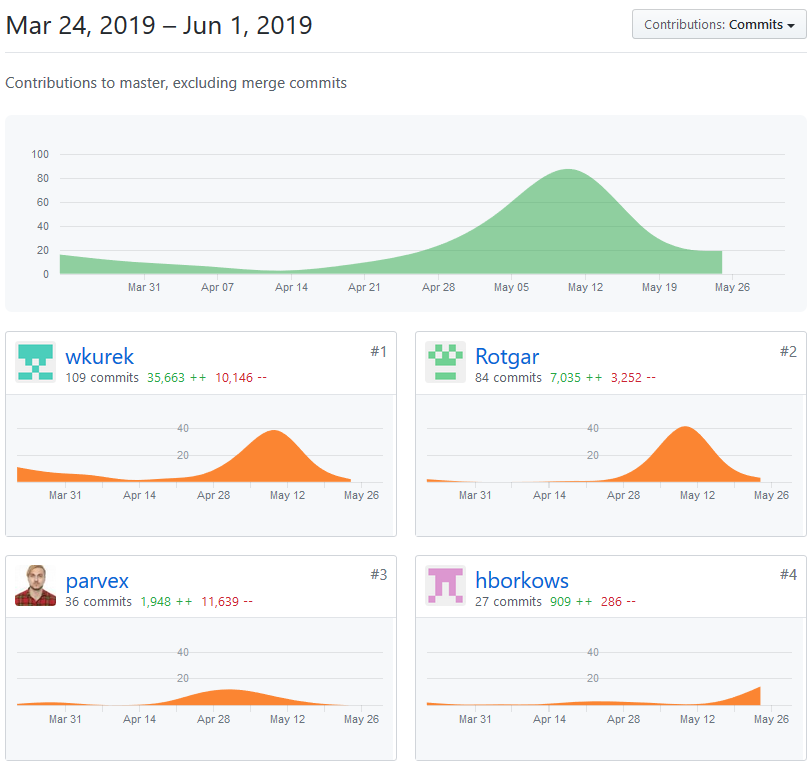
**Hour reporting system - dokumentacja**

