Connor Parrott

Henry Wright

Zachary Jewett

Montana State University

CSCI 483R - Dr. Izurieta

Spring 2023

Table of Contents

Introduction

Our memories and experiences shape who we are as a person. Our past defines our present as well as our future, and as we grow older, we in turn grow wiser. However, although the ability to retain memories is often taken for granted, it is not guaranteed. Alzheimer's Disease (AD) is a form of dementia that affects memory, thinking, and behavior. Beginning with a slow yet steady disintegration of brain neurons, Alzheimer's eventually leads to significant loss of brain volume (known as brain atrophy) when left untreated, gradually robbing those afflicted by the degenerative disease of their cognitive functions and independence. Unfortunately, Alzheimer's is not a rare disease. Roughly one out of every three senior citizens die with some form of dementia and the likelihood of developing the disease doubles every 5 years after the age of 65. As life expectancy increases, a surge in Alzheimer's is underway and will only continue to gain momentum. The preclinical portion of gaining the Food and Drug Administration's (FDA) approval for new AD drugs can take up to 13 years to determine the efficacy and safety. It is during this time that 99% of Alzheimer's Disease drugs fail.

Neurofluidic Diagnostics (NFD) is an entrepreneurial project from the Kunze Lab at Montana State University. Recognizing the lengthy process of obtaining effective treatment as a prevalent issue, NFD sought to take action to expedite the process. Through tremendous effort, NFD has developed groundbreaking medical technology that eases the hurdles presented by developing treatments for Alzheimer's and other degenerative brain conditions. Neurofludic Diagnostics seeks to help test candidate drugs, allowing researchers to make better decisions before they enter preclinical testing, potentially saving time, money, and even lives.

To shorten the 13 year timeframe, Neurofluidic Diagnostics is developing a groundbreaking “nanotechnology-based finger-printing” methodology to refine preclinical drug discovery efficacy reports. Working with NFD, our mission is to develop a web-based application to allow researchers to utilize Neurofluidic Diagnostics’ services. This software is being developed under the brand Neurodynamik. The capstone software development team’s objective is to produce a website to allow customers to utilize NFD’s services. The software will also allow clients to interact with their results via data visualizations. Essentially, Neurodynamik will serve as a bridge between therapeutic drug developers and Neurofluidic Diagnostics’ methodologies. To address these customer needs, we allow the user to submit an order based on their desired therapeutic drugs, cell types of interest, and biomarkers, charging them according to their selections. Once the request is submitted and approved by NFD, the customer will follow safe shipping protocol and send their drugs to NFD’s lab to be tested. A portal allows NFD employees to view the orders and update the client throughout the processing of each order. This portal is built into Neurodynamik utilizing a secure user-account system for both employees and users to ensure that personal information is kept private. We utilize React for client-side user interface composition and MongoDb Atlas to store users, orders, and results. After the sample processing is complete, users are able to generate data maps and raw data spreadsheets based on a pre-selected list of disease markers provided by NFD. These markers use cutting edge research-based technology, and the user’s will have the ability to download and have complete ownership of their individualized data results. Data maps and easily accessible data play a critical role in drug efficacy reports, thus creating appealing and useful data visuals is paramount to the success of this project. It is crucial for the data to be easily understandable and meaningful to a wide range of user’s, regardless of their background.

Prior to our assistance, there was a gap between NFD and their consumers. Their services were not easily accessible nor well advertised. By creating an easily accessible, user-friendly, efficient, and secure interface, we have allowed Neurofluidic Diagnostics the ability to expand their reach, leading to the growth of their consumer base and enterprise as a whole. Neurofluidic Diagnostic's customers will be able to spend more time developing viable solutions and ultimately progress in the fight against Alzheimer's.

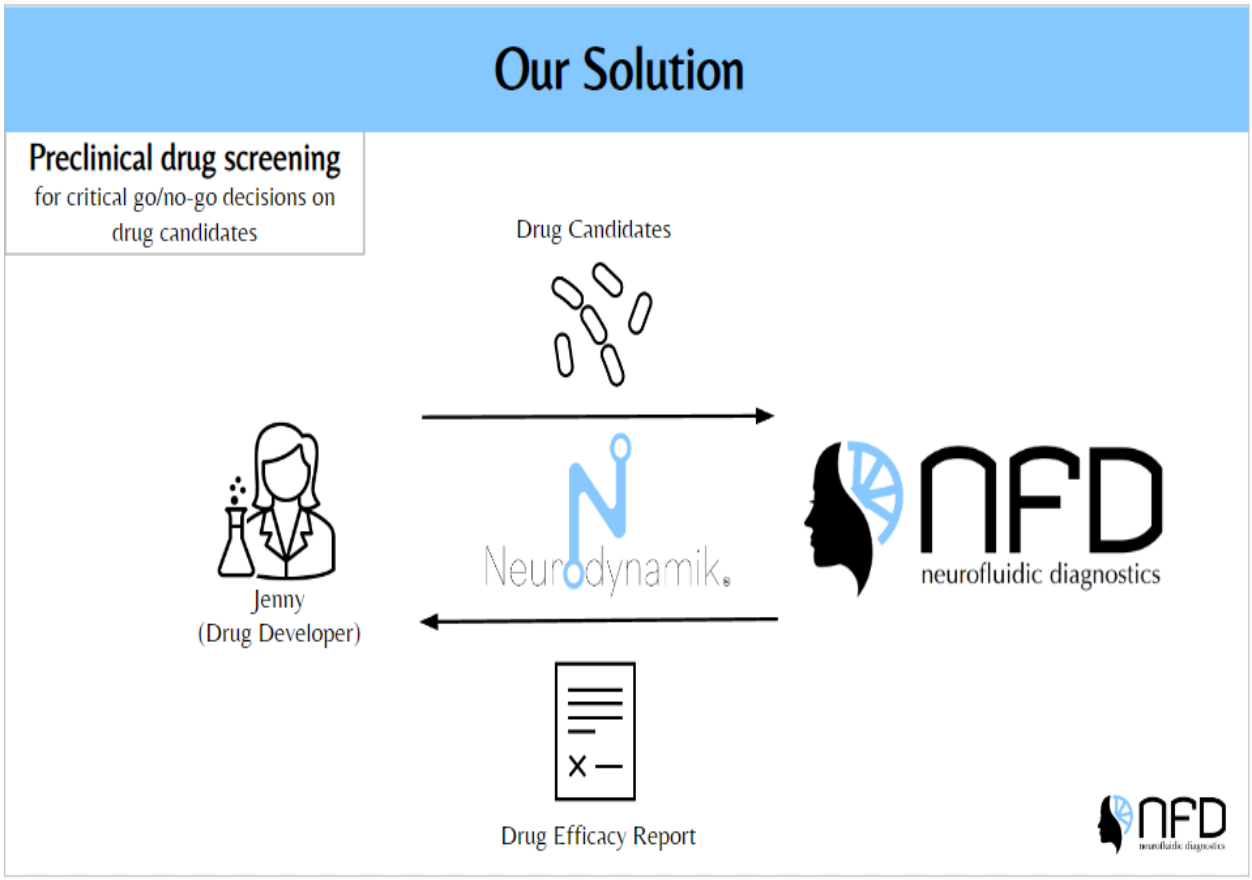
Background

Throughout this report there will be several important terms to understand. First, we will start by defining the actors in the process, which includes users, administrators, and customers. The Customer is any person or group of people who wish to use Neurofluidic Diagnostics' services. They are looking to get a drug tested using Neurofludic Diagnostics's proprietary technology and will be the primary recipient of NFD's service. The NFD Employee is anyone, or group that acts on NFD's behalf, to provide the information to the customer. They will have administrative access to the website and be able to act as a master account. The User is anyone using the web app, this includes both NFD employees and Customers. It is in reference to the collective person or group of people who intend to use the web app in the most general capacity. We also will require a Merchant Account actor who deal with processing and verifying payments, enabling customers to pay for NFD's services via the web platform.

Next we will define terms used to describe important and common processes. One of the most common processes is the Request. A Request is filled out by the Customer and includes the information needed by NFD to give the Customer the information they are looking for. This includes a set of Biomarkers (MicroRNA, Calcium Signaling, Extracellular Vesicles, and

Oxidative Stress), the cell types they wish for their drug to be tested on (Cortex, Hippocampus and Brainstem), the age of the cells (Number of Days in Vitro), and any safety information NFD needs to know about the drug (in terms of storage or danger to their employees).

Once this request has been verified by NFD and it has been paid for by the Customer, it becomes an Order. At this point the Customer will receive the go ahead to send their drugs to NFD. Once NFD receives the Order from the Customer they will update the status of the Order to keep the Customer in the loop. This will continue throughout the testing process until the final data is received and NFD posts the data to the Customer.



Qualifications

Connor Parrott

13334 W Hobble Creek Ct. Boise, ID 83713
(208) 809-7198
Email: connorparrott123@gmail.com
https://github.com/connorparrott/Sample-Work/branches

Intellectually Curious Computer Science student, with broad STEM background and experience, seeking internship in Robotics, Data Science and other Computer Science related fields.

EDUCATION:

Montana State University (Majoring in Computer Science minoring in Physics) 2021- Present
Data Structures and Algorithms, Software Engineering, Multivariable Calculus , Web Development, Biology II
GPA: 3.52
Current Expected Graduation Date May 2024

Associate of Arts in General Studies from Idaho State University 2018 - 2020
GPA: 3.88

WORK EXPERIENCE:

Hewlett Packard Intern 2018 - 2020

- ➢ Taught HP employees soldering, programing, 3D printing and other technical skills.
- ➢ Learned Python, C++ and implemented Arduinos and Raspberry Pi for testing.
- ➢ Translated HP printer testing algorithms to Python

Related Experience

First Robotics Competition Team Tators Robotics 2016 - 2020

- ➢ Gained hands-on mechanical design experience. Learned 3d modeling programs such as Solidworks, Onshape and Fusion 360.
- ➢ Responsible for the management of a mechanical design team, starting in 2017, of 2-3.
- ➢ Designed a variety of subsystems, one of the most successful being a custom vacuum module and robot lift that allowed for peak performance and robust failure prevention in competition.

Skills and Interests:

Proficient in MatLab, Python, Java, C/C++, JavaScript, HTML, Solidworks, Manual and CNC Machining, Physics. Interested in Astronomy, Problem Solving, Robotics and Mathematics

Henry Wright
henrywright302@gmail.com

5015 Larch Ave
Missoula, MT 59802

20 W Koch Street
Bozeman, MT 59715
(406) 552-3834

Education

MONTANA STATE UNIVERSITY Bozeman, MT
BS in Computer Science, Minor in Data Science May 2023
GPA: 3.71 / 4.00
Relevant Coursework: Honors Calculus I, Honors Calculus II, Discrete Structures, Data Structures and Algorithms, Computer Systems, Software Engineering, Computer Science Theory, Data Mining, Machine Learning, Computer Security

HELLGATE HIGH SCHOOL Missoula, MT
IB Diploma Recipient. ACT: 33 Math: 34 Science: 29 English: 34 Reading: 33 June 2019
National Merit Scholar Semi-Finalist. Member of Varsity Lacrosse, Basketball Team.
Sophomore, Junior Class President.

Technical Skills

Programming: Java, Python, C, C++, R, HTML, SQL
Operating Systems: Windows 10/8/7, MAC OS X, Linux

Relevant Projects

WEB ARCHIVER Java
Simple web archiver that downloads the contents of a given URL. Uses SHA-1 function to store archived websites under unique file names.

BATTLE BIT C
Recreation of the ‘Battleship’ board game. Rather than a standard 10x10 board, battleBit uses an 8x8 board to allow the board state to be stored in a set of 64-bit integers. Game is based on bit manipulation.

BASKETBALL PPG COMPARER Python
Given an input of two NBA basketball players, scrapes their career points per game average (PPG) from basketballreference.com and quizzes the user as to which player has a higher average.

VERTEX COVER/ INDEPENDENT SET Java
Given a graph, it utilizes an optimal solution seeking algorithm to determine the exact vertex cover and independent set, as well as greedy algorithms to quickly find the inexact vertex cover/ independent set.

Zachary D. Jewett

An overpriced apartment, Bozeman, MT, 5971 5
406.123.4567 - zdjewett@gmail.com - https://github.com/zdjewett

Education

Montana State University, Bozeman, MT *Computer Science Bachelor's* 2021 - Current

Great Falls College MSU, Great Falls, MT *Computer Programming Associate*'s 2019 - 2021

Experience

United States Postal Service, *Postal Support Clerk* June 2018 – August 2019
Great Falls, MT, 406.791.2597

Serving as a Postal Support Clerk, my main responsibilities included loading, unloading, and sorting various mail. I would sort packages, letters, and magazines to a vast number of towns throughout Montana and neighboring states. Trained as a General Expeditor, I was responsible for following a very strict schedule of arriving and departing delivery trucks; making sure every load was promptly and correctly handled and destined for the correct location. I also worked on mail sorting machines, a conveyor-belt system, and at manual sorting stations; often rotating through each position daily.
7+ more years of retail/logistic experience at businesses such as FedEx, Target, and Kmart.

What I've learned

Dependability – Being part of a very large team means depending on each other to complete work promptly and correctly. If one chain breaks it causes ripples throughout the entire organization. I strive to scrupulously complete every task I am assigned. By doing so, making my future assignments and my coworkers' responsibilities go that much smoother.
Presentation – You must look good to feel good. Having to communicate with hundreds of people every day taught me how important being well put together is. Having clean clothes and hygienic mannerisms helps me provide better service through increased confidence. Such cleanliness also reflects positively on the respectability of the company and my abilities to handle situations competently.
Pride – Reputation is crucial. It affects how people perceive you before they even meet you. Work provides its own influence on your reputation. A craftsman is judged on what they produce, and this pushes me to strive for an immaculate finish to any assignment. I want people to be impressed by what I can accomplish. Knowing I completed a job on time and under budget is extremely rewarding.

Related Projects

Great Falls Ice Coliseum Web App *JavaScript, PHP*
An original, from scratch project that served as a capstone project for programming associate degree. Features included encrypted login, account management, time-reactive calendar and events, dynamic page adjustments, searchable and filterable store, and many other standard website accouterments.

Paradox Interactive Modding *C++*
A perpetual hobby project for producing desktop game modifications that requires following their in-house Clausewitz Engine syntax in order to create, add, remove, and alter in-game events and images.

Addition Skills

American Sign Language, intermediate Swedish, USPS Certified Fork-lift Operator (PIV)

Work Schedule

Fall 2022 (August 25 - December 16)

- Week 1 - 4: Project Sponsor Presentations and Selection
- **Week 5 (September 25 - October 1): Abstract Submission**
- Week 6 (October 2 - October 8): NFD Meeting: Overall Comprehension
- Week 7 (October 9 - October 15): NFD Meeting: Biomarker Clarification
- **Week 8 (October 16 - October 22): Proposal Slide Check-in**
- Week 9 (October 23 - October 29): NFD Meeting: Nonfunctional requirements
- Week 10 (October 30 - November 5): NFD Meeting: Data Results and Graphs
- Week 11 (November 6 - November 12): NFD Meeting: Pricing Clarification
- Week 12 (November 13 - November 19): NFD Meeting: Faux Order Scenario
- Week 13 (November 20 - November 26): Fall break
- Week 14 (November 27 - December 3): Submit Written Proposal for Review
- **Week 15 (December 4 - December 10): Present Capstone Project**
- **Week 16 (December 11 - December 16): Present NFD with Prototype**

(December 17 - January 15) Winter Break; End of Semester

Spring 2023 (January 18 - May 11)

- Week 1 (January 18 - 21) Repository and file structure creation
- Week 2 (January 22 - 28) HTML and Database construction
- Week 3 (January 29 - February 4) Scripting Functionality requirements
- Week 4 (February 5 - 11) Test & debug Functionality requirements
- Week 5 (February 12 -18) Scripting non-functionality requirements
- Week 6 (February 19 - 25) Test & debug non-functionality requirements
- Week 7 (February 26 - March 4) Prototype design meeting with NFD
- **Week 8 March 5 - 11) 75k Challenge Application**
- Week 9 (March 12 - 18) Spring Break
- Week 10 (March 19 - 25) NFD requested tweaks
- Week 11 (March 26 - April 1) Optimization

- **Week 12 (April 2 - 8) Assumed Ruffatto Challenge (Canceled)**
- Week 13 (April 9 - 15) Code cleanup and explicit commenting
- **Week 14 (April 16 - 22) 75k Challenge Semi-finals (Top 16)**
- **Week 15 (April 23 - 29) 75k Challenge Finals (Top 8)**
- **Week 16 (April 30 - May 6) Final Prototype & Submission**

Timeline

Project Lifecycle Approach

We used the Agile approach for this project. We are justified in using Agile methodologies in our project lifecycle because we wanted to first and foremost prioritize stakeholder engagement and satisfaction. Through weekly meetings, we were able to achieve complete transparency and constant communication with our stakeholders regarding updates in requirements and changing expectations.. These frequent scrums kept NFD fully engaged in the process, and allowed our team to be incredibly adaptive to any necessary changes. This flexibility was another reason why we chose Agile, as we were able to quickly react to changes and alter our course without losing velocity. Going forward, as we begin software development, our Agile methodology will be justified for these same reasons. Utilizing sprints to create short-term deliverables will allow us to continue to have meaningful and frequent scrums, where we will continually showcase our progress ensuring stakeholder satisfaction. Furthermore, the feedback that we get from these meetings along with the short nature of our sprints will allow us to stay incredibly flexible.

We have also been careful to consider risks in security for both intellectual property and privacy to ensure both NFD and their customers are not at risk, by a weakness in our system. This is especially important as many drug candidates, whose data we are processing and storing, lack patents and are susceptible to being stolen if security is not taken into serious consideration. NFD is taking steps to protect their customers in the lab and we will ensure the customer’s safety online.

Proposal Statement

The Food and Drug Administration (FDA) must follow very strict safety protocols when determining the eligibility of new pharmaceuticals. Approximately 90% of all drugs and 99% of Alzheimer's Disease (AD) drugs fail preclinical trials. There are exponential amounts of chemical combinations that have unknown effects on the human body that have to be discovered, tested, and documented.

Neurofludic Diagnostics has developed a methodology that works with micropatterns, microchannels, and microfluids to test the requisiteness of new pharmaceutical drugs. Their mission is to alleviate the temporal and financial strain placed on institutions pursuing FDA approval.

Working with Montana State University's Kunze neuroscience lab and McCalla nanotechnology lab, we are developing a web-based application, Neurodynamik, to allow their technology to reach academic and commercial entities. The software serves as a bridge to allow these entities to test their drug candidates and ensure it has the efficacy rates expected before starting the long and expensive process of drug approval. Our goal is to make Neurodynamik appealing and accessible to ensure the best experience for Neurofluidic Diagnostics' customers. The software will allow the customer to quickly engage with NFD's service and their data results in a stress-free manner. User friendly interfaces and dynamic data visualization will allow customers to easily assess and interact with their results and apply them to drug efficacy reports.

Functionality Requirements

1. The user MUST be able to create an account and log into said account.
    a. The application MUST have a secure login.
    b. The application MUST differentiate between NFD employee users and customers.
2. The database SHOULD store user info including, company/institution name, contact info, name, email address, and customer ID.
3. The database MUST be able to generate and submit an order based on customer selections.
    a. The customer MUST be able to submit a safety data sheet (SDS).
    b. The customer CAN submit more than one drug candidate per order
4. The database MUST store therapeutic drug information including, drug name, drug ID, stock concentration, dilution solution (by volume or weight), and dilution series.
    a. The database SHOULD contain drug storage information.
5. The user MUST be able to select one, some, or all of three types of cells they wish for their drug to be tested on including: cortex, hippocampus, and brain stem.
6. The user MUST be able to select the cell ages (Day-in-vitro (DIV)) they want their drug to be tested on.
    a. The web application SHOULD default and constraint to a maximum DIV of 15.
7. The user MUST be able to select one, some, or all, of the following biomarkers to be used in their order: MicroRNA, Calcium Signaling, Extracellular Vesicles, Oxidative Stress.
    a. Regarding MicroRNA, the user CAN select a specific microRNA or the application SHOULD auto implement the default MicroRNA panel.

8. The web app MUST provide a final cost expected based on the user's request.
    a. Price MUST be a composite value decided by the number of drugs, how many concentrations of each drug, cell type selection(s), experiment length (DIV), sampling times (number of DIV collections), and biomarker selection(s).
    b. Price changes CAN be implemented by NFD employees.
9. Neurofluidic Diagnostics MUST be able to view order requests and download order details.
10. The customer SHOULD be able to view the status of their order.
    a. The customer CAN be notified of status changes to their order via email.
11. A NFD employee SHOULD be able to change an order's request status.
    a. Neurofluidic Diagnostics SHOULD change request status to "approved" or have "Clarification required" status.
        i. "Clarification required" CAN include messages from NFD.
    b. Neurofluidic Diagnostics CAN change request status to "samples received/processing".
    c. Neurofluidic Diagnostics CAN change request status to "completed."
12. A NFD employee MUST be able to upload completed data to the Web App.
    a. The web app SHOULD be able to display the correct order for the correct customer
13. A User CAN generate, view, and download data maps based on NFD's results.
14. A user CAN download the raw data from NFD's results in spreadsheet form or CSV form.

