End / Elapsed: 20220322 21:16:24.545 / 20220322 21:16:24.681 / 00:00:00.136
        - KEYWORD SeleniumLibrary . Location Should Be ${ERROR URL}          00:00:00.135
          Documentation:        Verifies that the current URL is exactly url.
          Start / End / Elapsed: 20220322 21:16:24.545 / 20220322 21:16:24.680 / 00:00:00.135
          21:16:24.573 INFO
```

We also conducted two different user tests and saw extremely positive results. We delivered a beta version of our web application to the client in mid March for them to try out. This gave them about 3 weeks of time to use the project daily. This beta test allowed us to have constant feedback from the client as well as address any concerns or initial setup issues that they could have. At the end of the three week period, we had them fill out a feedback form to gather their overall thoughts and experiences with the project. Their feedback is shown down below.

How often do you use the application?	What do you like about the application?	What do you dislike about the application?	Rate the usability of the app	Rate your overall experience using the app	Any additional feedback?
Daily	Automated and on demand summary of critical process metrics for ensuring production run integrity.	Inability to alter the reports through an admin portal rather than modifying the code directly.	Very usable	Very positive	Looks good!

In order to get more feedback on the project. We reached out to a mixture of computer and non computer majors at Iowa State University to get more, well-rounded feedback. Due to JEDA being a small company, they do not employ any developers and we wanted to conduct usability tests on others, not just our developer friends. In total, we had 10 students take our usability test where we had them complete three tasks. Register for a new account, reset their password, and generate a production run report. The results are listed below.

Task 1: Register for a new account.

How long did it take you to complete this task?	On a scale of 1-5, how difficult was this task? (5 being extremely hard)	Is there anything you would change to make this task an easier process?
Average completion time was 1.7 minutes.	Every user said 1.	None of the users listed additional feedback.

Task 2: Reset your password.

How long did it take you to complete this task?	On a scale of 1-5, how difficult was this task? (5 being extremely hard)	Is there anything you would change to make this task an easier process?
Average completion time was 1.8 minutes.	Every user except one said 1. The outlier said 2.	None of the users listed additional feedback.

Task 3: Log in and generate a production run report for Work Order 5281. Then download and email the report.

How long did it take you to complete this task?	On a scale of 1-5, how difficult was this task? (5 being extremely hard)	Is there anything you would change to make this task an easier process?
Average completion time was 1.33 minutes.	Every user said 1.	None of the users listed additional feedback.

Conclusion: Overall experience and any additional comments.

On a scale of 1-5, how would you rate your experience? (5 is extremely positive)	What did you dislike?	What did you like? Summary of the users' responses:	Any additional feedback?
Every user said 5.	No users listed any.	Simplicity, linear website, simple and easy to use, self explanatory, user friendly, and intuitive.	No responses.

4 Context of Work

4.1 Related Products

This project was requested by a client where they wanted a web application built from the ground up. Due to this, there were no related products that we started off with or based our project off of.

4.2 Related Literature

As stated in the above section, we built this project from the ground up. There were no research papers that we consulted before development.

5 Closing Material

5.1 Discussion

The result of this project was a web application accessible by members of the JEDA Polymers team, which accurately and quickly generates and displays a report of their product (plastic polymers). Our client has given us constant feedback and has been present for all of our modified design decisions and verification of project versions.

With our client's approval, we have marked this project as complete. Each requirement set by the client has been developed, tested, verified, and accepted. As we continue to collect feedback from JEDA Polymers, the client is consistently happy with our work. In the event of an unforeseen failure, we have left our names and contact details with the client to ensure that even after graduation, the project retains all of the hard work we put into it this year.

Additionally, with the project now completed. There is no plan for any future work. The client may decide that they might want more features in the future however they are satisfied at this time. We hope that if they do go forward with requesting new features, they resubmit another project proposal to a future Senior Design class. We wish that another group is able to receive an experience as good as ours.

5.2 Conclusion

With the conclusion of this project, our team can confidently say that we were able to deliver a more than satisfactory application to the client. After multiple meetings with our client, we have received extremely positive feedback and praise for our hard work. It was an honor to have this opportunity to work with a business to develop software that they will use daily for the coming years. Both 491 and 492 have taught us invaluable skills that we can utilize in our future careers and endeavors. Once again, thank you for this opportunity.

5.3 References

"IEEE Standard for Software and System Test Documentation," in IEEE Std 829-2008 , vol., no., pp.1-150, 18 July 2008, doi: 10.1109/IEEESTD.2008.4578383.

"IEEE Standard Specifications for Public-Key Cryptography," in IEEE Std 1363-2000 , vol., no., pp.1-228, 29 Aug. 2000, doi: 10.1109/IEEESTD.2000.92292.

"ISO/IEC/IEEE International Standard - Systems and software engineering — Developing information for users in an agile environment," in ISO/IEC/IEEE 26515:2018(E) , vol., no., pp.1-32, 20 Dec. 2018, doi: 10.1109/IEEESTD.2018.8584455.

Marvin, A., Wilkinson, P., Harwood, A., & Novak, M. (2009, November 17). *Easy approach to requirements syntax (EARS)*. IEEE Xplore. Retrieved September 20, 2022, from https://ieeexplore.ieee.org/document/5328509

5.4 Appendices

5.4.1 Operation Manual / Project README

Production Run Reports

This application is designed to save you time and effort. Instead of having to go into the database manually and crunch numbers, all it takes is a simple work order number and an optional line number. Next, a beautiful PDF of all your desired information is generated. The PDF is able to be downloaded or emailed.

Authentication

The authentication is very simple. A user only needs an email and password to create an account and login. We recommend using a strong password with 8+ characters consisting of letters/numbers/special characters.

- Once a user enters an email and password to create an account, that email will be sent an email authentication link.
- When the email authentication link is used, the user will be able to sign into the application.
- If a user forgets the password to their account, they can use the "Forgot Password" feature on the login page.
- The authentication uses a local database. If that database gets deleted or corrupted, the application will auto-generate a new local database. When this happens, users will have to create a new account.
    - There is no need to worry about security when this happens. This won't be a common occurrence, but there is a chance it will happen.

Usage

- Enter the desired work order number and an optional line number.
- Once the numbers are submitted, the application will generate the PDF.
    - Depending on the amount of data that is being pulled from the database, this could take up to 30 seconds.
- The application will display the generated PDF. The user can just view from the page, download the PDF, or email it.
- All of the work orders for that day will be emailed at midnight.
- The application has a log feature which logs all of the start and stop times and work order number of a generated PDF, and if any errors come up.
- To view the log files, the path is /ProductionRunReports/Logs/...

Customize appsettings.json

- appsettings.json is a file that will allow you to change some of the local configuration settings.
- The settings include logging, where the logs go, how big the log file sizes can get, how many log files at a time, email address and smtp settings, Canary database settings, nightly reports email, and database connection string.
- All of these settings are able to be changed if need be. However, if the appsettings.json is unreadable or missing settings, the default settings will be established.
- To view appsettings.json, the path is /ProductionRunReports/appsettings.json

Authors

- Colton Carlson
- Connor Linn
- Noah Meyer
- Hayden Havelka
- Jay Arnold