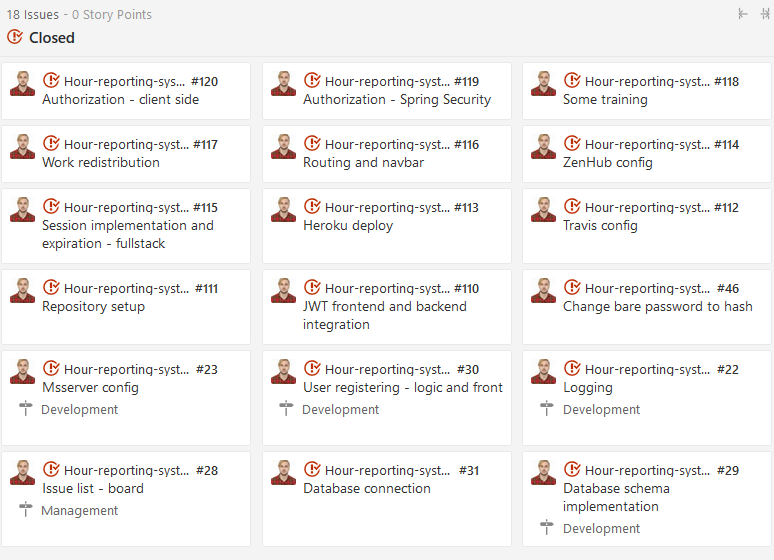
**Harmonogram i podział prac**

Dane z GitHub

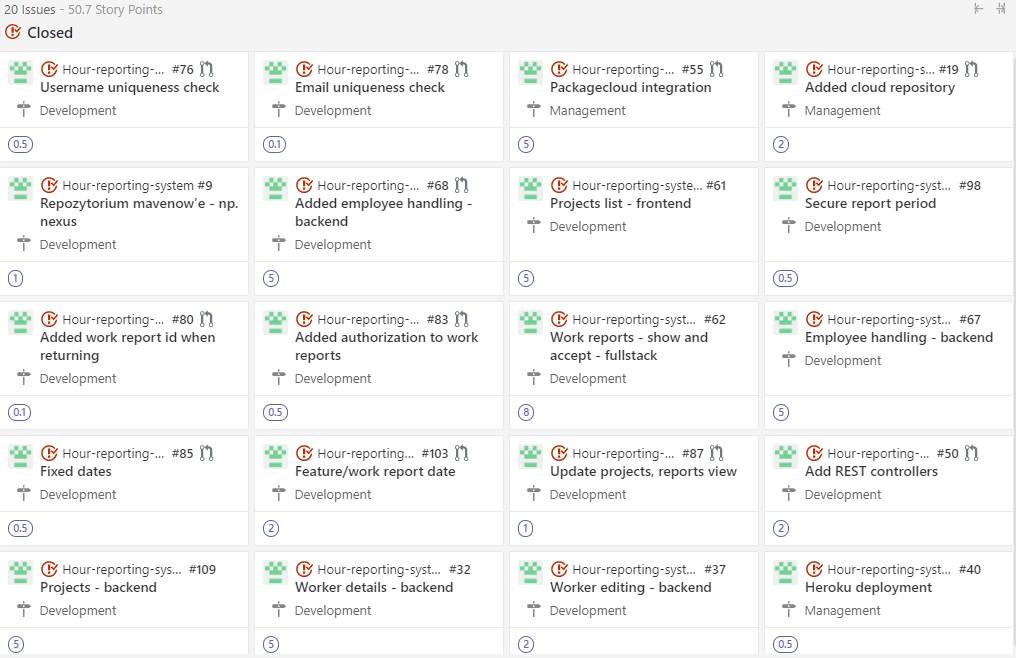


Dane z ZenHub

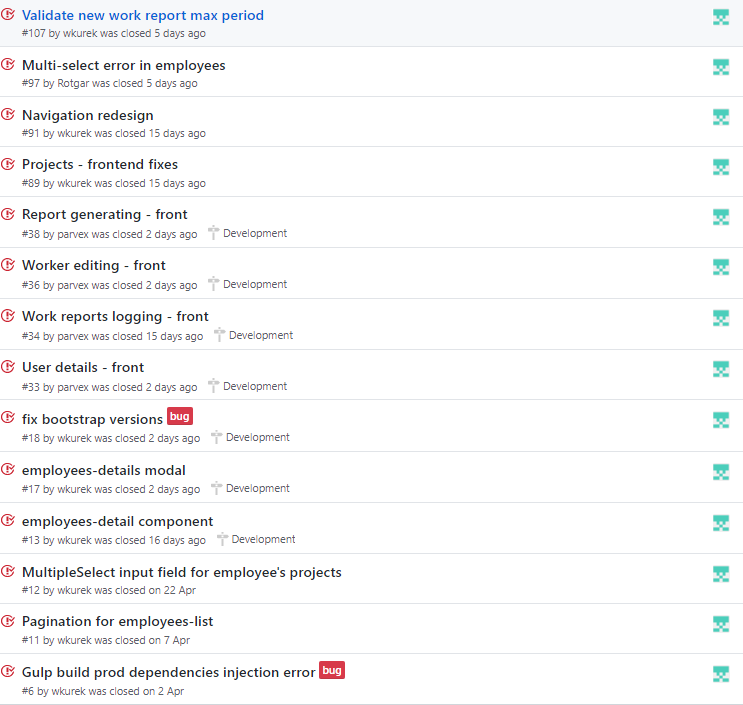
Zadania - Gabriel Rębacz



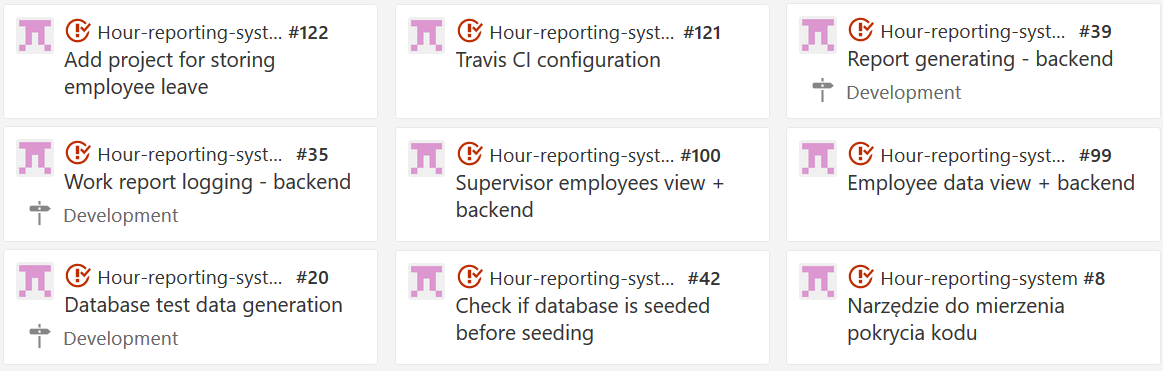
Zadania - Mateusz Krawczyk



Zadania - Wojciech Kurek



Zadania - Hubert Borkowski



**Funkcjonalności aplikacji**

1. Logowanie i autoryzacja użytkowników z użyciem loginu i hasła

2. Raportowanie godzin pracy nad wybranym projektem

a. Raportowanie pracy za okres jednego dnia

b. Raportowanie pracy za okres czasu więcej niż jednego dnia

c. Możliwość dodania komentarza z dodatkowymi informacjami do raportu

d. Wymuszanie pożądanych ram czasowych i relacji między datami za które składamy raport