Performance Requirements

According to Akamai (Akamai 2017) 47% of users expect a webpage to load within 3 seconds. Shorter load times correlate to better customer conversion rates. For example, dropping from a 8 second load time to a 2 second load time improved customer conversion rate by 74%. Some components of our application will require payment processing, data map generation, and large database queries; processes that often exceed the desired 2 second window. Simple load animations such as progress bars or rotating indicators easily alleviate any apprehension the customer may have about a process possibly being frozen. The Doherty Threshold states that computer response time should be under 400 milliseconds to make the application more addictive.

Interface Requirements

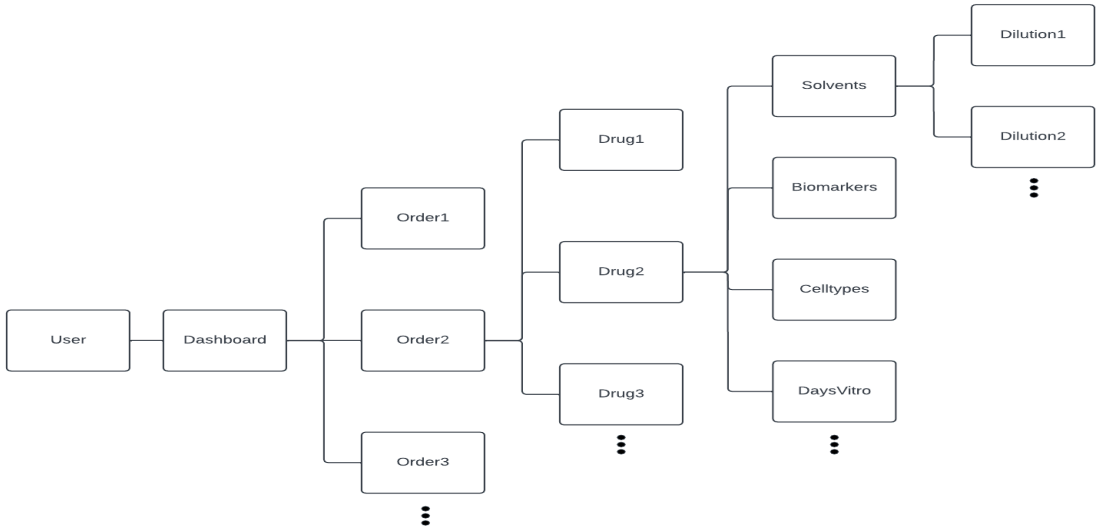
A user-friendly interface is a strongly desired component by Neurofluidics Diagnostics. The best way to meet this criteria is by utilizing an iterative user experience (UX) design thinking process. Particularly a process that includes the five steps: Empathize, Define, Ideate, Prototype, and Validate. The goal of this process is to find the intersections of usefulness, usability, and desirability to create a great iteration with the application. Producing a persona, business canvas models, mind-map, and prototypes allows us to summarize the results of user research into actionable qualitative and quantitative data. Following UX laws such as Fitts' Law, Common Region, Law of Parsimony, and Pareto Principles will make this application approachable and memorable.

An additional desired requirement for NFD was dynamic graph generation for the resulting data sets produced by their methodology. The graphs should also be intractable with selectable data points. The Chart.js library should be applicable to approach this requirement.. Furthermore, the graphs as well as the data should be exportable to PNG and CSV files respectable, or coupled as PDFs.

Architectural Design Documents

Methodology: Composite Pattern Summary

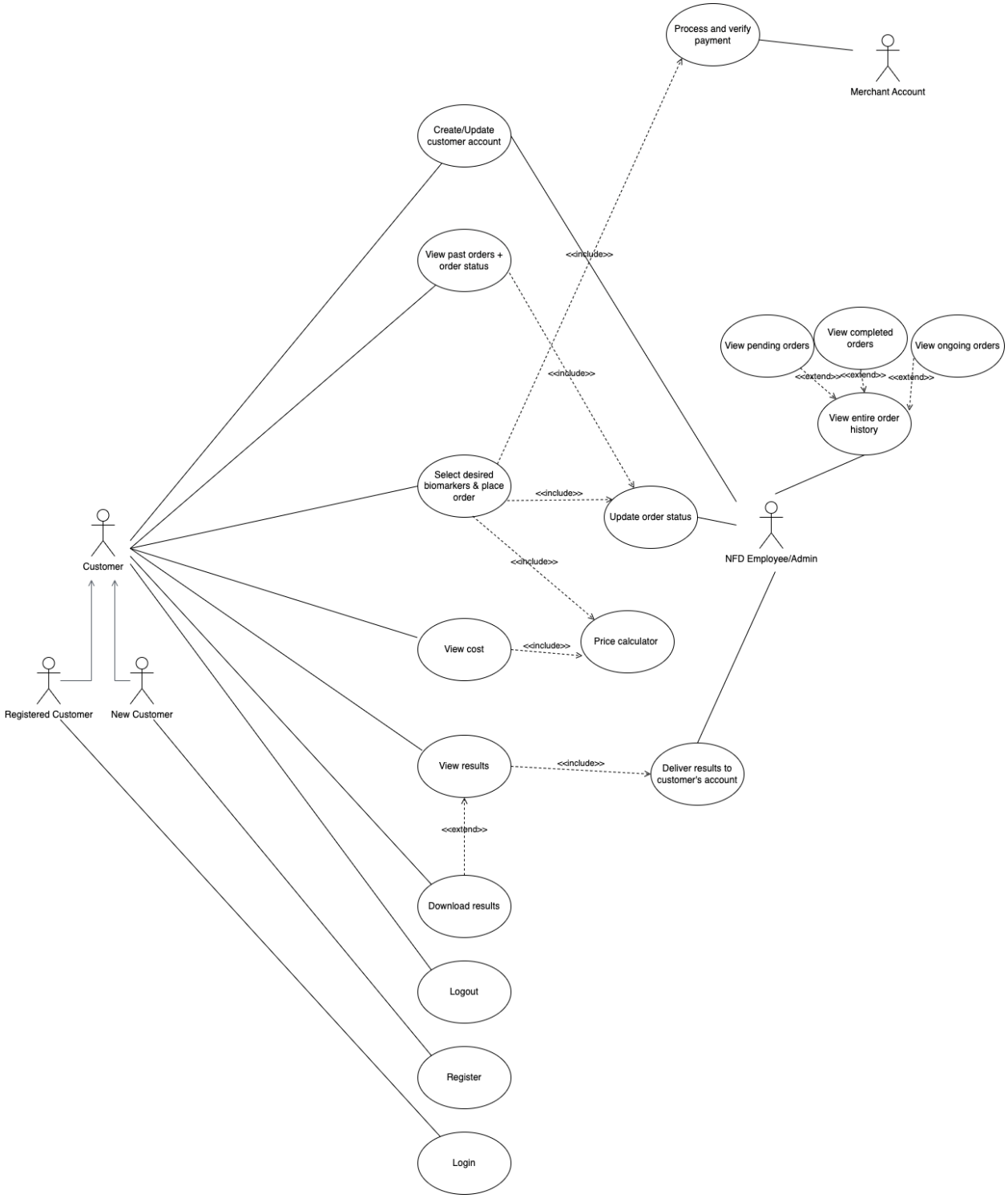
We have opted to use the Composite structural design pattern for the foundation of our core requirements. This creational pattern simplifies the creation of a complex object. The complex object for this project refers to a new order being submitted to NFD for approval. With the order having many parameters, with most being optional and having high degrees of variability, the tree-like hierarchy of the composite pattern will be a great asset. The composite pattern allows us to produce different types and representations of an object that can be treated in different ways with the same method. First we must define the composite ("trunk"), the sub-elements with children ("branches"), and the sub-elements with no children ("leaves"). The client only interacts with the composite component (Order) which receives parameters from the branches (Drugs) that fetches parameters returned by the leaves (Biomarkers, CellTypes, etc.)



Design Pattern Trade Offs

Advantages	Disadvantages
Scalable - easy to add new objects or groups to the tree-like structure, offering flexibility and expandability	**Overgeneralization -** harder to restrict classes with large functionality difference
Encapsulation - easy to manage and maintain objects and groups.	**Performance -** rely on run-time checks and can increase memory usage
Simplification - treats objects and groups uniformly	**Implementation -** initial complexity increases with each object

Use Case Diagrams



As a customer, I want to make a request so that I can test my drug before it goes to human trials.

1. Make Request
    a. Brief description/Goal:
        i. Collect/Enter drug test information needed by NFD and wanted by the customer.
    b. Related requirements:
        i. User Story 2, 3, 4, 5, 6, 7, 8
    c. Preconditions:
        i. Customer has an account and is logged in.
    d. Successful end condition:
        i. The customer submits the request to be reviewed by NFD
    e. Failed end condition:
        i. The customer cancels their request or NFD rejects their request
    f. Actors:
        i. NFD and the Customer
    g. Basic flow of events:
        i. Customer creates new request
        ii. Customer fills out request
        iii. Customer Submits request
        iv. NFD sends response
    h. Extension/Exception flow of events
        i. Customer creates new request
        ii. Customer fills out request
        iii. Customer Submits request
            1. Customer cancels their request
        iv. NFD accepts request
            1. NFD rejects request

As a Customer, I want to download the test results so that I can perform my own data analysis

1. Download Results
    a. Brief description/Goal:
        i. Download Raw data for further analysis
    b. Related requirements:
        i. User Story 1, 2, 3, 4, 5, 6, 7, 8, 14
    c. Preconditions:
        i. Customer Submitted a Request
        ii. NFD Approved Request
        iii. NFD Completed Request
    d. Successful end condition:
        i. Customer downloads raw data
    e. Failed end condition:
        i. Customer is unable to download raw data, no data to download
    f. Actors:
        i. NFD and the Customer
    g. Basic flow of events:

i. NFD sends the completed request to the Customer
ii. The Customer downloads the data

h. Extension/Exception flow of events
i. NFD sends the completed request to the customer
1. NFD fails to send the results to the customer
ii. The Customer downloads the data
1. The Customer is unable to download the data

As a user, I want to make an account so that information on my drug is secure and visible only by me.

1. Make Account
a. Brief description/Goal:
i. Sign up with personal information and create a username/password combo
b. Related requirements:
i. User Story 1, 2, 4, 10, 11, 12
c. Preconditions:
i. The user wants to do business with NFD
ii. Successful end condition:
iii. The user has created an account
d. Failed end condition:
i. The user is unable to create an account
e. Actors:
i. The user
f. Basic flow of events:
i. User enters personal/business info
ii. User enters username and password.
g. Extension/Exception flow of events
i. User enters personal/business info
ii. User enters username and password
1. Username or password are unavailable/do not meet security requirements.

As a Customer, I want to view the status of my request, so that I can see the progress being made and stay informed about the drug testing process.

2. View Results Online
a. Brief description/Goal:
i. On the customer's profile, they can view their entire order history, as well as the associated, up-to-date status of the order: 'Order Placed', 'Order Processed', or 'Order Complete'.
b. Related requirements:
i. User Story 11, 12, 14
c. Preconditions:
i. The Customer has created an account
ii. The Customer has submitted a request.
d. Successful end condition:

i. Customer is able to navigate to and view the status of their order

e. Failed end condition:
   i. Customer cannot view the status of their order
f. Actors:
   i. NFD, Customer
g. Basic flow of events:
   i. Customer makes a request
   ii. User selects the desired biomarkers that they wish to receive data for
   iii. Customer is charged for their results and their payment is verified
   iv. Their order is placed
   v. The Customer receives confirmation for their order and is updated throughout the process as they await their results. The status of their order will follow the following steps:
      1. First, their order is placed and is pending. The status will read 'Order Placed'
      2. Next, their order is processed or currently being processed. Status: 'Order Processed'
      3. Finally, their order will have the status 'Order Complete' when their results are available for viewing.
   vi. In X business days, the Customer's results are published to their profile and are able for viewing/downloading
h. Extension/Exception flow of events
   i. Customer places an order
      1. Customer cancels their order
   ii. The customer's profile displays their updated order history, with the new order having the status 'Order Placed'.
      1. Customer cancels order
   iii. NFD receives the order and updates the customer's specific order status to 'Order Processed'
   iv. NFD obtains the results for the customer
      1. NFD encounters an error while determining the results
   v. NFD sends the results to the customer, and updates the customer's specific order status to 'Order Complete'.

As a NFD employee, I want to upload data so that I can give the customer the results of the tests on their drug online.

3. View Results Online
   a. Brief description/Goal:
      i. NFD employees have the ability to upload the results of a customer's order
   b. Related requirements:
      i. User Story13, 14
   c. Preconditions:
      i. Customer has created an account and has placed an order
      ii. Customer's payment has been verified

iii. NFD has processed the customers order and has generated the results

d. Successful end condition:
   i. NFD employee is able to upload the customer's results so that the customer can view the results from their profile
e. Failed end condition:
   i. NFD is unable to upload the results, the customer is unable to get the results that they paid for
f. Actors:
   i. NFD, Customer
g. Basic flow of events:
   i. NFD employee uploads results to database with the customer's specific identification number associated with the results
   ii. Order status is updated to 'Completed'
   iii. Customer logs on to their account and is able to view their personalized results
h. Extension/Exception flow of events
   i. NFD employee processes the customer's results and appends them to the database
   ii. Customer has access to their results, and can view the graphical representation on their profile as well as download the raw data
      1. Customer is unable to access their results

As a NFD employee, I want to update the request status so that I can change the status of the request to keep my customer informed.

4. Update Request Status
   a. Brief description/Goal:
      i. Keep user who has placed an order "in the know" about the status/ timeline of their order from the moment the user places an order to the moment that their results are delivered
   b. Related requirements:
      i. User Story 11, 12, 13
   c. Preconditions:
      i. Customer has a desire to do business with NFD
      ii. Customer has an account created
      iii. Customer has placed an order for results and their payment has been verified
      iv. Customer is awaiting their results
      v. On the Customer's profile, they have a view of their order history, which has an associated status
      vi. NFD employee has access to all Customer orders
   d. Successful end condition:
      i. Customer's order status is continuously updated as needed throughout the entire order cycle, keeping the user informed and engaged in the process
   e. Failed end condition:
      i. Customer's specific order status is not updated and the Customer is not kept in the loop about their order

f. Actors:
    i. NFD, Customer
g. Basic flow of events:
    i. Customer creates/logs onto their account
    ii. Customer selects their desired biomarkers that meets their specific needs
    iii. Customer places an order
    iv. Customer pays for their order and their payment is verified
    v. Customer now is able to view the information of their latest order (as well as past orders), and can also view the order's associated status
    vi. Based on the status of the order, the status is automatically updated to read: 'Order Placed', 'Order Processed', 'Order Complete', as well as an estimated completion date (e.g. 2-3 business days).
h. Extension/Exception flow of events
    i. Customer places an order
        1. Customer cancels their order
    ii. The customer's profile displays their updated order history, with the new order having the status 'Order Placed'.
        1. Customer cancels order
    iii. NFD receives the order and updates the customer's specific order status to 'Order Processed'
    iv. NFD obtains the results for the customer
        1. NFD encounters an error while determining the results
    v. NFD sends the results to the customer, and updates the customer's specific order status to 'Order Complete'.

As a Customer, I want to view the cost of my drug tests, so that I can ensure I can ask for proper funding for my drug.

5. View Cost
    a. Brief description/Goal:
        i. Customer is able to view the pricing of their orders
    b. Related requirements:
        i. User Story 8, 9
    c. Preconditions:
        i. Customer is interested in doing business with NFD and have an account created
        ii. Customer is able to select the desired biomarkers that they wish to receive data for
        iii. Pricing calculator/ algorithm is implemented
    d. Successful end condition:
        i. Customer is able to see an accurate pricing based on their specific potential order
    e. Failed end condition:
        i. Customer is unable to see a price for their selected biomarkers
        ii. Alternatively, the customer is presented with an inaccurate price
    f. Actors:
        i. NFD, Customer, payment merchant account

g. Basic flow of events:
    i. Customer creates/ logs onto their account
    ii. Customers selects their desired biomarkers, determining the price of their order which is displayed on their screen. As they select/deselect biomarkers, the associated price is updated accordingly
    iii. Customer goes to place their order and they are charged the correct, previously displayed price
h. Extension/Exception flow of events
    i. Customer accesses their profile and goes to place an order. They select their desired biomarkers
    ii. An accurate pricing is displayed in accordance with the biomarkers being selected/deselected
        1. Pricing is inaccurate

As a Customer, I want to pay my bill so that I can pay for the service and information I am getting from NFD.

6. View Results Online
    a. Brief description/Goal:
        i. Customer is able to pay for their order
    b. Related requirements:
        i. User Story 7, 8, 9
    c. Preconditions:
        i. Customer is interested in doing business with NFD, has an account created, has internet access
        ii. Pricing calculator is implemented and functioning correctly, with the ability to correctly determine the price of a unique order
    d. Successful end condition:
        i. Customer is able to pay for their order
    e. Failed end condition:
        i. Customer's payment is unable to be processed when it should be
        ii. Customer's payment is not equal to the price that it should be based on their unique order
    f. Actors:
        i. NFD, Customer, payment processing merchant account
    g. Basic flow of events:
        i. Customer creates/logs onto their account
        ii. Customer places their order, consisting of their uniquely selected biomarkers
        iii. Customer pays (the correct price) for their order, and the validity of their payment is assessed
    h. Extension/Exception flow of events
        i. Customer selects their desired biomarkers and goes to check out
        ii. Customer pays for their order
            1. Customer is charged an inaccurate amount
            2. Customer is unable to pay due to an error on NFD's side

As a NFD employee, I want to view the complete list of order so that I can show a list of all the orders that are ongoing or completed

7. View List of Orders
    a. Brief description/Goal:
        i. For each unique order placed (across all customers), store order information in database, which can only be accessed by NFD employees/Admins and the user associated with the order
    b. Related requirements:
        i. User Story 2, 3, 4, 10
    c. Preconditions:
        i. Customer is interested in doing business with NFD, has an account created, has internet access
    d. Successful end condition:
        i. Entire history of orders is accurately stored in the database, which is updated to accuracy and is accessible by NFD employees
    e. Failed end condition:
        i. Database contains inaccurate account of orders
        ii. Database privileges are inaccurate
    f. Actors:
        i. NFD, customers
    g. Basic flow of events:
        i. Whenever a customer submits an order, it is added to a database and regularly and accurately updated for changing values (ex. Status, date_delivered).
        ii. NFD employee/Admin logs onto their account and can view a graphical representation of the database.
        iii. If they want to view all completed orders, they can add a filter to the database by clicking ‘Completed’. Alternatively, if the employee wants to see all orders that are ongoing, they may click the ‘Ongoing’ filter
    h. Extension/Exception flow of events
        i. NFD employee logs onto their account
        ii. NFD employee views all of the orders
            1. Orders are inaccurate (e.g. not up-to-date)

As a NFD employee, I want to view the pending orders separately, so that I can punctually deal with any new orders that need my attention.

8. View Pending Orders
    a. Brief description/Goal:
        i. NFD employees have the ability to separately view all new orders that have not yet been processed
    b. Related requirements
        i. User Story 10
    c. Preconditions:

i. Customer is interested in doing business with NFD, has an account created, has internet access
ii. Database is implemented and accurate up-to-date

d. Successful end condition:
i. NFD employee is able to view the accurate up-to-date pending orders separately from other types of orders

e. Failed end condition:
i. NFD employee views inaccurate list of pending orders
ii. List of pending orders is able to be viewed by those who should not have privileged access (e.g. customer is able to view all pending orders, even those which are not their own).
iii. List of pending orders is unable to be viewed

f. Actors:
i. NFD, customer

g. Basic flow of events:
i. Whenever a customer submits an order, it is added to a database initially set as a 'Pending' order
ii. NFD employees/Admins can log onto their account and view a graphical representation of all Pending orders, accurate up-to-date and separate from all 'Completed' and 'Ongoing' orders

h. Extension/Exception flow of events
i. NFD employee logs onto their account
ii. NFD employee views all of the orders
1. Orders are inaccurate (e.g. not up-to-date)

As an NFD employee, I want accounts to be authenticated so that the correct information is given to the correct user.

9. Authenticate User
a. Brief description/Goal:
i. Check whether the account is an employee or a customer and depending on what information that user is privy to (based on what they've paid for) ensure they have the correct permissions.

