1. Stworzyć **MessageService** który ma
   1. cztery pola prywatne :
      1. valueOne
      2. valueTwo
      3. decimalPlaces
      4. multiplier
   2. konstruktor który ustawia **decimalPlaces** = 2; **multiplier** = 1;
   3. prywatną metodę **calculateValues** która ustawia valueOne oraz valueTwo na:
      1. losowe liczby z zakresu **0.0 - 1.0**
      2. mnoży je przez pole **multiplier**
      3. ustawia miejsca po przecinku zgodnie z wartości **decimalPlaces**
   4. metodę **getValues,** która przy każdym wywołaniu oblicza **valueOne**, **valueTwo** oraz upakowuje wszystkie 4 pola w obiekt i zwraca go
2. Stworzyć **DecimalController ( /api/decimal )** który zawiera dwie metody:
   1. **setDecimal** - ustawia wartość decimal w **MessageService**
   2. **getDecimal** - zwraca wartość decimal
3. Stworzyć **MultiplierController ( /api/multiplier )** który zawiera dwie metody:
   1. **setMultiplier** - ustawia wartość multiplier w **MessageService**
   2. **getMultiplier** - zwraca wartość multiplier
4. Stworzyc **NumberController ( /api/number )** który zawiera dwie metody:
   1. **getCalculatedValues** – zwraca przeliczone dane w **calculateValues**
      1. valueOne
      2. valueTwo
5. Napisać testy dla każdej metody w **kontrolerach**
6. Napisać **filtr** który dla każdej metody **setDecimal** sprawdza czy zapytanie zawiera nagłówek
   1. **decimalPlaces**

I sprawdza czy wartości z tego nagłówka to “**1**”, “**2**”, “**3**” albo “**4**”.

Jeżeli nie to zwrócić informację, że kolor jest niepoprawny oraz ustawić status odpowiedzi na **400** - “Bad Request”.

1. Napisać filtr **timestampFilter** który dla zapytania **getCalculatedValues** zmierzy czas pomiędzy rozpoczęciem, a zakończeniem jego obsługi i wyświetli tą informacje w logach systemowych.
2. Dodać security do aplikacji - nie dodawać użytkowników.
3. Dodać klasę z uprawnieniami :
   1. DECIMAL\_READ
   2. DECIMAL\_WRITE
   3. MULTIPLIER\_READ
   4. MULTIPLIER\_WRITE
   5. ACCESS\_ALL
4. Zabezpieczyć **DecimalController** :
   1. **setDecimal** - w konfiguracji: DECIMAL\_WRITE, ACCESS\_ALL
   2. **getDecimal** - przy użyciu adnotacji @PreAuthorize w kontrolerze DECIMAL\_READ, ACCESS\_ALL
5. Zabezpieczyć **MultiplierController** :
   1. **setMultiplier** - w konfiguracji: MULTIPLIER\_WRITE, ACCESS\_ALL
   2. **getMultiplier** - przy użyciu adnotacji @PreAuthorize w kontrolerze MULTIPLIER\_READ, ACCESS\_ALL
6. W klasie konfiguracyjnej security stworzyć :
   1. @Bean - metodę **getInMemoryUserDetailsManager** zwracającą **InMemoryUserDetailsManager**
   2. Skonfigurować PasswordEncoder.
7. Stworzyć **RegisterService.**
8. wstrzyknąć **InMemoryUserDetailsManager**
9. Stworzyć metode **createUser** - która tworzy użytkownika przekazanego w argumencie
   1. userName
   2. password
   3. authorities

**( Możecie zamiast punktów 12 – 13 stworzyć użytkownikow i role na bazie danych.)**

1. Stworzyć **RegisterController ( /register ) (** dostęp dla wszystkich **)** z metodą
   1. createUser - która tworzy użytkownika
2. Stworzyć **UserService** z metodami
   1. getUser - zwraca użytkownika po jego userName
   2. deleteUser - szuka użytkownika po userName, jeśli istnieje to go usuwa
   3. updatePassword - szuka użytkownika po userName i aktualizuje jego hasło
3. Stworzyć **UserController ( /api/user )** z metodami dostępne tylko dal ACCESS\_ALL:
   1. getUser - zwraca użytkownika
   2. deleteUser - usuwa użytkownika
   3. updatePassword - aktualizuje hasło
4. Upakować odpowiedzi kontrolerów w obiekt **ResponseEntity**, ustawiając poprawne HTTPStatusy dla każdego przypadku oraz jeżeli zapytanie się nie powiodło dodać informacje co poszło nie tak.
5. Dopisać testy do brakujących kontrolerów / poprawić testy korzystając z adnotacji **@WithMockUser**
6. Stworzyć użytkowników:
   1. **userName**: admin; **password**: admin; **authorities**: ACCESS\_ALL
   2. **userName**: decimal; **password**: decimal; **authorities**: DECIMAL\_READ, DECIMAL\_WRITE
   3. **userName**: multi; **password**: multi; **authorities**: MULTIPLIER\_READ, MULTIPLIER\_WRITE