Dokumentacja końcowa

Artem Romanenko

numer konta:38185
semestr studiów:5
kierunek: Informatyka

Spis treści

Opis koncepcji

Tematem mojego projektu jest walka z propagandą.

Propaganda to celowe działanie zmierzające do ukształtowania określonych poglądów i zachowań zbiorowości ludzkiej lub jednostki.

Najlepszym sposobem walki z propagandą jest użycie kilku różnych źródeł informacji, czytanie różnych niepodległych opinii.

Celem mojego programu będzie udostępnienie jak najwięcej źródeł informacji o jednym zdarzeniu.

Rozwiązanie będzie podobne do witryny z wiadomościami tylko będzie zawierać tylko nagłówki i linki do różnych źródeł (gazety, programy nowostne, talk-show). Takim sposobem program będzie wspierać rozwój własnego zadania u użytkownika.

Przykład: Są wydawnictwa które wspierają rząd, jeśli osoba czyta tylko tylko taki wydawnictwa to jej opinia może stać pro rządową a osoby które czytają lub oglądają źródła internetowe(które zwykle krytykują rząd) to opinia osoby też może stać podobną. A prawda jak zawsze jest między tymi dwoma opiniami i żeby ją znaleźć trzeba czytać wszystkie opinii.

Funkcjonalność programu:

Użytkownik może analizować informację czytając różne opinie.

Użytkownik widzi jakie źródło informacji podoba się ludziom najwięcej (likes)

Użytkownik może otrzymywać nowości różnymi sposobami :

Strona internetowa
Chatbot - Telegram (wysłanie będzie automatyzowane)

Admin miszi tworzyć kontent tylko na stronie internetowej !

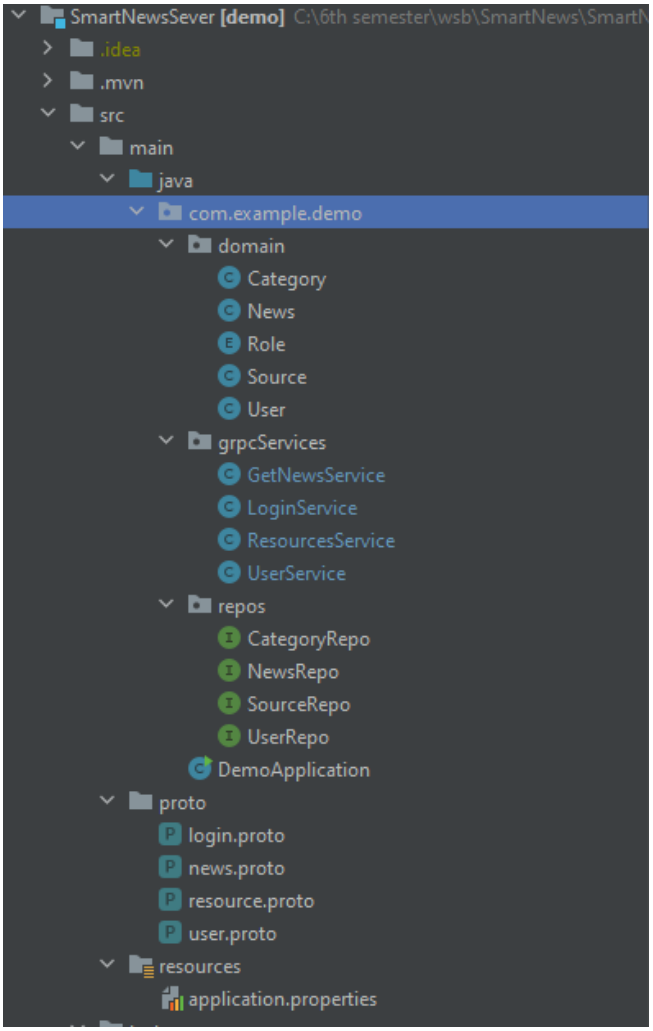
Zastosowane technologie

Programming language - Java
Framework used - Spring
Front-end framework - Vaadin
Database operations - Hibernate
Communication between applications - GRPC

Opis realizacji

Aplikacja składa się z 3 części server, web klient, bot dla sieci socjalnej.

Server -



Projekt składa się z :

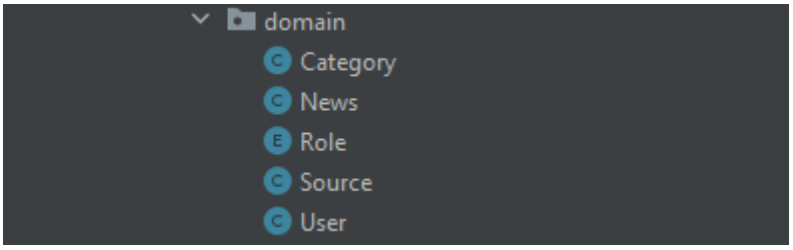
teczki domain w której jest opisana struktura bazy danych

teczki repos która zawiera interfejsy JpaRepository za ich pomocą komunikuję z bazą danych

teczki proto która zawiera .proto pliki potrzebne do opisu danych przesyłanych za pomocą GRpc.

teczki GrpcServices która zawiera implementację logiki i przesyła potrzebne dane za pomocą GRpc.

Struktura bazy danych



Baza danych składa się z 5ciu tablic, każda klasa definiuje jedną tablicę.

Klasa User

```
@Entity
@Table(name = "userok")
public class User implements UserDetails {
  @Id
  @GeneratedValue(strategy = GenerationType.AUTO)
  private Long id;
  private String username;
  private String password;
  private String email;
  private String token;

  @ElementCollection(targetClass = Role.class, fetch = FetchType.EAGER)
  @CollectionTable(name = "user_role", joinColumns = @JoinColumn(name = "user_id"))
  @Enumerated(EnumType.STRING)
  private Set<Role> roles = new TreeSet<>();

  public User(String username, String password, String email){
     this.username = username;
     this.password= password;
     this.email = email;
  }

  public User() {
```

```
  }

 //Getter and Setters, Equals and HashCode override
```

Jak można zrozumieć z powyższego kodu tabela User zawiera kolumny id, username, password, email, token(potrzebny dla autentykacji)

Enum Role

```
public enum Role implements GrantedAuthority {
  ADMIN, SUPERADMIN;

  @Override
  public String getAuthority() {
     return name();
  }

 public static Role parseRoleFromString(String role) throws Exception {
     switch (role){
        case("SUPERADMIN") : return Role.SUPERADMIN;
        case("ADMIN") : return Role.ADMIN;
        default: throw new Exception("Can't parse user's role");
     }
  }
}
```

Enum Role definiuje tablicę role która jest związana z tablicą user i w tej tablice przechowywuję informację o poziomie dostępu użytkownika.

Klasa Category

```
@Entity
public class Category {
  @Id
  @GeneratedValue(strategy = GenerationType.AUTO)
  private Long id;
  private String name;

  @OneToMany(mappedBy="category", fetch = FetchType.EAGER)
  private List<News> news;

  public Category(String name) {
     this.name = name;
  }
```

```
  public Category() {
  }
// getters setter equals and hashcode here
```

Klasa Category zawiera tylko swoją nazwę i listę obiektów news
Klasa News

```
@Entity
public class News {
  @Id
  @GeneratedValue(strategy = GenerationType.AUTO)
  private Long id;
  private String heading;

  @OneToMany(mappedBy="news", fetch = FetchType.EAGER)
  private List<Source> sources;

  @ManyToOne
  @JoinColumn(name="category_id")
  private Category category;

  public News() {
  }

  public News(String heading, Category category) {
     this.heading = heading;
     this.category = category;
  }
```

Klasa News zawiera nagłówek, obiekt Category z którym jest związana i listę obiektów Source

Klasa Source

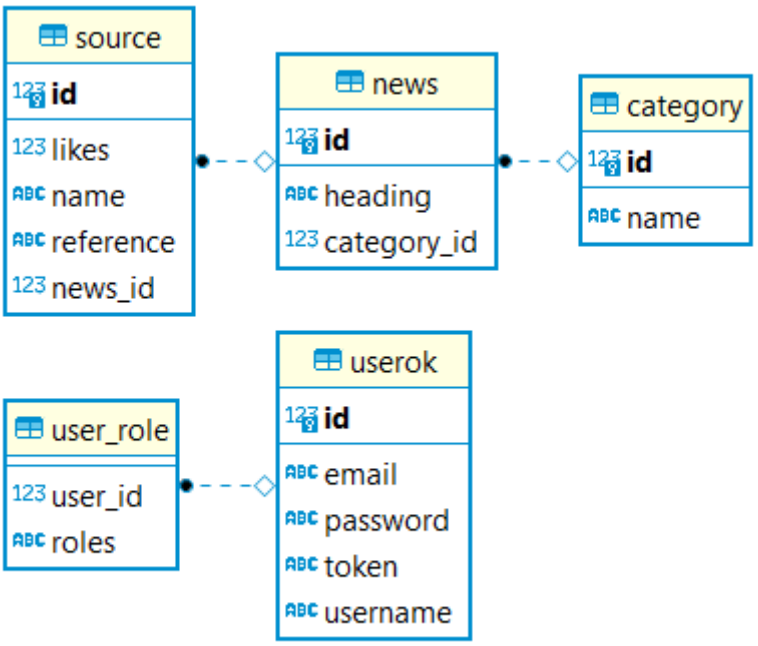
```
@Entity
public class Source {
  @Id
  @GeneratedValue(strategy = GenerationType.AUTO)
  private Long id;

  private String name;
  private String reference;
  private Integer likes;
  @ManyToOne
  @JoinColumn(name="news_id")
  private News news;
```

```
public Source() {
}

public Source(String name, String reference, News news) {
    this.name = name;
    this.reference = reference;
    this.news = news;
}
```

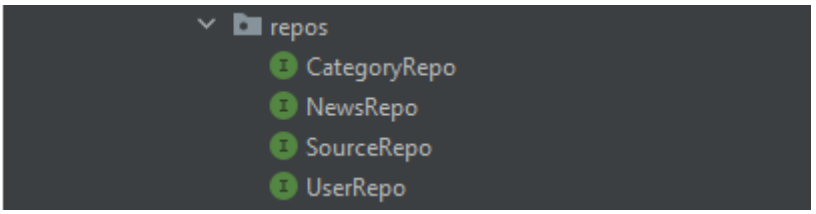
// getters, setter, equals and hashcode here

Klasa Source zawiera pola id, name, likes oraz obiekt news z którym jest związana

Diagram bazy danych



Repositories



CategoryRepo

```
public interface CategoryRepo extends JpaRepository<Category, Long> {
   Category findByName(String name);
}
```

Odpowiada za komunikację z tablicą category

NewsRepo

```
public interface NewsRepo extends JpaRepository<News, Long> {
}
```

Odpowiada za komunikację z tablicą news

SurceRepo

```
public interface SourceRepo extends JpaRepository<Source, Long> {
}
```

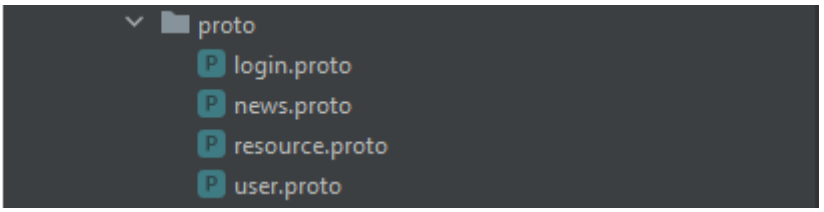
Odpowiada za komunikację z tablicą source

UserRepo

```
public interface UserRepo extends JpaRepository<User, Long> {
}
```

Odpowiada za komunikację z tablicą user

Pliki .proto



Pliki .proto używana technologiją Grpc dla zdefiniowania wszystkich możliwych żądań do serwera i odpowiedzi od serwera.

login.proto

```
syntax = "proto3";

// options for Java generated sources
option java_multiple_files = true;
option java_package = "com.thinhda.spring.grpc.core.model";


package sms.core;

message LoginRequest{
 string username = 1;
 string password = 2;
}

message LoginResponse{
 string token = 1;
}

//login service receives request that contains username and password and returns response with token,
// you should save token to the android session, you should add this token to each request, otherwise you will get 403 unauthenticated
service LoginService{
 rpc login (LoginRequest) returns (LoginResponse);
}
```

login.proto plik deklaruje że klient może wywołać na serwerze metodę login która przejmuje LoginRequest i zwraca LoginResponce.

user.proto

```
syntax = "proto3";

// options for Java generated sources
option java_multiple_files = true;
option java_package = "com.thinhda.spring.grpc.core.model";


package sms.core;

message User{
 string name =1;
 string password=2;
 string token = 3;
}
message AllUsersResponse{
 repeated User user = 1;
}
message GetUsersRequest{
 string token=1;
}

service UserService{
 rpc getAllUsers(GetUsersRequest) returns(AllUsersResponse);
}
```

user.proto deklaruje że klient może wywołać na serwerze metodę getAllUsers która przyjmuje obiekt GetAllUsersRequest i odpowiada obiektem AllUsersResponce.

resources.proto

```
syntax = "proto3";

// options for Java generated sources
option java_multiple_files = true;
option java_package = "com.thinhda.spring.grpc.core.model";
```

```
package sms.core;

message Resource{
  string name = 1;
  string reference = 2;
}
message AllResources{
  repeated Resource resource = 1;
}
message GetResourceByIdRequest{
  string id =1;
  string token =2;
}
message GetAllResourcesRequest{
  string token= 1;
}
message CreateResourceRequest{
  string name=1;
  string reference =2;
}
message CreateResourceResponse{
  bool created =1;
}

message EditResourceRequest{
  string id =1;
  string name = 2;
  string reference = 3;
}
message EditResourceResponse{
  bool updated =1;
}

service ResourcesService{
  rpc getAllResources(GetAllResourcesRequest) returns(AllResources);
  rpc getResourceById(GetResourceByIdRequest) returns(Resource);
  rpc createNewResource(CreateResourceRequest) returns(CreateResourceResponse);
  rpc editResource(EditResourceRequest) returns(EditResourceResponse);
}
```

resources.proto deklaruje że klient może wywołać metody getAllResources, getResourceById, createNewResource, editResource.

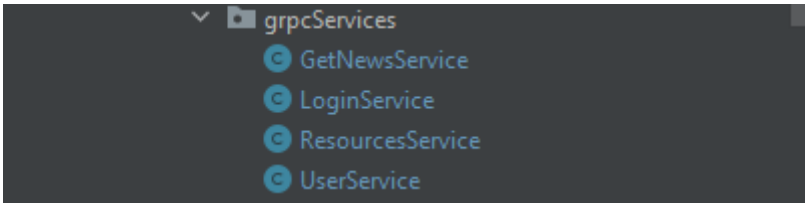
news.proto

// messages declaration omitted to save space

```
service NewsService{
 rpc getNewsByCategory(GetNewsByCategoryRequest) returns(MultipleNewsResponse);
 rpc getNewsById(GetNewsByIdRequest) returns(SingleNewsResponse);
 rpc getAllNews(GetAllNewsRequest) returns(MultipleNewsResponse);
 rpc getAllCategories(GetCategoriesRequest) returns(MultipleCategoriesResponse);
 rpc deleteCategory(DeleteCategoryRequest) returns(DeleteResponse);
 rpc deleteSource(DeleteSourceRequest) returns(DeleteResponse);
 rpc deleteNews(DeleteNewsRequest) returns(DeleteResponse);
 rpc getSourcesByNews(GetSourcesByNewsRequest) returns(MultipleSourcesResponse);
 rpc createCategory(CreateCategoryRequest) returns (CreateResponse);
 rpc createNews(CreateNewsRequest) returns (CreateResponse);
 rpc createSource(CreateSourceRequest) returns(CreateResponse);
 rpc editCategory(EditCategoryRequest) returns(EditResponse);
 rpc editSource(EditSourceRequest) returns(EditResponse);
 rpc editNews(EditNewsRequest) returns(EditResponse);
}
```

deklaruje metody które klient może wywołać na serwerze, metody operują z obiektami news, sources oraz category.