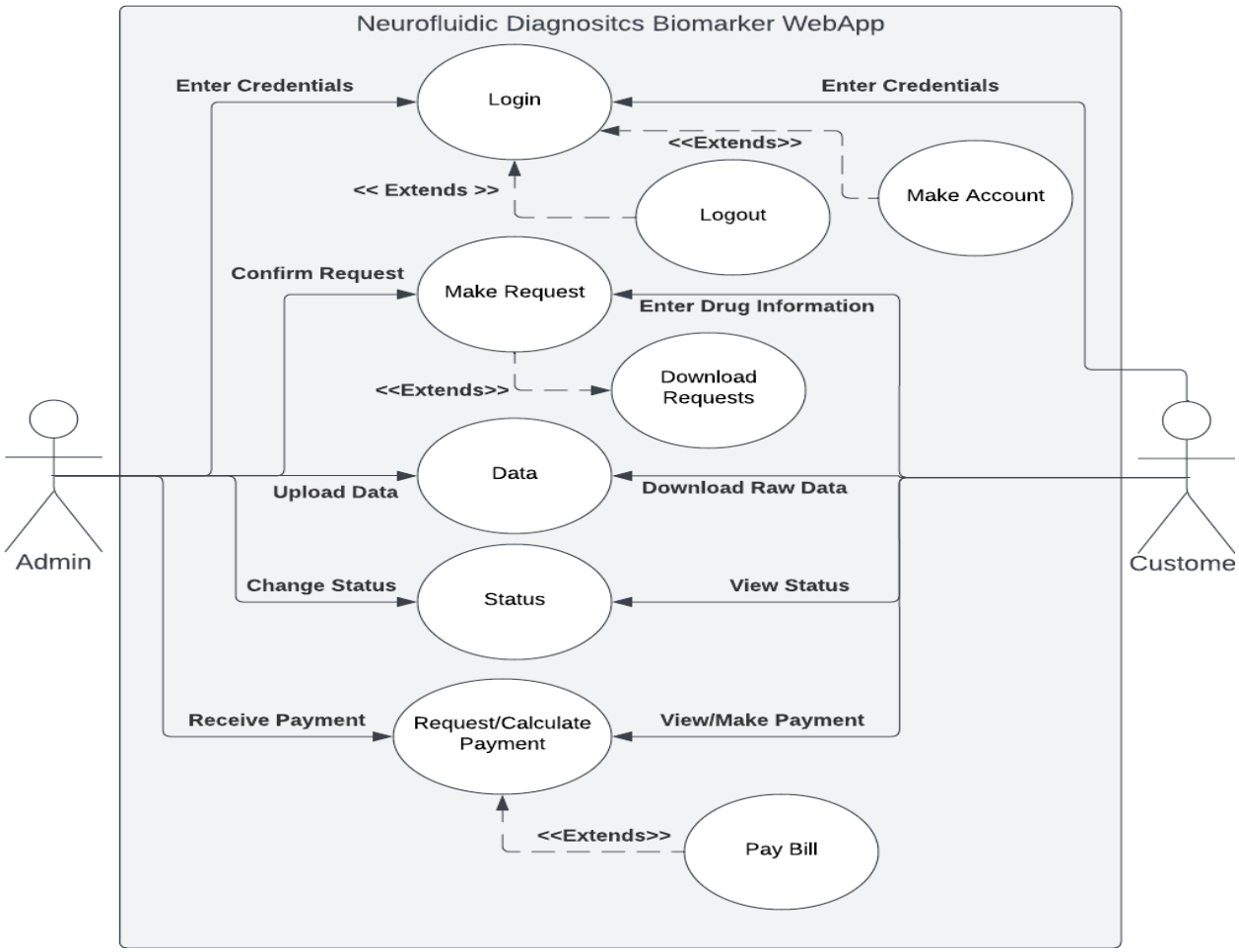
b. Related requirements:
i. User Story 1, 2, 3, 13

c. Preconditions:
i. Customer is interested in doing business with NFD, has an account created, has internet access
ii. Database of company's entire order history is accurate and up to date
iii. Customer has placed an order

d. Successful end condition:
i. A given account is correctly verified as being a employee's account or a customer's account, and account holds the correct privileges

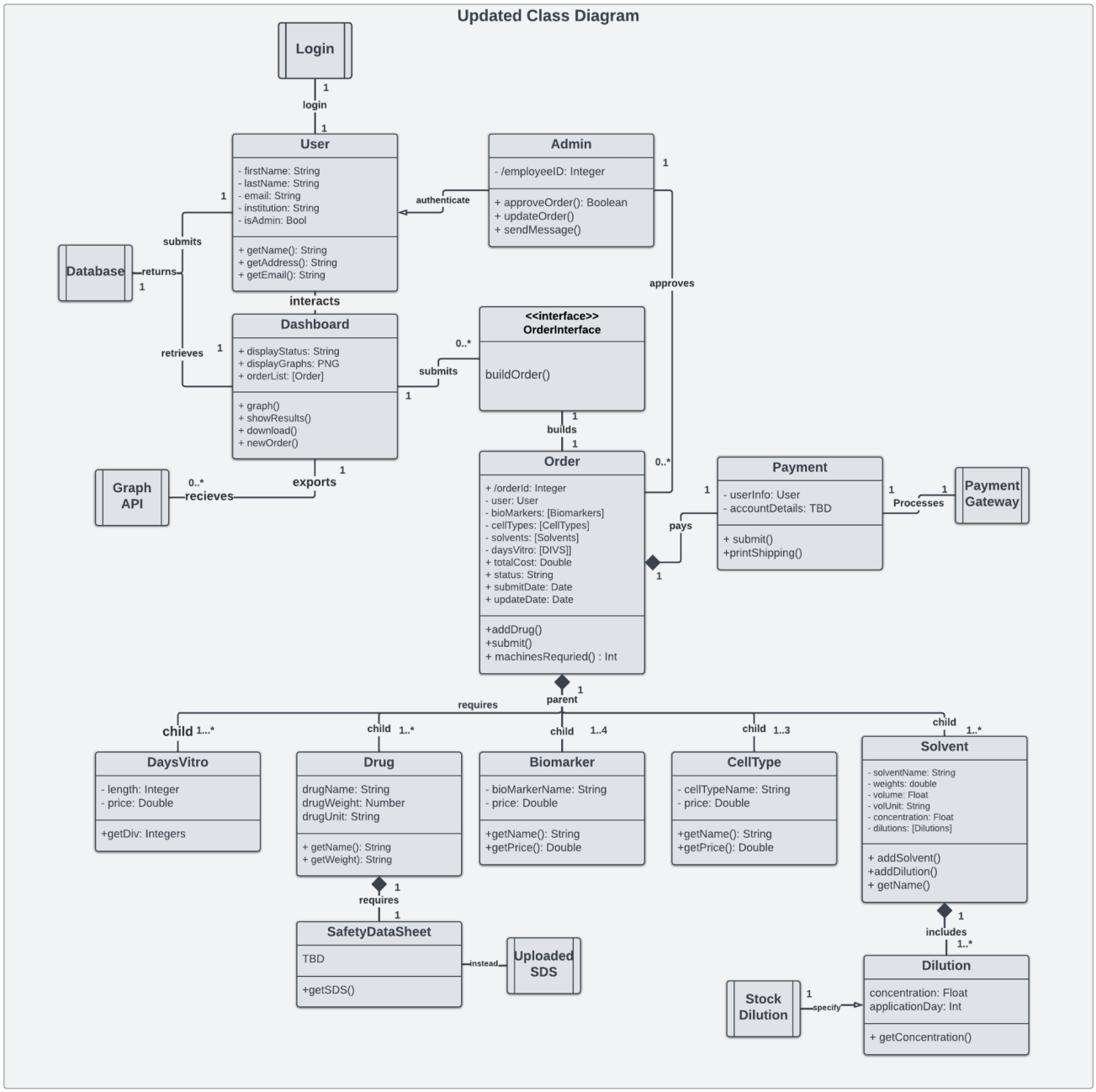
ii. The correct results are provided to the customer. The customer does not receive another customer's results. The customer only will receive results that are personalized for that specific customer.

e. Failed end condition:
   i. A given account is incorrectly verified; account holds incorrect privileges
   ii. A customer receives somebody else's results
f. Actors:
   i. NFD, Customer
g. Basic flow of events:
   i. A customer's account is correctly identified as being a customer, and they are provided with the associated privileges.
   ii. A customer has an associated identification number associated with their account and their orders, ensuring that they will receive their order and their order only.
   iii. In the event of a new NFD employee/ Admin, NFD employees will have the ability to elevate the permissions of a customer's profile so that the customer now is a new NFD employee/Admin.
h. Extension/Exception flow of events



Class Diagram

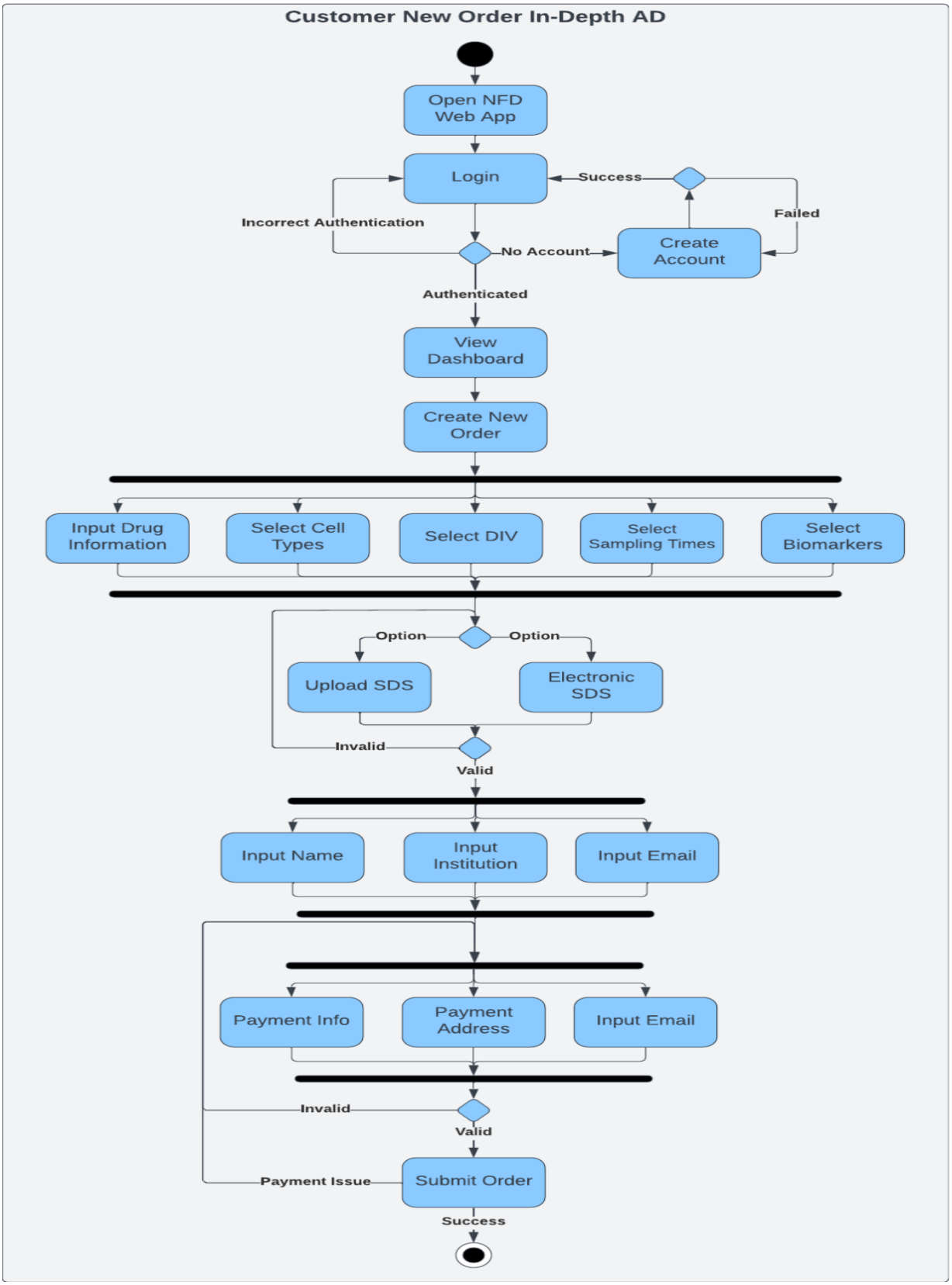
Class Diagram 1: *A class diagram that describes the relationships between User, Dashboard and Order objects while submitting a new order. In this instance, we are implementing a Composite design pattern to simplify the construction of a new Order(). This design pattern handles the large number of optional parameters that will compose an order. It provides an abstracted step-by-step methods to build the object and provide a method that returns the completed instance.*



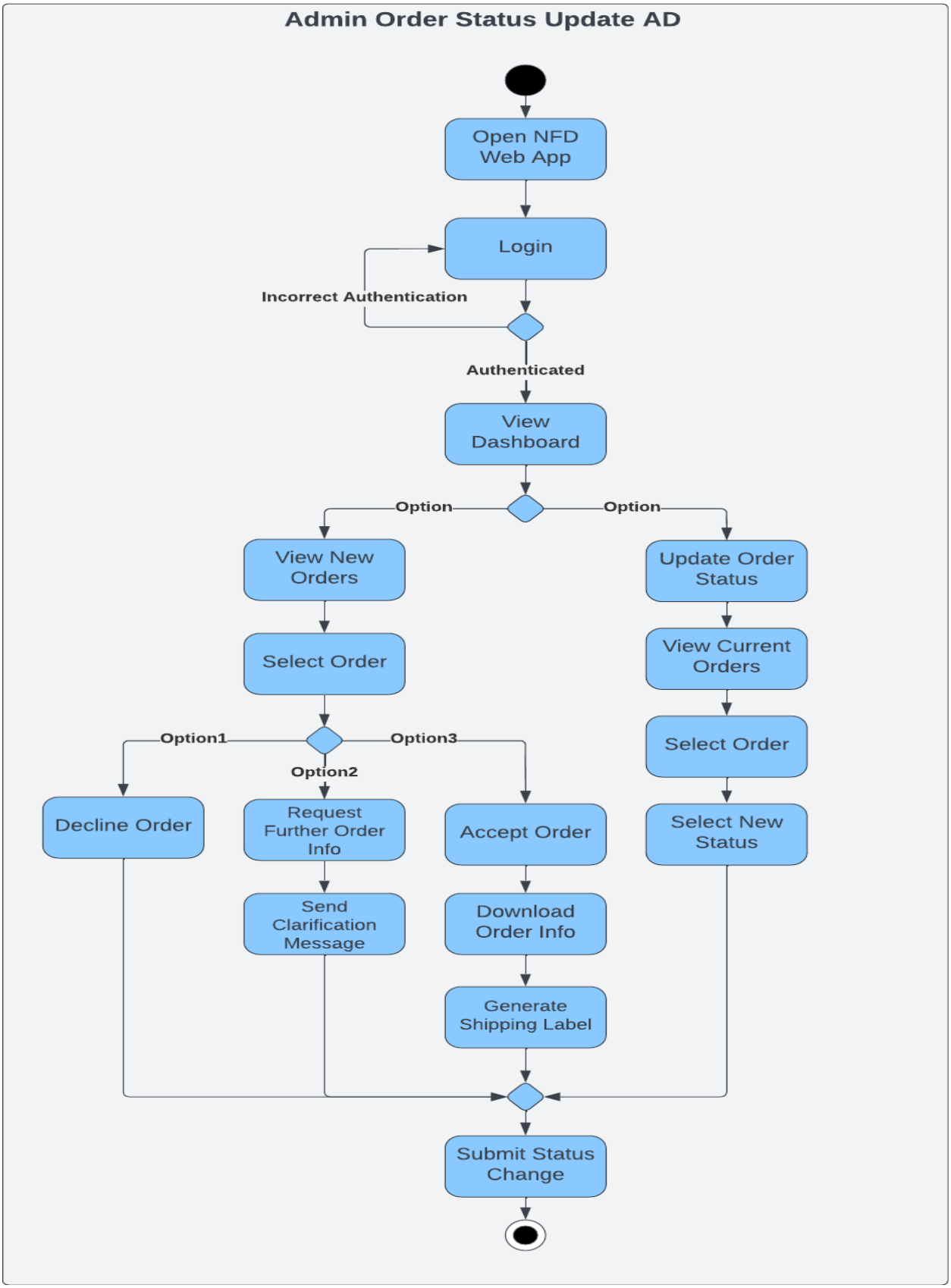
*The existence of setter methods is implicit for use with getters.

Activity Diagrams

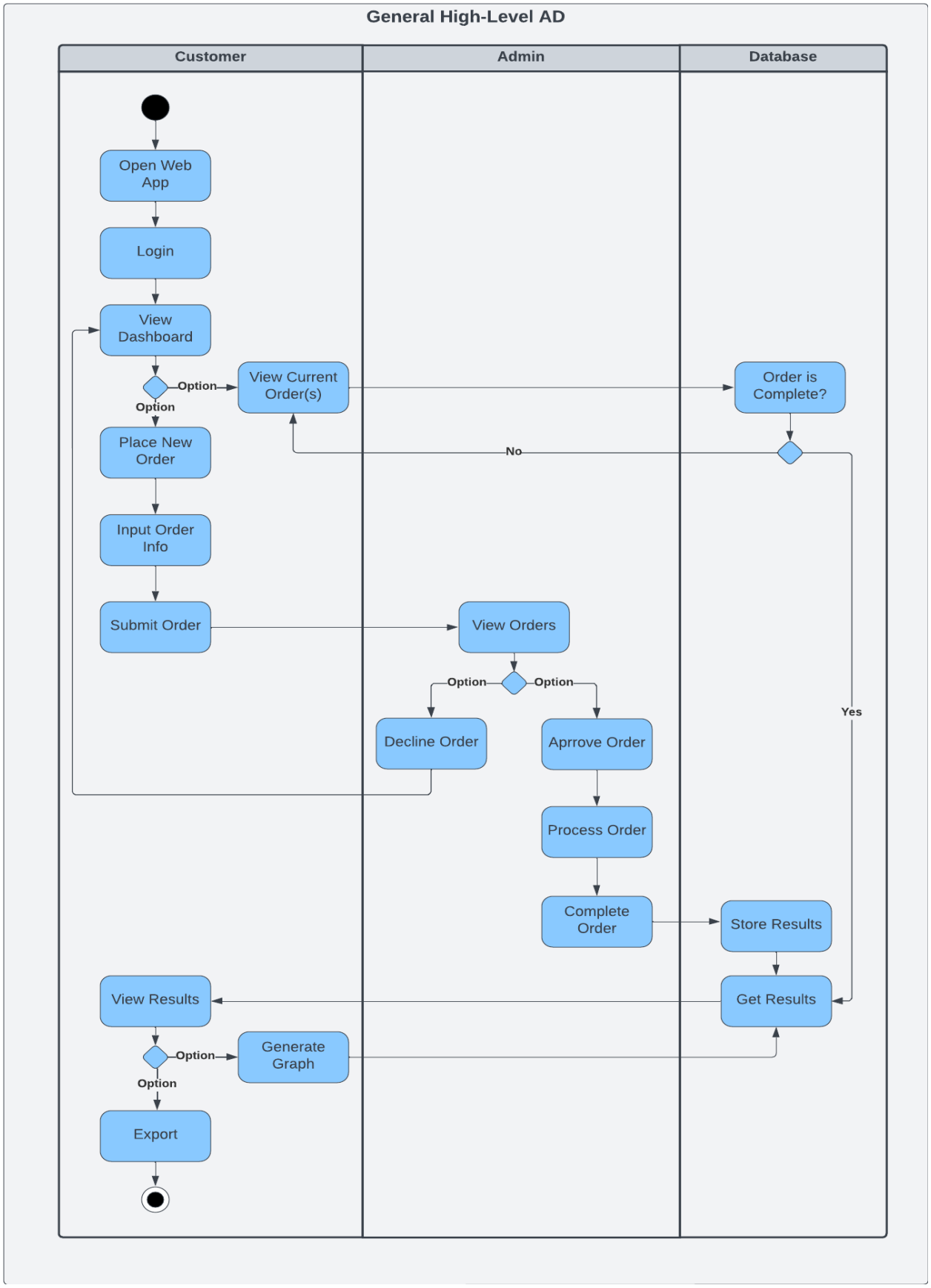
Activity Diagram 1: *This diagram follows a customer's path through the process of submitting a new order to Neurofludic Diagnostics.*



Activity Diagram 2: *This diagram follows an administrator (NFD employee) through the process of approving new orders and/or updating the status of current orders.*

