**ROZDZIAŁ 3**

**Badany Obszar**

**3.1. Start projektu**

3.1.1. Utworzenie projektu

Cała aplikacja postawiona będzie na szkielecie next.js. Do utworzenia projektu wykorzystany zostanie npm (node package menager). Jest to największe repozytorium oprogramowania, które umożliwia programistom open source z każdego miejsca na ziemi na dzielenie się oraz korzystanie z pakietów, a również organizacjom na zarządzanie prywatnymi projektami. Jej głównymi elementami są strony internetowe, interfejs wiersza poleceń (CLI) oraz rejestru.[[1]](#footnote-1) Zacząć należy od instalacji node, w tym celu należy wejść na stronę nodejs.org w celu pobrania pliku instalacyjnego. Node jest to środowisko, które uruchamia JavaScript zbudowane na silniku JavaScript V8[[2]](#footnote-2). Node package manager powinien się zainstalować wraz z zainstalowaniem się Node. W celu sprawdzenia poprawności zainstalowania się Node oraz NPM można wpisać następujące komendy wiersza poleceń „npm -v” oraz „node -v”. Literka „-v” jest to flaga, która oznacza version czyli wersję na początku npm potem node.

rys. 1 wersja node

Obraz zawierający tekst, Czcionka, zrzut ekranu, czarne

Opis wygenerowany automatycznie

rys. 2 Wersja npm

Obraz zawierający tekst, Czcionka, zrzut ekranu, czarne

Opis wygenerowany automatycznie

Tak samo jak na rysunku 1 jak i 2 widać, że ukazuje się wersja co znaczy, że node jak i npm zostały poprawnie zainstalowane. Następnie, można przystąpić do utworzenia projektu next.js. W tym celu należy utworzyć folder w prawidłowej ścieżce w której chcemy trzymać pliki. Można zrobić to na dwa sposoby przez wiersz poleceń i wpisanie komendy „mkdir [nazwa]”, lub przez prawy przycisk myszy następnie nowy i folder. Warto zaznaczyć, że wszystkie wymienione przykłady są prezentowane na systemie operacyjnym windows 11. Na Linuxie czy też MacOS, może się to nieco różnić. Jeśli folder już istnieje należy do niego wejść „cd [nazwa folderu]” i wpisać komendę, którą podaje oficjalna strona next.js „npx create-next-app@latest .”. Kropka na końcu oznacza, że projekt zostanie utworzony w aktualnej ścieżce w której tworzący projekt się znajduje. Następnie wyskoczy pytanie o paczkę należy wpisać „y” od yes czyli tak.

rys. 3 Tworzenie aplikacji w next.js

Obraz zawierający tekst, zrzut ekranu, oprogramowanie, Oprogramowanie multimedialne

Opis wygenerowany automatycznie

Następnie wyskoczą inne pytania konfiguracyjne jak preferowany język JavaScript/TypeScript czy sposób pisania stylów. Po poprawnie zainstalowanym projekcie na samym końcu wyskakuje zielony napisas „Success!” co oznacza sukces. Do wystartowania projektu w środowisku lokalnym wystarczy użyć komendy w folderze w którym został zainstalowany projekt „npm run dev”.

rys. 4 Uruchomienie aplikacji

Obraz zawierający tekst, zrzut ekranu, Czcionka

Opis wygenerowany automatycznie

Widać, że projekt uruchomił się na porcie 3000 po wejściu na stronę zostały wykonane pierwsze zapytania typu GET.

3.1.2. Architektura plików

Projekt przyjmie podstawową architekturę plików, którą zaleca producent.

rys. 5 Architektura plików

Obraz zawierający tekst, zrzut ekranu, oprogramowanie, Oprogramowanie multimedialne