Grpc services



NewsService implementuje metody zdefiniowane w news.proto

Ta klasa jest duże więc żeby oszczędzić miejsce pokaże implementację tylko jednej metody. Całą klasę można zobaczyć w kodzie dołączonym do sprawozdania.

```
@Override
public void getNewsByCategory(GetNewsByCategoryRequest request,
StreamObserver<MultipleNewsResponse> responseObserver) {
  String categoryName = request.getCategory();
  Category category = categoryRepo.findByName(categoryName);

  List<News> news = category.getNews();

  MultipleNewsResponse.Builder responseBuilder = MultipleNewsResponse.newBuilder();
  for(int i=0; i<news.size(); i++){
     com.thinhda.spring.grpc.core.model.News.Builder newsGrpcBuilder =
com.thinhda.spring.grpc.core.model.News.newBuilder()
          .setId(String.valueOf(news.get(i).getId()))
          .setHeading(news.get(i).getHeading());
     for(int j=0; j< news.get(i).getSources().size(); j++){
        com.example.demo.domain.Source source = news.get(i).getSources().get(j);
        com.thinhda.spring.grpc.core.model.Source sourceGrpc = Source.newBuilder()
             .setId(String.valueOf(source.getId()))
             .setName(source.getName())
             .setReference(source.getReference())
             .setLikes(source.getLikes())
             .build();
        newsGrpcBuilder.addSources(j, sourceGrpc);
     }
     com.thinhda.spring.grpc.core.model.News newsGrpc = newsGrpcBuilder.build();
     responseBuilder.addNews(i, newsGrpc);
  }
  MultipleNewsResponse response = responseBuilder.build();
  responseObserver.onNext(response);
  responseObserver.onCompleted();
}
```

Powyższa metoda przyjmuje obiekt GetNewsByKategoryRequest, “wyciąga” z niego nazwę kategorii, znajduje wszystkie obiekty news związane z tą kategorią i wysyła do klienta za pomocą metody onNext().
Pozostałe metody są podobne do tej.

LoginService

```
@GRpcService
public class LoginService extends LoginServiceGrpc.LoginServiceImplBase {
   private final UserRepo userRepo;

   public LoginService(UserRepo userRepo) {
       this.userRepo = userRepo;
   }
```

```
    @Override
    public void login(LoginRequest request, StreamObserver<LoginResponse>
responseObserver) {
        User user = userRepo.findByUsername(request.getUsername());
        LoginResponse.Builder responseBuilder = LoginResponse.newBuilder();
        if (user!=null&& user.getPassword().equals(request.getPassword())) {
            String uuid = UUID.randomUUID().toString();
            user.setToken(uuid);
            userRepo.save(user);
            responseBuilder.setToken(uuid);
        }else{
            responseBuilder.setToken("denied");
        }
        responseObserver.onNext(responseBuilder.build());
        responseObserver.onCompleted();
    }
}
```

Ta klasa zawiera metodę login która sprawdza przesłane username i password i wysyła klientowi token potrzebny dla dalszej autentykacji. Token będzie przechowywany w sesji użytkownika I dostępny do następnego logowania.

ResourcesService

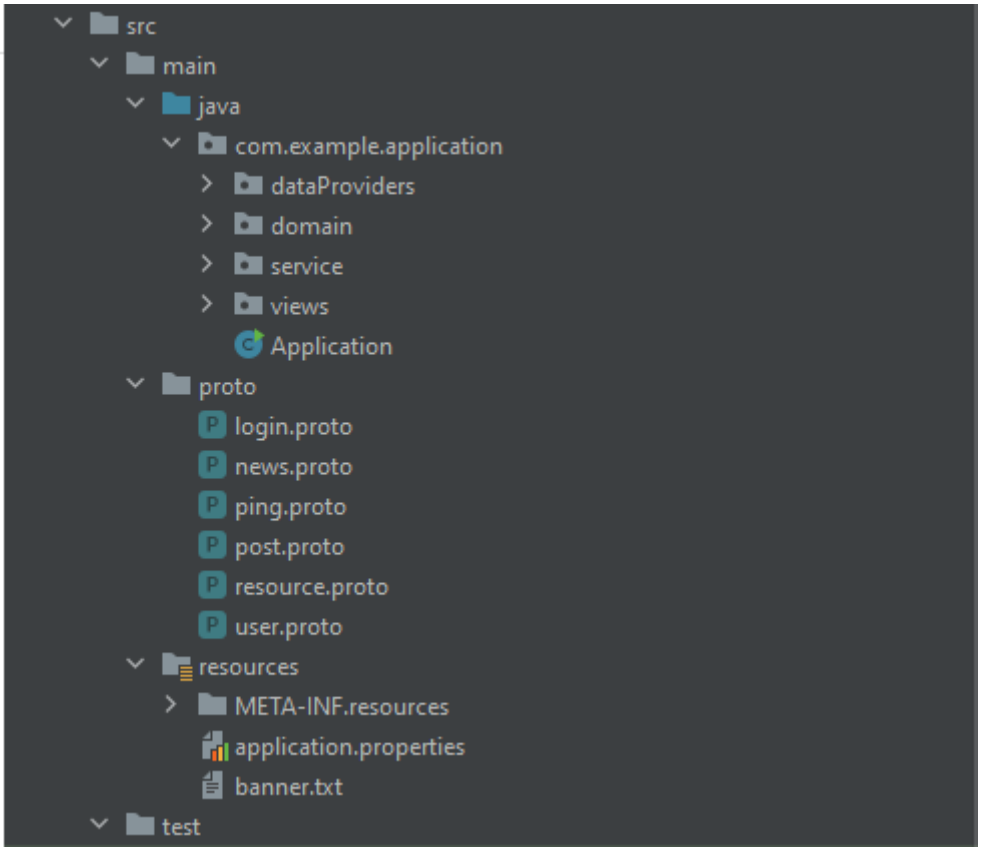
```
@GRpcService
public class ResourcesService extends ResourcesServiceGrpc.ResourcesServiceImplBase {
    private final ResourceRepo resourceRepo;

    public ResourcesService(ResourceRepo resourceRepo) {
        this.resourceRepo = resourceRepo;
    }

    @Override
    public void getAllResources(GetAllResourcesRequest request, StreamObserver<AllResources>
responseObserver) {
        List<com.example.demo.domain.Resource> resources = resourceRepo.findAll();
        AllResources.Builder response = AllResources.newBuilder();
        for (int i=0; i<resources.size(); i++){
            Resource resource = Resource.newBuilder()
                    .setName(resources.get(i).getName())
                    .setReference(resources.get(i).getReference())
                    .build();
            response.addResource(i, resource);
        }
        responseObserver.onNext(response.build());
        responseObserver.onCompleted();
    }
```

```
    @Override
    public void getResourceById(GetResourceByIdRequest request, StreamObserver<Resource>
responseObserver) {
        Long id = Long.valueOf(request.getId());
        com.example.demo.domain.Resource resource = resourceRepo.findById(id).get();
        if(resource ==null) {
            responseObserver.onError(new StatusException(Status.NOT_FOUND));
            return;
        }
        Resource response = Resource.newBuilder()
                .setName(resource.getName())
                .setReference(resource.getReference())
                .build();
        responseObserver.onNext(response);
        responseObserver.onCompleted();
    }

    @Override
    public void createNewResource(CreateResourceRequest request,
StreamObserver<CreateResourceResponse> responseObserver) {
        com.example.demo.domain.Resource resource = new
com.example.demo.domain.Resource(request.getName(), request.getReference());
        resourceRepo.save(resource);
        CreateResourceResponse response = CreateResourceResponse.newBuilder()
                .setCreated(true)
                .build();
        responseObserver.onNext(response);
        responseObserver.onCompleted();
    }

    @Override
    public void editResource(EditResourceRequest request,
StreamObserver<EditResourceResponse> responseObserver) {
        com.example.demo.domain.Resource resource  =
resourceRepo.findById(Long.parseLong(request.getId())).get();
        if (resource==null){
            responseObserver.onError(new StatusException(Status.NOT_FOUND));
            return;
        }else {
            resource.setName(request.getName());
            resource.setReference(request.getReference());
            resourceRepo.save(resource);
            EditResourceResponse response = EditResourceResponse.newBuilder()
                    .setUpdated(true)
                    .build();
            responseObserver.onNext(response);
            responseObserver.onCompleted();
        }
    }
```

Implementuje CRUD metody związane z obiektami Resource, editResource, crateResource, getById, getAll. Dane są przesyłane za pomocą technologii grpc.

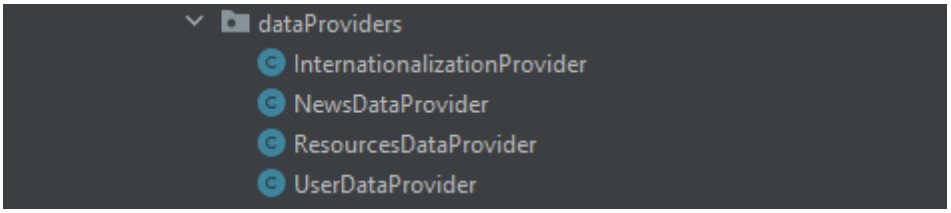
Web client

Struktura projektu



Projekt składa się z:
teczki dataProviders która zawiera klasy komunikujące z serwerem za pomocą Grpc
teczki domain zawierającej klasy reprezentujące obiekty z bazy danych na serwerze.
teczki views która zawiera klasy tworzące UI oraz logikę.
teczki proto zawirającej .proto pliki opisujące komukikacje z serverem.
teczki test zawirającej kalsy z unit oraz integration testami

Data providers



Folder zawiera klasy które dostarczają potrzebną informację z bazy danych serwera.

NewsDataProvider

Zawiera implementacje metod zdefiniowanych w klasie news.proto
klasa news.proto

```
service NewsService{
 rpc getNewsByCategory(GetNewsByCategoryRequest) returns(MultipleNewsResponse);
 rpc getNewsById(GetNewsByIdRequest) returns(SingleNewsResponse);
 rpc getAllNews(GetAllNewsRequest) returns(MultipleNewsResponse);
 rpc getAllCategories(GetCategoriesRequest) returns(MultipleCategoriesResponse);
 rpc deleteCategory(DeleteCategoryRequest) returns(DeleteResponse);
 rpc deleteSource(DeleteSourceRequest) returns(DeleteResponse);
 rpc deleteNews(DeleteNewsRequest) returns(DeleteResponse);
 rpc getSourcesByNews(GetSourcesByNewsRequest) returns(MultipleSourcesResponse);
 rpc createCategory(CreateCategoryRequest) returns (CreateResponse);
 rpc createNews(CreateNewsRequest) returns (CreateResponse);
 rpc createSource(CreateSourceRequest) returns(CreateResponse);
 rpc editCategory(EditCategoryRequest) returns(EditResponse);
 rpc editSource(EditSourceRequest) returns(EditResponse);
 rpc editNews(EditNewsRequest) returns(EditResponse);
}
```

Klasa NewsDataProvicer jest za duża żeby pokazać wszystkie metody które ona zawiera, dlatego pokaże tylko jedną pszykwadową metodę, całą klasę można zobaczyć w kodzie źródłowym.

```
@Service
public class NewsDataProvider {
  ManagedChannel channel;
  NewsServiceGrpc.NewsServiceBlockingStub stub;

  public NewsDataProvider() {
     channel = ManagedChannelBuilder.forTarget("localhost:6565").usePlaintext().build();
     stub = NewsServiceGrpc.newBlockingStub(channel);
  }

  public List<News> getNewsByCategory(String id){
     GetNewsByCategoryRequest request = GetNewsByCategoryRequest.newBuilder()
           .setCategory(id)
           .setToken("token").build();

     MultipleNewsResponse response  = stub.getNewsByCategory(request);

     List<News> news = new ArrayList<>();
     List<Source> sources = new ArrayList<>();
     for(int i=0; i< response.getNewsCount(); i++){
        for(int j=0; j<response.getNews(i).getSourcesCount();j++){
           sources.add(new Source(
                 response.getNews(i).getSources(j).getId(),
                 response.getNews(i).getSources(j).getName(),
                 response.getNews(i).getSources(j).getReference(),
                 response.getNews(i).getSources(j).getLikes()));
        }
        Category category = new Category(response.getNews(i).getCategory().getId(),
              response.getNews(i).getCategory().getName());

        news.add( new News(response.getNews(i).getId(),response.getNews(i).getHeading(), sources,
category));
        sources.clear();
     }
     return news;
  }
```

Metoda getNewsByCategory wysyła żądanie do serwera otrzymuje odpowiedź zawierającą listę obiektów news, wyciąga listę z obiektu responce i zwraca za pomocą return słowa kluczowego.

ResourcesDataProvider
implementuje metody zdefiniowane w pliku resources.proto
plik resource.proto

```
message Resource{
 string name = 1;
 string reference = 2;
}
message AllResources{
 repeated Resource resource = 1;
}
message GetResourceByIdRequest{
 string id =1;
 string token =2;
}
message GetAllResourcesRequest{
 string token= 1;
}
message CreateResourceRequest{
 string name=1;
 string reference =2;
}
message CreateResourceResponse{
 bool created =1;
}

message EditResourceRequest{
 string id =1;
 string name = 2;
 string reference = 3;
}
message EditResourceResponse{
 bool updated =1;
}

service ResourcesService{
 rpc getAllResources(GetAllResourcesRequest) returns(AllResources);
 rpc getResourceById(GetResourceByIdRequest) returns(Resource);
 rpc createNewResource(CreateResourceRequest) returns(CreateResourceResponse);
 rpc editResource(EditResourceRequest) returns(EditResourceResponse);
}
```

ResourcesDataProvider

```
@Service
public class ResourcesDataProvider {
```

```
  private final ManagedChannel channel;
  private final ResourcesServiceGrpc.ResourcesServiceBlockingStub stub;

  public ResourcesDataProvider() {
     this.channel = ManagedChannelBuilder.forTarget("localhost:6790").usePlaintext().build();
     this.stub = ResourcesServiceGrpc.newBlockingStub(this.channel);

  }

  public List<Resource> getAllResources(){

     GetAllResourcesRequest request =
GetAllResourcesRequest.newBuilder().setToken("token").build();

     AllResources response = stub.getAllResources(request);

     List<Resource> resources = new ArrayList<>();

     for(int i=0; i<response.getResourceCount(); i++) {
        resources.add(new Resource(response.getResource(i).getName(),
response.getResource(i).getReference()));
     }

     return resources;
  }

  public boolean createResource(String name, String reference){
     CreateResourceRequest request =
CreateResourceRequest.newBuilder().setName(name).setReference(reference).build();

     CreateResourceResponse response = stub.createNewResource(request);

     return response.getCreated();
  }

  public boolean editResource(String id, String name, String reference){
     EditResourceRequest request =
EditResourceRequest.newBuilder().setId(id).setName(name).setReference(reference).build();

     EditResourceResponse response = stub.editResource(request);

     return response.getUpdated();
  }
}
```

Metody klasy ResourceService wysyłają żądania do serwera i otrzymują odpowiedzi zawierające odpowidzi servera.

UserDataProvider implemetuje metodę login zdefiniowanej w pliku login.proto

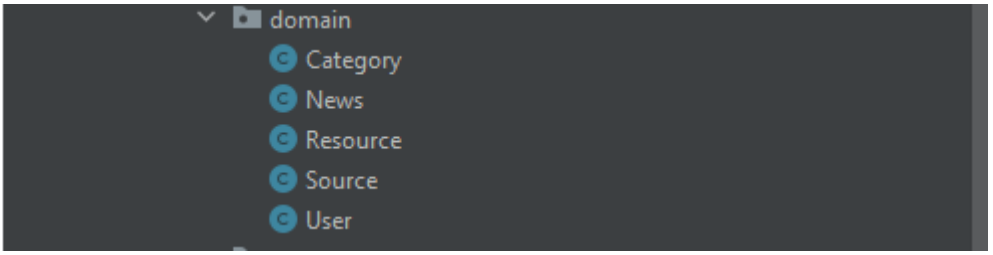
```
package sms.core;

message LoginRequest{
 string username = 1;
 string password = 2;
}

message LoginResponse{
 string token = 1;
}

//login service receives request that contains username and password and returns response with token,
// you should save token to the android session, you should add this token to each request, otherwise you will get 403 unauthenticated
service LoginService{
 rpc login (LoginRequest) returns (LoginResponse);
}
```

UserDataProvider

```
@Service
public class UserDataProvider{
   public String authenticate(String username, String password){
      ManagedChannel channel = ManagedChannelBuilder.forTarget("localhost:6565").usePlaintext().build();

      LoginServiceGrpc.LoginServiceBlockingStub stub = LoginServiceGrpc.newBlockingStub(channel);

      LoginRequest request = LoginRequest.newBuilder().setUsername("username").setPassword("Password").build();

      LoginResponse response = stub.login(request);

      channel.shutdownNow();

      return response.getToken();
   }
}
```

Metoda login wysyła do serwera username i passwrod wprowadzone przez użytkownika, otrzymuje w odpowiedzi token albo exception jeśli nazwa użytkownika lub chaslo jest nieprawidłowe

Domain



W folderze domain znajdują się klasy reprezentujące obiekty bazy danych serwera.