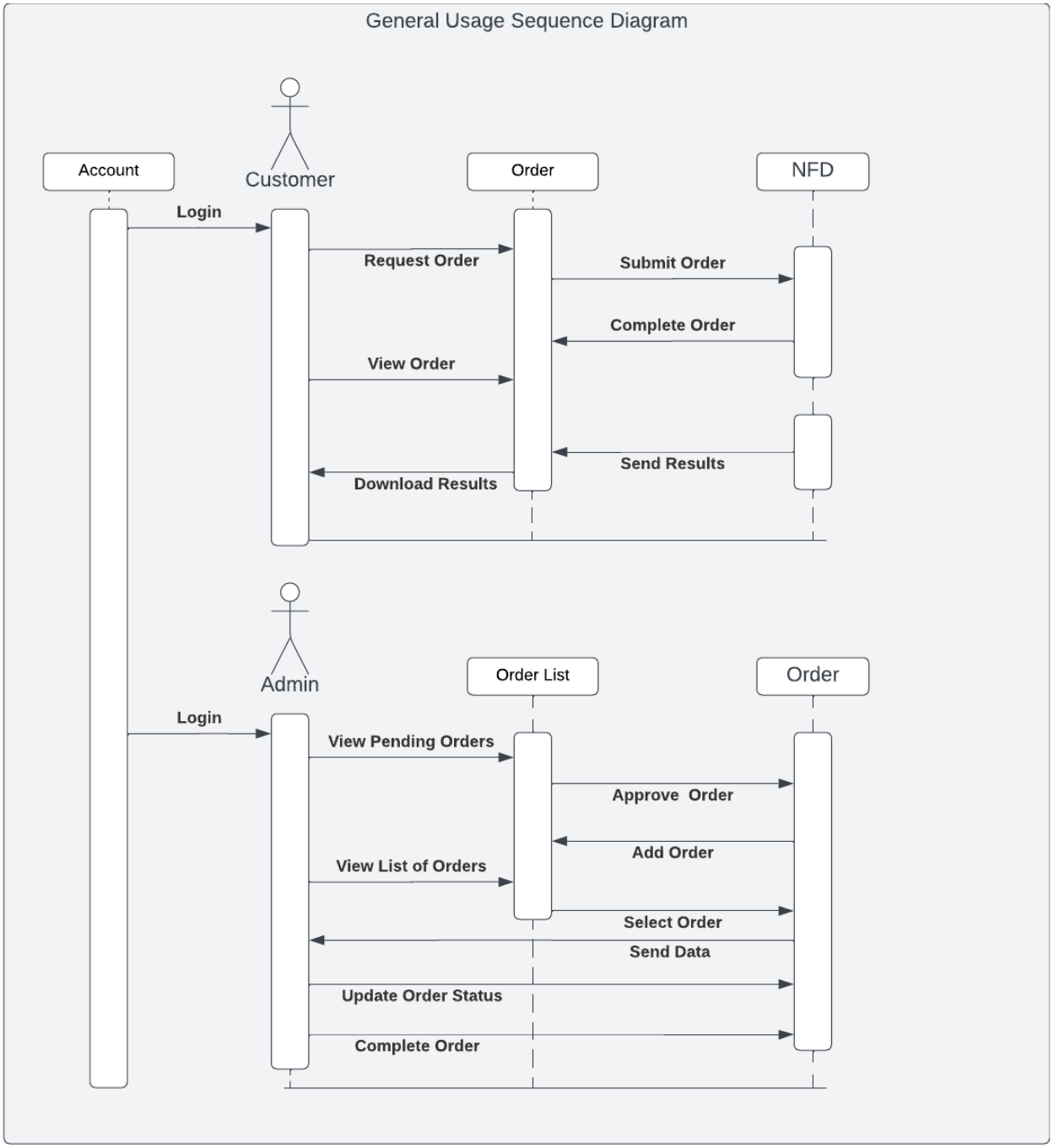


Activity Diagram 3: *This diagram provides a simplified version of how activities are shared between actors when using the NFD application.*



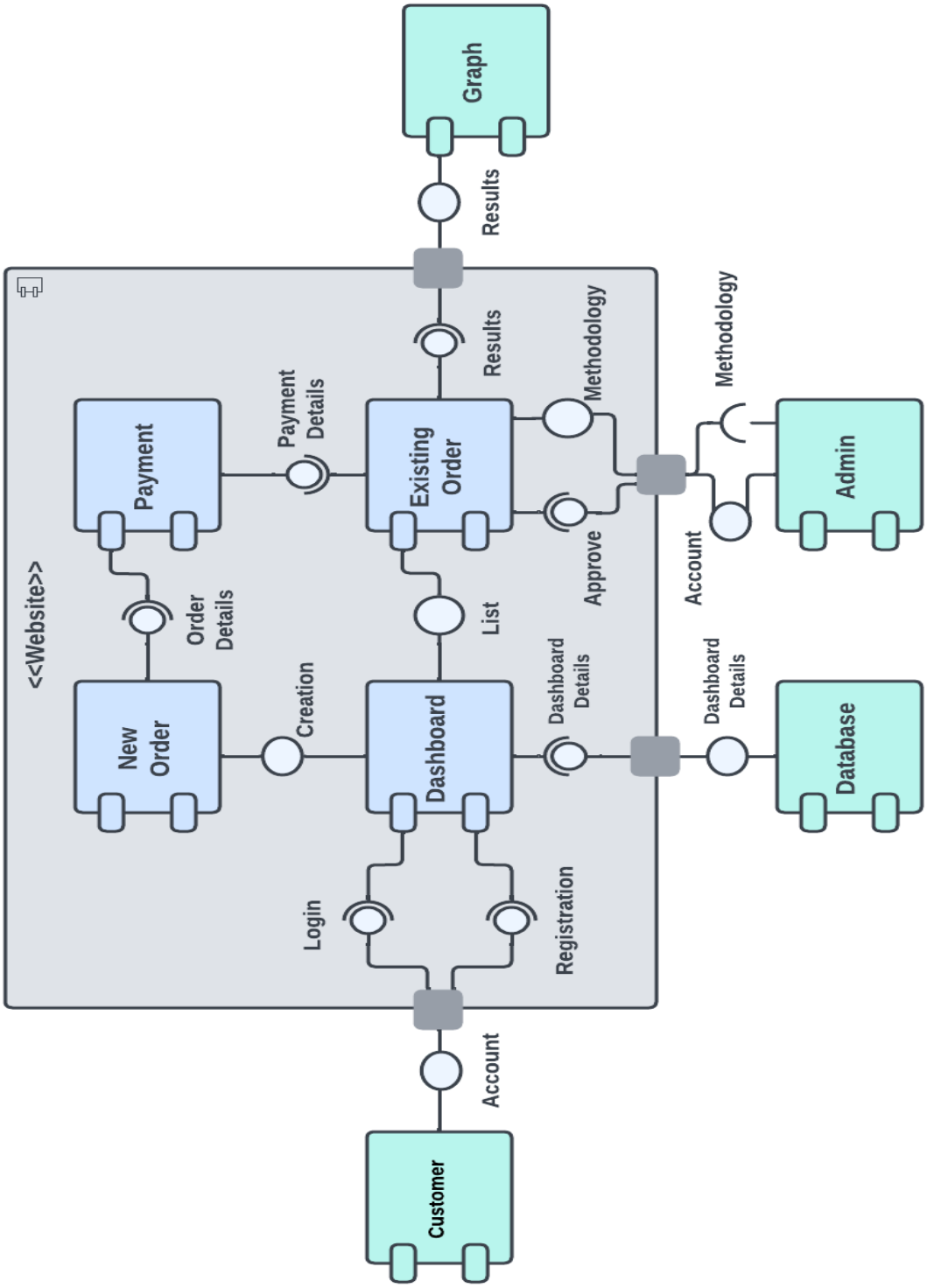
Sequence Diagrams

Sequence Diagram 1: *This sequence diagram couples two high-level instances of usage. The top shows a Customers sequence of events through placing a new order, while the later depicts an Admins acceptance of a new order.*



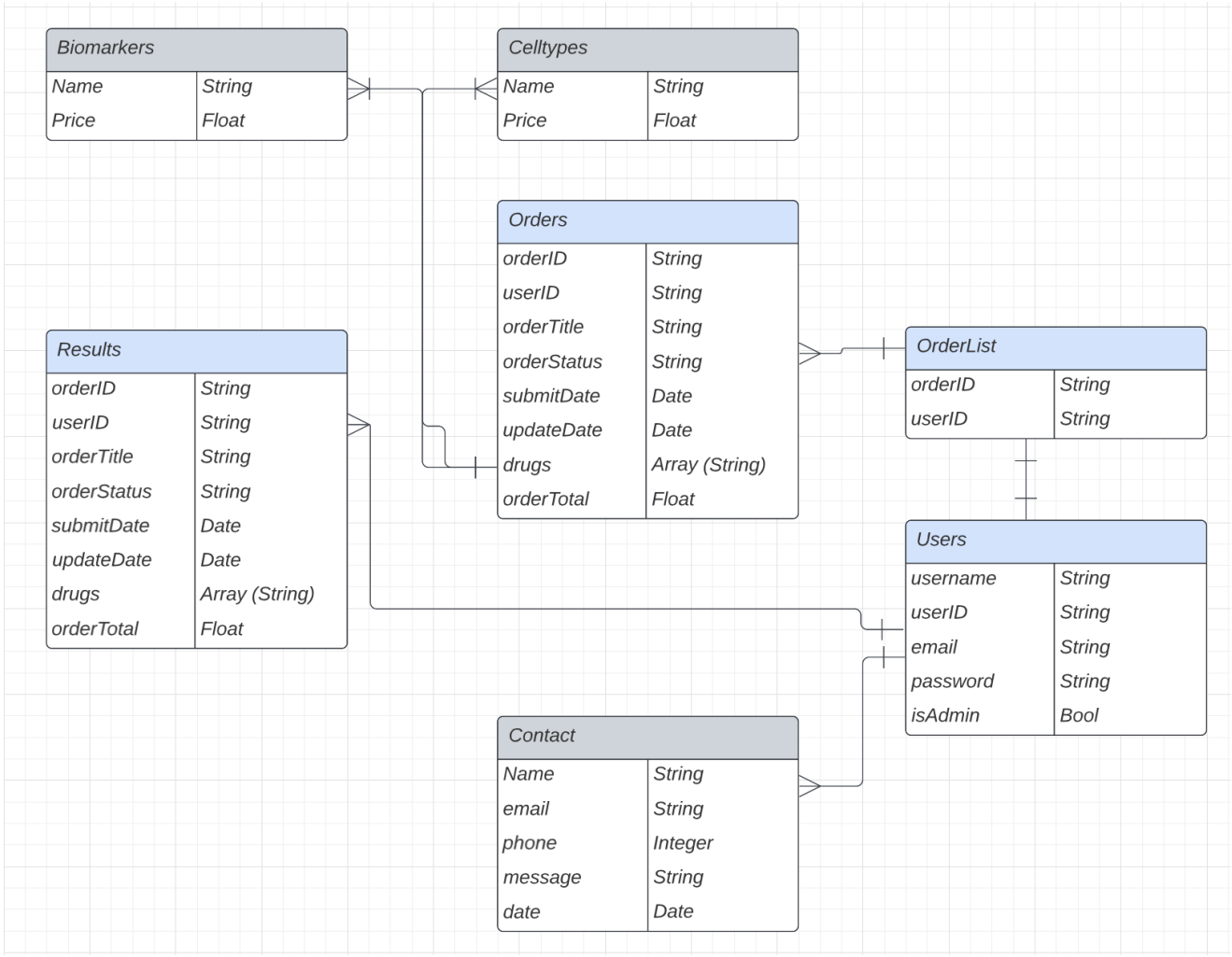
Component Diagram

Diagram describing the organization and dependencies of the internal software components and external actors



Database Entity Relationship Diagram

Diagram showing the structure and relationships between different components of the database.



Development Standards

Tools:

HTML & CSS - Essential webpage creation standardized languages

React.js - Client-side, component-based scripting language for dynamic content control

Node.js - run-time environment to build server-side applications

MongoDB - Source-available cross-platform document-oriented database program

Github - Version Control and Git management for application code collaboration

Diagrams.net & Lucidchart - Diagram and UML creation tools for documentation

Adobe XD - High-fidelity prototyping tool for mock ups and design

Packages:

Express - framework for Node.js for creating client-side web apps

Axios - promise-based HTTP client for making asynchronous HTTP requests

Mongoose - ODM for database structure schemas and models

CORS - allows apps running on one domain to securely access resources from another domain.

Bcrypt - password hashing functions

Dotenv - environment variables handling

Standards:

ISO 12207: Software Life Cycle Process - The most significant standard for software engineering. This standard expresses the importance of deciding methods, activities, and tasks involved in the project based on consultation. It also handles the assignment of accountability and explanation of tasks in order to meet minimum requirements.

ISO 29119: Testing - This standard represents the coding tenant for risk mitigation and avoidance. Steps include, testing based on keywords, documentation, testing techniques, and testing definition and concepts. Coupled with risk reports testing reports include testing case specifications and defect reporting with possible acceptance criteria.

ISO 27001: Information Security - This standard's purpose is to "secure company asset's and enhance its security procedures." (Bak 2022). Identification, classification, and labeling of core information assets and who has access to them is the initial step in this standard. Identification is followed by protection mechanism development. These mechanisms are iteratively tested with acceptance criteria throughout agile development.

Appendix

Documents provided by Neurofludic Diagnostics stakeholders throughout the project.

Client Document: Summary

CS Capstone Project Summary

Goals (from Anja):

1. Understand the data, its structure and come up with an easy to use, and efficient file format for data storage, transfer, and upload to a desktop app.

2. Develop a desktop app that can be integrated into a webpage, with login/logout, and user-specific functions/drop down menus based on different service requests.

3. Have a demonstrable software ready in March 2023, be able to present it at the Ruffato (at UofMontana), and 75k Challenge (MSU).

App requirements and User/Company interaction through the app

1. User logs into the account and starts a new request for services.
    a. Account will store user's info (company/institution name, contact info)
2. User is prompted to enter details (drop-down menus?):
    a. Therapeutic drugs to test (names/drug ID, stock concentration, dilution solution with either the volume to add to lyophilized drug or the drug weight and desired concentration, dilution series if desired)
    b. Cell types of interest (cortex, hippocampus, brain stem)
    c. Cell age(s) (which day(s) they want data from) - Example: day-in-vitro (DIV) 9, 11, and 13. We would probably want to have a maximum DIV based on our technology's limitations.
    d. Biomarkers of interest
        i. MicroRNA – optional to select specific microRNA or to use our default panel.
        ii. Calcium signaling
        iii. Extracellular vesicles
        iv. Oxidative stress
3. Based on user selections, pricing is calculated, and the user enters payment details.
    a. Price depends on the number of devices needed (number of drugs, how many concentrations of each drug, cell types), experiment length (max DIV), sampling times (how many DIVs to collect), and number of biomarkers (each with their own pricing).
4. User submits the request and is told that approval is pending.
5. Neurofluidic Diagnostics can view the request and download drug details.

6. Neurofluidic Diagnostics can change request status to “approved” or have some sort of “returned for clarification” type of status where we can send messages to the user.
7. If approved, the user is then prompted to ship lyophilized drugs to us.
8. Neurofluidic Diagnostics can change request status to “samples received/processing”.
9. Neurofluidic Diagnostics conducts experiments and uploads data into the app (what type of files do we need, file naming?).
10. Neurofluidic Diagnostics can change request status to “completed.”
11. Users can view data maps and download the raw data in spreadsheet form.

Client Document: Faux Scenario

Novartis has developed 3 drugs for Alzheimer's Disease research. Now, they want to know how their drugs alter various biomarker profiles in AD, using the data collected by Neurofluidic Diagnostics.

Drug Candidate number 1 (DC1) : They want this drug to be tested for all the biomarkers (EVs, miRNA, Oxidative Stress and calcium). They are only interested in Cortex and Hippocampus cells. They are interested in cells that are young and mature. They want DIV(days-in-vitro) 4, DIV9 and DIV15

Drug Candidate number 2 (DC2): They are only interested in the alterations of miRNA profiling when this drug is introduced. They are only interested in Cortex and Hippocampus cells. They are only interested in mature cells: DIV 15

Drug Candidate number 3(DC3) : They want to test this drug in 3 different concentrations for all the biomarkers. They are only interested in Cortex and Hippocampus cells. They want to test for only mature cells: DIV 15.

-What information does Neurofluidic Diagnostics need for each drug candidate?

- All the drugs will be sent in powder form to Neurofluidic Diagnostics. Novartis will have to attach an SDS (safety data sheet) for each drug, and an additional worksheet that comes with the information of what type and volume of solvent Neurofluidic Diagnostics should add for different concentrations.

Drug Candidate **(DC1)**

- Novartis will attach **SDS** for the DC1, and the sheet with **information regarding solvents**. They will manually enter the concentration they would like Neurofluidic Diagnostics to work with.
- **In the biomarkers tab**, Novartis will select EVs, miRNA, Oxidative Stress and Calcium data maps. (Remember, they wanted to see data maps with all biomarkers for this candidate.)
- Novartis will select **what type of cells** they want us to work with (Cortical, Hippocampus, Brain Stem). Each cell type will have individual pricing. Novartis is only interested in Cortical and Hippocampus cells.
- Novartis will enter the age of the cells they want us to work with. (DIV4, DIV9, DIV15)

Drug Candidate **(DC2)**

- Novartis will attach **SDS** for the DC2, and the sheet with **information regarding solvents**. They will manually enter the concentration they would like Neurofluidic Diagnostics to work with.
- **In the biomarkers tab**, Novartis will select only miRNA.

- Novartis will select **what type of cells** they want us to work with (Cortical, Hippocampus, Brain Stem). Each cell type will have individual pricing. Novartis is only interested in Cortical and Hippocampus cells.
- Novartis will enter the age of the cells they want us to work with. (DIV15).

Drug Candidate **(DC3)**

- Novartis will attach **SDS** for the DC3, and the sheet with **information regarding solvents**. They will manually enter the concentration they would like Neurofluidic Diagnostics to work with. **For this drug, Novartis wants to work with 3 different concentrations. They will have to manually enter the concentrations that they want us to work with.**
- **In the biomarkers tab**, Novartis will select all the biomarkers.
- Novartis will select **what type of cells** they want us to work with (Cortical, Hippocampus, Brain Stem). Each cell type will have individual pricing. Novartis is only interested in Cortical and Hippocampus cells.
- Novartis will enter the age of the cells they want us to work with. DIV15.

As they select these, a tiered-price structure will be activated and by the end of their entries they can see how much each drug will cost them to test; and they can see a total price by the end of entering all the drug candidates.

- Neurofluidic Diagnostics receives the information from their customers, Novartis. This information will be seen by everyone in the company, and the science team will design the experiments accordingly.
- When all the experiments are complete, the data maps will be uploaded to the customer's private page by Neurofluidic Diagnostics.
- Novartis will be able to track the process.
- Novartis will receive a notification once the data is uploaded.
- Novartis now can select individual drug candidates and the data map on these candidates.
- After Novartis sees the data map, they decide that Drug candidate number 1 and 2 is not as effective as they wanted them to be. They want to work closely with Drug Candidate number 3. Thank you neurofluidic diagnostics to help Novartis with this crucial decision before they lose billions of dollars!

References

Akamai Online Retail Performance Report. Akamai. 2017. Retrieved November 27, 2022, from https://www.akamai.com/newsroom/press-release/akamai-releases-spring-2017-state-of-online-retail-performance-report

Alzheimer's Disease Facts and Figures. Alzheimer's Disease and Dementia. (22AD). Retrieved November 29, 2022, from https://www.alz.org/alzheimers-dementia/facts-figures

Bąk, T. (2022). *Software Development Standards: ISO Compliance and Agile*. SoftKraft. Retrieved November 27, 2022, from https://www.softkraft.co/software-development-standards/#:~:text=Table%20of%20Contents-,What%20are%20software%20development%20standards%3F,the%20creation%20of%20software%20products

Freeman, E., & Robson, E. (2020). *Head First Design Patterns*. O'Reilly.

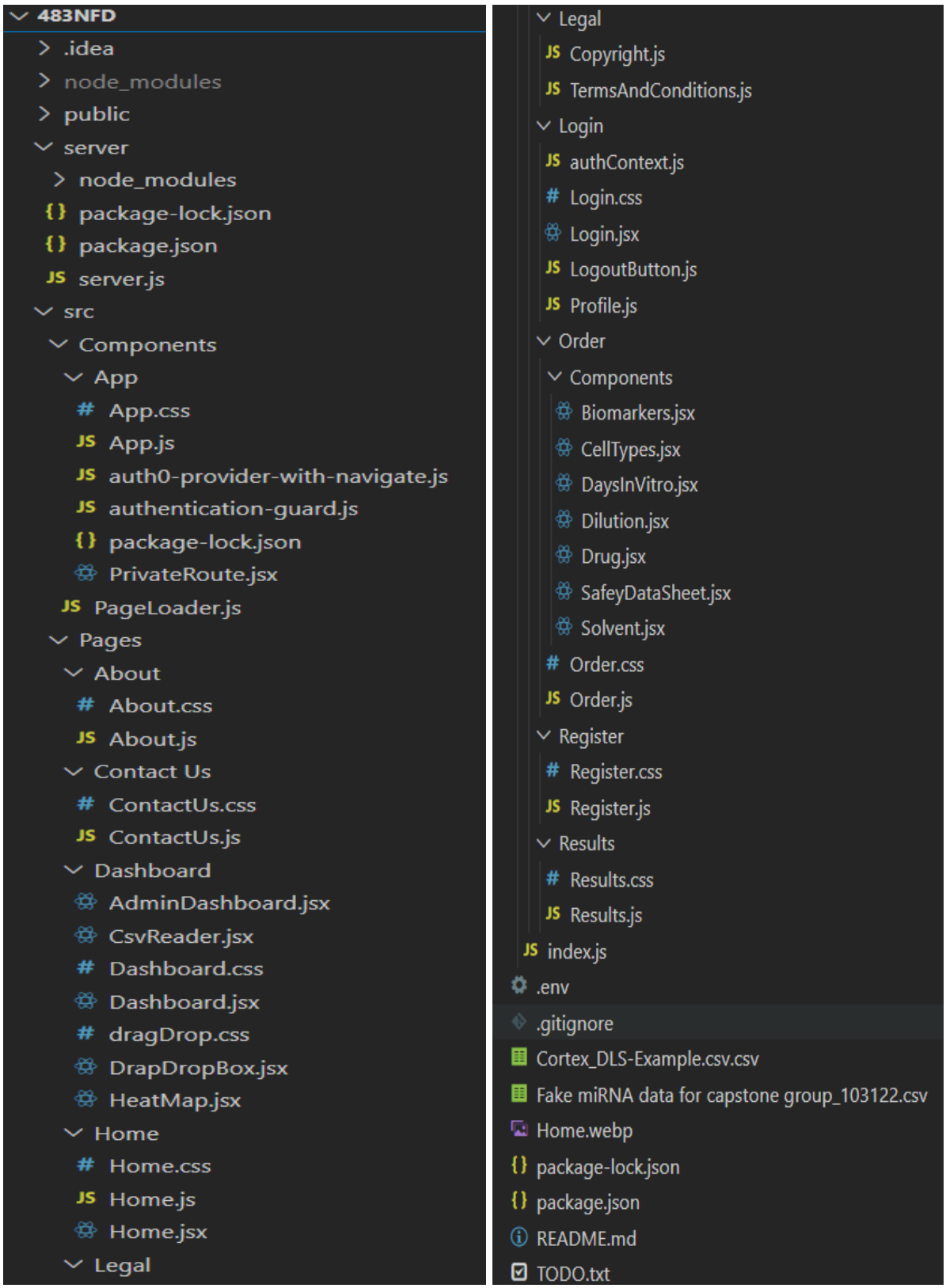
Miles, R., & Hamilton, K. (2006). *Learning Uml 2.0*. O'Reilly.