Opis wygenerowany automatycznie

Foldery takie jak .idea, .next czy node\_modules są to foldery konfiguracyjne szkieletu wiec one nie muszą zostać omówione, ponieważ w każdym projekcie są takie same w zależności oczywiście od wersji. Następnie folder app jest to routing aplikacji w którym zostaną utworzone foldery z nazwami podstron jak na przykład „about” w którym umieścić należy plik page.tsx z odpowiednim kodem TypeScriptowym. W takim przypadku endpoint „/about” (na przykład <http://localhost:3000/about>) powinien przenieść na odpowiednia podstronę. Folder „Components” zawiera wszystkie reużywalne komponenty które można następnie używać w całych featurach. Wiec odpowiednio folder features składa się na pliki zawierające połączenie wszystkich niezbędnych komponentów do utworzenia widoku. Folder Context zawiera wszystkie contexty czyli takie mechanizmy w react, które umożliwiają przekazywanie danych przez drzewo komponentów, bez konieczności ręcznego przekazywania propsów na każdym poziomie. Gdyby nie kontekst przekazywanie danych wyglądałoby w następujący sposób od rodzica do dziecka czyli w patrząc na drzewo komponentów od góry do dołu. Jednak by ułatwić przekaz niewygodnych danych jak motyw interfejsu, które są wymagane przez wiele komponentów w aplikacji stworzono kontekst. Zapewnia on również mechanizm dzielenia się wartościami między komponentami bez konieczności jawnego przekazywania danych przez każdy poziom drzewa.[[3]](#footnote-3) Hooks jest to folder zawierający wszystkie hooki, które służą do przetrzymywania stanu aplikacji bez konieczności pisania klas.[[4]](#footnote-4) Folder lib zawiera wszystkie konfiguracje bibliotek jak na przykład utworzenie klienta prisma czy opcje autoryzacji do biblioteki next auth. Folder messages zawiera jedynie pliki typu json, które zawierają tłumaczenia stron jeśli takie by wystąpiły. Prisma zawiera wszystkie pliki, użyte do obsługi bazy danych. Public zwiera zdjęcia ikony i tym podobne. Na końcu utils zawiera funkcję, które mogą być użyte w wielu komponentach jak na przykład przeliczenie jednostek. Z dalszych plików powinien interesować tylko plik tailwind.config.ts w którym zawarte są wartości styli nadawanych później w projekcie, .env w który zawarte są zawarte zmienne środowiskowe jak np. adres bazy danych czy domeny aplikacji oraz .gitignore w którym zapisuje się czego nie wysyłać do repozytorium z powodu za dużej ilości plików takich jak pliki konfiguracyjne.

3.1.3. Biblioteki

Tworząc projekt warto skorzystać z darmowych bibliotek ułatwiających i przyśpieszających pracę. Jedną z takich biblioteka jest biblioteka dzięki której możemy korzystać z gotowych komponentów i przerabiać je pod daną aplikację i dany wygląd. Biblioteka ta nazywa się NextUI. Do instalacji tej biblioteki jak i każdej zawartej w tym projekcie należy użyć node package managera wpisując komendę „npm install @nextui-org/react framer-motion”. Nastepnie wystarczy importować dany komponent „import {Button, ButtonGroup} from "@nextui-org/react";” i można go użyć w następujący sposób na przykładzie guzika.

rys. 6 Przykład użycia Buttona z NextUI

Obraz zawierający tekst, zrzut ekranu, Czcionka

Opis wygenerowany automatycznie

Dodatkowo można dodać jako props (np. color=”primary”), takie własności jak rozmiar, wariant promień zakrzywienia krawędzi, wstawienie ikony i wiele innych o których mówi dokumentacja. Ponadto możemy zmieniać wartości nadane już przez NextUI jak na przykład kolor primary może być kolorem który nadpiszemy w pliku tailwind.config.ts. Komponent ten funkcjonuje jak normalny tag HTML <buton/> czyli możemy przekazywać funkcję onClick, onChange itd. Biblioteką użytą do obsługi wysyłania na serwer zapytań klienta będzie axios. Jest to klient HTTP opartym na obietnicach (z ang. Promise-based) przeznaczonym do pracy zarówno w node.js, jak i w przeglądarce. Jest narzędziem izomorficznym, co znaczy, że może być używane w obu środowiskach za pomocą tego samego kodu. Na serwerze wykorzystuje natywny moduł http w node.js, podczas gdy w przeglądarce korzysta z żądań XMLHttpRequest.[[5]](#footnote-5) Do zainstalowania wystarczy użyć komendy „npm install axios”. Nastepnie można przejść już do użycia axiosa. Na przykładzie zapytania typu POST