Category

```
public class Category {
  private String id;
  private String name;

  public Category(String id, String name) {
     this.id = id;
     this.name = name;
  }


//getters , setters , equals , hashcode
```

News

```
public class News {
  private String id;
  private String heading;
  private List<Source> sources;
  private Category category;

  public News(String id, String heading, List<Source> sources, Category category) {
     this.id = id;
```

```
        this.heading = heading;
        this.sources = sources;
        this.category = category;
    }

    //getters , setters , equals , hashcode
```

Resource

```
@Entity
public class Resource {
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private Long id;

    private String name;
    private String reference;

    public Resource() {
    }

    public Resource(String name, String reference) {
        this.name = name;
        this.reference = reference;
    }

//getters , setters , equals , hashcode
```

Source

```
public class Source {
    private String id;
    private String name;
    private String reference;
    private int likes;

    public Source(String id, String name, String reference, int likes) {
        this.id = id;
        this.name = name;
        this.reference = reference;
        this.likes = likes;
    }
//getters , setters , equals , hashcode
```

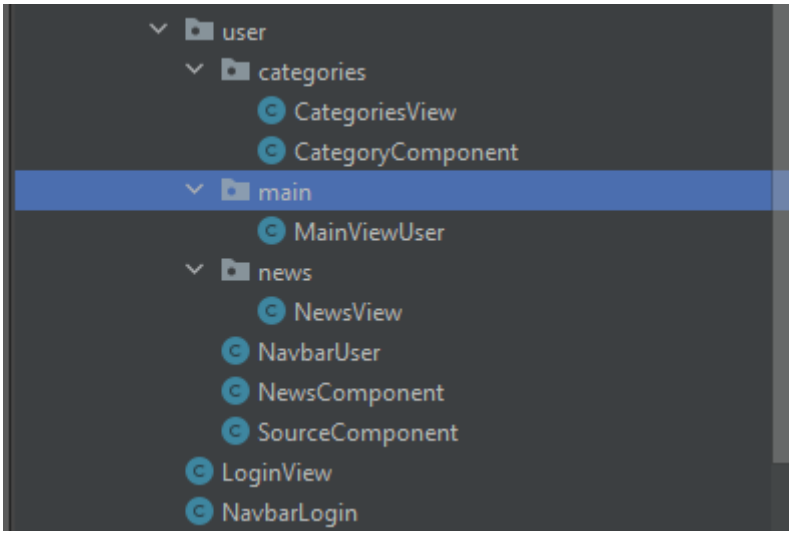
User

```
public class User {
  private Long id;

  private String username;
  private String password;
  private String token;

  public User() {
  }

  public User(String username, String password, String token) {
     this.username = username;
     this.password = password;
     this.token = token;
  }

  public Long getId() {
     return id;
  }
```

Views

Folder views zawiera dwie części user i admin, user zawiera odpowiadające za UI oraz logikę dla użytkowników , admin zawiera klasy tworzące UI i logikę dla użytkowników z rolą ADMIN.

User



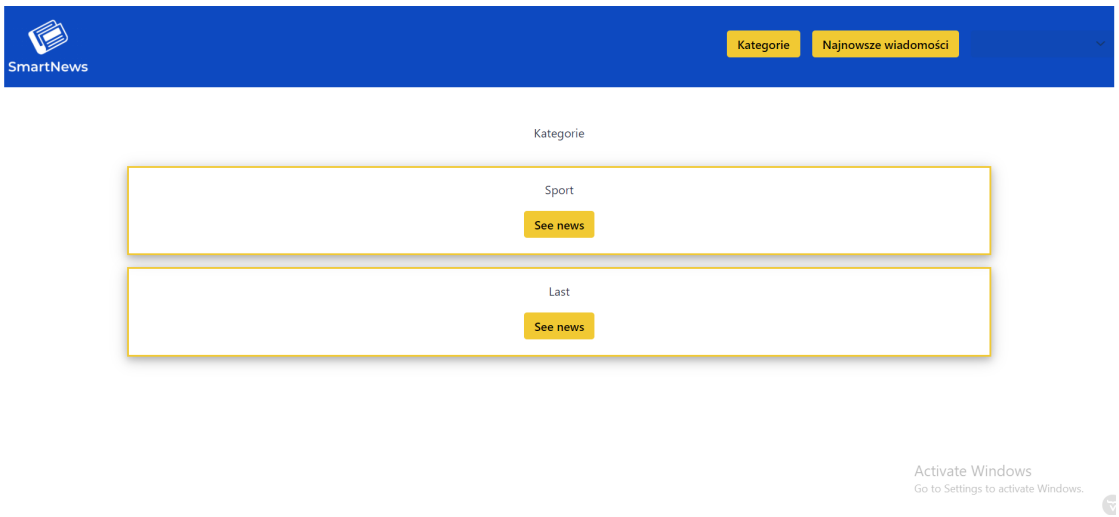
W folderze categories znajdują się klasa CategoriesView, która opisuję stronę na której będą wyświetlona lista kategorii.

```
 @Route("categories")
public class CategoriesView extends VerticalLayout {
   private Label categoryLabel;
   public CategoriesView(InternationalizationProvider internationalizationProvider,
                         NewsDataProvider newsDataProvider){
      //get selected language;
      String language;
      try {
         language = VaadinSession.getCurrent().getAttribute("language").toString();
      } catch (Exception ex) {
         language = "Polski";
         VaadinSession.getCurrent().setAttribute("language", "Polski");
      }
      this.add(new NavbarUser(internationalizationProvider));
      VerticalLayout centered = new VerticalLayout();
      centered.setWidth("80%");
```

```
        this.add(centered);
        this.setAlignItems(Alignment.CENTER);
        VerticalLayout labelLayout = new VerticalLayout();
        labelLayout.setAlignItems(Alignment.CENTER);
        categoryLabel = new
Label(internationalizationProvider.getCategoriesButtonString().get(language));
        labelLayout.add(categoryLabel);
        centered.add(labelLayout);
        this.add(centered);
        List<Category> categories = newsDataProvider.getAllCategories();
        if(categories.size()==0){
            this.add(new Label("No categories jet"));
        }else{
            for(Category category : categories){
                centered.add(new CategoryComponent(category));
            }
        }
    }
```

I klasy CategoryComponent którą reprezentuje poszczególne kategorii na stronie.

```
public class CategoryComponent extends VerticalLayout {
    private Label nameLabel;
    private Button seeNewsButton;
    public CategoryComponent(Category category){
        nameLabel = new Label(category.getName());
        seeNewsButton = new Button("See news", event->{
            VaadinSession.getCurrent().setAttribute("category", category.getName());
            UI.getCurrent().navigate("news");
        });
        seeNewsButton.getStyle().set("color", "black");
        seeNewsButton.getStyle().set("background-color", "#F1C933");
        this.add(nameLabel, seeNewsButton);
        //style
        this.getStyle().set("box-shadow", "0 4px 10px 0 rgba(0,0,0,0.2), 0 4px 20px 0 rgba(0,0,0,0.19)");
        this.getStyle().set("border-style", "solid");
        this.getStyle().set("border-width", "3px");
        this.getStyle().set("border-color", "#F1C933");
        this.setAlignItems(Alignment.CENTER);
    }
}
```

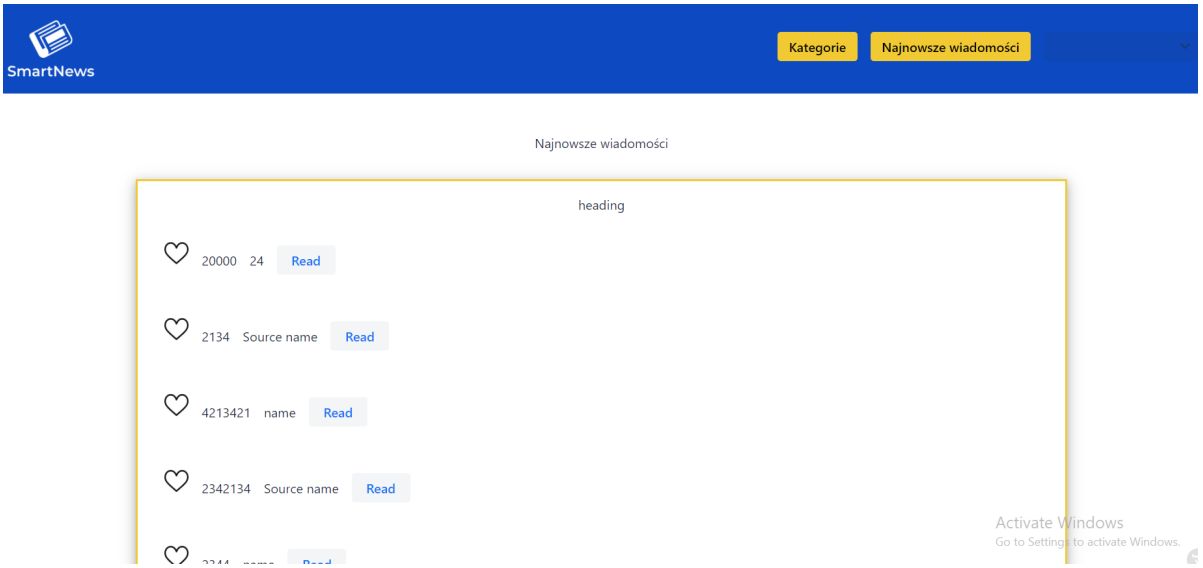


strona categories

W folderze main znajduje się klasa MinViewUser która definiuje główną dla użytkowników stronę, wykorzystuje component NewsComponent dla wyświetlenia ostatnich nowości.

```
@Route("")
public class MainViewUser extends VerticalLayout {
  private NavbarUser navbarUser;
  //UI components
  private Label lastNews;

  public MainViewUser(InternationalizationProvider internationalizationProvider,
                      NewsDataProvider newsDataProvider){
    //get selected language;
    String language;
    try {
      language = VaadinSession.getCurrent().getAttribute("language").toString();
    } catch (Exception ex) {
      language = "Polski";
      VaadinSession.getCurrent().setAttribute("language", "Polski");
    }
    this.navbarUser = new NavbarUser(internationalizationProvider);
    VerticalLayout centered = new VerticalLayout();
    VerticalLayout labelLayout = new VerticalLayout();
    labelLayout.setWidth("100%");
    labelLayout.setAlignItems(Alignment.CENTER);
    lastNews = new Label(internationalizationProvider.getLastButtonStrings().get(language));
    labelLayout.add(lastNews);
```

```
        centered.setWidth("80%");
        this.setAlignItems(Alignment.CENTER);
        centered.add(labelLayout);
        List<News> news = newsDataProvider.getNewsByCategory("Last");
        if (news !=null || news.size()!=0) {
            for (News n : news) {
                centered.add(new NewsComponent(internationalizationProvider, n, newsDataProvider));
            }
        }else{
            centered.add(new Label("No news jet"));
        }
        this.add(navbarUser, centered);
    }
}
```

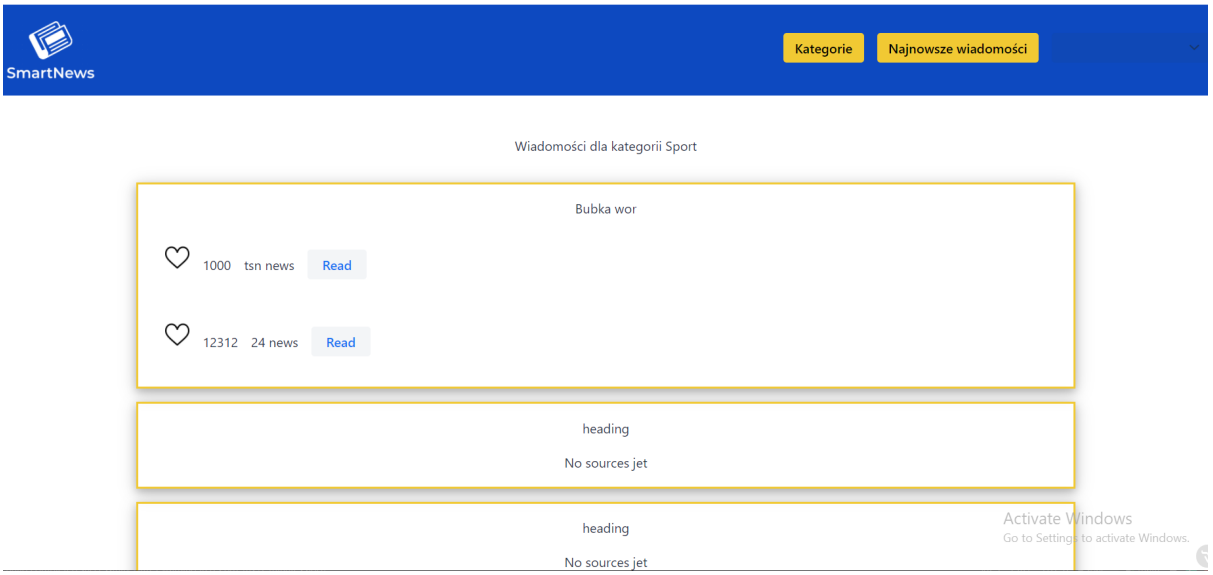


strona main

W folderze news znajdują się klasa NewsView która opisuje UI strony news i korzysta z klasy NewsComponent dla wyświetlenia nowości.

```
@Route("news")
public class NewsView extends VerticalLayout {
    private Label newsLabel;
    public NewsView(InternationalizationProvider internationalizationProvider,
                    NewsDataProvider newsDataProvider){
        //get selected language;
        String language;
        try {
            language = VaadinSession.getCurrent().getAttribute("language").toString();
        } catch (Exception ex) {
            language = "Polski";
            VaadinSession.getCurrent().setAttribute("language", "Polski");
        }
        String categoryName = (String) VaadinSession.getCurrent().getAttribute("category");
```

```
        this.add(new NavbarUser(internationalizationProvider));
        VerticalLayout centered = new VerticalLayout();
        centered.setWidth("80%");
        this.setAlignItems(Alignment.CENTER);
        VerticalLayout labelLayout = new VerticalLayout();
        newsLabel = new Label(internationalizationProvider.getNewsLabelStrings().get(language)+" "+
categoryName);
        labelLayout.setWidth("100%");
        labelLayout.setAlignItems(Alignment.CENTER);
        labelLayout.add(newsLabel);
        centered.add(labelLayout);
        for (News news: newsDataProvider.getNewsByCategory(categoryName)){
            centered.add(new NewsComponent(internationalizationProvider, news, newsDataProvider));
        }
        this.add(centered);
    }
}
```



strona news

Klasa NewsComponent odpowiada za wyświetlenie przekazywanego w construktor obiektu news.

```
public class NewsComponent extends VerticalLayout {

    //UI components
    private Label heading;

    public NewsComponent(InternationalizationProvider internationalizationProvider,
                         News news,
```

```
                    NewsDataProvider newsDataProvider){
        this.setAlignItems(Alignment.CENTER);
        heading=new Label(news.getHeading());
        this.add(heading);
        List<Source> sources = newsDataProvider.getSourcesByNews(news);
        if(sources !=null && sources.size()!=0) {
            for (Source source : sources) {
                this.add(new SourceComponent(internationalizationProvider, source, newsDataProvider));
            }
        }else {
            this.add(new Label("No sources jet"));
        }
        this.getStyle().set("box-shadow", "0 4px 10px 0 rgba(0,0,0,0.2), 0 4px 20px 0 rgba(0,0,0,0.19)");
        this.getStyle().set("border-style", "solid");
        this.getStyle().set("border-width", "3px");
        this.getStyle().set("border-color", "#F1C933");
    }
}
```

Używa klasę SourceComponent dla wyświetlania obiektów source związanych z newsem.

```
public class SourceComponent extends VerticalLayout {
    private final InternationalizationProvider internationalizationProvider;
    private final NewsDataProvider newsDataProvider;
    //UI components
    private Image displayImage;
    private Label name;
    private Button read;
    private Label likes;

    public SourceComponent(InternationalizationProvider internationalizationProvider,
                           Source source,
                           NewsDataProvider newsDataProvider){
        this.internationalizationProvider = internationalizationProvider;
        this.newsDataProvider = newsDataProvider;
        //UI initialization
        displayImage = new Image("https://sklquest.com/static/uploads/like.png", "empty hard");
        displayImage.setHeight("30px");
        displayImage.setWidth("30px");
        displayImage.addClickListener(i->like(source));
        name = new Label(source.getName());
        read  =new Button("Read" , event->{
            UI.getCurrent().getPage().executeJavaScript("window.open(\""+source.getReference()+"\",
\"_blank\");");
        });
        likes = new Label(String.valueOf(source.getLikes()));
        HorizontalLayout layout =new  HorizontalLayout();
        layout.add(displayImage, likes, name, read);
        layout.setWidth("100%");
        layout.setAlignItems(Alignment.BASELINE);
```

```
      this.add(layout);
  }

  private void like(Source source){
      displayImage.setSrc("https://sklquest.com/static/uploads/heart.png");
      likes.setText(String.valueOf(source.getLikes()+1));
      boolean edited = newsDataProvider.editSource(source.getId(), source.getName(),
source.getReference());
  }
}
```