Web Development. IONOS Digital Guide. (2021). Retrieved November 30, 2022, from https://www.ionos.com/digitalguide/websites/web-development/

Shvets, O. (n.d.). *Composite*. Refactoring Guru. Retrieved April 30, 2023, from https://refactoring.guru/design-patterns/composite

Source Code

File Structure



Javascript & Reacts

App.js

```
import './App.css';
import {Link } from 'react-router-dom';
import React, {Fragment, useContext} from 'react';
import {Routes, Route} from 'react-router-dom';
import Home from '../../Pages/Home/Home';
import Dashboard from '../../Pages/Dashboard/Dashboard.jsx';
import Login from '../../Pages/Login/Login.jsx';
import Order from '../../Pages/Order/Order.js';
import Results from '../../Pages/Results/Results.js';
import Register from '../../Pages/Register/Register.js';
import About from '../../Pages/About/About';
import ContactUs from "../../Pages/Contact Us/ContactUs";
import TermsAndConditions from "../../Pages/Legal/TermsAndConditions";
import Copyright from "../../Pages/Legal/Copyright";
import { AuthContext } from '../../Pages/Login/authContext';
import LogoutButton from "../../Pages/Login/LogoutButton";

window.onscroll = function(){stickyHeader()};
var header = document.getElementById("header")

const navButtons = document.querySelectorAll('.nav-button');

navButtons.forEach(button => {
    button.addEventListener('click', () => {
        navButtons.forEach(otherButton => {
            otherButton.classList.remove('active');
        });
        button.classList.add('active');
    });
});

function stickyHeader() {
    const header = document.getElementById('header');
    const sticky = header.offsetTop;

    if (window.pageYOffset >= sticky) {
        header.classList.add('sticky');
        header.style.top = 0;
        document.body.style.paddingTop = `${header.offsetHeight}px`;
    } else {
        header.classList.remove('sticky');
        header.style.top = null;
        document.body.style.paddingTop = 0;
    }
}

function App() {
    const { isLoggedIn } = useContext(AuthContext);
    const { currentUser } = useContext(AuthContext);

    //hook for authentication flag
```

```
    if(isLoggedIn){
        return (
            <div className="App">
                <div className="Header" id={"header"}>
                    <img className="Logo" src="NFD_Logo.png"></img>
                    <div className="NavBar">
                        <Link to="/home"><button className='navButton'
id='homeButton'>Home</button></Link>
                        <Link to="/about"><button className='navButton'
id='aboutButton'>About</button></Link>
                        <Link to="/dashboard"><button className='navButton'
id='dashButton'>Dashboard</button></Link>
                        <Link to="/order"><button className='navButton'
id='orderButton'>Order</button></Link>
                        <Link to="/logout"><button className='navButton'
id='logoutButton'>Logout</button></Link>
                    </div>
                    <div className='clear'></div>
                </div>

                <div className="Content">
                    <div>
                        <Fragment>
                            <Routes>
                                <Route path="/home" element={<Home />}></Route>
                                <Route path="/" element={<Home />}></Route>
                                <Route path="/register" element={<Register />}></Route>
                                <Route path="/dashboard" element={<Dashboard />}></Route>
                                <Route path="/order" element={<Order />}></Route>
                                <Route path="/results" element={<Results />}></Route>
                                <Route path="/logout" element={<LogoutButton />}></Route>
                                <Route path="/about" element={<About />}></Route>
                                <Route path="/contact-us" element={<ContactUs />}></Route>
                                <Route path="terms-and-conditions" element={<TermsAndConditions
/>}></Route>
                                <Route path="/copyright-information" element={<Copyright />}></Route>
                            </Routes>
                        </Fragment>
                    </div>
                </div>

                <div className="Footer">
                    <img className="footerLogo" src="LOGO.png"></img>
                    <div className={"footerButtons"}>
                        <Link to="/about"><button className='Button'
id='aboutButton'>About</button></Link>
                        <Link to="/contact-us"><button className='Button' id='aboutButton'>Contact
Us</button></Link>
                        <Link to="/terms-and-conditions"><button className='Button'
id='aboutButton'>Terms and Conditions</button></Link>
                        <Link to="/copyright-information"><button className='Button'
id='aboutButton'>Copyright Information</button></Link>
                    </div>
                    <div className={"caption"}>(c) 2023, Neurofluidic Diagnostics</div>
                </div>
            </div>
        )
    } else{
        return (
```

```
            <div className="App">
                <div className="Header" id={"header"}>
                    <img className="Logo" src="NFD_Logo.png"></img>
                    <div className="NavBar">
                        <Link to="/home"><button className='navButton'
id='homeButton'>Home</button></Link>
                        <Link to="/about"><button className='navButton'
id='aboutButton'>About</button></Link>
                        <Link to="/login"><button className='navButton'
id='loginButton'>Login</button></Link>
                    </div>
                    <div className='clear'></div>
                </div>

                <div className="Content">
                    <div>
                        <Fragment>
                            <Routes>
                                <Route path="/home" element={<Home />}></Route>
                                <Route path="/" element={<Home />}></Route>
                                <Route path="/register" element={<Register />}></Route>
                                <Route path="/results" element={<Results />}></Route>
                                <Route path="/login" element={<Login />}></Route>
                                <Route path="/about" element={<About />}></Route>
                                <Route path="/contact-us" element={<ContactUs />}></Route>
                                <Route path="terms-and-conditions" element={<TermsAndConditions
/>}></Route>
                                <Route path="/copyright-information" element={<Copyright />}></Route>
                            </Routes>
                        </Fragment>
                    </div>
                </div>

                <div className="Footer">
                    <img className="footerLogo" src="LOGO.png"></img>
                    <div className={"footerButtons"}>
                        <Link to="/about"><button className='Button'
id='aboutButton'>About</button></Link>
                        <Link to="/contact-us"><button className='Button' id='aboutButton'>Contact
Us</button></Link>
                        <Link to="/terms-and-conditions"><button className='Button'
id='aboutButton'>Terms and Conditions</button></Link>
                        <Link to="/copyright-information"><button className='Button'
id='aboutButton'>Copyright Information</button></Link>
                    </div>
                    <div className={"caption"}>(c) 2023, Neurofluidic Diagnostics</div>
                </div>
            </div>
        )
    }
}

export default App;
```

Index.js

```
import React from 'react';
import ReactDOM from 'react-dom/client';
import App from './Components/App/App.js';
import {BrowserRouter as Router} from 'react-router-dom';
import {GoogleOAuthProvider} from '@react-oauth/google';
import {AuthProvider} from "./Pages/Login/authContext";

const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(
    <Router>
        <AuthProvider>
            <GoogleOAuthProvider clientId={`${process.env.REACT_APP_GOOGLE_API_TOKEN}`}>
                <App />
            </GoogleOAuthProvider>
        </AuthProvider>

    </Router>
);
```

About.js

```
import React, { useState } from 'react';
import './About.css';

function About() {
  const [description, setDescription] = useState('');
  const [image, setImage] = useState();
  function loaded() {
      setDescription('Calcium plays a crucial role in the function of neurons. It is involved in processes
such as neurotransmitter release, synaptic plasticity, and gene expression. However, abnormal calcium
homeostasis has been implicated in the pathogenesis of several neurodegenerative diseases. For example, in
Alzheimer\'s disease, calcium dysregulation can lead to the activation of enzymes that generate toxic
amyloid-beta peptides. In Parkinson\'s disease, disrupted calcium signaling can contribute to the
formation of Lew bodies, and abnormal protein aggregates that are a hallmark of the disease.');
      setImage('Calcium.jpg');
  }


    function updateDescription(service) {
        const panels = document.querySelectorAll('.servicePanel > div');
        panels.forEach(panel => panel.classList.remove('selected-panel'));


        // This is a temporary hack to allow the description and image to be loaded on first go around. If
removed, remove the second <p> and second <img> below.
        let temp = document.getElementById('loadingDES')
        temp.style.display = "none";
        temp = document.getElementById('loadingIMG')
        temp.style.display = "none";
```

```
        if (service === "Calcium") {
            setDescription('Calcium plays a crucial role in the function of neurons. It is involved in
processes such as neurotransmitter release, synaptic plasticity, and gene expression. However, abnormal
calcium homeostasis has been implicated in the pathogenesis of several neurodegenerative diseases. For
example, in Alzheimer\'s disease, calcium dysregulation can lead to the activation of enzymes that
generate toxic amyloid-beta peptides. In Parkinson\'s disease, disrupted calcium signaling can contribute
to the formation of Lew bodies, and abnormal protein aggregates that are a hallmark of the disease.');
            setImage('Calcium.jpg');
            panels[0].classList.add('selected-panel');
        } else if (service === 'Extracellular Vesicles') {
            setDescription('Extracellular vesicles (EVs) are small membranous structures that are released
by cells into the extracellular environment. They contain a variety of biomolecules such as proteins,
lipids, and nucleic acids, and have been implicated in the pathogenesis of various neurodegenerative
diseases. EVs are thought to play a role in the spread of misfolded proteins between cells, a key feature
of diseases such as Alzheimer\'s, Parkinson\'s, and Huntington\'s. Additionally, EVs may be involved in
the immune response to neurodegeneration, and have the potential as diagnostic and therapeutic targets in
these diseases.');
            setImage('EVs.jpg');
            panels[1].classList.add('selected-panel');
        } else if (service === 'MicroRNA') {
            setDescription('MicroRNAs (miRNAs) are small RNA molecules that regulate gene expression by
binding to messenger RNAs (mRNAs) and inhibiting their translation into proteins. Dysregulation of miRNA
expression has been implicated in the pathogenesis of various neurodegenerative diseases, including
Alzheimer\'s, Parkinson\'s, and Huntington\'s. In neurodegenerative diseases, specific miRNAs have been
shown to be dysregulated in affected brain regions, suggesting a role in disease pathogenesis.');
            setImage('MicroRNA.jpg');
            panels[2].classList.add('selected-panel');
        } else if (service === 'Oxidative Stress') {
            setDescription('Hypoxia and oxidative stress are two processes that are closely related and
can contribute to the pathogenesis of neurodegenerative diseases. Hypoxia, or a lack of oxygen, can occur
as a result of reduced blood flow to the brain, which can happen in conditions such as stroke or vascular
dementia. Hypoxia can lead to the generation of reactive oxygen species (ROS) in cells, which can cause
oxidative stress. ');
            setImage('Oxidative Stress.png');
            panels[3].classList.add('selected-panel');
        } else {
            setDescription('No services');
        }
    }

    return (
    <div className="About">
      <div>
        <img src="NFD_Team.jpg"/>
      </div>

        <div className={"About"}>

            <div className={"inline"}>
                <div className={"description"}>
                    <h2>About</h2>
```

```
                    Neurofluidic Diagnostics (NFD) is a spin-off idea turned entrepreneurial project from
the Kunze Lab at Montana State University, recognizing the lengthy process of obtaining effective
treatment as a prevalent issue. NFD sought to take action to expedite the process. NFD has developed a
medical technology that could provide more effective treatments for Alzheimer's and other degenerative
brain conditions. Neurofluidic Diagnostics seeks to help test candidate drugs, allowing researchers to
make better decisions before they enter preclinical testing, potentially saving time, money, and even
lives.
                </div>
                <div className={"visual"}>
                    <img src="about.jpg"/>
                </div>
            </div>
        </div>


      <div className="inline-wrapper">
          <div className="servicePanel">
            <div>
              <button className="panelButton" onClick={() =>
updateDescription('Calcium')}>Calcium</button>
            </div>
            <div>
              <button className="panelButton" onClick={() => updateDescription('Extracellular
Vesicles')}>Extracellular Vesicles</button>
            </div>
            <div>
              <button className="panelButton" onClick={() =>
updateDescription('MicroRNA')}>MicroRNA</button>
            </div>
            <div>
              <button className="panelButton" onClick={() => updateDescription('Oxidative
Stress')}>Oxidative Stress</button>
            </div>
          </div>
          <div>
              <p className="service-description">{description}</p>
              <p id='loadingDES' className="service-description">
                  Calcium plays a crucial role in the function of neurons. It is involved in processes
such as neurotransmitter release, synaptic plasticity, and gene expression. However, abnormal calcium
homeostasis has been implicated in the pathogenesis of several neurodegenerative diseases. For example, in
Alzheimer\'s disease, calcium dysregulation can lead to the activation of enzymes that generate toxic
amyloid-beta peptides. In Parkinson\'s disease, disrupted calcium signaling can contribute to the
formation of Lew bodies, and abnormal protein aggregates that are a hallmark of the disease.
              </p>
          </div>
          <div>
              <img id='loadingIMG' src="calcium.jpg"/>
              <img src={image} className="serviceImg"/>
          </div>
      </div>
```

```
    </div>
  );
}

export default About;
```

Contact.js

```
import React, { useState } from 'react';
import './ContactUs.css';
import { Link } from "react-router-dom";

function ContactUs() {
  const [name, setName] = useState('');
  const [email, setEmail] = useState('');
  const [phone, setPhone] = useState('');
  const [message, setMessage] = useState('');

  const handleSubmit = async (event) => {
    event.preventDefault();

    // Create a new document object with the form data
    const doc = {
      name: name,
      email: email,
      phone: phone,
      message: message
    };

    // Send a POST request to the backend API to insert the document into MongoDB
    try {
      const response = await fetch(`${process.env.REACT_APP_REACT_LOCAL}/api/contact`, {
        method: 'POST',
        headers: { 'Content-Type': 'application/json' },
        body: JSON.stringify(doc)
      });

      if (!response.ok) {
        throw new Error('Failed to submit form');
      }

      // Redirect the user to a confirmation page or show a success message
      alert('Form submitted successfully!');
      setName('');
      setEmail('');
      setPhone('');
      setMessage('');

    } catch (error) {
      console.error(error);
```

```
      alert(error.message);
    }
  };

  return (
    <div>
      <div className="contact-us">
        <form onSubmit={handleSubmit}>
          <div htmlFor="name" className={"title"}>Name/Institution:</div><br/>
          <input type="text" className="input" value={name} onChange={(e) => setName(e.target.value)}
/><br/>
          <div htmlFor="email"  className={"title"}>Email:</div><br/>
          <input type="text" className="input" value={email} onChange={(e) => setEmail(e.target.value)}
/><br/>
          <div htmlFor="phone" className={"title"}>Phone Number:</div><br/>
          <input type="text" className="input" value={phone} onChange={(e) => setPhone(e.target.value)}
/><br/>
          <div htmlFor="message" className={"title"}>Message:</div><br/>
          <textarea className="message" value={message} onChange={(e) => setMessage(e.target.value)}
placeholder="What can we help you with?"></textarea>
          <button className="Button" type="submit">Send</button>
        </form>
      </div>
    </div>
  );
}

export default ContactUs;
```

AdminDashboard.jsx

```
import { useState, useEffect, Fragment } from 'react';
import axios from 'axios';
import './Dashboard.css';
import Papa from 'papaparse';

const AdminDashboard = () => {

const [dashboardList, setDashboardList] = useState([])
const [selectedOrder, setSelectedOrder] = useState([]);
const [selectedBiomarker, setSelectedBiomarker] = useState('');
const [newStatus, setNewStatus] = useState('');
const [file, setFile] = useState(null);
// const [csvData, setCsvData] = useState([]);
// const [headers, setHeaders] = useState([]);

const [toggledState, setToggledState] = useState([])

const newOrderAlert = document.getElementById('statusChangeDiv')
```

```
const darkenDiv = document.getElementById('darkDiv')

useEffect(() => {
    axios.get(`${process.env.REACT_APP_REACT_LOCAL}/api/admindashboard`)
    .then(response => {
      setDashboardList(response.data)
      setToggledState(new Array(response.data.length).fill(false));

    })
    .catch(error => {
      console.log(error);
    });
  },[]);

function checkToggle(position) {
    const updatedToggleState = toggledState.map((item, index) =>
    index === position ? !item : false);
    setToggledState(updatedToggleState);
}

function handleStatusChange(newStatus, index) {
    setNewStatus(newStatus);
    setSelectedOrder(dashboardList[index]);
    newOrderAlert.style.display = "block";
    darkenDiv.style.display = "block";
}

function cancelStatusChange() {
    darkenDiv.style.display = "none";
    newOrderAlert.style.display = "none";
    setSelectedOrder('');
    setNewStatus('');
}

async function confirmStatusChange() {
    try {
        const response = await fetch(`${process.env.REACT_APP_REACT_LOCAL}/api/updateorder`, {
          method: 'PUT',
          headers: { 'Content-Type': 'application/json' },
          body: JSON.stringify( {
            orderID: selectedOrder.orderID,
            orderStatus: newStatus,
            updateDate : new Date().toLocaleString().replace(/:\d{2}\s/, ' ').replace(/\//g, '-')

          })
        });

        const data = await response.json();

        if (!response.ok) {
          throw new Error(data.message);
```

```
        }

      } catch (error) {
        console.log(error.message);
      }
      const index = dashboardList.findIndex(order => order.orderID === selectedOrder.orderID);
      dashboardList[index].orderStatus = newStatus;
      setDashboardList([...dashboardList]);
      cancelStatusChange();

}

function handleFileUpload(event, drug, bioID) {
    setSelectedBiomarker(bioID)
    setFile(event.target.files[0]);

    const reader = new FileReader();
    reader.onload = (event) => {
      let text = event.target.result;

      // Add quotation marks around any 0,0 that occur
      text = text.replace(/(?<=^|,)(0,0)(?=,|$)/g, '"$1"');

      Papa.parse(text, {
        header: true,
        dynamicTyping: true,
        complete: (result) => handleCSVData(result, bioID),
      });
    };
    reader.readAsText(event.target.files[0]);
  };

  const handleCSVData = async (result, bioID) => {
    try {
      // send the parsed data to the server
      const response = await axios.post(`${process.env.REACT_APP_REACT_LOCAL}/api/resultsup`, {
        method: 'POST',
        headers: { 'Content-Type': 'application/json' },
          orderID: selectedOrder.orderID,
          biomarker: bioID,
          data: result.data
      });
      if(response.ok) {
        alert("Result upload Successful")
      }
    } catch (error) {
      console.error(error);
    }
  };

    return (
```

```
      <div>
      <div>
          <h1>Admin Dashboard</h1>
          <table className='dashTable'>
<thead>
  <tr className='tableHeaderRow'>
    <th>Order</th>
    <th>OrderID</th>
    <th>Status</th>
    <th>Total</th>
    <th>Date Submitted</th>
    <th>Date Updated</th>
    <th></th>
  </tr>
</thead>
<tbody>
  {dashboardList.map((order, index) => {

    return (
      <Fragment key={index}>
        <tr className='tableRow'>
          <td>
          <button
        className='toggleInfoButton'
        onClick={() => checkToggle(index)}
      >
        {toggledState[index] ? 'X' : 'V'}
      </button>
            {index + 1}
          </td>
          <td>{order.orderID.slice(-5)}</td>
          <td>
            {order.orderStatus}
            <br />
            <select
              placeholder='Change Status'
              name='adminOrderStatus'
              className='statusOption'
              onChange={(e) => handleStatusChange(e.target.value, index)}
            >
              <option value='' defaultValue='none'>
                --change status--
              </option>
              <option value='Approved - Awaiting Sample'>
                Approved - Awaiting Sample
              </option>
              <option value='Sample Received'>Sample Received</option>
              <option value='Processing Sample'>Processing Sample</option>
              <option value='Completed'>Completed</option>
              <option value='Rejected'>Rejected</option>
            </select>
```

```
            </td>
            <td>{order.orderTotal}</td>
            <td>{order.submitDate}</td>
            <td>{order.updateDate}</td>
            <td></td>
          </tr>
          {toggledState[index] && (
            <tr>
              <td colSpan={7}>
                <div className='subOrderHeader'>
                  <table className='subOrderInfo' id='subBioTable'>
                    <thead>
                      <tr>
                        <th>Name</th>
                        <th>Weight</th>
                        <th>Biomarkers</th>
                        <th>CellTypes</th>
                        <th>Solvents</th>
                      </tr>
                    </thead>
                    <tbody className='subInfoTable'>
                      {order.drugs.map((drug, index2) => (
                        <tr key={index2}>
                          <td>{drug.name}</td>
                          <td>
                            {drug.weight}
                            {drug.unit}
                          </td>
                          <td>
                            {drug.biomarkers.map((bio, index3) => (
                              <div key={index3}>{bio.name}</div>
                            ))}
                          </td>
                          <td>
                            {drug.cellTypes.map((cells, index3) => (
                              <div key={index3}>{cells.name}</div>
                            ))}
                          </td>
                          <td>
                            {drug.solvents.map((solvs, index3) => (
                              <div key={index3}>
                                {solvs.name} {solvs.volume} {solvs.concentration}
                              </div>
                            ))}
                          </td>
                        </tr>
                      ))}
                    </tbody>
                  </table>
                </div>
              </td>
```

```
            </tr>
          )}
        </Fragment>
      );
    })}
  </tbody>
  </table>
        </div>
        <div className='darkenDiv' id="darkDiv">
        </div>

        <div className='statusChangeDiv' id="statusChangeDiv">
  <div className='statusDivContent'>
    <h4>Confirm Status Change?</h4>
    <h4>{selectedOrder.orderTitle}</h4>
    <h5>From: {selectedOrder.orderStatus}</h5>
    <h5>To: {newStatus}</h5>
    {newStatus === "Completed" && (
      <div>
        {selectedOrder.drugs.map(drug => (
          <div key={drug.id}>
            <h6>{drug.drugName}</h6>
            {drug.biomarkers.map(biomarker => (
              <div key={biomarker.id}>
                <label className="bioUpload"
htmlFor={`fileInput_${drug.id}_${biomarker.id}`}>{biomarker.name}:</label>
                <input type="file" accept=".csv" id={`fileInput_${drug.id}_${biomarker.id}`} onChange={(e)
=> handleFileUpload(e, drug.id, biomarker.name)} />
              </div>
            ))}
          </div>
        ))}
      </div>
    )}
    <button className='statusChangeButton cancelButton' onClick={cancelStatusChange}>Cancel</button>
    <button className='statusChangeButton confirmButton' onClick={confirmStatusChange}>Confirm</button>
  </div>
</div>
        </div>
    );
};

export default AdminDashboard
```