3. Przeglądanie istniejących raportów w widoku kalendarza lub listy

a. Przeglądanie raportów pracowników przez ich przełożonego

b. Filtrowanie prezentowanych raportów względem:

i. Ram czasowych (daty od, data do)

ii. Pracowników

iii. Projektów

4. Modyfikowanie istniejących raportów przez ich właścicieli (możliwa modyfikacja liczby godzin oraz komentarza).

5. Przeglądanie szczegółów raportów z poziomu kalendarza i listy.

6. Raportowanie dni wolnych (specjalny projekt Employee Leave)

7. Zatwierdzanie raportu godzin pracy pracowników przez ich przełożonych

8. Dodawanie nowych pracowników (użytkowników) przez administratora

a. Przypisywanie projektów do pracownika

b. Przypisywanie przełożonego do pracownika

c. Przypisywanie ról dla pracownika (USER, ADMIN, SUPERVISOR)

9. Przeglądanie podlegających pracowników przez przełożonego

a. Możliwość filtrowania pracowników po następujących kryteriach:

i. Imię

ii. Nazwisko

iii. Projekty

iv. Email

10. Możliwość przeglądania oraz edycji pracownika

11. Możliwość przeglądania statystyk dla podległych pracowników. Statystyki rozumiemy jako: projekty, w których bierze udział, wykorzystane dni urlopu oraz zalogowane godziny pracy nad projektami.

12. Możliwość przeglądania własnych statystyk. Statystyki rozumiemy jako: projekty w których bierze udział, wykorzystane dni urlopu oraz zalogowane godziny pracy nad projektami.

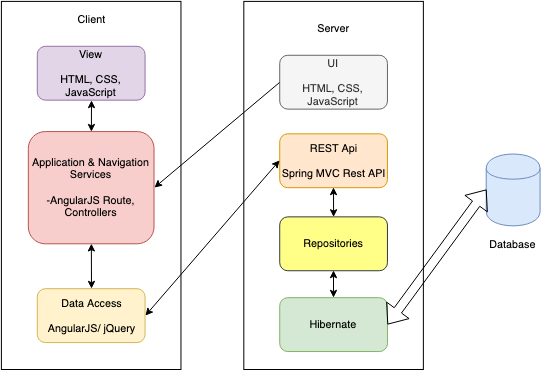
13. Możliwość dodawania nowych projektów oraz edycji istniejących

14. Możliwość przeglądania pracowników przypisanych do danego projektu przez ich przełożonego

15. Możliwość przeglądania raportów pracy złożonych przez pracowników w ramach danego projektu przez ich przełożonego

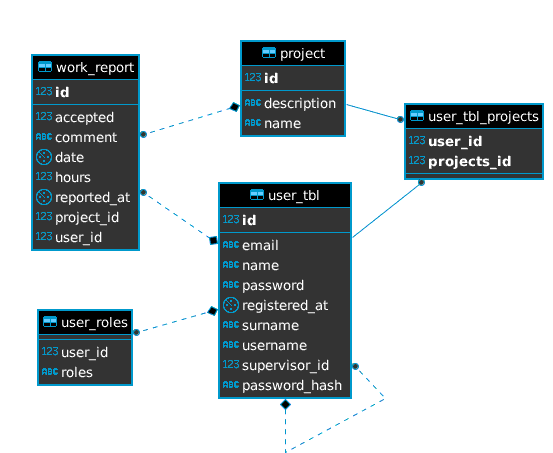
16. Możliwość akceptowania/odrzucania raportów pracy złożonych przez pracowników w ramach danego projektu przez ich przełożonego

**Architektura systemu**



System składa się z trzech warstw Persistence, Backend i Frontend. Projekt systemu został zaimplementowany jako projekt Maven.

* Persistence

W warstwie trwałości zastosowana została baza danych SQL Server firmy Microsoft, która na potrzeby aplikacji demo jest hostowana w chmurze Azure. W przypadku komercyjnego wykorzystania można również skorzystać z rozwiązań chmurowych, w zależności od potrzeb istnieje również możliwość wykorzystania bazy hostowanej na prywatnych maszynach (on premise).

* Backend

Serwer systemu został zaimplementowany w języku Java z wykorzystaniem framework’a Spring Web MVC, który umożliwia tworzenie aplikacji webowych we wzorcu projektowym Model - View - Controller.

Serwer wystawia REST API pozwalające na wymianę informacji z aplikacją kliencką. Endpointy REST API mają zaimplementowane uwierzytelnianie i autoryzację, która jest zaimplementowana za pomocą Spring Security oraz Json Web Token. W momencie poprawnego uwierzytelnienia w metodzie authorize API, serwer przesyła JWT token do klienta, który jest załączany do nagłówka zapytań HTTP w celu uwierzytelnienia komunikacji. Token jest ważny przez 15 minut.