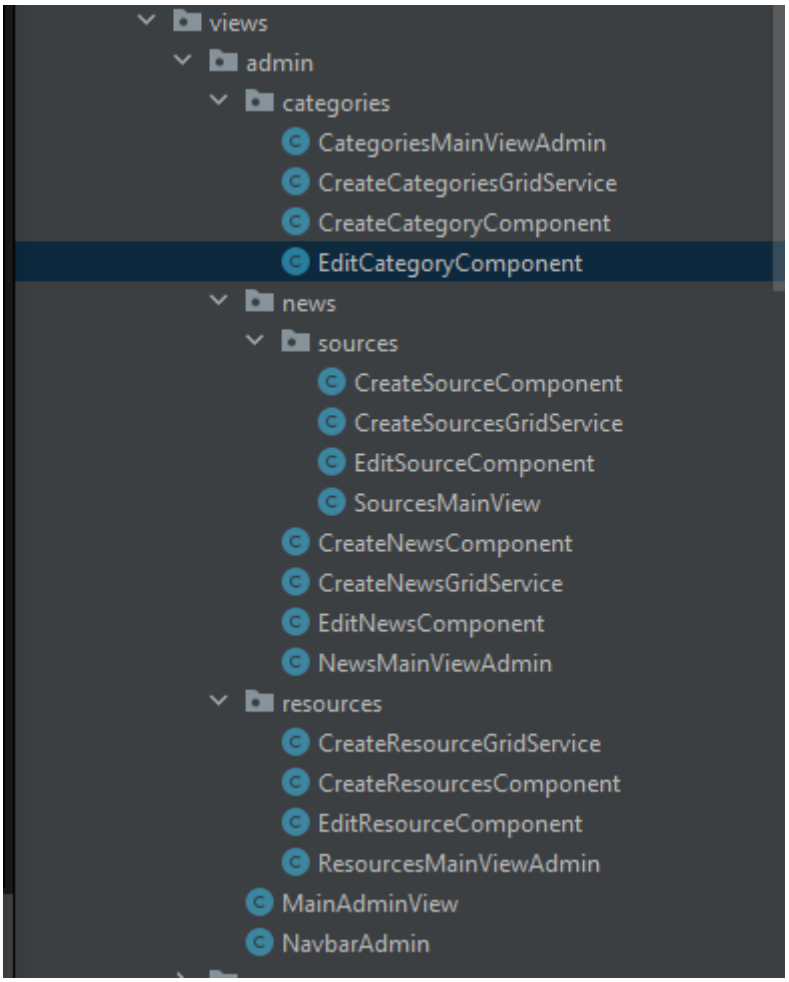
Klasa LoginView odpowiada za stronę dla logowania do części przeznaczonej dla adminów.

```
@Route(value = "login")
@PageTitle("Login")
public class LoginView extends Div {
  public static final String ROUTE = "login";

  public LoginView(UserDataProvider userDataProvider,
            AuthService authService) {
    //block to center the form
    VerticalLayout centered = new VerticalLayout();
    centered.setAlignItems(FlexComponent.Alignment.CENTER);
    centered.setWidth("100%");
    //create navbar
    //form
    VerticalLayout  verticalLayout = new VerticalLayout();
    verticalLayout.setWidth("auto");
    verticalLayout.setPadding(true);
    com.example.application.views.NavbarLogin navbarView = new
com.example.application.views.NavbarLogin();
    TextField userNameTextField = new TextField("identifier");
    PasswordField passwordField = new PasswordField("password");
    Button submitButton = new Button("Увійти" , event -> {
      try {
        String token = userDataProvider.authenticate(userNameTextField.getValue(),
passwordField.getValue());
        VaadinSession.getCurrent().setAttribute("token", token);
        authService.authenticate(userNameTextField.getValue());
      } catch (Exception e) {
        Notification.show("Wrong id or password");
      }
      UI.getCurrent().navigate("personal");
    });
    submitButton.getStyle().set("color", "black");
    submitButton.getStyle().set("background-color", "#F1C933");
    VerticalLayout centerSubmitButton = new VerticalLayout();
    centerSubmitButton.setAlignItems(FlexComponent.Alignment.CENTER);
```

```
    centerSubmitButton.add(submitButton);
    verticalLayout.add(userNameTextField, passwordField, centerSubmitButton);
    centered.add(verticalLayout);
    add(navbarView,centered);
  }
}
```

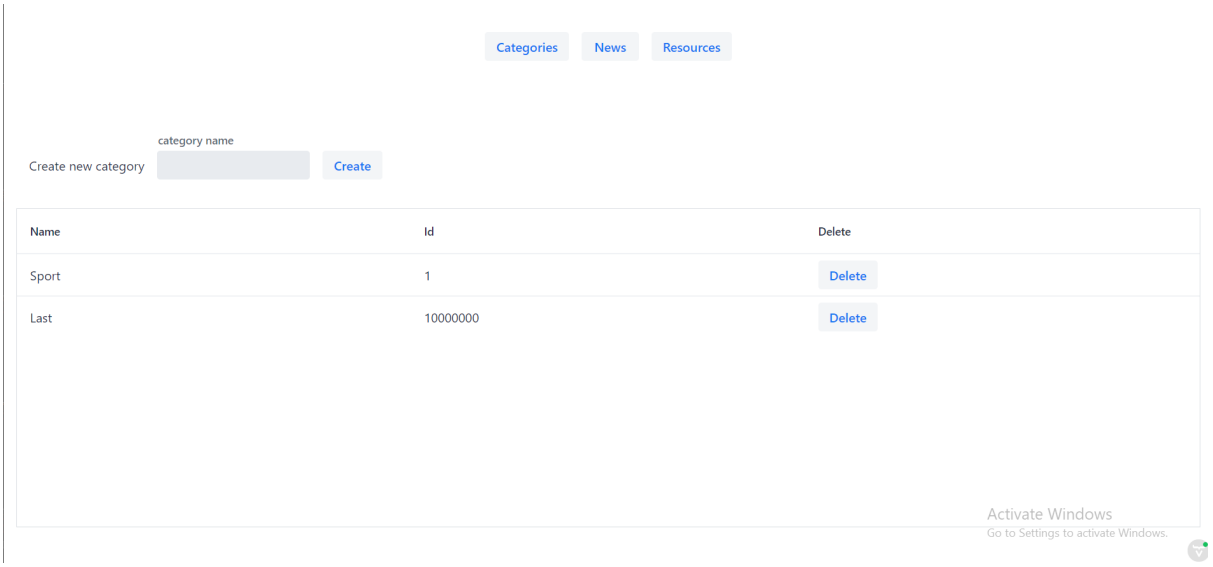
strona login

Admin

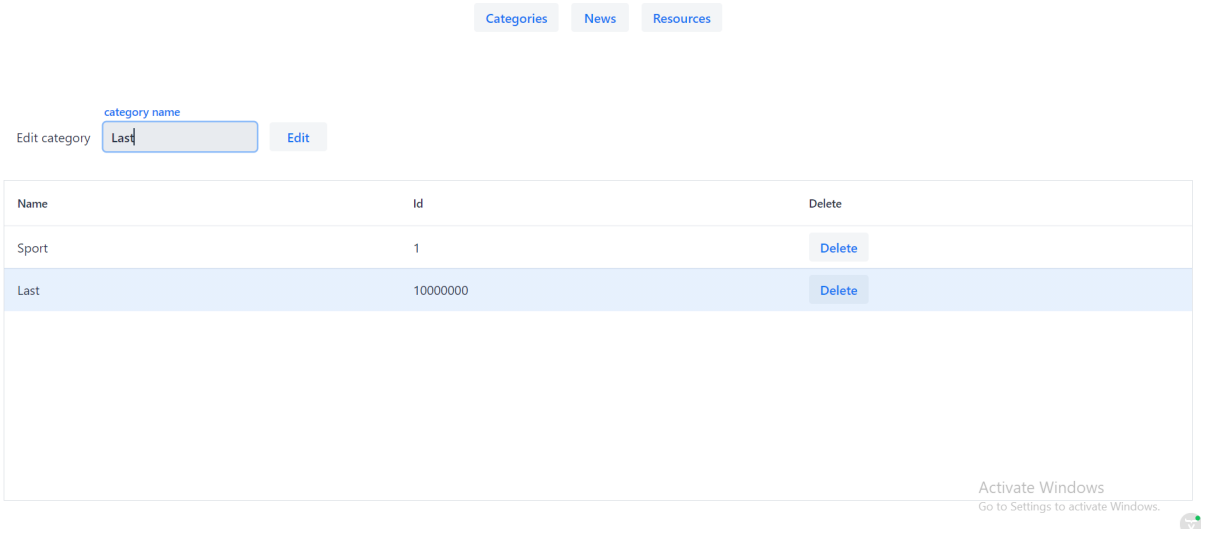


Część aplikacji admin zawiera 4 foldery, każdy odpowiada za jedną stronę.

Folder categories odpowiada za stronę categories admin.



Na stronie możemy zobaczyć wszystkie kategorie, usunąć kategorie, edytować kategorie.



Aby edytować kategorie trzeba kliknąć na kategorie lewym przyciskiem.

Klasa CategoriesMainViewAdmin

```
public class CategoriesMainViewAdmin extends VerticalLayout {
  private final NavbarAdmin navbarAdmin;
  private final NewsDataProvider newsDataProvider;
  private Grid<Category> grid;
  private HorizontalLayout modificationComponentDisplayed;


  public CategoriesMainViewAdmin(NewsDataProvider newsDataProvider,
                                 CreateCategoriesGridService createCategoriesGridService){
    this.newsDataProvider = newsDataProvider;
    //UI initialization
    this.navbarAdmin = new NavbarAdmin();
    this.grid = createCategoriesGridService.createGrid();
    CreateCategoryComponent createCategoryComponent = new CreateCategoryComponent(grid,
this.newsDataProvider);
    this.modificationComponentDisplayed = createCategoryComponent;
    grid.addItemClickListener(item -> this.itemClickEvent(item.getItem(),
createCategoryComponent));
    this.displayUiComponents();
  }

  private void itemClickEvent(Category category, CreateCategoryComponent
createCategoryComponent){
    this.modificationComponentDisplayed = new
EditCategoryComponent(category,newsDataProvider, grid, this, navbarAdmin,
createCategoryComponent );
    this.removeAll();
    this.displayUiComponents();
  }

  private void displayUiComponents(){
    this.add(navbarAdmin, modificationComponentDisplayed, grid);
  }
}
```

Odpowiada za tworzenie i wyświetlanie komponentów tworzących stronę

CreateCategoryGridService

```
@Service
public class CreateCategoriesGridService {
  private final NewsDataProvider newsDataProvider;

  private Grid<Category> grid;

  public CreateCategoriesGridService(NewsDataProvider newsDataProvider) {
     this.newsDataProvider = newsDataProvider;
  }
```

```
  public Grid<Category> createGrid(){
     grid = new Grid<>();
     grid.setItems(newsDataProvider.getAllCategories());
     grid.addColumn(Category::getName).setHeader("Name");
     grid.addColumn(Category::getId).setHeader("Id");
     grid.addComponentColumn(this::createDeleteButton).setHeader("Delete");
     return grid;
  }

  private Button createDeleteButton(Category category){
     return new Button("Delete", event->{
        newsDataProvider.deleteCategory(category);
     });
  }
}
```

Pobiera dane z serwera za pomocą wcześniej opisanie klasy newsDataProvied i używa pobrane dane do utworzenia obiektu Grid wyświetlającego ich na ekranie.

CreateCategoryComponent

```
public class CreateCategoryComponent extends HorizontalLayout {
  private final NewsDataProvider newsDataProvider;
  private Grid<Category> grid;

  private Label createLabel;
  private TextField categoryName;
  private Button submitButton;

  public CreateCategoryComponent(Grid<Category> grid,
                                 NewsDataProvider newsDataProvider){
     this.newsDataProvider = newsDataProvider;
     this.grid = grid;

     createLabel = new Label("Create new category");

     createLabel = new Label("Create new category");
     categoryName = new TextField("category name");
     submitButton = new Button("Create", event->{
        this.createNewCategory();
     });
     this.add(createLabel , categoryName, submitButton);
     this.setWidth("100%");
     this.setPadding(true);
     this.setAlignItems(Alignment.BASELINE);

  }
```

```
  public void createNewCategory(){
     boolean created = newsDataProvider.createCategory(categoryName.getValue());
     if (!created) Notification.show("Category was not created, wrong data");
     categoryName.setValue("");
     grid.setItems(newsDataProvider.getAllCategories());
  }

  public Label getCreateLabel() {
     return createLabel;
  }

  public TextField getCategoryName() {
     return categoryName;
  }

  public Button getSubmitButton() {
     return submitButton;
  }
```

Odpowiada wyświetlenie komponentu odpowiadającego za utworzenie nowego obiektu category. żeby stworzyć nowy obiekt w bazie danych serwera używa klasę NewsDataProwider.

EditCatagoryComponent

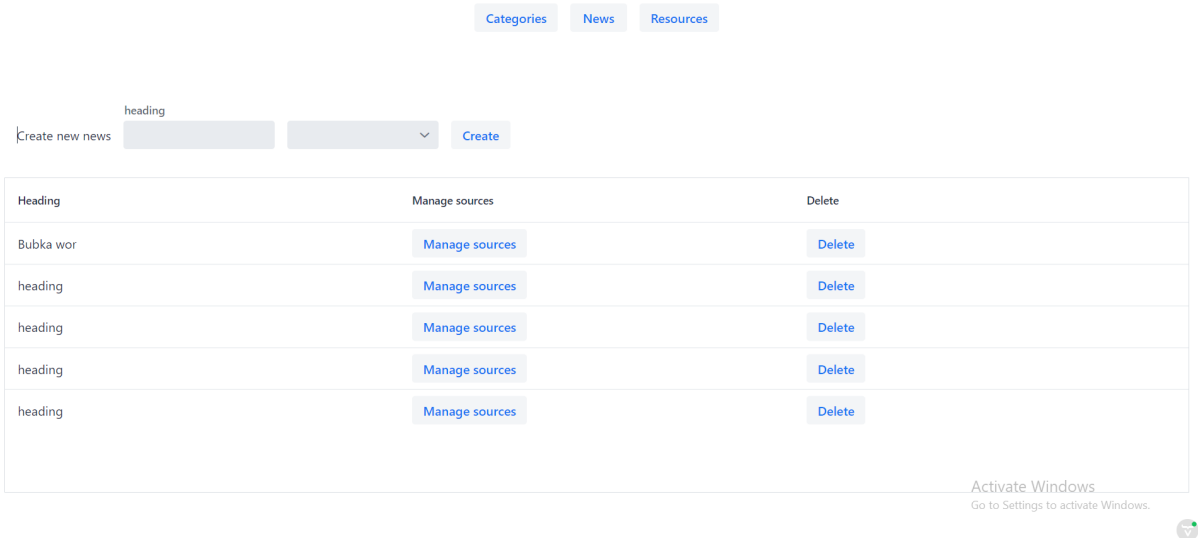
```
public class EditCategoryComponent extends HorizontalLayout {
   private final Category category;
   private final NewsDataProvider newsDataProvider;
   private final Grid<Category> grid;
   private final CategoriesMainViewAdmin adminView;
   private final NavbarAdmin navbarAdmin;
   private final CreateCategoryComponent createCategoryComponent;
   //UI elements
   private Label editLabel;
   private TextField categoryName;
   private Button submitButton;

   public EditCategoryComponent(Category category,
                                NewsDataProvider newsDataProvider,
                                Grid<Category> grid,
                                CategoriesMainViewAdmin categoriesMainViewAdmin,
                                NavbarAdmin navbarAdmin,
                                CreateCategoryComponent createCategoryComponent){
      this.category = category;
      this.newsDataProvider = newsDataProvider;
      this.grid = grid;
      this.adminView = categoriesMainViewAdmin;
      this.navbarAdmin = navbarAdmin;
      this.createCategoryComponent = createCategoryComponent;
```

```
        //UI initialization
        editLabel = new Label("Edit category");
        categoryName = new TextField("category name");
        submitButton = new Button("Edit", event->{
            this.editCategory();
        });
        this.add(editLabel,categoryName, submitButton);
        this.setWidth("100%");
        this.setPadding(true);
        this.setAlignItems(Alignment.BASELINE);
    }

    public void editCategory(){
        if (categoryName.getValue().equals("")) {
            Notification.show("Category name can not be empty");
        }else {
            boolean edited = newsDataProvider.editCategory(category.getId(), categoryName.getValue());
            if (!edited) Notification.show("Category was not edited, wrong data")
                    ;
            grid.getDataProvider().refreshItem(category);
            adminView.removeAll();
            adminView.add(navbarAdmin, createCategoryComponent, grid);
        }
    }

    public Label getEditLabel() {
        return editLabel;
    }

    public TextField getCategoryName() {
        return categoryName;
    }

    public Button getSubmitButton() {
        return submitButton;
    }
}
```

Komponent odpowiada za element edytujący kategorie po kliknięciu na nią.