Dashboard.jsx

```jsx
import { useState, useEffect, useContext, Fragment} from 'react';
import './Dashboard.css';
import axios from 'axios';
import AdminDashboard from './AdminDashboard';
import { AuthContext } from '../../Pages/Login/authContext';

const Dashboard = () => {

const [dashboardList, setDashboardList] = useState([]);
const [toggledState, setToggledState] = useState([])
const { isAdmin } = useContext(AuthContext);
const { currentUser } = useContext(AuthContext);

  useEffect(() => {
    axios.get(`${process.env.REACT_APP_REACT_LOCAL}/api/dashboard`, {
      params: {
        userID: currentUser
      }
    })
    .then(response => {
      setDashboardList(response.data)
      setToggledState(new Array(response.data.length).fill(false));

    })
    .catch(error => {
      console.log(error);
    });
  },[]);

// function generateHeatMap() {

// }

function checkToggle(position) {
  const updatedToggleState = toggledState.map((item, index) =>
  index === position ? !item : false);
  setToggledState(updatedToggleState);
}

if (isAdmin) {
  return(
  <AdminDashboard />
  )
}

else {
  return (
    <div>
    <div>
```

```
      <h1>Dashboard</h1>
      <table className='dashTable'>
<thead>
  <tr className='tableHeaderRow'>
    <th>Order</th>
    <th>OrderID</th>
    <th>Status</th>
    <th>Total</th>
    <th>Date Submitted</th>
    <th>Date Updated</th>
    <th></th>
  </tr>
</thead>
<tbody>
  {dashboardList.map((order, index) => {
    return (
      <Fragment key={index}>
        <tr className='tableRow'>
          <td>
          <button
        className='toggleInfoButton'
        onClick={() => checkToggle(index)}
      >
        {toggledState[index] ? 'X' : 'V'}
      </button>
            {index + 1}
          </td>
          <td>{order.orderID.slice(-5)}</td>
          <td>{order.orderStatus}</td>
          <td>{order.orderTotal}</td>
          <td>{order.submitDate}</td>
          <td>{order.updateDate}</td>
          <td>
            <button className='dashButton downloadButton'></button>
            <button className='dashButton graphButton'></button>
            <button className='dashButton csvButton'></button>
          </td>
        </tr>
        {toggledState[index] && (
          <tr>
            <td colSpan={7}>
              <div className='subOrderHeader'>
                <table className='subOrderInfo'>
                  <thead>
                    <tr>
                      <th>Name</th>
                      <th>Weight</th>
                      <th>Biomarkers</th>
                      <th>CellTypes</th>
                      <th>Solvents</th>
                    </tr>
```

```
                    </thead>
                    <tbody className='subInfoTable'>
                      {order.drugs.map((drug, index2) => (
                        <tr key={index2}>
                          <td>{drug.name}</td>
                          <td>
                            {drug.weight}
                            {drug.unit}
                          </td>
                          <td>
                            {drug.biomarkers.map((bio, index3) => (
                              <div key={index3}>{bio.name}</div>
                            ))}
                          </td>
                          <td>
                            {drug.cellTypes.map((cells, index3) => (
                              <div key={index3}>{cells.name}</div>
                            ))}
                          </td>
                          <td>
                            {drug.solvents.map((solvs, index3) => (
                              <div key={index3}>
                                {solvs.name} ({solvs.volume}{solvs.volUnit}
{solvs.concentration}{solvs.concUnit})
                              </div>
                            ))}
                          </td>
                        </tr>
                      ))}
                    </tbody>
                  </table>
                </div>
              </td>
            </tr>
          )}
        </Fragment>
      );
    })}
  </tbody>
  </table>
  </div>
  </div>
);
}
};

export default Dashboard
```

Home.js

```
import React, { useState } from 'react';
import './Home.css';
import {Link} from "react-router-dom";

function Home() {
  return (
    <div className="home">
        <img src="Main.jpg"/>
      <div>
          <div className={"Slogan"}> Providing comprehensive data maps for Alzheimer's Disease,
              enabling cost-effective drug-candidate screening.</div>
      </div>
        <div className={"services"}>
            <div className={"inline"}>
                <div className={"description"}>
                    <h2>Services</h2>
                    Our company specializes in biomarker profile maps designed to optimize drug
development for neurodegenerative diseases. We provide comprehensive analysis and evaluation of potential
drug candidates, enabling our clients to make informed decisions and during the drug development process.
Our team of experts utilizes cutting-edge technology and industry-standard methods to ensure the accuracy
and reliability of our results.
                    <br/> <Link to="/about"><button className='Button' id='aboutButton'>Learn
More</button></Link>
                </div>
                <div className={"visual"}>
                    <img src={"services-img.jpg"}/>
                </div>
            </div>
        </div>

        <div className={"problem"}>
            <div className={"inline"}>
                <div className={"visual"}>
                    <img src={"alzheimerProjections.jpg"}/>
                </div>
                <div className={"description"}>
                    <h2>Problem</h2>
                    Alzheimer’s disease (AD) is a prevalent form of dementia affecting people above the
age of 65 years. Currently, there are six million Americans living with this form of dementia, and
according to the National Institutes of Health (NIH) that number is expected to rise to about 12.5 million
people by 2050. This high prevalence within neurodegenerative diseases is creating huge health care and
financial burdens for our society. Unfortunately, early diagnostics and drug development for AD are still
insufficient in lowering health care burden. Furthermore, the costs for developing effective drug
treatments for AD per patient are one of the highest in the pharmaceutical industry.
                </div>
            </div>
        </div>
        <div className={"solution"}>
            <div className={"inline"}>
```

```
                <div className={"description"}>
                    <h2>Solution</h2>
                    To enhance early-diagnostics for AD, our company, Neurofluidic Diagnostics (NFD), uses a unique in-vitro screening platform that enables multiple biomarker profiling during disease progression. Our technology involves a neurodegenerative disease model within living neurons in culture that allows us to quantify efficacy of AD drug candidates based on data mapping. The resulting visual data maps are available to our customers through a web-based application, called “Neurodynamik”. Furthermore, Neurodynamik offers customizable testing solutions for individual drug candidates that will allow customers to make crucial go/no-go decision towards clinical drug testing. Our innovative testing platform has the potential to save customers millions of dollars in drug development costs.
                </div>
                <div className={"visual"}>
                    <img src={"Solution.jpg"}/>
                </div>
            </div>
        </div>
    </div>
  );
}

export default Home;
```

Copyright.js

```
function Copyright(){
    return(
        <div>
            <h1>
                COPYRIGHT INFORMATION
            </h1>
            <h2>
                © 2023 Neurofluidic Diagnostics. All Rights Reserved.
            </h2>

            <div>
                All content on this website, including text, graphics, logos, images, and software, is the property of Neurodynamik or its content suppliers and is protected by copyright law. The compilation of all content on this website is the exclusive property of Neurodynamik and is protected by copyright law.

                You may not reproduce, modify, or distribute any of our content without our express written permission. Unauthorized use of any content on this website may violate copyright, trademark, and other laws.

                If you believe that any content on this website infringes on your copyright, please contact us at [Email Address]. We take all claims of copyright infringement seriously and will promptly investigate any such claims.

                Thank you for respecting our intellectual property rights.
            </div>
```

```
        </div>
    );
}
export default Copyright;
```

TermsAndConditions.js

```
function TermsAndConditions(){
    return(
        <div>
                <h1>
                        TERMS AND CONDITIONS
                </h1>
                <h2>
                        Introduction
                </h2>
                <div>
                        Welcome to Neurodynamik! These terms and conditions govern your use of our website
and the products and services we offer. By using our website and purchasing our products, you agree to
these terms and conditions.
                </div>
                <h2>
                        Acceptance of Terms
                </h2>
                <div>
                        By using our website and purchasing our products, you acknowledge that you have
read and agree to these terms and conditions. If you do not agree to these terms and conditions, you may
not use our website or purchase our products.
                </div>
                <h2>
                        User Accounts
                </h2>
                <div>
                        In order to purchase our products, you must create an account on our website. You
are responsible for maintaining the confidentiality of your account information, including your username
and password.
                </div>
                <h2>
                        Payment and Billing
                </h2>
                <div>
                        We accept payment through our secure online checkout system. By purchasing our
products, you agree to pay the full price listed on our website, including any applicable taxes and
shipping fees. We reserve the right to change our prices at any time without notice.
                </div>
                <h2>
                        Intellectual Property
                </h2>
                <div>
                        All content on our website, including text, graphics, logos, images, and software,
```

```
is the property of our company or our affiliates and is protected by copyright law. You may not reproduce, modify, or distribute any of our content without our express written permission.
                </div>
                <h2>
                        User Content
                </h2>
                <div>
                        By submitting user-generated content to our website, such as product reviews or comments, you grant us a non-exclusive, royalty-free, perpetual, and irrevocable license to use, modify, and distribute your content in any way we see fit.
                </div>
                <h2>
                        User Conduct
                </h2>
                <div>
                        You agree to use our website and products only for lawful purposes and in a manner that does not infringe on the rights of others. You may not engage in any fraudulent, abusive, or harassing behavior.
                </div>
                <h2>
                        Privacy Policy
                </h2>
                <div>
                        Our privacy policy outlines how we collect, use, and protect your personal information. By using our website and purchasing our products, you agree to our privacy policy.
                </div>
                <h2>
                        Disclaimers and Limitations of Liability
                </h2>
                <div>
                        We make no warranties or guarantees regarding the quality or performance of our products. We are not liable for any damages or losses arising from your use of our website or products, including any direct, indirect, incidental, or consequential damages.
                </div>
                <h2>
                        Termination and Suspension
                </h2>
                <div>
                        We reserve the right to terminate or suspend your account at any time, for any reason, without notice.
                </div>
                <h2>
                        Governing Law and Dispute Resolution
                </h2>
                <div>
                        These terms and conditions are governed by the laws of the state of [State], and any disputes arising from these terms and conditions shall be resolved in the state or federal courts located in [City], [State].
                </div>
                <h2>
                        Changes to Terms and Conditions
```

```
                </h2>
                <div>
                        We reserve the right to change or update these terms and conditions at any time
without notice. Your continued use of our website and products following any such changes constitutes your
acceptance of the revised terms and conditions.
                </div>
                <h2>
                        Contact Information
                </h2>
                <div>
                        If you have any questions or concerns about these terms and conditions, please
contact us at [Email Address].

                        Thank you!
                </div>
            </div>
    );
}
export default TermsAndConditions;
```

authContext.js

```
import {createContext, useContext, useState} from 'react';

export const AuthContext = createContext();

export const AuthProvider = ({ children }) => {
    const [isLoggedIn, setIsLoggedIn] = useState(false);
    const [currentUser, setCurrentUser] = useState(null);
    const [isAdmin, setIsAdmin] = useState(false);
    const MyComponent = () => {
        const MyComponent = () => {
            const { isLoggedIn, setIsLoggedIn } = useContext(AuthContext);
        };
    };
    return (
        <AuthContext.Provider value={{ isLoggedIn, setIsLoggedIn, currentUser, setCurrentUser, isAdmin,
setIsAdmin }}>
            {children}
        </AuthContext.Provider>
    );
};
```

Login.jsx

```
import React, {useState, useContext} from 'react';
import { GoogleLogin } from '@react-oauth/google';
import {Link, useNavigate} from 'react-router-dom';
import { FcGoogle } from 'react-icons/fc';
import jwt_decode from "jwt-decode";
import './Login.css';
```

```
import { AuthContext } from './authContext';

const Login = () => {
    const [username, setUsername] = useState('');
    const [email, setEmail] = useState('');
    const [password, setPassword] = useState('');
    const [error, setError] = useState('');
    const navigate = useNavigate();
    const { setIsLoggedIn } = useContext(AuthContext);
    const { setCurrentUser } = useContext(AuthContext);
    const { setIsAdmin } = useContext(AuthContext);

    const handleSubmit = async (event) => {
        event.preventDefault();
        try {
            // Submit post request
            const response = await fetch(`${process.env.REACT_APP_REACT_LOCAL}/api/login`, {
                method: 'POST',
                headers: { 'Content-Type': 'application/json' },
                body: JSON.stringify({ username, email, password })
            });

            // This line extracts the JSON data from the response object.
            // The await keyword makes the function wait for the data to be extracted before moving on to
the next line of code.
            const data = await response.json();

            if (!response.ok) {
                throw new Error(data.message);
            }
            setIsLoggedIn(true);
            setCurrentUser(data.user.userID);
            setIsAdmin(data.user.isAdmin);
            navigate("/home");

        } catch (error) {
            setError(error.message);
        }
    };
    function handleCredentialResponse(response) {

        localStorage.setItem('user', JSON.stringify(response.profileObj));
        var decodedHeader = jwt_decode(response.credential);
        console.log(decodedHeader);
        const { name, sub, email, email_verified } = decodedHeader
        console.log(name)

        const doc = {
            _id: sub,
            _type: 'user',
            userName: name,
```

```
            email: email,
            email_verified: email_verified
        }

        setIsLoggedIn(true);
        navigate("/home");
    };

    return (
        <div className="Login">
            <h1>Login</h1>
            <form onSubmit={handleSubmit}>
                <input
                    type="text"
                    value={username}
                    onChange={(event) => setUsername(event.target.value)}
                    placeholder="Username/Email"
                />
                <input
                    type="password"
                    value={password}
                    onChange={(event) => setPassword(event.target.value)}
                    placeholder="Password"
                />
                <button type="submit">Login</button>
            </form>
            <br/>
            <Link to="/register"><div>Don't have an account yet?</div></Link>

            {/*Login with Google*/}
            <br/>
            <GoogleLogin
                clientId= '143526042201-lpkm6hgoslltopgops5ucm2ahv5r4qe.apps.googleusercontent.com'
                onSuccess={(response) => {
                    handleCredentialResponse(response);
                }}
                onError={() => {
                    console.log('Login Failed');
                }}
                cookiePolicy={'single_host_origin'}
            />
        </div>
    )
}

export default Login;
```

LogOutButton.js

```
import React, { useContext } from 'react';
```

```
import { AuthContext } from './authContext';
import { useNavigate } from 'react-router-dom';

const LogoutButton = () => {
    const { setIsLoggedIn } = useContext(AuthContext);
    const navigate = useNavigate();

    const handleLogout = () => {
        setIsLoggedIn(false);
        navigate('/home');
    };
    return (
        <div>
            <div className={"a"}>
                Are you sure you want to logout? <br/>
            </div>
                <button className={"Button"} onClick={handleLogout}>Logout</button>
        </div>
    )};

export default LogoutButton;
```

Profile.js

```
import React, { useState } from "react";
import { useAuth0 } from "@auth0/auth0-react";

function Profile() {
  const { user } = useAuth0();
  const [firstName, setFirstName] = useState(user.given_name);
  const [lastName, setLastName] = useState(user.family_name);
  const [email, setEmail] = useState(user.email);

  const handleUpdate = (e) => {
    e.preventDefault();
    // TODO: Save updated information to MongoDB
  };

  return (
    <div className="Profile">
      <h1>Profile Page</h1>
      <form onSubmit={handleUpdate}>
        <label>
          First Name:
          <input
            type="text"
            value={firstName}
            onChange={(e) => setFirstName(e.target.value)}
          />
```

```
        </label>
        <label>
          Last Name:
          <input
            type="text"
            value={lastName}
            onChange={(e) => setLastName(e.target.value)}
          />
        </label>
        <label>
          Email:
          <input
            type="email"
            value={email}
            onChange={(e) => setEmail(e.target.value)}
          />
        </label>
        <button type="submit">Update</button>
      </form>
    </div>
  );
}

export default Profile;
```

Order.js

```
import {useState, useEffect, useCallback, useContext} from 'react';
import './Order.css';
import Drug from './Components/Drug'
import { useNavigate } from 'react-router-dom';
import axios from 'axios';
import { AuthContext } from '../../Pages/Login/authContext';

const Order = () => {
  const navigate = useNavigate();
  const { currentUser } = useContext(AuthContext);
  const [order, setOrder] = useState({
    userID: currentUser,
    orderId: '',
    orderTitle: '',
    orderTotal: 0,
    orderStatus: 'Incomplete',
    drugs: [],
  });

  const [drugs, setDrugList] = useState([]);
  const [user, setUser] = useState({});

  function handleAddDrug(drugs) {
    setDrugList({ ...drugs, drugs})
```

```
  setOrder({ ...order, drugs:[] });
}

function handleUpdateDrugs(newDrugsList) {
  setOrder({...order, drugs: newDrugsList});
}

const handleOrderChange = useCallback((field, value) => {
  setOrder(prevOrder => ({ ...prevOrder, [field]: value }));
},[])

const handleUserChange = useCallback((field, value) => {
  setUser(prevUser => ({ ...prevUser, [field]: value }));
  if (field === 'userID') {
    handleOrderChange('userID', value);
  }
},[handleOrderChange])

useEffect(() => {
  async function fetchData() {
    const userResponse = await axios.get(`${process.env.REACT_APP_REACT_LOCAL}/api/user`, {
      params: {
        userID: currentUser
      }
    });
    const fieldNames = Object.keys(userResponse.data[0]);
    fieldNames.forEach(key => {
      const value = userResponse.data[0][key];
      handleUserChange(key, value)
    })
  }
  fetchData();
  },[]);

const [error, setError] = useState('');

function errorReset() {
  var errs = document.getElementsByClassName('errorText');
  for(var x =0; x < errs.length; x++) {
    errs[x].style.display = 'none';
  }
}

const handleSubmit = async (event) => {
  event.preventDefault();
  handleOrderChange('orderStatus', 'sending');
  errorReset();
  try {
    const response = await fetch(`${process.env.REACT_APP_REACT_LOCAL}/api/order`, {
      method: 'POST',
      headers: { 'Content-Type': 'application/json' },
```

```
        body: JSON.stringify({
          userID: order.userID,
          orderTitle: order.orderTitle,
          orderTotal: order.orderTotal,
          orderStatus: order.orderStatus,
          drugs: order.drugs,
        }),
      });


      // This line extracts the JSON data from the response object.
      // The await keyword makes the function wait for the data to be extracted before moving on to the
next line of code.
      console.log(response)
      if(response.ok) {
        alert("Order Successful")
        navigate("/dashboard")
      }
      if (!response.ok) {
        throw new Error();
      }
    } catch (error) {
      setError(error.message);
    }
  };

return (
  <div className="Order">
    <form onSubmit={handleSubmit}>
      <div className="orderInfoSection">
       <h3>Order Info</h3>
        <div>
          <label htmlFor="orderTitle" className="orderInfoField">
            Order Title:
            <input className="inputField" id="orderTitle" type="text" name="orderTitle" onChange={(e) =>
handleOrderChange('orderTitle', e.target.value)} />
            <p className="errorText titleErr" id="orderTitleErr">Cannot be Empty</p>
          </label>
        </div>
        <div>
          <label htmlFor="firstName" className="orderInfoField">
            First Name:
            <input className="inputField" id="firstName" type="text" name="firstName" onChange={(e) =>
handleUserChange('firstName', e.target.value)} />
            <p className="errorText nameErr" id="firstNameEmptyErr">Cannot be Empty</p>
            <p className="errorText nameErr" id="firstNameNonAlphaErr">Cannot contain numbers or special
characters</p>
          </label>
        </div>
        <div>
          <label htmlFor="lastName" className="orderInfoField">
```

```
            Last Name:
            <input className="inputField" id="lastName" type="text" name="lastName"  onChange={(e) =>
handleUserChange('lastName', e.target.value)} />
            <p className="errorText nameErr" id="lastNameErr">Cannot be Empty</p>
            <p className="errorText nameErr" id="firstNameNonAlphaErr">Cannot contain numbers or special
characters</p>
          </label>
        </div>
        <div>
          <label htmlFor="email" className="orderInfoField">
            Email:
            <input className="inputField" id="email" type="email" name="email"  onChange={(e) =>
handleUserChange('email', e.target.value)} />
            <p className="errorText emailErr" id="emailValidErr">Invalid email format</p>
            <p className="errorText emailErr" id="emailUniqueErr">Email already exists</p>
          </label>
        </div>
        <div>
          <label htmlFor="institution" className="orderInfoField">
            Institution:
            <input className="inputField" id="institution" type="text" name="institution" onChange={(e) =>
handleUserChange('institution', e.target.value)} />
            <p className="errorText instErr" id="institutionErr">Cannot be Empty</p>
          </label>
        </div>
        </div>
        <div>
          <Drug onUpdateDrugs={handleUpdateDrugs}/>
        </div>
      <button className="Button" type="submit">Submit</button>
    </form>
  </div>
);
}

export default Order;
```