rys. 7 Przykład użycia axios

Obraz zawierający tekst, zrzut ekranu, Czcionka, design

Opis wygenerowany automatycznie

W tym przykładzie została użyta metoda POST do wysłania danych określonych w data na backend znajdujący się pod danym endpointem. W tym przypadku musiała zostać utworzona instancja axiosa w której można określić stały adres i wpisywać tylko endpointy. W takim podstawowym użyciem można dodać jeszcze takie parametry jak headers czy parametry. Biblioteką użytą do warunkowego stylowania w pracy, został wykorzystana bilbioteka clsx. Do instalacji wystarczy użyć komendy „npm install clsx”. Stylowanie warunkowe jest bardzo przydatne do na przykład nadawania kolorów komponentów, które są odpowiedzialne za zadania, jak nadanie priorytetu zadaniu, gdzie zadanie z mniejszym priorytetem ma kolor zielony, a z większym priorytetem kolor czerwony. Biblioteka bcrypt została użyta do zaszyfrowania haseł trzymanych w bazie danych. Do obsługi i walidacji formularzy zostały użyte biblioteki zod oraz react hook form. Pozwalają one na pobranie odpowiednich typów danych z formularzy deklarując je szybciej. Po walidacji zwracane, jeśli występują błędy na przykład zły format emailu czy niespełnienie warunków hasła typu wielka litera, znak specjalny itd. Do zarządzania stanem całej aplikacji czyli pisaniem odpowiednich hooków, które pomagają pobierać dane i przechowywać użyto biblioteki react query. Później we wszystkich komponentach wystarczy użyć hooka, który dostarcza biblioteka, do użycia danych. W projekcie zastosowane jest również tłumaczenie, a do tego została użyta biblioteka next-intl. Tutaj należy ustawić w odpowiedni sposób plik layout, który definiuje RootLayout czyli główny układ strony co znaczy, że należy nadać odpowiedni kontekst stronie zależnie jaki stan języka strony jest obecnie aktywny. Opisany wyżej katalog messages uczestniczy w tworzeniu tłumaczenia gdyż w nim zawarte są pliki JSON z tłumaczeniami na wybrane języki, których potrzebuje aplikacja. W tym przypadku jest to angielski oraz polski czyli pliki kolejno nazywać się będą en.json oraz pl.json. Należy również ustawić middleware, który odpowiedzialny jest za obsługę tłumaczenia w tym przypadku jest połączony z autoryzacją wiec więcej o nim w kolejnym podrozdziale.

3.1.4. Autoryzacja

Autoryzacja jest bardzo ważną częścią projektu. Już przy rejestracji użytkownik musi wpisać odpowiednie dane jak: imię, nazwisko, email, hasło, zaakceptować warunki korzystania oraz wybrać jedną z dwóch opcji, czy jest kierowcą czy osobą zamawiającą. Jest to o tyle ważne, że widok i pokazywane dane będą pokazywane w odpowiedniej ilość i sposób dla danej kategorii. To autoryzacji została użyta odpowiednia biblioteka, która pozwala zrobić autoryzacje we szkielecie next.js. Pozwala na obsługę dowolnej usługi OAuth oraz obsługuje OAuth 1.0, 1.0A, 2.0 oraz OpenID Connect. Next auth posiada na dodanie odpowiednich providerów logowania takich jak github czy google. Wystarczy na podstawie githuba zrobić OAuth aplikacje w ustawieniach dla deweloperów, podać odpowiedni adres URL i można cieszyć się logowaniem przy użyciu github, oczywiście przez dodanie odpowiedniego providera w kodzie.