News

Folder news odpowiada za stronę newsAdmin



Na stronie możemy zobaczyć wszystkie wiadomości z bazy danych serwera, usunąć kategorie, edytować kategorie, utworzyć nową kategorie.

Po kliknięciu na “ManageResources” otworzy się strona dla zarządzania źródłami dołączonymi do newsa.

NewsMainViewAdmin

```
public class NewsMainViewAdmin extends VerticalLayout {
   private final NewsDataProvider newsDataProvider;

   private final NavbarAdmin navbarAdmin;
   private Grid<News> grid;
   private HorizontalLayout modificationComponentDisplayed;

   public NewsMainViewAdmin(NewsDataProvider newsDataProvider,
                            CreateNewsGridService createNewsGridService){
      this.newsDataProvider = newsDataProvider;
      //UI initialization
      navbarAdmin = new NavbarAdmin();
      grid = createNewsGridService.createGrid();
      CreateNewsComponent createNewsComponent = new
CreateNewsComponent(newsDataProvider, grid);
      modificationComponentDisplayed = createNewsComponent;
```

```
    grid.addItemClickListener(item->this.itemClickEvent(item.getItem(), createNewsComponent));
    this.displayUIComponents();
  }

  private void itemClickEvent(News news, CreateNewsComponent createNewsComponent){
    EditNewsComponent editNewsComponent = new EditNewsComponent(news,
        newsDataProvider,
        grid,
        this,
        navbarAdmin,
        createNewsComponent);
    modificationComponentDisplayed = editNewsComponent;
    this.removeAll();
    this.displayUIComponents();
  }

  private void displayUIComponents(){
    this.add(navbarAdmin, modificationComponentDisplayed, grid);
  }
```

Odpowiada za tworzenie i wyświetlanie komponentów tworzących stronę, używa klasy EditNewsComponent, CreateNewasComponent, CreateNewsGridService

CreateNewsGridService

```
@Service
public class CreateNewsGridService  {
  private final NewsDataProvider newsDataProvider;

  private Grid<News> grid;

  public CreateNewsGridService(NewsDataProvider newsDataProvider) {
    this.newsDataProvider = newsDataProvider;
  }

  public Grid<News> createGrid(){
    this.grid = new Grid<News>();
    grid.setItems(newsDataProvider.getAllNews());
    grid.addColumn(News::getHeading).setHeader("Heading");
    grid.addComponentColumn(this::createManageSourcesButton).setHeader("Manage sources");
    grid.addComponentColumn(this::createDeleteButton).setHeader("Delete");
    return grid;
  }

  private Button createManageSourcesButton(News news){
    return new Button("Manage sources", event->{
      VaadinSession.getCurrent().setAttribute("newsId", news.getId());
      UI.getCurrent().navigate("admin/news/sources");
```

```
        });
    }

    private Button createDeleteButton(News news){
        return new Button("Delete", event->{
            newsDataProvider.deleteNewsWithSources(news);
        });
    }
}
```

Odpowiada wyświetlenie komponentu odpowiadającego za utworzenie nowego obiektu news. żeby stworzyć nowy obiekt w bazie danych serwera używa klasę NewsDataProwider.

EditNewsComponent

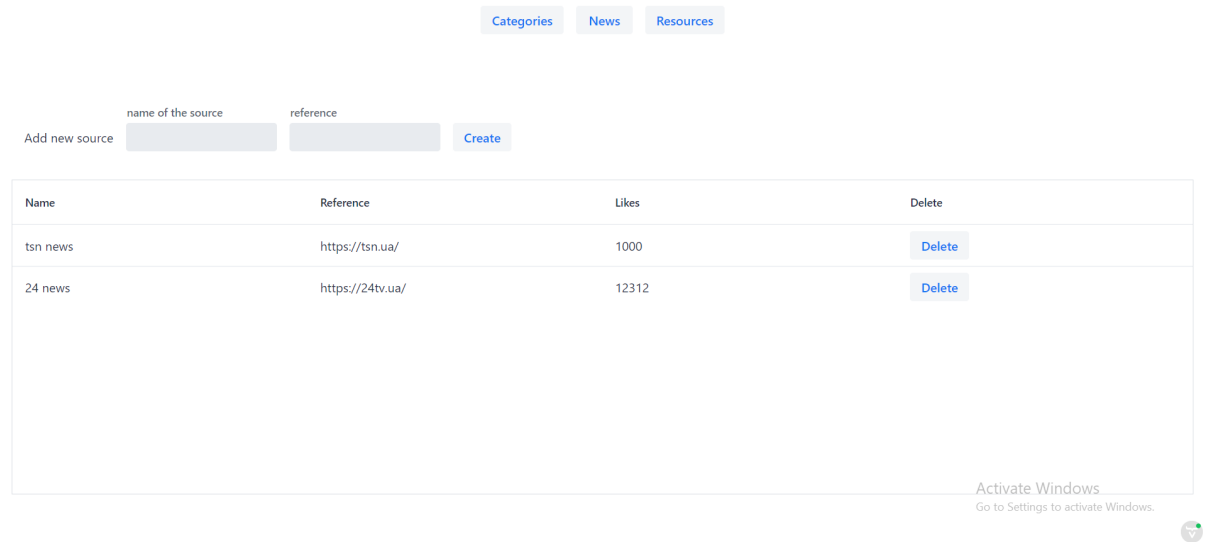
```
public class EditNewsComponent extends HorizontalLayout {
    private final News news;
    private final NewsDataProvider dataProvider;
    private final Grid<News> grid;
    private final NavbarAdmin navbarAdmin;
    private final CreateNewsComponent createNewsComponent;
    private final NewsMainViewAdmin adminView;
    //UI components
    private Label editLabel;
    private TextField heading;
    private Select<String> categorySelect;
    private Button submitButton;

    public EditNewsComponent(News news,
                             NewsDataProvider dataProvider,
                             Grid<News> grid,
                             NewsMainViewAdmin adminView,
                             NavbarAdmin navbarAdmin,
                             CreateNewsComponent createNewsComponent){
        this.news = news;
        this.adminView = adminView;
        this.dataProvider = dataProvider;
        this.grid = grid;
        this.navbarAdmin = navbarAdmin;
        this.createNewsComponent = createNewsComponent;
        //UI initialization
        editLabel= new Label("Create new news");
        heading = new TextField("heading");
        categorySelect = new Select<>();
        categorySelect.setItems(dataProvider.getCategoriesNames());
        categorySelect.setLabel("category");
        submitButton = new Button("Create", event->{
            this.editNews();
        });
```

```
        this.add(editLabel, heading, categorySelect, submitButton);
        this.setWidth("100%");
        this.setPadding(true);
        this.setAlignItems(Alignment.BASELINE);
    }

    public void  editNews(){
        if(heading.getValue().equals("") || categorySelect.getValue().equals("") ){
            Notification.show("Fields can not be empty");
        }else{
            boolean edited = dataProvider.editNews(news.getId(), heading.getValue(),
categorySelect.getValue());
            if(!edited) Notification.show("Category was not edited, wrong data")
                    ;
            grid.setItems(dataProvider.getAllNews());
            adminView.removeAll();
            adminView.add(navbarAdmin, createNewsComponent, grid);
        }
    }
```

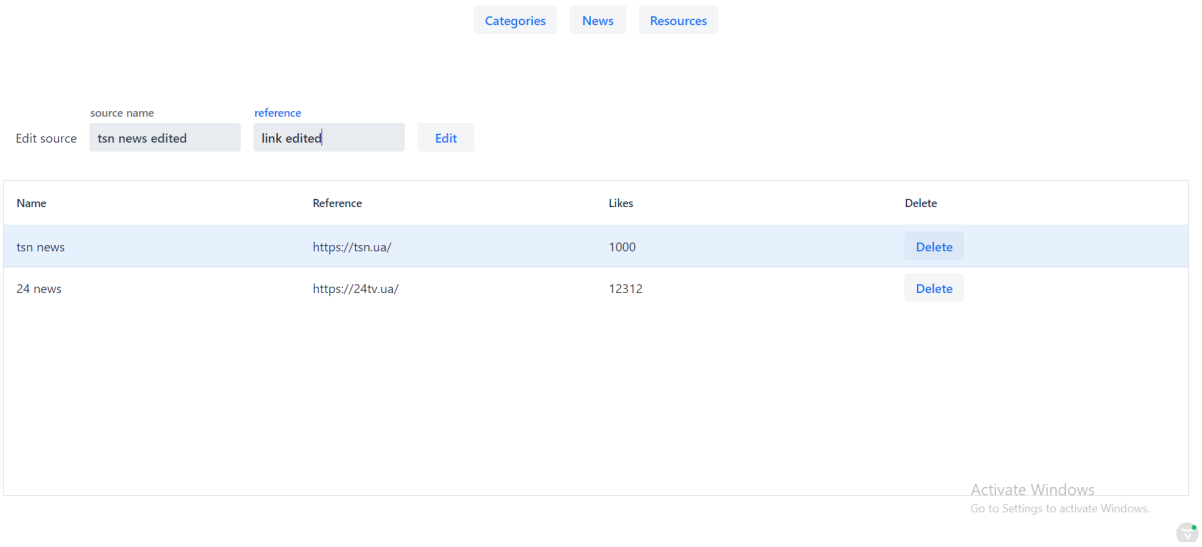
Komponent odpowiada za element edytujący nowość po kliknięciu na nią.

Sources

Folder news odpowiada za stronę sourcesAdmin



Edytować elementy można kliknięciem na nich



SourcesMainViewAdmin

```
public class SourcesMainView extends VerticalLayout {
  private final NewsDataProvider newsDataProvider;

  private final NavbarAdmin navbarAdmin;
  private Grid<Source> grid;
  private HorizontalLayout modificationComponentDisplayed;

  public SourcesMainView(NewsDataProvider newsDataProvider,
                         CreateSourcesGridService createSourcesGridService){
    this.newsDataProvider = newsDataProvider;
    //UI initialization
    navbarAdmin = new NavbarAdmin();
    String newsId = (String) VaadinSession.getCurrent().getAttribute("newsId");
    News news = newsDataProvider.getNewsById(newsId);
    grid = createSourcesGridService.createGrid(news);
    CreateSourceComponent createSourceComponent = new
CreateSourceComponent(newsDataProvider, grid, news);
    this.modificationComponentDisplayed = createSourceComponent;
    grid.addItemClickListener(item->this.itemClickEvent(item.getItem(), createSourceComponent));
    this.displayUIComponents();
  }
```

```
  private void itemClickEvent(Source source, CreateSourceComponent createSourceComponent){
     this.modificationComponentDisplayed = new EditSourceComponent(source, newsDataProvider,
grid, this, navbarAdmin, createSourceComponent);
     this.removeAll();
     this.displayUIComponents();
  }

  private void displayUIComponents(){
     this.add(navbarAdmin, modificationComponentDisplayed, grid);
```

Odpowiada za tworzenie i wyświetlanie komponentów tworzących stronę, używa klasy EditSourceComponent, CreateSourceComponent, CreateComponentGridService

CreateSourceCoponent

```
public class CreateSourceComponent extends HorizontalLayout {
  private final NewsDataProvider newsDataProvider;
  private final Grid<Source> grid;
  private final News news;
  //UI components
  private Label createLabel;
  private TextField nameField;
  private TextField referenceField;
  private Button createButton;

  public CreateSourceComponent(NewsDataProvider newsDataProvider, Grid<Source> source,
News news){
     this.newsDataProvider = newsDataProvider;
     this.grid = source;
     this.news = news;
     //UI initialization
     this.createLabel = new Label("Add new source");
     this.nameField = new TextField("name of the source");
     this.referenceField = new TextField("reference");
     this.createButton = new Button("Create", event->{
        this.addSource();
     });
     this.add(createLabel , nameField, referenceField, createButton);
     this.setWidth("100%");
     this.setPadding(true);
     this.setAlignItems(Alignment.BASELINE);
  }
```

```
  public void addSource(){
     boolean created = newsDataProvider.createSource(nameField.getValue(),
referenceField.getValue(), news.getId());
     if(!created) Notification.show("Error!!")
           ;
     nameField.setValue("");
     referenceField.setValue("");
     grid.setItems(newsDataProvider.getSourcesByNews(news));
  }
}
```

Odpowiada wyświetlenie komponentu odpowiadającego za utworzenie nowego obiektu source związanego z wcześniej wybranym obiektem news. żeby stworzyć nowy obiekt w bazie danych serwera używa klasę NewsDataProwider.

EditSourceComponent

```
public class EditSourceComponent extends HorizontalLayout {
  private final Source source;
  private final NewsDataProvider newsDataProvider;
  private final Grid<Source> grid;
  private final SourcesMainView mainView;
  private final NavbarAdmin navbarAdmin;
  private final CreateSourceComponent createSourceComponent;

  private Label editLabel;
  private TextField nameField;
  private TextField referenceField;
  private Button submitButton;

  public EditSourceComponent(Source source,
                     NewsDataProvider newsDataProvider,
                     Grid<Source> grid,
                     SourcesMainView mainView,
                     NavbarAdmin navbarAdmin,
                     CreateSourceComponent createSourceComponent){
     this.source = source;
     this.newsDataProvider = newsDataProvider;
     this.grid = grid;
     this.mainView = mainView;
     this.navbarAdmin = navbarAdmin;
     this.createSourceComponent = createSourceComponent;
     //UI initialization
     editLabel = new Label("Edit source");
     nameField = new TextField("source name");
     referenceField = new TextField("reference");
     submitButton = new Button("Edit", event->{
```

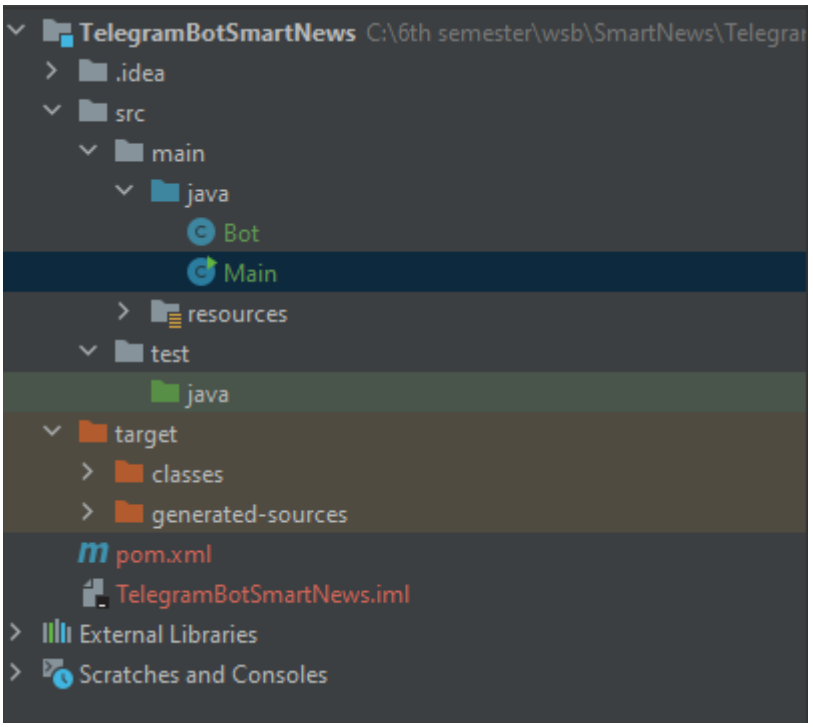
```
        this.editSource();
      });
      this.add(editLabel,nameField, referenceField, submitButton);
      this.setWidth("100%");
      this.setPadding(true);
      this.setAlignItems(Alignment.BASELINE);
  }

  public void editSource(){
      boolean edited = newsDataProvider.editSource(source.getId(), nameField.getValue(),
referenceField.getValue());
  }
```

Komponent odpowiada za element edytujący obiekt source po kliknięciu na niego.