Biomarkers.jsx

```
import {useState, useEffect} from 'react';
import axios from 'axios';

const Biomarkers = ({getBiomarkers}) => {

    const [biomarkers, setBiomarkers] = useState([]);

    useEffect(() => {
        axios.get(`${process.env.REACT_APP_REACT_LOCAL}/api/biomarkers`)
        .then(response => {
```

```
          setBiomarkers(response.data)
        })
        .catch(error => {
          console.log(error);
        });
      },[]);

    const [checkedBiomarkers, setCheckedBiomarkers] = useState([]);

    function handleCheckChange(e, index) {
        const biomarkerName = e.target.value;
        if(e.target.checked) {
            const checkedBiomarker = biomarkers.find((biomarker) => biomarker.name === biomarkerName);
            setCheckedBiomarkers([...checkedBiomarkers, checkedBiomarker])
        }
        else {
            const removedBiomarker = [...checkedBiomarkers];
            removedBiomarker.splice(index,1);
            setCheckedBiomarkers(removedBiomarker);
        }
        getBiomarkers(checkedBiomarkers);
    }

    useEffect(() => {
        getBiomarkers(checkedBiomarkers);
      }, [checkedBiomarkers]);

    return (
        <div className="drugOrderComp" id="biomarkers">
            <h3>BioMarkers</h3>
            <div className='bioWrapper'>
                {biomarkers.map((bio, index) => {
                    return (
                        <div key={index} className='bioMarkerItem'>
                                <label htmlFor={bio.name}>{bio.name}
                                    <input type="checkbox"
                                           name={bio.name}
                                           value={bio.name}
                                           checked={checkedBiomarkers.some((selected) => selected.name ===
bio.name)}
                                           onChange={ (e) => handleCheckChange(e, index)}/>
                                </label>
                        </div>
                    );
                })}
                </div>
        </div>
    );
};

export default Biomarkers
```

CellTypes.jsx

```
import {useState, useEffect} from 'react';
import axios from 'axios';

const CellTypes = ({getCellTypes}) => {
    const [cellTypes, setCellTypes] = useState([]);

    useEffect(() => {
        axios.get(`${process.env.REACT_APP_REACT_LOCAL}/api/celltypes`)
        .then(response => {
          setCellTypes(response.data)
        })
        .catch(error => {
          console.log(error);
        });
      },[]);

    const [checkedCellTypes, setCheckedCellTypes] = useState([]);

    function handleCheckChange(e, index) {
        const cellTypeName = e.target.value;
        if(e.target.checked) {
            const checkedCellType = cellTypes.find((cellType) => cellType.name === cellTypeName);
            setCheckedCellTypes([...checkedCellTypes, checkedCellType])
        }
        else {
            const removedCellType = [...checkedCellTypes];
            removedCellType.splice(index,1);
            setCheckedCellTypes(removedCellType);
        }
        getCellTypes(checkedCellTypes);
    }

    useEffect(() => {
        getCellTypes(checkedCellTypes);
      }, [checkedCellTypes]);

    return (
        <div className="drugOrderComp" id="celltypes">
            <h3>CellTypes</h3>
            <div className='cellWrapper'>
                {cellTypes.map((cell, index) => {
                    return (
                        <div key={index} className='cellTypeItem'>
                            <label htmlFor={cell.name}>{cell.name}
                               <input type="checkbox"
                               name={cell.name}
                               value={cell.name}
```

```
                                checked={checkedCellTypes.some((selected) => selected.name === cell.name)}
                                onChange={ (e) => handleCheckChange(e, index)}
                                />
                            </label>
                        </div>
                    );
                })}
                </div>a
        </div>
    );
}

export default CellTypes
```

DaysInVitro.jsx

```
import React from "react";
import {useState, useEffect} from 'react';

const DaysInVitro = ({addDaysVitro}) => {

    const [daysVitro, setDaysVitroList] = useState([]);
    const divLengthMultiplier = .05;

    useEffect(() => {
        setDaysVitroList([{value: 0, price: 10}])
    }, []);

    function handleAddDaysVitro() {
        const newDaysVitro = {
            value: 0,
            price: 10
        }

        setDaysVitroList([...daysVitro, newDaysVitro]);
        addDaysVitro([...daysVitro, newDaysVitro]);
    }

    function handleRemoveDaysVitro(index) {
        const removedDaysVitro = [...daysVitro];
        removedDaysVitro.splice(index, 1);
        setDaysVitroList(removedDaysVitro);
        addDaysVitro(removedDaysVitro);
    }

    function handleChange(index, field, value) {
        const updatedLength = [...daysVitro];
        updatedLength[index][field] = value
        updatedLength[index]['price'] = value * (1 + divLengthMultiplier);
```

```
        setDaysVitroList(updatedLength) ;
        addDaysVitro(updatedLength);
    }

    return (
        <div className="drugOrderComp" id="daysInVitro">
            <h3>DaysInVitro</h3>
                {daysVitro.map((days, index) => {
                    return (
                        <div key={index} className="daysVitroItem">
                            <label htmlFor={"daysVitro" + {index}}>DiV:
                                <input type="number"
                                    className="inputField"
                                    name={'daysVitro' + {index}}
                                    value={days.length}
                                    onChange={ (e) => handleChange(index, 'value', e.target.value)}>
                                </input>
                                <button type="button" className='daysVitroButton orderButton removeButton
Button'onClick={() => handleRemoveDaysVitro(index)}>X</button>
                            </label>
                        </div>
                    )
                })}
                <button type="button"  className='daysVitroButton orderButton Button'
onClick={handleAddDaysVitro}>Add DaysInVitro</button>
        </div>
    )
}

export default DaysInVitro
```

Dilution.jsx

```
import { useState, useEffect} from 'react';

const Dilution = ({addDilution}) => {

    const [dilutionList, setDilutionList] = useState([]);

    useEffect(() => {
        setDilutionList([])
    }, []);

    function handleChange(index, field, value) {
        const updatedDilutions = [...dilutionList];
        updatedDilutions[index][field] = value;
        setDilutionList(updatedDilutions);
        addDilution(updatedDilutions); // Call the callback function to update the drug component
    }
```

```
    const handleAddDilution = () => {
        const newDilution = {
            concentration: 0.0,
            volUnit: "mL"
        }
        setDilutionList([...dilutionList, newDilution]);
        addDilution([...dilutionList, newDilution]);
    }

    const handleRemoveDilution = (index) => {
        const updatedDilutions = [...dilutionList];
        updatedDilutions.splice(index, 1);
        setDilutionList(updatedDilutions);
    }

    return(
        <div className="dilutionsDiv">
            {dilutionList.map((dilution, index) => (
                <div key={index}>
                    <h3>Dilutions:</h3>
                    <label htmlFor={"dilutionConc"}>{index+1}. Concentration:
                        <input type="number" value={dilution.concentration} className="inputField
dilutionField" name={"dilutionConc"} onChange={(e) => handleChange(index, 'concentration',
e.target.value)}/>
                        <select value={dilution.unit} name={"dilutionConcUnit"} className="dilutionField"
onChange={(e) => handleChange(index, 'concUnit', e.target.value)}>
                            <option value="µM">µM</option>
                            <option value="nM">nM</option>
                        </select>
                        <button type="button" className="dilutionButton orderButton removeButton Button"
onClick={() => handleRemoveDilution(index)}>X</button>
                    </label>
                    <br/>
                </div>
            ))}
            <br/>
            <button type="button" className='dilutionButton orderButton Button'
onClick={handleAddDilution}>Add Dilution</button>
        </div>
    );
}

export default Dilution;
```

Drug.jsx

```
import React from "react";
import {useState, useEffect} from 'react';
import Biomarkers from "./Biomarkers";
import CellTypes from "./CellTypes";
```

```
import DaysInVitro from "./DaysInVitro";
import SafetyDataSheet from "./SafeyDataSheet";
import Solvent from "./Solvent";

const Drug = ({onUpdateDrugs}) => {

    const [drugsList, setDrugList] = useState([]);
    const [total, setTotal] = useState(0);

    useEffect(() => {
        setDrugList([{name: "", weight: 0.0, unit: 'ug', biomarkers:[], cellTypes:[], solvents:[],
daysVitro:[]}])
    }, []);

    function handleAddDrug(drug){
        const newDrug = {
            name: "",
            weight: 0.0,
            unit: "ug",
            biomarkers: [],
            cellTypes: [],
            solvents: [],
            daysVitro: [],
          };
        setDrugList([...drugsList, newDrug]);
        onUpdateDrugs([...drugsList, newDrug]);
    }

    function handleRemoveDrug(index) {
        const removedDrugs = [...drugsList];
        removedDrugs.splice(index, 1);
        setDrugList(removedDrugs);
        onUpdateDrugs(removedDrugs);
      }

      function handleChange(index, field, value) {
        const updatedDrugs = [...drugsList];
        updatedDrugs[index][field] = value;
        setDrugList(updatedDrugs);
        onUpdateDrugs(updatedDrugs);
      }

      function handleBioMarkers(bios, index) {
        const updatedBios = [...drugsList];
        updatedBios[index].biomarkers = bios;
        setDrugList(updatedBios);
        onUpdateDrugs(updatedBios);
      }

      function handleCellTypes(cells, index) {
        const updatedCells = [...drugsList];
```

```
        updatedCells[index].cellTypes = cells;
        setDrugList(updatedCells);
        onUpdateDrugs(updatedCells);
      }

      function handleAddSolvents(solvs, index) {
        const updatedDrugs = [...drugsList];
        updatedDrugs[index].solvents = solvs;
        setDrugList(updatedDrugs);
        onUpdateDrugs(updatedDrugs);
    }

      function handleAddDaysVitro(vitro, index) {
        const updatedVitro = [...drugsList];
        updatedVitro[index].daysVitro = vitro;
        setDrugList(updatedVitro);
        onUpdateDrugs(updatedVitro);
      }

    //   function handleSafetyDataSheet() {
    //   }

    return (
        <div className="drugOrderComp" id="drugInfo">

            {drugsList.map((drug, index) => {
                return(
                    <div key={index} className="drugInfoDiv">
                        <h3>Drug{index+1} Info</h3>
                        <div>
                            <label htmlFor="drugName" className="drugInfoField">Drug Name
                                <input className="inputField" id="drugName" type="text" name="drugName"
value={drug.name} onChange={(e) => handleChange(index, 'name', e.target.value)} />
                                <p className="errorText" id="drugNameErr">Cannot be Empty</p>
                            </label>
                            <br/>

                            <label htmlFor="drugWeight" className="drugInfoField">Drug Weight
                                <input className="inputField" id="drugWeight" type="text"
name="drugWeight" value={drug.weight} onChange={(e) => handleChange(index, 'weight', e.target.value)} />
                                <p className="errorText" id="drugWeightErr">Cannot be Empty</p>
                                <select name="drugUnit" className="drugInfoField" id="drugUnit"
onChange={(e => handleChange(index, 'unit', e.target.value))}>
                                    <option value="µg">µg</option>
                                    <option value="mg">mg</option>
                                </select>
                            </label>
                        </div>
                        <div>
                            <SafetyDataSheet/>
                        </div>
```

```
                        <div>
                            <Biomarkers getBiomarkers={(bios) => handleBioMarkers(bios, index)}/>
                        </div>
                        <div>
                            <CellTypes getCellTypes={(cells) => handleCellTypes(cells, index)}/>
                        </div>
                        <div>
                            <Solvent solvents={drug.solvents} addSolvent={(solvents) =>
handleAddSolvents(solvents, index)} />
                        </div>
                        <div>
                            <DaysInVitro addDaysVitro={(daysVitro) => handleAddDaysVitro(daysVitro,
index)}/>
                        </div>
                        <button type="button" className="drugButton orderButton removeButton
Button"onClick={() => handleRemoveDrug(index)}>Remove Drug</button>
                        <hr/>
                </div>
                )
            })}
            <button type="button" className='drugButton orderButton Button' onClick={handleAddDrug}>Add
Another Drug</button>
            <hr/>
        </div>
    )
}

export default Drug
```

SafetyDataSheet.jsx

```
import React, {useState} from "react";

const SafetyDataSheet = () => {

    const [sdsPDFfile, setSdsPdfFile] = useState(null);
    const [pdfFileError, setPdfFileError] = useState('');

    const fileType =['application/pdf'];

    function importSDS(e) {
        //TODO implement drag and drop file box
        handleFileChange()
        let selectedFile = e.target.files[0];
        if(selectedFile && fileType.includes(selectedFile.type)) {
            let reader = new FileReader();
            reader.readAsDataURL(selectedFile);
            reader.onloadend = (e) => {
                setSdsPdfFile(e.target.result);
                setPdfFileError('');
```

```
            }
        }
        else {
            setSdsPdfFile(null);
            setPdfFileError('Please select a valid PDF');
        }
    }

    // function electronicSDS() {
    //     //TODO implemet electronic SDS form
    // }

    // function validateSDS() {
    //     //TODO implement SDS check
    // }

    // const handleDrag = (e) => {
    //     e.preventDefault();
    // }

    // const handleDrop = (e) => {
    //     e.preventDefault();
    //     const file = e.dataTransfer.files[0];
    //     handleFileChange(file);
    // }

    const handleFileChange = (file) => {
        if(file.type === "application/pdf") {
            setSdsPdfFile(file);
        }
        else {
            setSdsPdfFile(null);
            setPdfFileError('Please select a valid PDF');
        }
    }

    return(
        <div className="drugOrderComp" id="safetyDataSheet">
            <h3>Safety Data Sheet</h3>
            <div>
                <label htmlFor="sdsUpload">Upload Safety Data Sheet:
                <input type="file" className="sdsButton formButton" id="sdsUploadButton" onChange={(e) =>
importSDS(e)}/>
                </label>
                {/* <div className="drag&Drop" onDrop={handleDrop} onDrag={handleDrag}
                    style={{ border: "1px solid gray", padding: "20px", margin: "20px 0" }}>
                <p>Drag&Drop Here</p>
                {sdsPDFfile && (
                    <div>
                        <p>Selected File: {sdsPDFfile.name}</p>
                    </div>
```

```
                 )}
                </div> */}

                {/* <button className="sdsButton formButton" id="sdsElectronicButton" onClick={(e) => {
                    Open blank electronic SDS(?)
                     OSHA SDS Guidelines
https://www.osha.gov/sites/default/files/publications/OSHA3514.pdf
                }}
                >Electronic Form
                </button> */}
                {/* <label htmlFor="sdsCheckbox"></label>
                <input type="checkbox" id="sdsCheckbox" name="sdsCheckbox" value="sdsCheckbox"></input>
*/}
            </div>

        </div>
    )
}

export default SafetyDataSheet;
```

Solvent.jsx

```
import { useState, useEffect } from "react";
import Dilutions from "./Dilution";

function Solvent({addSolvent}) {

//regarding Solvents
const [solventsList, setSolventList] = useState([]);

    useEffect(() => {
        setSolventList([{name: "", volume: 0.0, volUnit: "mL", concentration: 0.0, concUnit: "nm",
dilutions:[]}])
    }, []);

    function handleAddSolvent() {
        const newSolvent = {
            name: "",
            volume: 0.0,
            volUnit: "mL",
            concentration: 0.0,
            concUnit: "nm",
            dilutions:[]
        }
        setSolventList([...solventsList, newSolvent]);
        addSolvent([...solventsList, newSolvent]);
    }

    function handleRemoveSolvent(index) {
```

```
        const removedSolvents = [...solventsList];
        removedSolvents.splice(index, 1);
        setSolventList(removedSolvents);
        addSolvent(removedSolvents);
    }

    function handleChange(index, field, value) {
        const updatedSolvents = [...solventsList];
        updatedSolvents[index][field] = value;
        setSolventList(updatedSolvents);
        addSolvent(updatedSolvents); // Call the callback function to update the drug
    }

    const handleAddDilutions = (dilutions, index) => {
        const updatedSolvents = [...solventsList];
        updatedSolvents[index].dilutions = dilutions;
        setSolventList(updatedSolvents);
        addSolvent(updatedSolvents);
    }

    return (
    <div className="solventDiv">
        <h3>Solvents:</h3>
        {solventsList.map((solvent, index) => {
            return (
            <div key={index} className="solventsDiv">
                <div className="solventField">
                    <h5>Solvent{index+1}</h5>
                    <label htmlFor={"solventName" + {index}}>Compound:
                    <input className="inputField" type="text" name={"solventName" + {index}}
value={solvent.name} onChange={(e) => handleChange(index, 'name', e.target.value)}
                    />
                    </label>
                </div>

                <div className="solventField">
                    <label htmlFor={"solventVolume" + {index}}>Volume:
                    <input className="inputField" type="number" name={"solventVolume" + {index}}
value={solvent.volume} onChange={(e) => handleChange(index, 'volume', e.target.value)}/>
                    <select className="solventSelect" name={"solventVolUnit" + {index}}
value={solvent.volUnit} onChange={(e) => handleChange(index, 'volUnit', e.target.value)}>
                        <option value="mL">mL</option>
                        <option value="L">L</option>
                    </select>
                    </label>
                </div>
                <div className="solventField">
                    <label htmlFor={"solventConcentration" + {index}}>Concentration:
                    <input className="inputField" type="number" name={"solventConcentration" + {index}}
value={solvent.concentration} onChange={(e) => handleChange(index, 'concentration', e.target.value)}/>
                    <select className="solventSelect" value={solvent.concUnit} onChange={(e) =>
```

```
handleChange(index, 'concUnit', e.target.value)}>
                        <option value="µM">µM</option>
                        <option value="nM">nM</option>
                    </select>
                    </label>
                </div>
                <h3 id="dilutionHeader">Dilutions</h3>
                <div className="solventField">
                    <Dilutions addDilution={(dilutions) => handleAddDilutions(dilutions, index)} />
                    <button type="button" className='solvent orderButton removeButton Button' onClick={()
=> handleRemoveSolvent(index)}>Remove Solvent</button>
                </div>
            </div>
        )
        })}
        <button type="button" className='solventButton orderButton Button' onClick={handleAddSolvent}>Add
Solvent</button>
    </div>

  );
}

export default Solvent;
```

Register.js

```
import React, { useState } from "react";
import {Link, useNavigate} from 'react-router-dom';
import './Register.css';

const Register = () => {
  const [username, setUsername] = useState('');
  const [email, setEmail] = useState('');
  const [password, setPassword] = useState('');
  const [error, setError] = useState('');
  const navigate = useNavigate();

  const handleSubmit = async (event) => {
    event.preventDefault();
    try {
      // Submit post request
      const response = await fetch(`${process.env.REACT_APP_REACT_LOCAL}/api/register`, {
        method: 'POST',
        headers: { 'Content-Type': 'application/json' },
        body: JSON.stringify({ username, email, password })
      });

      // This line extracts the JSON data from the response object.
      // The await keyword makes the function wait for the data to be extracted before moving on to the
next line of code.
```

```
      const data = await response.json();

      if (!response.ok) {
        throw new Error(data.message);
      }
      navigate("/home");
    } catch (error) {
      setError(error.message);
    }
  };

  return (
    <div className="Register">
      <h1>Register</h1>
      {error && <p className="error">{error}</p>}
      <form onSubmit={handleSubmit}>
        <input
          type="text"
          value={username}
          onChange={(event) => setUsername(event.target.value)}
          placeholder="Username"
        />
        <input
          type="email"
          value={email}
          onChange={(event) => setEmail(event.target.value)}
          placeholder="Email"
        />
        <input
          type="password"
          value={password}
          onChange={(event) => setPassword(event.target.value)}
          placeholder="Password"
        />
        <button type="submit">Register</button>
      </form>
      <br/>
      <Link to="/Login"><div>Already have an account?</div></Link>
    </div>
  );
};

export default Register;
```