rys. 8 GithubProvider

Obraz zawierający tekst, zrzut ekranu, Czcionka

Opis wygenerowany automatycznie

NextAuth obsługuje również uwierzytelnianie za pomocą adresu e-mail bez hasła. Biblioteka wspiera uwierzytelnianie bezstanowe z dowolnym backendem jak Active Directory czy LDAP oraz innych. Obsługuje tokeny JWT oraz sesje w bazie danych. Stworzony pod kątem Serverless, lecz działa wszędzie (AWS Lambda, Docker, itd.). Autoryzacja przy użyciu nextAuth, może być używana z użyciem bazy danych lub bez jej użycia. Stosuje koncept BYOD (z ang. Bring Your Own Database) co znaczy, że może zostać użyty z dowolną bazą danych. Posiada wbudowaną obsługę MySQL, Postgres, MariaDB, SQL Server, SQLite oraz MongoDB. Wykorzystuje tokeny Cross-Site Request Forgery (CSRF) do zapytań typu POST. Polityka plików cookie domyślnie dązy do stosowania polityki, która jest najbardziej restrykcyjna odpowiedniej dla każdego pliku cookie. Podczas gdy JSON Web Tokens są włączone, domyślnie zostają szyfrowane (JWE) przy pomocy A256GCM. Dla ułatwienia deweloperom automatycznie generuje symetryczne klucze do podpisywania i szyfrowania. Platforma oferuje synchronizację kart oraz wiadomości keepalive, aby wspierać krótkotrwałe sesje. Stosowane są najnowsze wytyczne publikowane przez Open Web Application Security Project. Zaawansowane opcje pozwalają na definiowanie własnych procedur obsługi kontroli, które konta mogą się logować, kodować i dekodować tokeny JSON Web Tokens oraz ustawić niestandardowe zasady bezpieczeństwa plików cookie i właściwości sesji, co pozwala kontrolować dostęp i częstotliwość walidacji sesji.[[6]](#footnote-6) By wprowadzić autoryzację NextAuth do aplikacji należy w pierwszym kroku po zainstalowaniu biblioteki, dodać API route pod ścieżką „/app/api/auth/[…nextauth]”, plik route.ts w którym zawarta będzie inicjalizacja NextAuth.

rys. 9 Inizacjilacja Next auth

Obraz zawierający tekst, zrzut ekranu, oprogramowanie, Oprogramowanie multimedialne

Opis wygenerowany automatycznie

Jak widać jest definiowana funkcja o nazwie „handler”, która jest wynikiem wywołania fukcji NextAuth z argumentem authOptions, który został stworzony w innym katalogu dla utrzymania czystości kodu. Następnie zostaje wyeksportowane jako GET oraz POST co znaczy, że funkcja obsługuje zarówno żądania typu GET jak i również POST. Do prawidłowego skonfigurowania opcji autoryzacji należy podać strategię sesji w tym przypadku będzie to JWT. Następnie będzie to podanie odpowiedniego klucza, który pozwoli aplikacji rozpoznać zautoryzowaną sesję. Dobry klucz można wygenerować przy użyciu wiersza poleceń, przy pomocy komendy „openssl rand -base64 32”. Później w kolei jest zdefiniowanie callbacków. Callback session, który jest wywoływany poczas tworzenia sesji i w przypadku ten aplikacji będzie zwracał całą sesję rozszerzoną o dane usera oraz callback jwt, który jest wywoływany podczas generowania JWT i zwraca całe dane o tokenie rozszerzone i id usera, jego adres e-mail oraz imię. Następnie na samym końcu należy sprecyzować providery jak wspomniane szybciej można użyć githuba lub googla lecz w tej aplikacji użyty będzie tylko ten który będzie stworzony w formularzu czyli logowanie przez podanie adresu email oraz hasła. Należy również stworzyć asynchroniczną funkcję do autoryzacji która sprawdzi czy hasło i email zostały podane, następnie jeśli warunek się spełni, sprawdzi czy użytkownik o podanym adresie e-mail istnieje w bazie, kolejnym krokiem jest sprawdzenie czy hasło użytkownika się zgadza jeśli tak sesja zostanie zwrócona, a jeśli nie można zwrócić błąd, który następnie zostanie wyświetlony użytkownikowi. Gdy wszystkie te rzeczy zostały wykonane, można teraz rozszerzyć architekturę plików o foldery private, gdzie zostaną utrzymywane podstrony, które dostępne będą tylko po autoryzacji oraz public, gdzie zawarte będą podstrony do których niewymagana jest autoryzacja w tym przypadku strona logowania oraz rejestracji.