CreateSourceGridService

```
@Service
public class CreateSourcesGridService {
  private final NewsDataProvider newsDataProvider;

  private Grid<Source> grid;

  public CreateSourcesGridService(NewsDataProvider newsDataProvider) {
      this.newsDataProvider = newsDataProvider;
  }

  public Grid<Source> createGrid(News news){
      this.grid = new Grid<>();
      grid.setItems(newsDataProvider.getSourcesByNews(news));
      grid.addColumn(Source::getName).setHeader("Name");
      grid.addColumn(Source::getReference).setHeader("Reference");
      grid.addColumn(Source::getLikes).setHeader("Likes");
      grid.addComponentColumn(this::createDeleteButton).setHeader("Delete");
      return grid;
  }

  private Button createDeleteButton(Source source){
      return new Button("Delete", event->{
        newsDataProvider.deleteSource(source);
      });
  }
```

Chatbot



Program składa się z klasy Bot która reprezentuje bota w API dostarczanym przez telegram.

API podłączone za pomocą dependency

```
<dependencies>
  <dependency>
     <groupId>org.telegram</groupId>
     <artifactId>telegrambots</artifactId>
     <version>4.1.2</version>
```

```
  </dependency>
</dependencies>
```

Klasa Bot:

Klasa implementuje klasę abstrakcyjną TelegramLongPollingBot i nadpisuje 3 metody tej klasy.
getBotUsername() zwraca nazwę bota
getBotToken() zwraca token bota podzielony przez api

onUpdateReceived() wywołuje się za każdym razem kiedy użytkownik wysyła wiadomość botu. Moja napisana metoda zapisuje chat Id użytkownika do bazy danych.

```
public class Bot extends TelegramLongPollingBot {
  private final Set<String> chats = new HashSet<>();
  private boolean fetched = false;

  public String getBotUsername() {
     return  "SmartNewsBot";
  }

  public String getBotToken() {
     return "1712124889:AAEswbVUfjWhItQDmt74qHQ9nevaiYO8YYE";
  }

  public void onUpdateReceived(Update update){
     if(fetched) {
        processMessage(update);
     }else{
        try {
           this.getChats();
           this.fetched = true;
           processMessage(update);
        } catch (IOException e) {
           e.printStackTrace();
        }
     }
  }

  private void getChats() throws IOException {
     Path path = Paths.get("C:\\Users\\ArtemCodeMachine\\Desktop\\users.txt");
```

```
        this.chats.addAll(Arrays.asList(Files.readAllLines(path,
Charset.defaultCharset()).get(0).split("/")));
    }

    public void processMessage(Update update){
        SendMessage message = new SendMessage();
        message.setChatId(String.valueOf(update.getMessage().getChatId()));
        Long chatId = update.getMessage().getChatId();
        if (update.hasMessage()) {
            //create and
            if (!this.chats.contains(String.valueOf(update.getMessage().getChatId()))) {
                try {
                    Path path = Paths.get("C:\\Users\\ArtemCodeMachine\\Desktop\\users.txt");
                    String s = String.valueOf(update.getMessage().getChatId());
                    Files.write(path, (s + "/").getBytes(UTF_8), StandardOpenOption.CREATE,
StandardOpenOption.APPEND);
                    message.setText("You are subscribed for updates");
                } catch (IOException ex) {
                    ex.printStackTrace();
                    message.setText("IO error");
                }
            } else {
                message.setText("You are already subscribed for news");
            }
        } else {
            message.setText("Error");
        }
        try {
            execute(message);
        } catch (TelegramApiException e) {
            e.printStackTrace();
        }
    }
}
```

Klasa Mian

Zawiera metodę public static system main dla uruchomienia programu.

Metoda main najpierw rejestruje bota jako aktywnego w sieci a potem sprawdza czy serwer nie zostawił czegoś do wysłania. Kiedy

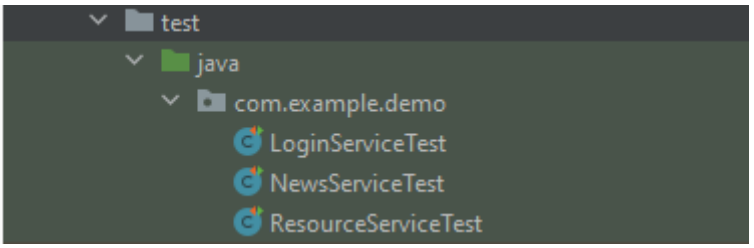
pojawia się nowość do wysłania wysłał ją do wszystkich użytkowników zapisanych w bazie danych.

```
public class Bot extends TelegramLongPollingBot {
   private final Set<String> chats = new HashSet<>();
   private boolean fetched = false;

   public String getBotUsername() {
       return  "SmartNewsBot";
   }

   public String getBotToken() {
       return "1712124889:AAEswbVUfjWhItQDmt74qHQ9nevaiYO8YYE";
   }

   public void onUpdateReceived(Update update){
       if(fetched) {
           processMessage(update);
       }else{
           try {
               this.getChats();
               this.fetched = true;
               processMessage(update);
           } catch (IOException e) {
               e.printStackTrace();
           }
       }
   }

   private void getChats() throws IOException {
       Path path =
Paths.get("C:\\Users\\ArtemCodeMachine\\Desktop\\users.txt");
       this.chats.addAll(Arrays.asList(Files.readAllLines(path,
Charset.defaultCharset()).get(0).split("/")));
   }

   public void processMessage(Update update){
       SendMessage message = new SendMessage();
       message.setChatId(String.valueOf(update.getMessage().getChatId()));
       Long chatId = update.getMessage().getChatId();
       if (update.hasMessage()) {
           //create and
           if
(!this.chats.contains(String.valueOf(update.getMessage().getChatId()))) {
               try {
                   Path path =
Paths.get("C:\\Users\\ArtemCodeMachine\\Desktop\\users.txt");
                   String s =
String.valueOf(update.getMessage().getChatId());
                   Files.write(path, (s + "/").getBytes(UTF_8),
StandardOpenOption.CREATE, StandardOpenOption.APPEND);
                   message.setText("You are subscribed for updates");
```

```
            } catch (IOException ex) {
                ex.printStackTrace();
                message.setText("IO error");
            }
        } else {
            message.setText("You are already subscribed for news");
        }
    } else {
        message.setText("Error");
    }
    try {
        execute(message);
    } catch (TelegramApiException e) {
        e.printStackTrace();
    }
}
```

Testowanie aplikacji

Server aplikacja



Dla sprawdzenia działania aplikacji napisałem 3 klasy testowe.

NewsServiceTest zawiera Metody do testowania NewsService

Pszyklad :

```
@Test
public void getNewsByCategory(){
  GetNewsByCategoryRequest request = GetNewsByCategoryRequest.newBuilder()
        .setToken("token")
        .setCategory("Sport")
        .build();
  ManagedChannel channel;
  NewsServiceGrpc.NewsServiceBlockingStub stub;
  channel = ManagedChannelBuilder.forTarget("localhost:6565").usePlaintext().build();
  stub = NewsServiceGrpc.newBlockingStub(channel);
  MultipleNewsResponse response = stub.getNewsByCategory(request);

Assert.assertTrue(response.getNewsCount()==categoryRepo.findByName("Sport").getNews().size());
}
@Test(expected = StatusRuntimeException.class)
public void getNewsByCategory_CategoryNameDoNotExist(){
  GetNewsByCategoryRequest request = GetNewsByCategoryRequest.newBuilder()
        .setToken("token")
        .setCategory("Sportttt")
        .build();
  ManagedChannel channel;
  NewsServiceGrpc.NewsServiceBlockingStub stub;
  channel = ManagedChannelBuilder.forTarget("localhost:6565").usePlaintext().build();
  stub = NewsServiceGrpc.newBlockingStub(channel);
  MultipleNewsResponse response = stub.getNewsByCategory(request);
}
@Test(expected = StatusRuntimeException.class)
```

```
public void getNewsByCategory_ExistedToken_ExceptionReturned(){
  GetNewsByCategoryRequest request = GetNewsByCategoryRequest.newBuilder()
        .setToken("t")
        .setCategory("Sport")
        .build();
  ManagedChannel channel;
  NewsServiceGrpc.NewsServiceBlockingStub stub;
  channel = ManagedChannelBuilder.forTarget("localhost:6565").usePlaintext().build();
  stub = NewsServiceGrpc.newBlockingStub(channel);
  MultipleNewsResponse response = stub.getNewsByCategory(request);
}
```

Cały kod klasy można zobaczyć w gistie oraz w kodzie źródłowym.

https://gist.github.com/Aretomko/22f1d9e4d9c4675438a304301301f41a

ResourceServicTest zawiera metody testujące ResourceService.

Pszykład:

```
@Test
public void getAllResources_CorrectRequest_AllResourcesListReturned(){
  GetAllResourcesRequest request = GetAllResourcesRequest.newBuilder()
        .setToken("token")
        .build();

  ManagedChannel channel;
  channel = ManagedChannelBuilder.forTarget("localhost:6565").usePlaintext().build();
  ResourcesServiceGrpc.ResourcesServiceBlockingStub stub=
ResourcesServiceGrpc.newBlockingStub(channel);
  AllResources response = stub.getAllResources(request);
  Assert.assertEquals(response.getResourceCount(), resourceRepo.findAll().size());
}
@Test(expected = io.grpc.StatusRuntimeException.class)
public void getAllResources_TokenDoesNotExist_ExceptionReceived(){
  GetAllResourcesRequest request = GetAllResourcesRequest.newBuilder()
        .setToken("tokedfdfsdf3n")
        .build();
  ManagedChannel channel;
  channel = ManagedChannelBuilder.forTarget("localhost:6565").usePlaintext().build();
  ResourcesServiceGrpc.ResourcesServiceBlockingStub stub=
ResourcesServiceGrpc.newBlockingStub(channel);
  AllResources response = stub.getAllResources(request);
```

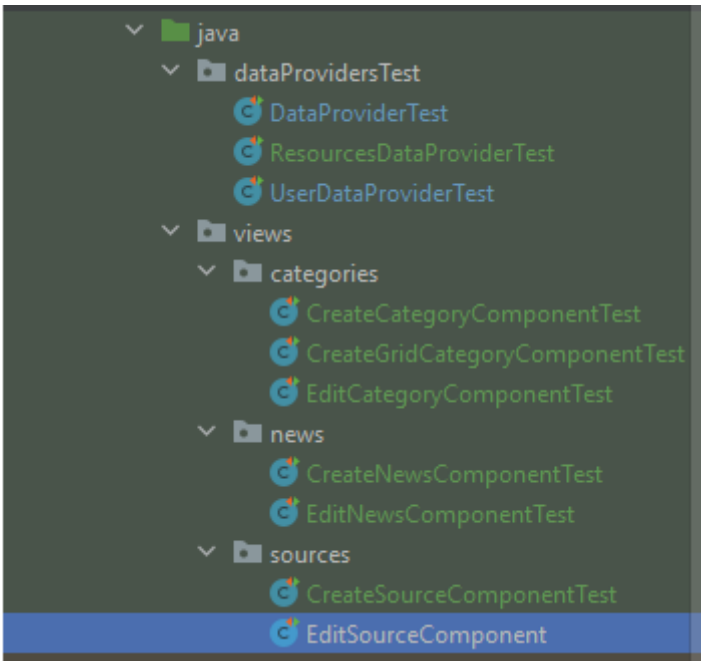
}

LoginServiceTest zawiera metody testujące metodę login w klasie LoginService

```
@RunWith(SpringRunner.class)
@SpringBootTest(classes = DemoApplication.class)
@AutoConfigureMockMvc
public class LoginServiceTest {
  @Test
  public void login_rightCredentials_UUIDReturned(){
     LoginRequest request = LoginRequest.newBuilder()
           .setUsername("admin")
           .setPassword("123")
           .build();
     ManagedChannel channel;
     LoginServiceGrpc.LoginServiceBlockingStub stub;
     channel = ManagedChannelBuilder.forTarget("localhost:6565").usePlaintext().build();
     stub = LoginServiceGrpc.newBlockingStub(channel);

     LoginResponse response = stub.login(request);
     Assert.assertTrue(!response.getToken().equals("denied"));
  }
  @Test
  public void login_UsernameDoNotExist_UUIDReturned(){
     LoginRequest request = LoginRequest.newBuilder()
           .setUsername("a2342342342423")
           .setPassword("123")
           .build();
     ManagedChannel channel;
     LoginServiceGrpc.LoginServiceBlockingStub stub;
     channel = ManagedChannelBuilder.forTarget("localhost:6565").usePlaintext().build();
     stub = LoginServiceGrpc.newBlockingStub(channel);

     LoginResponse response = stub.login(request);
     Assert.assertTrue(response.getToken().equals("denied"));
  }
  @Test
  public void login_WrongPassword_UUIDReturned(){
     LoginRequest request = LoginRequest.newBuilder()
           .setUsername("admin")
           .setPassword("12323342342342424")
           .build();
     ManagedChannel channel;
     LoginServiceGrpc.LoginServiceBlockingStub stub;
     channel = ManagedChannelBuilder.forTarget("localhost:6565").usePlaintext().build();
     stub = LoginServiceGrpc.newBlockingStub(channel);

     LoginResponse response = stub.login(request);
     Assert.assertTrue(response.getToken().equals("denied"));
```

```
  }
}
```

Web klient.



folder dataProvwidersTest zawiera testy dla NewsDataProvider, ResourcesDataProvider, UserDataProvider

```
@RunWith(SpringRunner.class)
@SpringBootTest(classes = App.class)
@AutoConfigureMockMvc
public class DataProviderTest {
  @Autowired
  private NewsDataProvider newsDataProvider;

  @Test
  public void getNewsByCategory_CorrectRequest_NewsSetProvided(){
     List<News> news = newsDataProvider.getNewsByCategory("Sport");
     Assert.assertTrue(news.size()>0);
  }
```

```
    @Test
    public void getNewsById_CorrectRequest_NewsReceived(){
        News news = newsDataProvider.getNewsById("2");
        Assert.assertTrue(news !=null);
    }

    @Test
    public void getAllNews_CorrectRequest_NewsReceived(){
        List<News> news = newsDataProvider.getAllNews();
        Assert.assertTrue(news.size()>0);
    }
    @Test
    public void getSourcesByNews_CorrectRequest_SourcesReceived(){
        List<Source> sources =
newsDataProvider.getSourcesByNews(newsDataProvider.getNewsById("2"));
        Assert.assertTrue(sources.size()>0);
    }
    @Test
    public void getGetAllCategorized_CorrectRequest_CategoriesListReceived(){
        List<Category> categories = newsDataProvider.getAllCategories();
        Assert.assertTrue(categories.size()>0);
    }
    @Test
    public void CreateCategory_CorrectRequest_NewCategoryCreated(){
        boolean created = newsDataProvider.createCategory("New Category");
        Assert.assertTrue(created);
    }
    @Test
    public void CreateSource_CorrectRequest_NwSourceCreated(){
        boolean created = newsDataProvider.createSource("New source", "reference", "2");
        Assert.assertTrue(created);
    }
    @Test
    public void CreateNews_CorrectRequest_NewsCreated(){
        boolean created = newsDataProvider.createNews("heaidng", "Sport");
        Assert.assertTrue(created);
    }
    @Test
    public void EditCategory_CorrectRequest_CategoryEdited(){
         boolean edited = newsDataProvider.editCategory("1", "Sport");
         Assert.assertTrue(edited);
    }
    @Test
    public void EditSource_CorrectRequest_CategoryEdited(){
        boolean edited = newsDataProvider.editSource("4", "editedNme", "reference");
        Assert.assertTrue(edited);
    }
    @Test
    public void EditNews_CorrectRequest_CategoryEdited(){
        boolean edited = newsDataProvider.editNews("2", "heading", "Sport");
        Assert.assertTrue(edited);
```

```
  }
```

ResourcesDataProvider:

```
@RunWith(SpringRunner.class)
@SpringBootTest(classes = App.class)
@AutoConfigureMockMvc
public class ResourcesDataProviderTest {
  @Autowired
  private ResourcesDataProvider resourcesDataProvider;

  @Test
  public void getAllResourcesTest(){
     List<Resource> resources = resourcesDataProvider.getAllResources();
     Assert.assertTrue(resources.size()>0);
  }
  @Test
  public void createResource(){
     boolean created = resourcesDataProvider.createResource("name", "reference");
     Assert.assertTrue(created);
  }
  @Test
  public void editResource(){
     boolean edited = resourcesDataProvider.editResource("26", "new name", "new reference");
     Assert.assertTrue(edited);
  }
}
```