Server.js

```
const express = require('express');
const mongoose = require('mongoose');
const bcrypt = require('bcryptjs');
```

```
const cors = require('cors');
const app = express();
const dotenv = require('dotenv');
dotenv.config();
app.use(cors());

// Set up the session middleware

// User
const userSchema = new mongoose.Schema({
    // Required fields:
    username: { type: String, required: true, unique: true },
    password: { type: String, required: true },
    email: { type: String, required: true, unique: true },
    userID: {type: String, required: true, unique: true},
    isAdmin: { type: Boolean, required: true },

    // Non-required fields:
    firstName: { type: String, required: false },
    lastName: { type: String, required: false },
    address: { type: String, required: false },
    phoneNumber: { type: Number, required: false }

}, { versionKey: false });

//Biomarkers
const biomarkerSchema = new mongoose.Schema({
  name: String,
  price: Number
});

const cellTypeSchema = new mongoose.Schema({
  name: String,
  price: Number
});
const dilutionSchema = new mongoose.Schema({
  concentration: {type: Number, required: true},
  unit: {type: String, required: false},
})

const solventSchema = new mongoose.Schema({
  name: {type: String, required: true},
  volume: {type: Number, required: true},
  unit: {type: String, required: false},
  concentration: {type: Number, required: true},
  dilutions: [dilutionSchema],
})

const daysVitroSchema = new mongoose.Schema({
  length: {type: Number, required: true},
  price: {type: Number, required: true},
```

```
})

const drugSchema = new mongoose.Schema({
  name: { type: String, required: true },
  weight: { type: Number, required: true },
  unit: { type: String, required: false },
  biomarkers: [biomarkerSchema],
  cellTypes: [cellTypeSchema],
  solvents: [solventSchema],
  daysVitro: {daysVitroSchema}
});

const orderSchema = new mongoose.Schema({
  userID: { type: String, required: true },
  orderID: { type: String, required: true },
  orderTitle: { type: String, required: true },
  orderTotal: { type: Number, required: true },
  orderStatus: { type: String, required: true },
  drugs: [drugSchema],
  submitDate: { type: String, required: false },
  updateDate: { type: String, required: false },

});

const resultSchema = new mongoose.Schema({
  orderID: {type: String, required: true},
  biomarker: {type: String, required: true},
  data: {
    type: mongoose.Schema.Types.Mixed,
    required: true,
  },
});

const contactSchema = new mongoose.Schema({
  name: { type: String, required: true },
  email: { type: String, required: true },
  phone: { type: String, required: false },
  message: { type: String, required: true },
  date: { type: Date, default: Date.now }
});

// Create 'User' model
const User = mongoose.model('User', userSchema);

// 'Order' model
const Order = mongoose.model('Order', orderSchema);

const Biomarker = mongoose.model('Biomarker', biomarkerSchema);

const CellType = mongoose.model('CellType', cellTypeSchema);
```

```
const Results = mongoose.model('Results', resultSchema);

const Contact = mongoose.model('Contact', contactSchema)

// Middleware to set Cross-Origin-Opener-Policy header
app.use((req, res, next) => {
  res.setHeader('Cross-Origin-Opener-Policy', 'same-origin-allow-popups');
  next();
});

// "... enables the Express middleware to parse incoming JSON data."
app.use(express.json());

app.get('/', (req, res) => {
  res.send('Need to address this use case. Redirect to login page?');
});


app.post('/api/login', async (req, res) => {
  // Get the username and password from the request body
  const { username, password } = req.body;

  try {
    // Find the user in the database by their username
    const user = await User.findOne({ username });

    if (!user) {
      // If the user does not exist in the database, send a JSON response with an error message
      return res.status(401).json({ message: 'Login failed: User not found' });
    }

    // Check if the password is correct
    const isMatch = await bcrypt.compare(password, user.password);

    if (isMatch) {
        //req.session.userId = User.findOne({ username }).userID;
      // If the password is correct, send a JSON response with a success message
      return res.status(200).json({ message: 'Login successful', user});
    } else {
      // If the password is incorrect, send a JSON response with an error message
      return res.status(401).json({ message: 'Login failed: Incorrect password' });
    }
  } catch (error) {
    // If there's an error while querying the database, send a JSON response with an error message
    return res.status(500).json({ message: 'Internal server error' });
  }
});

app.post('/api/register', async (req, res) => {
  // Get username, password, and email
  const { username, password, email } = req.body;
```

```
  try {
    // Verify that username is unique
    const existingUser = await User.findOne({ username: username });
    if (existingUser) {
      return res.status(400).json({ message: 'Username already exists' });
    }

    // Verify that email is unique
    const existingEmail = await User.findOne({ email: email });
    if (existingEmail) {
      return res.status(400).json({ message: 'Email already exists' });
    }

    // Hash the password
    const salt = await bcrypt.genSalt(10);
    const hashedPassword = await bcrypt.hash(password, salt);

    // Generate UUID
    const { v4: uuidv4 } = require('uuid');
    const uuid = uuidv4();

    // Create a new user object with hashed password
    const newUser = new User({
        username: username,
        password: hashedPassword,
        email: email,
        userID: uuid,
        isAdmin: false
    });

    // Save the new user to the database
    await newUser.save();

    return res.status(200).json({ message: 'User registered successfully' });
  } catch (error) {
    // If there's an error while querying the database or hashing the password, send a JSON response with
an error message
    return res.status(500).json({ message: 'Internal server error' });
  }
});

app.post('/api/order', async (req, res) => {
  const { userID, orderTitle, orderTotal, orderStatus, drugs} = req.body;
  const serializedDrugs = drugs.map(drug => {
    return {
      name: drug.name,
      weight: drug.weight,
      unit: drug.unit,
      biomarkers: drug.biomarkers,
      cellTypes: drug.cellTypes,
```

```
      solvents: drug.solvents,
      daysVitro: drug.daysVitro
    }
  });

  try {

    // Generate UUID
    const { v4: uuidv4 } = require('uuid');
    const uuid = uuidv4();

    // Create a new user object with hashed password
    const newOrder = new Order({
      orderID: uuid, // Alternative just have the order # as the ID, but I think this way is better
because customers will see it
      userID: userID,
      orderTitle: orderTitle,
      orderTotal: orderTotal,
      orderStatus: orderStatus,
      drugs: serializedDrugs,
      submitDate: new Date().toLocaleString().replace(/:\d{2}\s/, ' ').replace(/\//g, '-'),
      updateDate: new Date().toLocaleString().replace(/:\d{2}\s/, ' ').replace(/\//g, '-')
    });

    await newOrder.save(function (err) {
      if (err) {
        console.log('Error saving new order', err);
      } else {
        console.log('New order saved');
      }
    });

    // Save the new user to the database
    return res.status(200).json({ message: 'Order submitted successfully' });
  } catch (error) {
    // If there's an error while querying the database or hashing the password, send a JSON response with
an error message
    return res.status(500).json({ message: 'Internal server error' });
  }
});


const PORT = 3001;
const msg = `Running on PORT ${PORT}`;

mongoose.connect(`${process.env.REACT_APP_ATLAS_URI}`, {
  useNewUrlParser: true,
  useUnifiedTopology: true,
});

app.listen(PORT, () => {
```

```
  console.log(msg);
});

app.get('/api/biomarkers', async (req, res) => {
  Biomarker.find({})
    .then((data) => {
      res.json(data);
    })
    .catch((err) => {
      console.log('error: ', err);
    });
});

app.get('/api/celltypes', async (req, res) => {
  CellType.find({})
    .then((data) => {
      res.json(data);
    })
    .catch((err) => {
      console.log('error: ', err);
    });
});

app.get('/api/user', async (req, res) => {
  const { userID } = req.query;
  User.find({"userID" : userID})
    .then((data) => {
      res.json(data);
    })
    .catch((err) => {
      console.log('error: ', err);
    });
});

app.get('/api/admindashboard', async (req, res) => {
  Order.find({})
    .then((data) => {
      res.json(data);
    })
    .catch((err) => {
      console.log('error: ', err);
    });
});

app.get('/api/dashboard', async (req, res) => {
  const { userID } = req.query;
  Order.find({"userID" : userID})
    .then((data) => {
      res.json(data);
    })
    .catch((err) => {
```

```
      console.log('error: ', err);
    });
});

app.get('/api/results', async (req, res) => {
  const { orderID } = req.query;
  Results.findOne({"orderID" : '525a5c12-0a39-4d19-acbc-e3b4641b47b3'})
  .select('-_id -orderID -biomarker -data.Header')
  .then((result) => {
    res.json(result.data);
  })
  .catch((err) => {
    console.log('error: ', err)
  });
})

app.post('/api/contact', async (req, res) => {
  // Get input values from request body
  const { name, email, phone, message } = req.body;

  try {
    // Create a new contact object
    const newContact = new Contact({
      name: name,
      email: email,
      phone: phone,
      message: message,
      date: new Date()
    });

    // Save the new contact to the database
    await newContact.save();

    return res.status(200).json({ message: 'Contact information saved successfully' });
  } catch (error) {
    // If there's an error while saving the contact information to the database, send a JSON response with
an error message
    return res.status(500).json({ message: 'Internal server error' });
  }
});


app.put('/api/updateorder', async (req, res) => {
  const { orderID, orderStatus, updateDate } = req.body;
  try {
    const order = await Order.updateOne({"orderID" : orderID},
    {"$set" : {"orderStatus" : orderStatus, "updateDate" : updateDate} }
  );
    if (!order) {
      return res.status(404).send('Order not found');
    }
```

```
    res.send(order);
  } catch (err) {
    console.error(err);
    res.status(500).send('Server Error');
  }
});

app.post('/api/resultsup', async (req, res) => {
  console.log(req.body)
  const {orderID, biomarker, data } = req.body;
  try {
    // insert the parsed data into MongoDB
    const newResult = new Results({
      orderID: orderID,
      biomarker: biomarker,
      data: data,
    });

    await newResult.save(function (err) {
      if (err) {
        console.log('Error uploading result', err);
      } else {
        console.log('New result saved');
      }
    });
  } catch (error) {
    return res.status(500).json({ message: 'Internal server error' });
  }
});

// app.get('/api/resultdown', async (req, res) => {
//   const { orderID } = req.query;
//   Results.find({"orderID" : orderID})
//   .then((data) => {
//     res.json(data);
//   })
//   .catch((err) => {
//     console.log('error: ', err)
//   });
// })
```

CSS

App.css

```css
body {
    font-family: "Helvetica Neue", Helvetica, Arial, sans-serif;
}

.Button {
    width: auto;
    height: 30px;
    font-size: 16px;
    color: black;
    padding: 5px 10px;
    background-color: #89c8ff;
    border: none;
    margin: 10px;
    cursor: pointer;
}

.Button:hover {
    background-color: #6aa7e0;
}

.navButton {
  width: 120px;
  height: 40px;
  font-size: 16px;
  color: black;
  background-color: white;
  border: none;
    margin: 25% 5px 5px;
    cursor: pointer;
}

.navButton:hover {
  background-color: #89c8ff;
}
.Name {
  font-size: 30px;
  font-weight: bold;
}

.navButton:active {
   background-color: #89c8ff;
}

.sticky {
    position: fixed;
    top: 0;
}
```

```
.sticky + .Content {
    padding-top: 102px;
}

/*
main 3
*/
.Header {
  justify-content: space-between;
  align-items: center;
  background-color: white;
    padding: 10px 16px;
  border-bottom: solid black 5px;
    margin-left: 15%;
    margin-right: 15%;
}
    /*
    * sub groups
    */
    .Logo{
        width: 12.5%;
        height: 20%;
        margin: 15px;
    }

    .footerLogo{
        width: 7.5%;
        height: 15%;
        margin: 15px;
    }

    .NavBar {
        display: flex;
        align-items: center;
        float:right;
      }
      .clear{
          clear:both;
      }

.Content{
    display: block;
    margin-left: 15%;
    border-bottom: 5px solid black;
    width: 70%;
    padding: 16px;
}

.Footer{
    width: 70%;
```

```
    margin-left: 15%;
    margin-top: 5px;
    display: block;
}
.footerButtons{
    align-content: center;
    float: right;
}
```

About.css

```
.servicePanel{
    margin-top: 50px;
}

.panelButton.active {
    background-color: lightblue;
}
.selected-panel {
    background-color: lightgrey;
}
.inline-wrapper {
    display: flex;
    margin-left: 0%;
    width: 96.5%;
}

.panelButton {
    width: 90%;
    height: 40px;
    font-size: 16px;
    color: black;
    background-color: #89c8ff;
    border: none;
    margin: 5px 5px 5px;
    cursor: pointer;
}

.panelButton:hover {
    background-color: lightblue;
}

.service-description{
    width: 90%;
    text-align: left;
    margin-top: 75px;
    margin-left: 5%;
}

.serviceImg{
```

```
    width: 300px;
    margin-top: 0%
}
```

ContactUs.css

```
.input{
    margin-bottom: 3%;
    width: 33%;
    padding: 12px 20px;
    box-sizing: border-box;
    border: 2px solid #ccc;
    border-radius: 4px;
    background-color: #f8f8f8;
}

.title{
    font-weight: bold;
    margin-left: 0;
}

.message{
    width: 100%;
    height: 150px;
    padding: 12px 20px;
    box-sizing: border-box;
    border: 2px solid #ccc;
    border-radius: 4px;
    background-color: #f8f8f8;
    resize: none;
}
```

Dashboard.css

```
.dashNav {
    list-style: none;
    display: inline-flex;
    border: solid 2px black;
}

.dashNavItem {
    padding: 5px 10px 5px 10px;
    border: solid 1px black;
}

.dashboardList {
    list-style: none;
}

.table{
```

```
    padding: 15px;
    text-align: center;
}

.orderInfo {
    display: inline-flex;
    padding: 5px 10px 5px 10px;
}

.dashListItem {
    border: solid 1px black;
}

.tableHeaderRow {
    background-color: #89c8ff;
}

.tableRow:nth-child(even) {
    background-color: Lightgrey;
}

.subOrderInfo {
    width: 100%;
    text-align: center;
    border-style: ridge;
    border-width: 1px;
    border-color: #89c8ff;
    border-top: none;
}

.dashButton {
    border-radius: 50%;
    align-items: center;
    width: 3vw;
    height: 3vw;
    max-width: 20vw;
    max-height: 20vw;
    background-size: 75%;
    background-position: center;
    background-repeat: no-repeat;
    display: inline-flex;
}

.downloadButton {
    background-image: url("../../../public/downloadIcon.svg");
}

.graphButton {
    background-image: url("../../../public/graphIcon.svg");
}
```

```
.csvButton {
    background-image: url("../../../public/csvIcon.svg");
}

.statusChangeDiv {
    display: none;
    position: fixed;
    margin: 0;
    top: 50%;
    left: 30%;
    min-width: fit-content;
    min-height: fit-content;
    width: 50%;
    z-index: 9999 !important;
    background-color: white;
    border-radius: 25px;
    border: 2px solid #89c8ff;
    text-align: center;
    padding-bottom: 15px;
}

.darkenDiv {
    display: none;
    position: fixed;
    top: 0px;
    left: 0px;
    width: 100%;
    height: 100%;
    background-color: rgba(0, 0, 0, .5);
    opacity: 1;
}

.statusChangeButton {
    width: 35%;
    padding: 10px;
    margin-top: 20px;
    margin-left: 5px;
    margin-right: 5px;
    font-size: 16px;
    background-color: #89c8ff;
    color: black;
    border-radius: 5px;
    border: none;
    cursor: pointer;
}

#confirmButton:disabled {
    width: 35%;
    padding: 10px;
    margin-top: 20px;
    margin-left: 5px;
```

```
    margin-right: 5px;
    font-size: 16px;
    background-color: lightgrey;
    color: white;
    border-radius: 5px;
    border: none;
}

#confirmButton:enabled {
    width: 35%;
    padding: 10px;
    margin-top: 20px;
    margin-left: 5px;
    margin-right: 5px;
    font-size: 16px;
    background-color: #89c8ff;
    color: black;
    border-radius: 5px;
    border: none;
    cursor: pointer;
}

.statusDivContent input{
    content: 'Select some files';
    display: inline-block;
    background: linear-gradient(top, #f9f9f9, #e3e3e3);
    border: 1px solid #999;
    border-radius: 3px;
    padding: 5px 8px;
    outline: none;
    white-space: nowrap;
    cursor: pointer;
    text-shadow: 1px 1px #fff;
    font-weight: 700;
    font-size: 10pt;
}
```

Home.css

```
.home {
  background-color: white;
  border-top: none;
}

body {
  margin: 0;
}
```

```
img {
  width: 90%;
  margin-left: 5%;
  /*margin-top: 25px;*/
  margin-bottom: 25px;
}

.Slogan {
  font-size: 34px;
  margin: 5%;
  text-align: center;
  font-weight: bold;
  margin-top: 0;
}
.caption {
  font-size: 15px;
  text-align: center;
}

.services {
  width: 100%;
}

.description {
 width: 50%;
  margin-top: 5%;
  margin-bottom: 5%;
  padding: 10px;
}

.visual {
  margin-top: 25px;
  padding: 10px;
  display: flex;
  justify-content: center;
  align-items: center;
  width: 50%;
}

.inline {
  display: flex;
}

.srv-btn {
  align-items: center;
  background-color: aliceblue;
  border: 2px solid black;
  color: black;
  padding: 20px;
  text-align: center;
```

```
  text-decoration: none;
  display: inline-block;
  font-size: 16px;
  margin: 12px 4px;
  border-radius: 50%;
}

.srv-btn:hover {
  background-color: lightblue;
}

.service-description {
  text-align: center;
}

.content {
  padding-top: 32px;
}
```

Login.css

```
.Login {
  display: flex;
  flex-direction: column;
  align-items: center;
  margin-top: 20px;
}

.Login form {
  display: flex;
  flex-direction: column;
  align-items: center;
  background-color: #89c8ff;
  padding: 20px;
  border-radius: 10px;
  box-shadow: 2px 2px 10px rgba(0, 0, 0, 0.3);
}

.Login input[type="text"],
.Login input[type="email"],
.Login input[type="password"] {
  width: 100%;
  padding: 10px;
  margin-bottom: 20px;
  font-size: 16px;
  border-radius: 5px;
  border: none;
  box-shadow: 2px 2px 5px rgba(0, 0, 0, 0.3);
}
```

```
.Login button[type="submit"] {
  width: 100%;
  padding: 10px;
  margin-top: 20px;
  font-size: 16px;
  background-color: white;
  color: black;
  border-radius: 5px;
  border: none;
  cursor: pointer;
}

.Login .error {
  margin-top: 20px;
  color: red;
}
```

Order.css

```
.Order {
    border: .1em solid #85c7ff;
    border-radius: 5em;
    width: 100%;
    display: flex;
    justify-content: center;
    box-shadow: 0 0 2px 3px #85c7ff;
}

h3{
    display: flex;
    flex-direction: row;
    justify-content: center;
    text-align: center;
    background-color: #85c7ff;
    border-radius: 5em;
    padding: .5em;
}

.orderCategory{
    border: 1px solid black;
}

.bioWrapper {
    display: flex;
    flex-wrap: wrap;
    justify-content: center;
}
```

```
.bioMarkerItem {
    width: calc(50% - 10px);
    margin: 5px;
    box-sizing: border-box;
    justify-content: center;
    text-align: center;
}

.inputField {
    height: 20px;
    flex: 0 0 200px;
    margin-left: 10px;
}

.errorText {
    display: none;
}

.cellWrapper {
    display: flex;
    justify-content: center;
  }

.cellTypeItem {
    padding: 0px 20px 0px 10px;
    display: inline-block;
}

#removeSolventButton {
    display: none;
}

#dilutionHeader {
    display: none;
}

.daysVitroItem {
    display: flex;
    align-items: center;
}

.orderInfoField {
    display: flex;
    flex-wrap: wrap;
    align-items: center;
  }

  .orderInfoLabel {
    text-align: right;
    margin-right: 10px;
  }
```

```
  .orderInfoInput {
    flex: 0 0 50%;
  }
```

Register.css

```
.Register {
  display: flex;
  flex-direction: column;
  align-items: center;
  margin-top: 20px;
}

.Register form {
  display: flex;
  flex-direction: column;
  align-items: center;
  background-color: #89c8ff;
  padding: 20px;
  border-radius: 10px;
  box-shadow: 2px 2px 10px rgba(0, 0, 0, 0.3);
}

.Register input[type="text"],
.Register input[type="email"],
.Register input[type="password"] {
  width: 100%;
  padding: 10px;
  margin-bottom: 20px;
  font-size: 16px;
  border-radius: 5px;
  border: none;
  box-shadow: 2px 2px 5px rgba(0, 0, 0, 0.3);
}
.Register button[type="submit"] {
  width: 100%;
  padding: 10px;
  margin-top: 20px;
  font-size: 16px;
  background-color: black;
  color: white;
  border-radius: 5px;
  border: none;
  cursor: pointer;
}
.Register .error {
  margin-top: 20px;
  color: red;
}
```