rys. 10 Rozszerzony folder app

Obraz zawierający tekst, zrzut ekranu, Czcionka, numer

Opis wygenerowany automatycznie

Gdy wszystkie kroki zostaną wykonane, można w prawidłowy sposób użyć funkcji signIn, która dostarcza producent NextAuth, która odwoła się do funkcji autoryzującej i podczas gdy wszystko będzie się zgadzać przeniesie do odpowiedniej postronny z katalogu private. Na końcu zostaje skonfigurować middleware.

rys. 11 MIddleware

Obraz zawierający tekst, zrzut ekranu, oprogramowanie

Opis wygenerowany automatycznie

W tym przypadku jak wspomniane było w podrozdziale wyżej odpowiedziany on jest za obsługę tłumaczenia oraz przekierowywanie ze strony prywatnych gdy nie uzyskana jest autoryzacja. W przypadku tej aplikacji jeśli w url bez autoryzacji, ktoś będzie chciał przenieść się na endpoint na przykład „/orders”, gdzie ta podstrona jest prywatna, zostanie przekierowany na podstronę logowania.

3.1.5. Ważne funkcjonalności

Ważnymi funkcjonalnościami są te co za pewne pojawią się w aplikacji jako pierwsze. Jest to związane ze słabymi rozwiązaniami obecnie panującymi na rynku. Ale odpowiedzieć należy na pytanie „Która z funkcjonalności jest najważniejsza dla aplikacji oraz przedsiębiorstwa?”. A więc biorą pod uwagę braki w płynności oraz ogólne słabe działanie aplikacji z których przedsiębiorstwo korzystało dotychczas oraz charakter w jaki przedsiębiorstwo funkcjonuje, można stwierdzić, że jedną z najważniejszych funkcjonalności będzie ta w której dodawane są towary do odebrana w dany dzień. Kierowcy oraz osoby zamawiające będą miały płynie działające narzędzie do wymiany informacji co przyśpieszy pracę, zwiększy efektywność oraz ekonomiczność jazdy kierowców, a to wszystko związane jest z zmniejszeniem prawdopodobieństwa niezadziałania aplikacji i spowodowania, że kierowca będzie musiał się wracać na daną hurtownie z której towar został już odebrany.

**3.2. System na diagramach**

3.2.1. Diagram przypadków użycia

Rozważając diagram przypadków użycia (z ang. use case), należy najpierw rozpatrzyć co oznaczają poszczególne znaki. Kwadrat opisany „system” jest to cały system aplikacji, wszystko co jest wewnątrz jego krawędzi dzieje się w systemie. Okręgi z małym kółkiem na obwodzie, znajdujące się w systemie nazwane są przypadkiem operacyjnym. Opisują zbiór działań, które podmioty wykonują, oraz które mogą ale nie muszą przynosić widoczny wynik, mającego wartość dla jednego lub kilku aktorów lub innych interesariuszy. Działania w przypadku operacyjnym mają miejsce jedynie w wyniku działania w przypadku użycia lub są inicjowane przez obserwatora lub czasomierz. Czasomierz jest to symbolizacja wewnętrznego zegara systemu lub konkretnego interwału czasu odzwierciedlonego w implementującym oprogramowaniu. Jest on na wykresie opisywany jako kwadrat z opisem „<<Timer>>” . Obserwator jest to komponent systemowy, który uruchamia przypadki operacyjne w reakcji na zdarzenie wewnętrzne lub zewnętrzne. Reprezentowany jest również jako kwadrat z opisem „<<Observer>>”[[7]](#footnote-7). Występują również aktorzy, czyli osoby bądź systemy zewnętrzne, które biorą udział w funkcjonowaniu systemu. Jest oznaczony na diagramie jako postać tzw. „patyczak”. Oraz występują również połączenia, pomiędzy aktorami, a poszczególnymi przypadkami użycia oraz pomiędzy przypadkami użycia.