UserDataProviderTest:

```
@RunWith(SpringRunner.class)
@SpringBootTest(classes = App.class)
@AutoConfigureMockMvc
public class UserDataProviderTest {
  @Autowired
  private UserDataProvider userDataProvider;

  @Test
  public void login() {
     String token = userDataProvider.authenticate("admin", "123");
     Assert.assertTrue(!token.equals("denied"));
  }
}
```

Folder views.categories zawiera testy dla klas znajdujących się w folderze admin.categories

CreateCategoryComponentTest

```
@RunWith(SpringRunner.class)
@SpringBootTest(classes = App.class)
@AutoConfigureMockMvc
public class CreateCategoryComponentTest {
  @Autowired
  private NewsDataProvider newsDataProvider;
  @Test
  public void  createNewCategoryTest_CorrectDataProvided_CategoryCreated() {
     CreateCategoryComponent createCategoryComponent = new CreateCategoryComponent(new
Grid<Category>(), newsDataProvider);
     int numberOfCategoriesBefore = newsDataProvider.getAllCategories().size();
     createCategoryComponent.setCategoryName(new TextField("New category name"));
     createCategoryComponent.createNewCategory();
     List<Category> categories = newsDataProvider.getAllCategories();
     categories.sort(Comparator.comparing(Category::getId).reversed());
     newsDataProvider.deleteCategory(categories.get(0));
     Assert.assertEquals(categories.size()-1, numberOfCategoriesBefore);
  }
  @Test
  public void  createNewCategoryTest_CorrectDataProvided_CategoryWithProvidedNameCreated() {
     CreateCategoryComponent createCategoryComponent = new CreateCategoryComponent(new
Grid<Category>(), newsDataProvider);
     int numberOfCategoriesBefore = newsDataProvider.getAllCategories().size();
     String categoryName = "New category name";
     createCategoryComponent.setCategoryName(new TextField(categoryName));
     createCategoryComponent.createNewCategory();
     List<Category> categories = newsDataProvider.getAllCategories();
     categories.sort(Comparator.comparing(Category::getId).reversed());
     newsDataProvider.deleteCategory(categories.get(0));
     Assert.assertEquals(categories.get(0).getName(), categoryName);
  }
  @Test
  public void  createNewCategoryTest_EmptyNameField_CategoryNotCreated() {
     CreateCategoryComponent createCategoryComponent = new CreateCategoryComponent(new
Grid<Category>(), newsDataProvider);
     int numberOfCategoriesBefore = newsDataProvider.getAllCategories().size();
     String categoryName = "New category name";
     createCategoryComponent.setCategoryName(new TextField(categoryName));
     createCategoryComponent.createNewCategory();
     List<Category> categories = newsDataProvider.getAllCategories();
```

```
        categories.sort(Comparator.comparing(Category::getId).reversed());
        newsDataProvider.deleteCategory(categories.get(0));
        Assert.assertEquals(categories.size(), numberOfCategoriesBefore);
    }
}
```

EditCategoryComponetTest:

```
@RunWith(SpringRunner.class)
@SpringBootTest(classes = App.class)
@AutoConfigureMockMvc
public class EditCategoryComponentTest {
    @Autowired
    private NewsDataProvider newsDataProvider;

    @Test
    private void editCategory(){
        Category category = newsDataProvider.getAllCategories().get(0);
        EditCategoryComponent editCategoryComponent = new EditCategoryComponent(
                category,
                newsDataProvider,
                new Grid<Category>(),
                new CategoriesMainViewAdmin(newsDataProvider, new
CreateCategoriesGridService(newsDataProvider)),
                new NavbarAdmin(),
                new CreateCategoryComponent(new Grid<Category>(), newsDataProvider)
        );
        String name= "Edited Name";

        editCategoryComponent.setCategoryName(new TextField(name));

        editCategoryComponent.editCategory();

        Category editedCategory = newsDataProvider.getAllCategories().get(0);

        Assert.assertEquals(name , editedCategory.getName());
    }
    @Test
    private void editCategory_NameNotProvided_NameDoNotChange(){
        Category category = newsDataProvider.getAllCategories().get(0);
        EditCategoryComponent editCategoryComponent = new EditCategoryComponent(
                category,
                newsDataProvider,
                new Grid<Category>(),
                new CategoriesMainViewAdmin(newsDataProvider, new
CreateCategoriesGridService(newsDataProvider)),
                new NavbarAdmin(),
                new CreateCategoryComponent(new Grid<Category>(), newsDataProvider)
        );
```

```
        String name= "Edited Name";

        editCategoryComponent.setCategoryName(new TextField(name));

        editCategoryComponent.editCategory();

        Category editedCategory = newsDataProvider.getAllCategories().get(0);

        Assert.assertEquals(category.getName() , editedCategory.getName());
    }

}
```

folder news zawiera testy dla klas znajdujących się w folderze admin.news

CreateNewsComponentTest

```
public class CreateNewsComponentTest {
    @Autowired
    private NewsDataProvider newsDataProvider;
    @Test
    public void createNewsComponent_ValidData_NewsObjectCreated(){
        CreateNewsComponent createNewsComponent = new CreateNewsComponent(
                newsDataProvider,
                new Grid<News>()
        );
        int newsCountBeforeCreation = newsDataProvider.getAllCategories().size();
        String newHeading= "edited";
        createNewsComponent.setHeading(new TextField(newHeading));
        Select<String> categorySelect = new Select<String>();
        categorySelect.setItems(newsDataProvider.getCategoriesNames());
        categorySelect.setValue(newsDataProvider.getAllCategories().get(0).getName());
        createNewsComponent.setCategorySelect(categorySelect);
        createNewsComponent.createNews();
        Assert.assertEquals(newsCountBeforeCreation+1, newsDataProvider.getAllCategories().size());
    }
    @Test
    public void createNewsComponent_ValidData_NewsWithRightHeadingCreated(){
        CreateNewsComponent createNewsComponent = new CreateNewsComponent(
                newsDataProvider,
                new Grid<News>()
        );
        int newsCountBeforeCreation = newsDataProvider.getAllCategories().size();
        String newHeading= "edited";
        createNewsComponent.setHeading(new TextField(newHeading));
        Select<String> categorySelect = new Select<String>();
        categorySelect.setItems(newsDataProvider.getCategoriesNames());
        categorySelect.setValue(newsDataProvider.getAllCategories().get(0).getName());
```

```
        createNewsComponent.setCategorySelect(categorySelect);
        createNewsComponent.createNews();
        List<News> news = newsDataProvider.getAllNews();
        news.sort(Comparator.comparing(News::getId).reversed());
        //getting lastly created object
        Assert.assertEquals(news.get(0).getHeading(), newHeading);
    }
    @Test
    public void createNewsComponent_ValidData_NewsWithRightCategoryCreated(){
        CreateNewsComponent createNewsComponent = new CreateNewsComponent(
                newsDataProvider,
                new Grid<News>()
        );
        int newsCountBeforeCreation = newsDataProvider.getAllCategories().size();
        String newHeading= "edited";
        createNewsComponent.setHeading(new TextField(newHeading));
        Select<String> categorySelect = new Select<String>();
        Category category = newsDataProvider.getAllCategories().get(0);
        categorySelect.setItems(newsDataProvider.getCategoriesNames());
        categorySelect.setValue(category.getName());
        createNewsComponent.setCategorySelect(categorySelect);
        createNewsComponent.createNews();
        List<News> news = newsDataProvider.getAllNews();
        news.sort(Comparator.comparing(News::getId).reversed());
        //getting lastly created object
        Assert.assertEquals(news.get(0).getCategory().getName(), category.getName());
    }
    @Test
    public void createNewsComponent_EmptyHeading_NewsShouldNotBeCreated(){
        CreateNewsComponent createNewsComponent = new CreateNewsComponent(
                newsDataProvider,
                new Grid<News>()
        );
        int newsCountBeforeCreation = newsDataProvider.getAllCategories().size();
        String newHeading= "";
        createNewsComponent.setHeading(new TextField(newHeading));
        Select<String> categorySelect = new Select<String>();
        categorySelect.setItems(newsDataProvider.getCategoriesNames());
        categorySelect.setValue(newsDataProvider.getAllCategories().get(0).getName());
        createNewsComponent.setCategorySelect(categorySelect);
        createNewsComponent.createNews();
        Assert.assertEquals(newsCountBeforeCreation, newsDataProvider.getAllCategories().size());
    }

}
```

folder sources zawiera testy dla klas znajdujących się w folderze main.admin.news.sources

EditSourceComponentTest

```
@RunWith(SpringRunner.class)
@SpringBootTest(classes = App.class)
@AutoConfigureMockMvc
public class EditSourceComponent {
  @Autowired
  private NewsDataProvider newsDataProvider;
  @Test
  public void editSource_validData_NameUpdated(){
     News news = newsDataProvider.getAllNews().get(0);
     Source source = newsDataProvider.getSourcesByNews(news).get(0);
     com.example.application.views.admin.news.sources.EditSourceComponent
editSourceComponent =new
com.example.application.views.admin.news.sources.EditSourceComponent(
           source,
           newsDataProvider,
           new Grid<Source>(),
           new SourcesMainView(
                newsDataProvider,
                new CreateSourcesGridService(newsDataProvider)
                ),
           new NavbarAdmin(),
           new CreateSourceComponent(newsDataProvider, new Grid<Source>(), news)
           );
     String editedName = "edited name";
     editSourceComponent.setNameField(new TextField(editedName));
     String editedReference= "EditedReference";
     editSourceComponent.setReferenceField(new TextField(editedReference));
     editSourceComponent.editSource();
     Source sourceAfterEdit = newsDataProvider.getSourcesByNews(news).get(0);
     Assert.assertEquals(sourceAfterEdit.getName(), editedName);
  }
  @Test
  public void editSource_validData_ReferenceUpdated(){
     News news = newsDataProvider.getAllNews().get(0);
     Source source = newsDataProvider.getSourcesByNews(news).get(0);
     com.example.application.views.admin.news.sources.EditSourceComponent
editSourceComponent =new
com.example.application.views.admin.news.sources.EditSourceComponent(
           source,
           newsDataProvider,
           new Grid<Source>(),
           new SourcesMainView(
                newsDataProvider,
                new CreateSourcesGridService(newsDataProvider)
           ),
           new NavbarAdmin(),
           new CreateSourceComponent(newsDataProvider, new Grid<Source>(), news)
     );
     String editedName = "edited name";
     editSourceComponent.setNameField(new TextField(editedName));
     String editedReference= "EditedReference";
```

```
        editSourceComponent.setReferenceField(new TextField(editedReference));
        editSourceComponent.editSource();
        Source sourceAfterEdit = newsDataProvider.getSourcesByNews(news).get(0);
        Assert.assertEquals(sourceAfterEdit.getReference(), editedReference);
    }
}
```

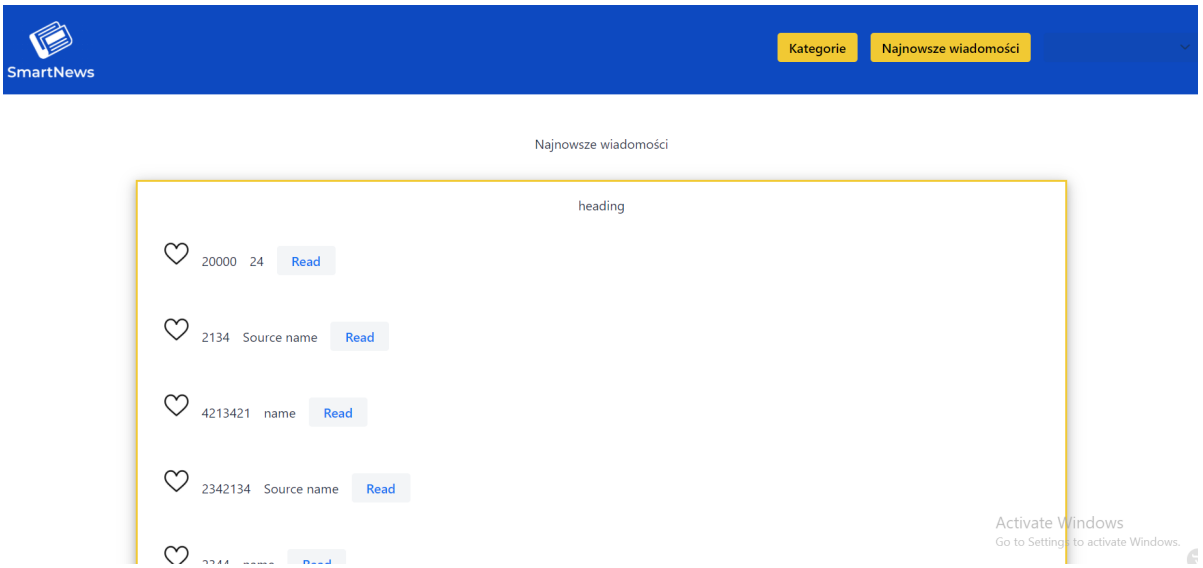
CreateSourceComponentTest

```
@RunWith(SpringRunner.class)
@SpringBootTest(classes = App.class)
@AutoConfigureMockMvc
public class CreateSourceComponentTest {
    @Autowired
    private NewsDataProvider newsDataProvider;
    @Test
    public void addSource_ValidData_SourceCreated(){
        News news = newsDataProvider.getAllNews().get(0);
        CreateSourceComponent createSourceComponent = new CreateSourceComponent(
                newsDataProvider,
                new Grid<Source>(),
                news
        );
        int numberOfSourcesBefore = newsDataProvider.getSourcesByNews(news).size();
        String name = "new name";
        createSourceComponent.setNameField(new TextField(name));
        String reference = "new reference";
        createSourceComponent.setReferenceField(new TextField(reference));
        createSourceComponent.addSource();
        Assert.assertEquals(numberOfSourcesBefore+1,
newsDataProvider.getSourcesByNews(news).size());
    }
    @Test
    public void addSource_ValidData_SourceWithRightNameCreated(){
        News news = newsDataProvider.getAllNews().get(0);
        CreateSourceComponent createSourceComponent = new CreateSourceComponent(
                newsDataProvider,
                new Grid<Source>(),
                news
        );
        int numberOfSourcesBefore = newsDataProvider.getSourcesByNews(news).size();
        String name = "new name";
        createSourceComponent.setNameField(new TextField(name));
        String reference = "new reference";
        createSourceComponent.setReferenceField(new TextField(reference));
        createSourceComponent.addSource();
        List<Source> sources = newsDataProvider.getSourcesByNews(news);
        sources.sort(Comparator.comparing(Source::getId).reversed());
        Assert.assertEquals(sources.get(0).getName(), name);
```

```
    }
    @Test
    public void addSource_ValidData_SourceWithRightReferenceCreated(){
        News news = newsDataProvider.getAllNews().get(0);
        CreateSourceComponent createSourceComponent = new CreateSourceComponent(
                newsDataProvider,
                new Grid<Source>(),
                news
        );
        int numberOfSourcesBefore = newsDataProvider.getSourcesByNews(news).size();
        String name = "new name";
        createSourceComponent.setNameField(new TextField(name));
        String reference = "new reference";
        createSourceComponent.setReferenceField(new TextField(reference));
        createSourceComponent.addSource();
        List<Source> sources = newsDataProvider.getSourcesByNews(news);
        sources.sort(Comparator.comparing(Source::getId).reversed());
        Assert.assertEquals(sources.get(0).getReference(), reference);
    }
}
```

Instrukcje obsługi

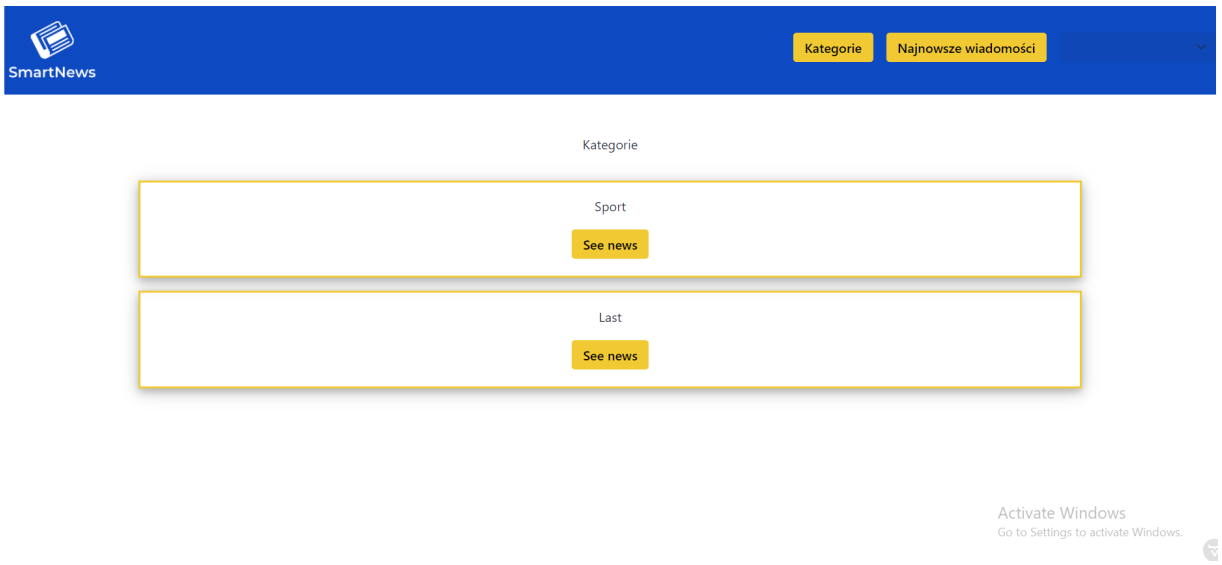
Instrukcje dla użytkowników:



