Politechnika Rzeszowska
Wydział Elektrotechniki i Informatyki

***Katedra Informatyki i Automatyki***

# Techniki multimedialne

## Post Mortem

## Webchat

**Wykonał: Sendera Wojciech**
**III EF-DI**

**Rzeszów 2018**

## Temat projektu:

Podstawowym założeniem projektu było stworzenie działającego komunikatora, dzięki któremu dwie osoby mogłyby wysyłać do siebie wiadomości w postaci niesformatowanego tekstu.

## Podsumowanie:

Implementacja projektu „Webchat" zakończyła się sukcesem. Udało się stworzyć działającą aplikację obsługująca porozumiewanie się nieograniczonej liczby osób. Lista rozmawiających jest wyświetlana w czasie rzeczywistym i aktualizowana również w przypadku opuszczenia przez użytkownika pokoju rozmów. Użytkownicy na starcie wybierają wyświetlaną nazwę, pod jaką będą występować z punktu widzenia innych rozmawiających.

## Co zadziałało:

Podczas wykonywania projektu za największy sukces uznaję się wykonanie minimalnego działającego produktu już przy pierwszej iteracji programistycznej.
W projekcie sprawdziło się podejście modułowe, dzięki czemu nowe funkcjonalności oraz usprawniona technologia dodawana była stopniowo, co zaowocowało przejrzystym i działającym kodem na każdym etapie produkcji.
Pierwszym krokiem milowym było stworzenie widoku pokoju czatowego w strukturze React.js. Następnie dodana została część serwerowa, napisana przy użyciu Node.js. Następnie dodany został wzorzec projektowy „Flux", a co za tym idzie biblioteka Redux.js, idealnie pasująca do jednostronnie przepływającego strumienia danych. Następnie stworzono moduł w części funkcjonalnej za pomocą biblioteki obsługującej WebSockets. Dzięki temu uzyskano połączenie użytkowników w czasie rzeczywistym. Udało się również zaimplementować część testów jednostkowych dla poszczególnych widoków. Na końcu stworzony został moduł logowania po stronie front-end.

## Wnioski:

Mimo krótkiego czasu wykonywania projektu (24 godziny) wyciągnięto szereg wniosków, które mogą ułatwić wykonywanie przyszłych projektów. Ze względu na niedoświadczenie przy pisaniu aplikacji użytkowych tracono wiele czasu na testowanie różnych rozwiązań i wybieranie tych, które najlepiej pasują do danego typu problemu. Jednym z pierwszych zwycięstw było odkrycie alternatywnych menadżerów paczek, takich Yarn czy Webpack. Poszerzyło to mocno zakres materiału odpowiedniego do wykorzystania w projekcie. Kolejnym było porównanie dostępnych bibliotek obsługujących WebSockets. Ze względu na małą skale projektu większość dostępnych rozwiązań wydawała się odpowiednia, jednak na końcu

zdecydowano się na ws.js, kompaktowej lecz niezwykle wydajnej biblioteki na licencji MIT. Wartym wzmianki jest również problem, jaki wystąpił z automatycznymi weryfikatorami poprawności oraz czystości kodu takimi jak Eslint.js. Mimo swoich bezapelacyjnych zalet często zawadzały one podczas wykonywania projektu, ze względu na niekonsekwentne podążanie za wzorcami.

Na końcu warto wspomnieć o etapie, na jakim zakończył się projekt. Ze względu na ograniczenie czasowe nie została zaimplementowana baza danych. Mimo to, zostało przeprowadzone rozpoznanie w technologiach dostępnych i wybrany został PostGreSQL, ze względu na swoją wydajność przy dużej ilości powtarzalnych danych oraz zaufanie wśród środowiska open-source. Rozważane na początku rozwiązania NoSQL zostały odrzucone ze względu na wydajność. Najszybszym do zaimplementowania rozwiązania byłą by prawdopodobnie baza MongoDb, opierająca się właśnie na technologiach NoSQL. Kolejną rozważaną opcją była baza danych kierowana wydarzeniami. Koncept ten miał się opierać na zapisywaniu wszystkich akcji użytkowników i na tej podstawie wyciąganiu potrzebnych informacji. Został jednak odrzucony ze względu na zbyt duże skomplikowanie implementacji. Ze względu jednak na tą ciekawą koncepcję w projekcie znajduję „saga", czyli wzorzec opierający się na obsługiwaniu logiki za pomocą wydarzeń, gdzie dla każdej transakcji może zostać wykonana kontr transakcja cofająca dotychczasowe zmiany.

Myśląc jeszcze dalej w przyszłość w planach mogło by być niezliczona ilość funkcjonalności, takich jak:

- tworzenie własnych (prywatnych) pokoi czatowych,
- zapis historii użytkowników,
- interfejs programistyczny aplikacji udostępniający wszystkie funkcjonalności (np. CRUD),
- czatowanie nie tylko za pomocą tekstu ale również np. obrazków
- motywy wizualne po stronie użytkownika (personalizacja)
- panel administratora pokoju czatowego

## Optymalizacja oraz czysty kod:

Ze względu na użycie najnowszych technologii front-end aplikacja kliencka występuje w formie SPA (ang. Single Page Application, pol. aplikacja jednej strony), dlatego po pierwszym, inicjalnym ładowaniu na przeciętnym sprzęcie użytkowym czas odpowiedzi jest niezauważalny. Dodatkowo ze względu na jednostronny przepływ danych („Flux"), w każdym momencie stan aplikacji jest łatwo sprawdzalny. Dodatkowo podczas dynamicznych zmian na obiektowym modelu dokumentu (ang. DOM), ponownie rysowane na ekranie są tylko te elementy, które uległy jakiejś zmianie.

Udało się również zachować zasady czystego kodu. Większość nazw funkcji i zmiennych jest łatwo identyfikowalna, a przez to łatwa w późniejszej modyfikacji. Wiele dostępnych funkcji jest typu czystego, a więc działa tylko na obiektach dostępnych w zakresie swojego własnego działania. Udało się napisać testy jednostkowe do wszystkich podstawowych widoków. Udało się również zastosować bezstanowe podejście do obiektów tworzonych przez React.js, co jest nowoczesnym i pożądanym podejściem.

## Finalny wykres Gantt'a:

| WEB CHAT | 1 | 24 | |
|---|---|---|---|
| Wykonanie diagramu Gantta | 1 | 2 | Complete |
| Rozpoznanie w instniejących rozwiązaniach oraz wybór technologi | 2 | 5 | Complete |
| Wykonanie modelu bazy danych | 5 | 5 | Complete |
| Wykonanie części back-endowej | 5 | 7 | Complete |
| Testy części back-endowej | 7 | 8 | Complete |
| Wykonanie części front endowej | 8 | 16 | Complete |
| Testy aplikacji | 16 | 17 | Complete |
| Wykonanie dokumentacji | 17 | 23 | Complete |
| Poprawki | 23 | 24 | Complete |

*Rysunek 1: Finalny wykres Gantt'a*