rys. 12 Diagram przypadków użycia

Obraz zawierający tekst, diagram, linia, zrzut ekranu

Opis wygenerowany automatycznie

W tym systemie biorą udział trzech aktorów, dwóch interesariuszy: kierowcy oraz osoby zamawiające, jak i platforma zakupowa przedsiębiorstwa, na której osoby bądź firmy zamawiający towar składają swoje zamówienia. Połączenie z platforma w dalszej możliwej do osiągnięcia części rozwoju systemu będzie miało znaczący wpływ na dodawanie zamówień na konkretny dzień. Osoby zamawiające będą miały wgląd na całość zamówień, gdyż będą mogły dzięki temu w łatwy sposób zamawiać towar z dalszych hurtowni, a kierowcy potrzebują tylko widoku swoich zamówień, które będą posegregowane w taki sposób, że dostosują się do wybranego przez kierowcę rejonu w którym jeździ w dany dzień. Zazwyczaj jest to obszar na stałe wiec nie w ustawieniach będzie jedynie możliwość zmiany jeśli przyjdzie taka konieczność, żeby system nie musiał pytać przy każdym logowaniu. Kolejnym przypadkiem użycia jest lista zakupów, do której będą mieli dostęp interesariusze, w taki sposób, że kierowcy będą mogli przeglądać i zaznaczyć jako kupione, a osoby zamawiające dodawać oraz edytować listy zakupowe. Kalendarz czyli na diagramie przypadek użycia dodaj obecność, będzie tylko dostępny dla kierowców ze względu na ich umowy i to, że najzwyczajniej osoby zamawiające tej funkcjonalności nie potrzebują do swojej pracy. Zostają jeszcze listy zadań do wykonania i tutaj podobnie jak listy zakupowe, osoby odpowiedzialne za zamawianie będą miały dostęp do edycji, dodawania czy napisania notatki do zadań oraz nadawania priorytetu, a kierowcą potrzebny jest jedynie dostęp do przeglądania zadań oraz oznaczenia wykonanych jako zrobione.

3.2.2. Diagram związków encji

Model danych stanowi istotny element procesu projektowania koncepcyjnego, skupiając się na określeniu, jak informacja należy przechowywać w bazie danych. Model danych służy do zaprojektowania odpowiednich tabel. Ten proces obejmuje tworzenie diagramów encji i relacji, jak również używanie języka UML jak i innych narzędzi do opisu struktury danych przed stworzeniem fizycznej bazy. Modelem danych nazwane są również techniki lub narzędzia wykorzystywane do przedstawienia logicznej struktury danych, przed przekształceniem ich w kod programów. Celem tego procesu jest zapewnienie kompletnego i dokładnego odwzorowania wszystkich danych potrzebnych w bazie danych. Dzięki używaniu prostych notacji i języka zrozumiałego dla użytkowników końcowych, model danych może być łatwo weryfikowany przez nich pod kątem poprawności. Diagram związków encji jest graficznym narzędziem, które pomaga wizualizować strukturę danych oraz relacje między nimi, reprezentując świat rzeczywisty jako zbiór encji i relacji. Podsumowując, model danych ma za zadanie precyzyjnie uchwycić strukturę danych, obejmując typy encji, atrybuty, relacje oraz wszelkie ograniczenia występujące w systemie.[[8]](#footnote-8)

