Dokumentacja projektowa
„Komunikator partnerski”

Identyfikacja zagadnienia biznesowego (problemu)

Aplikacja w swojej idei nadrzędnej ma odpowiadać na potrzebę, bardzo konkretną potrzebę wyartykułowaną przez moją piękniejszą połowę. Potrzeba ta, w kontekście aplikacji webowych, wydaje się możliwa do realizacji, w stosunkowo prosty sposób. Jednocześnie w szerszym kontekście wydaje się, że aplikacja mogłaby znaleźć zastosowanie na szerszym rynku, ale nie to powinno stanowić jej cel na etapie podstawowym.

Potrzebą biznesową, jaką planowana aplikacja ma spełniać jest umożliwienie w relacji partnerskiej, mówiąc brzydko, wzajemnego składania sobie „ticketów”, czyli abstrakcyjnych zadań, niewerbalnie, ale właśnie poprzez aplikację

Wymagania systemowe i funkcjonalne

Obligatoryjne wymagania systemowe aplikacji:
Back-end: środowisko uruchomieniowe Node.js + Express.js,
Front-end: biblioteka React.js

Ze względu, iż dysponuję własnym serwerem, wymagania środowiskowe zostały dopasowane do jego środowiska a zatem jakikolwiek system pozwalający na „hostowanie” aplikacji napisanej w oparciu o „node.js”. W przypadku kontynuacji projektu należy rozważyć konteneryzację i „hostowanie” np. poprzez „framework” Heroku, czy z wykorzystaniem środowiska Azure.

b2. Wymagania funkcjonalne

Określenie potencjalnych „user stories”:

- Jako jeden z dwojga użytkowników, loguję się do aplikacji, by sprawdzić moje zadania na dziś
- Jako jeden z dwojga użytkowników, loguję się do aplikacji by dodać zadanie partnerowi i sprawdzić stan jego wykonania
- Jako jeden z dwojga użytkowników, loguję się do aplikacji by zmienić status swojego zadania
- Jako administrator mam dostęp do bazy danych by móc dokonywać wpisów korekcyjnych (mała skala aplikacji)
- Jako jeden z dwojga użytkowników, loguję się do aplikacji by zmienić opis istniejącego zadania
- Jako jeden z dwojga użytkowników, loguję się do aplikacji loguje się by zamknąć zadanie

Na podstawie powyższych, historyjek użytkownika, określono następujące wstępne, **minimalne** wymagania funkcyjne:

- Wymóg implementacji autentykacji, i autoryzacji (Ze względu na początkowo niekomercyjny charakter, raczej w podstawowym wymiarze, z możliwością rozbudowy w przypadku potrzeby skalowania)
- Wymóg posiadania interfejsu graficznego dla każdego z użytkowników, wyświetlającego zadania wraz z podziałem tych zadań na kategorię („do zrobienia”,” pracuję nad tym”,” zrobione”. Interfejs powinien umożliwiać zmianę kategorii, edycję i korekty
- Wymóg przechowywania stanu aplikacji. Ze względu na potrzebę przechowywania stanów (zadań, użytkowników, grup), niezbędna jest implementacja przechowywania danych w bazie. Ze względu nałatwość implementacyjną i niski koszt zdecydowano się wykorzystać bazę NoSql, a konkretnie „mongodb”
- Wymóg dostępności aplikacji poprzez zewnętrzne ip (aplikacja powinna być dostępna z każdej przeglądarki)
- Architektura serwer-klient, rozdzielna, ale hostowana z jednego urządzenia, a zatem należy rozwiązać problem poprawnej implementacji „cors”. Server powinien uzewnętrzniać do „frontendu” dostęp do bazy i przejąć zarządzanie danymi. „Frontend” czyli aplikacja webowa w „Reactie” powinien móc komunikować się z serwerem, zarówno by zapisywać nowe zadania, jak i by umożliwiać ich modyfikację, jak i odczyt danych z bazy.

Harmonogram prac

Wstępnie zarysowane zadania do wykonania:

- „Frontend” (2-4 tygodni)
    1. Stworzenie pierwszej strony funkcyjnej, „dashboard”, wraz z podstawową formą. Zarysowanie podziału na grupy zadań
    2. „Zamokowanie” komunikacji dashboardu z bazą(tymczasowe)
    3. Stworzenie ekranu logowania wraz formą umożliwiającą przekazanie poświadczeń
    4. „Zamokowanie” poświadczeń(tymczasowe)
    5. „Zamapowanie” obiektów biznesowych do „Reacta”
- „Backend” (2-4 tygodni)
    1. Stworzenie Serwera
    2. Stworzenie bazy danych
    3. Podpięcie serwera do bazy, definicja „endpointów”
- Integracja (2 tygodnie)
    1. Podpięcie Frontendu (komunikacji z „endpointami” serwera)
    2. Weryfikacja i testy
    3. Poprawki

Analiza zagadnienia i jego modelowanie

Obiekty biznesowe:

Tabela(kolekcja) **tasks**:
Pojedynczy **task** pola:
_id typ *guid, (indexer)*
id typ *guid,*
owner typ *string,*
group typ *string,*
name typ *string,*
isComplete *typ bool*

Tabela(kolekcja) „groups”:
Pojedyncza grupa, pola:
_id typ *guid, (indexer)*
id typ *guid,*
name typ string,
id typ *guid,*
owner typ *string*

Tabela(kolekcja) „users”
Pojedynczy user, pola:
_id typ *guid, (indexer)*
name typ *string,*
id typ *guid,*
owner typ *string*

Tabela(kolekcja) „comments”
Pojedynczy komentarz, pola:
_id typ *guid, (indexer)*
task typ *string,*
id typ *guid,*
owner typ *string,*
content typ string

Podsumowanie

Aplikacja okazała się spełniać wymagania minimalne ale wymaga poprawy w następujących polach:

Dostęp do danych partnera, wylogowywanie, oraz usuwanie danych rozumiane jako możliwość ich edycji z frontendu.