Konfiguracja aplikacji oparta jest o Spring Boot, czyli platformę znacznie ułatwiającą wszelkie konfiguracje komponentów Spring, co pozwala zwiększyć produktywność i zmniejsza czas developmentu.

REST API komunikuje się z warstwą persistence poprzez serwisy, które zawierają repozytoria, które zaimplementowane zostały za pomocą framework’a Hibernate. Hibernate automatycznie generuje schemat bazy danych na podstawie obiektów encyjnych, implementuje podstawowe zapytania oraz mapuje automatycznie otrzymane dane z bazy na obiekty klas encyjnych(ORM). W razie potrzeby pozwala na napisanie niestandardowych zapytań w łatwy sposób.

* Frontend

Aplikacja frontendowa jest serwowana przez serwer i znajduje się w folderze resources aplikacji webowej. Wykonana ona została w języku JavaScript i wykorzystany został framework AngularJs. Wykorzystane zostały dodatkowe biblioteki Angular UI, które zawierają moduły routingu(najważniejsze) - ui route, stylów - ng bootstrap oraz inne. Aplikacja kliencka komunikuje się z serwerem wysyłając zapytania za pomocą funkcji bibliotek AngularJs ($http). Otrzymane dane prezentowane są za pomocą kontrolerów (JavaScript) w widokach do nich przypisanych (template - HTML). Budowanie aplikacji klienckiej jest zautomatyzowane przy budowanie całego projektu za pomocą biblioteki GulpJs. Można również zbudować ją w trybie developerskim co ułatwia debugowania i przyśpiesza proces tworzenia. Wszystkie zapytania do serwera są uwierzytelniane przez token otrzymany w trakcie logowania. Informacje o użytkowniku i o tokenie są przechowywane w ciasteczkach. Po 15 minutach jeśli aplikacja otrzyma w zapytaniu błąd 401 - następuje wyczyszczenie sesji i aplikacja wymaga ponownego logowania. Widoki wystylizowane są za pomocą biblioteki ng bootstrap oraz dodatkowe style znajdują się w pliku style.css.

**Dokumentacja techniczna**

Dzięki wykorzystaniu Spring Boot oraz Maven konfiguracja oraz budowanie aplikacji jest bardzo łatwe. Do budowy wykorzystać można komendę “mvn clean install” z poziomu głównego folderu. Pobrane zostaną wszelkie potrzebne zależności. Podpięcie do konfiguracji Mavena pluginów - Node.js, Gulp.js oraz Bower, pozwala na pobranie wymaganych paczek oraz budowę i późniejsze serwowanie warstwy frontendu za pomocą komendy gulp serve z poziomu src/main/frontend/ . Dzięki temu można tworzyć frontend lokalnie na porcie 3000.

Do budowy aplikacji wymagane jest dostarczenie informacji dotyczących bazy danych. Zawarte są one w application.properties. Należy zdefiniować podstawowe wartości: driver wymagany do połączenia z wybranym serwerem, url do bazy oraz login i hasło. W trakcie kompilacji Spring Boot nawiązuje połączenie z bazą. W przypadku braku połączenia, bądź błędnej konfiguracji, kompilacja nie powiedzie się i zwrócony zostanie błąd.

Po poprawnej kompilacji i uruchomieniu, Spring Boot automatycznie skonfiguruje i wystawi aplikację za pomocą Tomcata, lokalnie na porcie 8080.

Aplikacja odbiera dane dostarczane na end-pointach przy pomocy REST controllerów (/restController). W zależności od podstawowych metod HTTP: GET, POST, PUT, DELETE przesyłane są one dalej do warstwy service (/service). Po sprawdzeniu i przetworzeniu, odpowiednie dane wysłane zostają do warstwy repository(/repository). Przy pomocy JPA oraz Hibernate tworzone są @Query, czyli zapytania napisane zgodnie ze składnią dostarczonego języka JPQL/HQL, który umożliwia na pisanie zapytań krótko i efektywnie. Spring Security umożliwia sprawdzenie i uwierzytelnienie użytkownika wysyłającego requesty, dzięki czemu pobrane zostają tylko dane do których faktycznie ma dostęp. Pobrane dane są zwracane poprzez warstwy z powrotem do REST controllerów i wystawiane na end-pointach do wykorzystania w widokach przy pomocy AngularJS.

Do transportu danych między warstwami wykorzystane zostały obiekty - DTO (/dto).

W (/domain) znajdują się klasy reprezentujące tabele w bazie danych.

W plikach: .travis.yml, .travis.setting.xml, codecov.yml zapisane są konfiguracje wykorzystanych technologii: travis, jacoco, codecov, packagecloud.io - repozytorium artefaktów.