rys. 13 Diagram związków encji

Obraz zawierający tekst, zrzut ekranu, diagram, Plan

Opis wygenerowany automatycznie

Diagram ten został wygenerowany przy użyciu schematu prisma, mianowicie na samym początku należy przemyśleć oraz rozpisać wszystkie modele prsima gdzie należy sprecyzować nazwę oraz typ. Należy przy pomocy rozszerzenia do edytora tekstu jakim jest visual studio code, został wygenerowany diagram związków encji. Jak widać na rysunku podstawowa aplikacja posiada pięć modeli, które będą obsługiwać trzy podstawowe funkcjonalności. Zaczynając od użytkownika, można zauważyć, że posiada on najwięcej modeli danych. Zaczynając od podstawowych jak imię i nazwisko czy kończąc na nieobowiązkowym polu numeru rejestracyjnego. Nie jest on obowiązkowy ze względu na osoby kupujące, które nie mają sprecyzowanego numeru, ponieważ nie mają przypisanego samochodu do siebie. Relacje występują między modelem „Presence” i „User” oraz między „Product” i „ShoppingList”. Model obecności musi zostać w połączeniu z użytkownikiem gdyż potrzeba jest zaznaczania, w które dni ile pracował dany kierowca. Model danych produkty połączony jest relacją z listami zakupowymi, gdyż należy posegregować odpowiednie produkty do odpowiedniego sklepu. Warte zaznaczenia jest również typy takie jak Unit, gdyż jest on zapisany jako enum, gdzie sprecyzowane są konkretne typy używane w przedsiębiorstwie takie jak: sztuki kartony bindy butelki opakowania i ryzy w schemacie prisma.

3.2.3. Diagram klas

Przy tym rodzaju diagramu warto zaznaczyć, że aplikacja jest napisana w frameworku, który wspomaga pisanie kodu funkcyjne zamiast obiektowego. Również backend tworzony w api routes, które dostarcza nam szkielet są pisane stylem funkcyjnym. Jako iż nie ma sprecyzowanych obiektów, praca przedstawi diagram klas wspomagając się modelami schematu prisma oraz funkcjonalnościami dostępnymi w aplikacji.

rys. 14 Diagram klas

Obraz zawierający tekst, diagram, Karteczka samoprzylepna, pismo odręczne

Opis wygenerowany automatycznie

Jak widać klasy są bardzo podobne do modeli schematu prisma. Kluczowe klasy obejmują: Auth, User, Task, ShoppingList, Product, Presence, SalesMan oraz Driver, które pełnią określoną funkcje. Moduł autoryzacji (Auth) umożliwia uwierzytelnianie użytkowników i przechowuje zalogowanego użytkownika w sesji. Klasa User reprezentuje użytkowników systemu, z którymi związane są zadania, listy zakupów oraz obecności. Zadania są opisane przez klasę Task, a listy zakupów przez ShoppingList, z możliwością dodawania, edycji i usuwania zarówno zadań, jak i produktów. Klasa Product reprezentuje produkty w listach zakupów, z możliwością dodawania, edycji, usuwania oraz oznaczania jako kupione. Obecności pracowników są rejestrowane przez klasę Presence, a kierowcy są reprezentowani przez klasę Driver, związaną z rekordami obecności.

**3.3. Badania do aplikacji**

3.3.1. Podmiot obserwacji

W aplikacji internetowej wspomagającej pracę kierowców w przedsiębiorstwie zajmującym się sprzedażą artykułów biurowych, głównymi podmiotami są osoby korzystające z aplikacji czyli kierowcy oraz osoby zamawiające. Należy spojrzeć na rynek takich aplikacji oraz na główne wymagania interesariuszy i porównać je ze sobą. Następnym podmiotem jest sam proces logistyczny, który polega na odbiorze towaru, kupnie, segregacji oraz dowozu do klienta. Przedsiębiorstwo posiada swoją platformę zakupową, którą można połączyć z aplikacją. Kierowcy zaczynają pracę koło godziny ósmej. Pierwszym zadaniem jest odbiór towaru z hurtowni, następnie wszyscy kierowcy spotykają się w firmie, gdzie wymieniają się towarami, dopakowują zamówienia z dnia poprzedniego oraz pakują swoje samochody. Koło godziny jedenastej kierowcy wyjeżdżają już na swoją trasę. Natomiast osoby odpowiedzialne za zamawianie od godziny ósmej zaczynają zamawiać zamówienia, gdzie od godziny ósmej do dziesiątej, odpowiedzialni są za sprawdzenie towaru oraz dopisanie odpowiednich pozycji do kupienia. Aktualna efektywność procesów logistycznych zależna jest od ilości dziennych zamówień. Firma dowozi na następny dzień zamówienia złożone do godziny szesnastej. Jeśli dużo zamówień się uzbiera pojawia się chaos towarowy oraz efektywność jazdy kierowców jak i ekonomiczność jest na niskim poziomie. Zauważyć trzeba, że występuje różnica urządzeń na których pracują poszczególne osoby. Kierowcy głównie czerpią informacje o towarze z telefonów, a osoby zamawiające z komputerów.