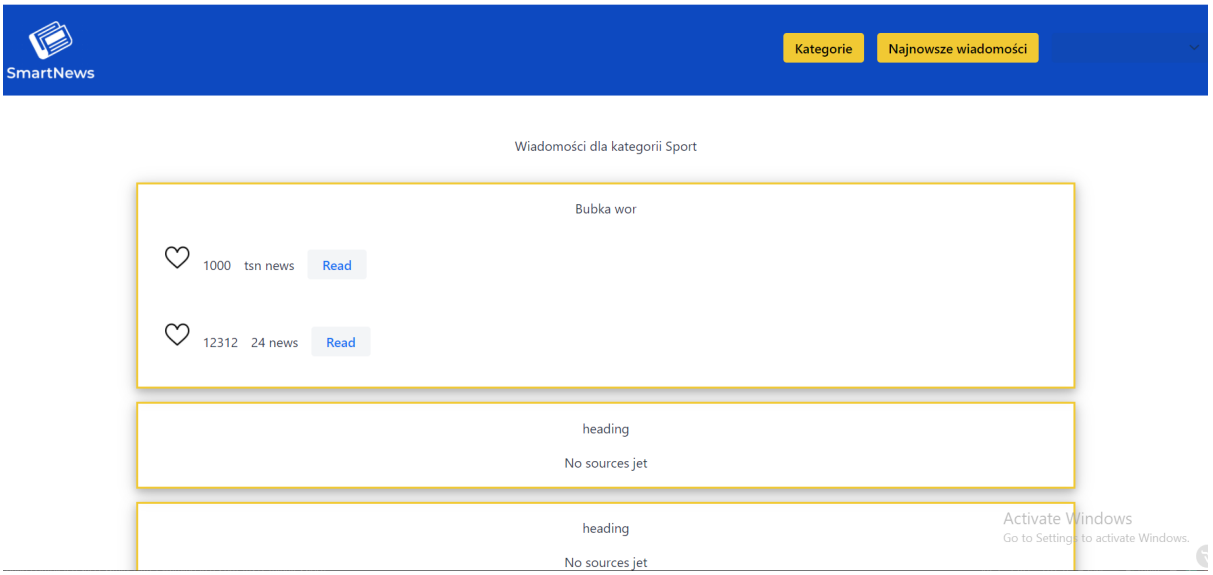
Na stronie głównej użytkownik może zobaczyć najnowsze wiadomości dodane do aplikacji.

Dla każdej nowości użytkownik widzi listę źródeł opisujących powyższy problem.

Użytkownik może przeczytać każde źródło informacji a potem zaznaczyć które jemu podobało najbardziej.



Na stronie “Kategorie” użytkownik może zobaczyć wszystkie dostępne kategorie nowości.



Po wybraniu kategorii użytkownik może zobaczyć wszystkie nowości związane z wybraną kategorią

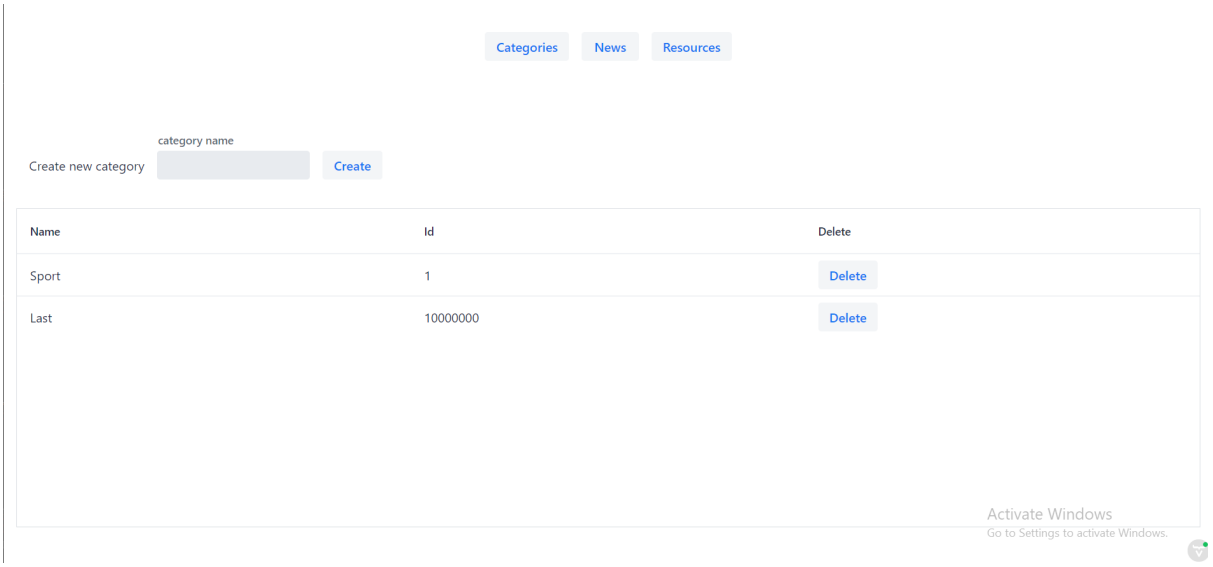
Dla każdej nowości użytkownik widzi listę źródeł opisujących powyższy problem.

Użytkownik może przeczytać każde źródło informacji a potem zaznaczyć które jemu podobało najbardziej.

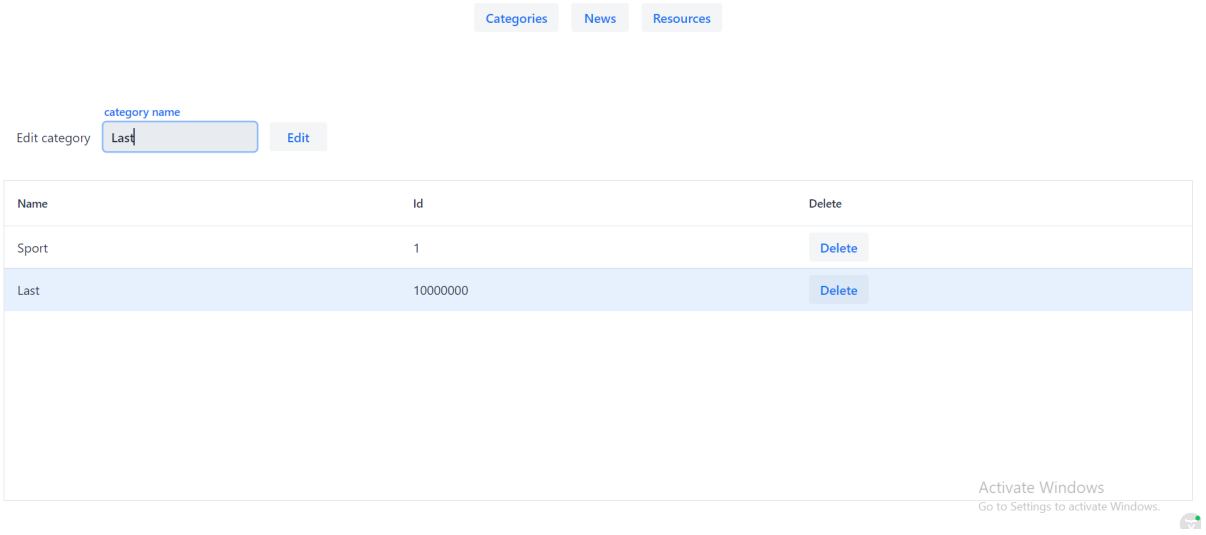
Użytkownik może zmienić język strony za pomocą wyboru języka w pasku nawigacyjnym.

Instrukcje dla adminów:

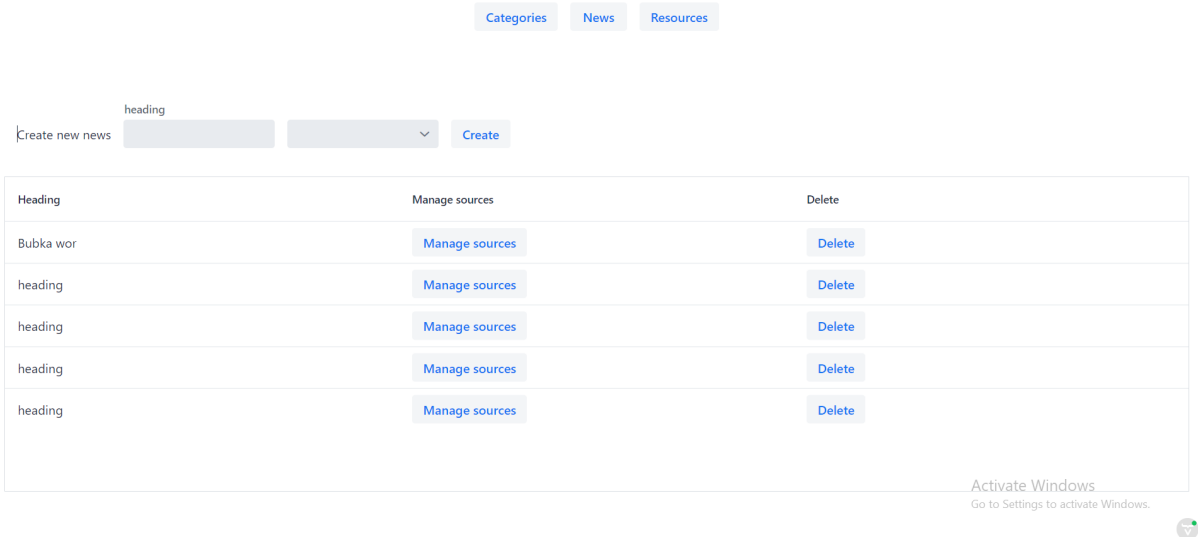
Dlatego żeby żeby używać funkcyjność dla adminów użytkownik musi się najpierw zalogować za pomocą strony login.



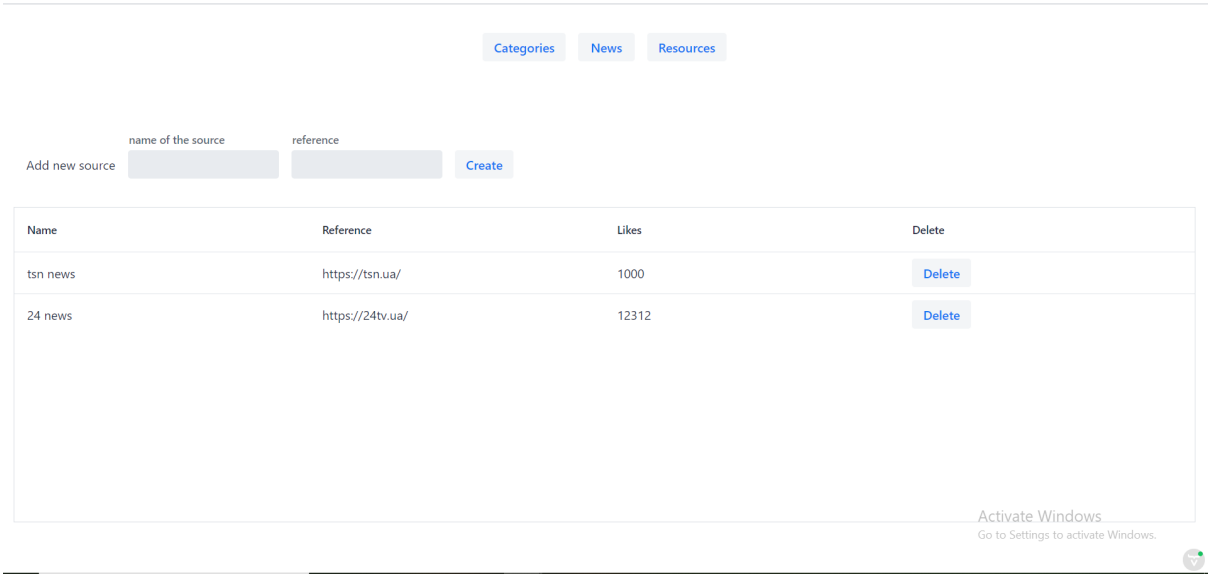
Za pomocą strony “Categories” administrator może dodawać nowe kategorie, edytować istniejące kategorie klikając na nich i usunąć kategorie za pomocą przycisku “delete”



Żeby edytować kategorie trzeba kliknąć na potrzebną kategorie w liście.

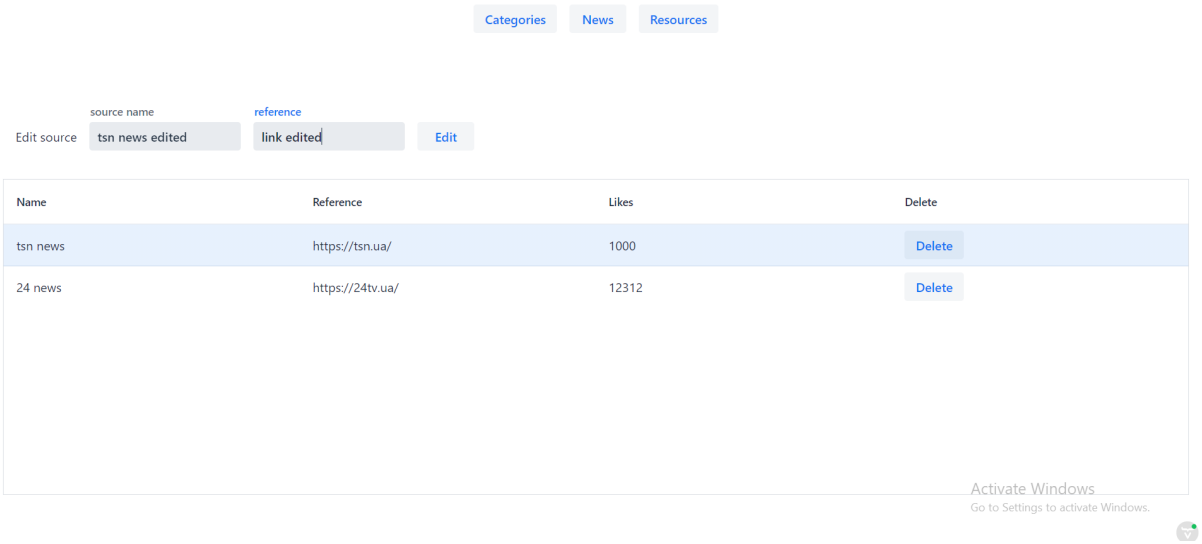


Za pomocą strony “News” administrator może dodawać nowe nowości , edytować istniejące klikając na nich i usunąć nowość za pomocą przycisku “delete”.



Za pomocą strony sources możemy dodawać źródła do utworzonej wcześniej nowości. Możemy również edytować i usunąć źródła.

Edytować elementy można kliknięciem na nich



Opis przebiegu projektu i jego realizacji

Opis napotkanych problemów i sposobu ich rozwiązania

Największym problemem, z jakim się spotkałem, jest to, że aplikacje powinny reagować i szybko reagować na interakcje użytkowników. Zwykłe aplikacje wiosenne z wyrenderowanymi dokumentami html, które pisałem wcześniej, nie zapewniają wymaganej szybkości reakcji.

Najlepszym sposobem rozwiązania tego problemu było użycie części frameworka frontendowego. Jednak nauka całej nowej frameworku jest bardzo czasochłonna. Aby szybciej rozwiązać ten problem, zdecydowałem się użyć frameworka Vaadin. Pozwala nam tworzyć świetne responsywne aplikacje frontendowe w połączeniu z back-endem przy użyciu tylko języka Java.

To rozwiązanie było świetne, mogłem szybko stworzyć połączoną aplikację front-end i back-end używając tylko javy - języka, który znam. Byłem znacznie szybszy niż nauka nowego odrębnego języka i frameworka.