3.3.2. Cele obserwacji

Celami obserwacji jest zebranie opinii o korzystaniu z aplikacji z których korzysta aktualnie przedsiębiorstwo oraz czego oczekiwali by interesariusze od nowo powstałej aplikacji dostosowanej do charakteru przedsiębiorstwa. Należy również zaobserwować w jaki sposób użytkownicy korzystają z aplikacji dotychczasowych, żeby dostosować aplikację w taki sposób, żeby przejście i dostosowanie się do nowej aplikacji było jak najłatwiejsze oraz trwało jak najkrótszy okres czasu. Obserwację również powinny przynieść dokładny wgląd w proces logistyczny co pozwoli na początkowej fazie tworzenia aplikacji oraz podczas jej rozwijania na dostosowywanie funkcjonalności w taki sposób by były skierowane do przedsiębiorstwa w którym aplikacja będzie wspomagać.

3.3.3. Miejsce i czas dokonywania obserwacji

Obserwacje można dokonywać w każdy dzień w którym przedsiębiorstwo pracuje tzn. od poniedziałku do piątku w godzina ósma szesnasta. Należy wziąć również pod uwagę dni w których jest mniej zamówień jak i takie w którym zamówień jest sporo, żeby zobaczyć czy do przedsiębiorstwa wkrada się chaos. Obserwacji dokonywać również należy w poszczególne okresy czasu dla przykładu od ósmej do dziesiątej gdy towary z zamówień są dopiero odbierane z hurtowni bądź kupowane, żeby sprawdzić czy proces logistyczny jest efektywny.

3.3.4. Sposób dokonywania obserwacji

Obserwacje trzeba gromadzić w postaci tabeli w której należy wziąć pod uwagę każdego interesariusza aplikacji, zdając odpowiednie pytania jak na przykład „Czy jesteś zadowolony z rozwiązań obecnie na rynku? Oceń od 1 do 5 gdzie 1 to niezadowolenie, a 5 to duże zadowolenie”. Przykładowa tabela może wyglądać następująco.

Tabela 1 Przykładowa tabela do zbierania wyników

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Numer pracownika | Zadowolenie z obecnych aplikacji | | | | |
| 1 | 2 | 3 | 4 | 5 |
| 1  2  3  4  5 |  |  |  |  |  |
| Razem |  |  |  |  |  |

Jak widać obserwacja może przybrać anonimowości najważniejsze jest zebranie ogólnego rezultatu obserwacji dostosowując pytania do celów obserwacyjnych.

1. Dane za stroną internetową: <https://docs.npmjs.com> (dostęp 23.04.2024 r.) [↑](#footnote-ref-1)
2. Dane za stroną internetową: https://nodejs.org (dostęp 23.04.2024 r.) [↑](#footnote-ref-2)
3. Dane za stroną internetową: https://legacy.reactjs.org/ (dostęp 23.04.2024 r.) [↑](#footnote-ref-3)
4. Ibidem. (dostęp 23.04.2024 r.) [↑](#footnote-ref-4)
5. Dane za stroną: [https://axios-http.com](https://axios-http.com/) (dostępne 23.04.2024 r.) [↑](#footnote-ref-5)
6. Dane za stroną: <https://next-auth.js.org/> (dostęp 25.04.2024 r.) [↑](#footnote-ref-6)
7. Saqib Iqbal, Issam Al-Azzoni, Gary Allen, Hikmat Ullah Khan, *Extending UML Use Case Diagrams to Represent Non-Interactive Functional Requirements,* 2020r. s.102 [↑](#footnote-ref-7)
8. Nor Anisah Mohd Saada, Mageswary Muniandib, *The Reflections on the using of Oracle Data Modeler*

   *in Creating Entity Relationship Diagram (ERD), aUngku Omar Polytechnic, 2020r. s.21* [↑](#footnote-ref-8)