Refaktoryzacja SnapRead do Clean Architecture

Nowa struktura pakietów:

```
com.snapread.dev/
├── domain/                         # Warstwa Domain (centrum cebuli)
│   ├── entities/
│   │   ├── User.java
│   │   ├── Car.java
│   │   ├── Invoice.java
│   │   ├── VehicleInspection.java
│   │   ├── InsuranceAC.java
│   │   └── InsuranceOC.java
│   ├── valueobjects/
│   │   ├── Email.java
│   │   ├── Password.java
│   │   ├── Money.java
│   │   ├── VIN.java
│   │   └── LicensePlate.java
│   ├── enums/
│   │   ├── UserRole.java
│   │   ├── Fuel.java
│   │   └── InvoiceStatus.java
│   └── exceptions/
│       ├── DomainException.java
│       ├── UserNotFoundException.java
│       └── InvalidCarDataException.java
│
├── application/                    # Warstwa Application Core
│   ├── interfaces/                 # Interfejsy (porty)
│   │   ├── repositories/
│   │   │   ├── UserRepository.java
│   │   │   ├── CarRepository.java
│   │   │   └── InvoiceRepository.java
│   │   ├── services/
│   │   │   ├── EmailService.java
│   │   │   ├── PdfGeneratorService.java
│   │   │   └── OcrService.java
│   │   └── security/
│   │       └── AuthenticationService.java
│   ├── usecases/                   # Use Cases (Command/Query Handlers)
│   │   ├── user/
│   │   │   ├── RegisterUserUseCase.java
│   │   │   ├── LoginUserUseCase.java
│   │   │   └── GetUserProfileUseCase.java
│   │   ├── car/
│   │   │   ├── AddCarUseCase.java
│   │   │   ├── UpdateCarServiceUseCase.java
│   │   │   └── GetUserCarsUseCase.java
│   │   └── invoice/
│   │       ├── CreateInvoiceUseCase.java
│   │       ├── ProcessInvoiceOcrUseCase.java
│   │       └── GenerateChartDataUseCase.java
│   ├── dto/                         # Data Transfer Objects
│   │   ├── UserRegistrationDTO.java
│   │   ├── UserLoginDTO.java
│   │   ├── InvoiceDTO.java
│   │   └── ChartDTO.java
│   └── validation/
│       ├── UserValidator.java
│       ├── CarValidator.java
│       └── InvoiceValidator.java
│
├── infrastructure/                  # Warstwa Infrastructure
│   ├── persistence/                 # Implementacje repozytoriów
│   │   ├── jpa/
│   │   │   ├── entities/           # JPA Entities (różne od domain entities)
│   │   │   │   ├── UserJpaEntity.java
│   │   │   │   ├── CarJpaEntity.java
│   │   │   │   └── InvoiceJpaEntity.java
│   │   │   ├── repositories/
│   │   │   │   ├── UserJpaRepository.java
│   │   │   │   ├── CarJpaRepository.java
│   │   │   │   └── InvoiceJpaRepository.java
│   │   │   └── adapters/           # Adaptery implementujące interfejsy z application
│   │   │       ├── UserRepositoryAdapter.java
│   │   │       ├── CarRepositoryAdapter.java
│   │   │       └── InvoiceRepositoryAdapter.java
│   │   └── mappers/                # Mapowanie między JPA a Domain
│   │       ├── UserMapper.java
│   │       ├── CarMapper.java
│   │       └── InvoiceMapper.java
│   ├── external/                   # Zewnętrzne serwisy
│   │   ├── email/
│   │   │   └── SmtpEmailService.java
│   │   ├── ocr/
│   │   │   ├── PythonOcrService.java
│   │   │   └── JsonProcessingService.java
│   │   └── pdf/
│   │       └── ITextPdfGeneratorService.java
│   ├── security/
│   │   ├── jwt/
│   │   │   ├── JwtService.java
│   │   │   └── JwtFilter.java
│   │   ├── config/
│   │   │   └── SecurityConfig.java
│   │   └── AuthenticationServiceImpl.java
│   └── config/
│       ├── DatabaseConfig.java
│       └── ApplicationConfig.java
│
└── presentation/                    # Warstwa Presentation (UI/API)
    ├── rest/                      # REST Controllers
    │   ├── auth/
    │   │   ├── AuthController.java
    │   │   └── request/
    │   │       └── LoginRequest.java
    │   ├── user/
    │   │   └── UserController.java
    │   ├── car/
    │   │   └── CarController.java
    │   ├── invoice/
    │   │   ├── InvoiceController.java
    │   │   └── ChartController.java
    │   └── admin/
    │       ├── AdminUserController.java
    │       ├── AdminCarController.java
    │       └── AdminInvoiceController.java
    ├── config/
    │   └── WebConfig.java
    └── exception/
        └── GlobalExceptionHandler.java
```

Krok po kroku refaktoryzacji:

Krok 1: Stwórz warstwę Domain

Przed (obecna struktura):

java

```java
// com.snapread.dev.auth.model.User
public class User {
    private String email;
    private String password;
    // gettery/settery bez logiki
}
```

Po (Clean Architecture):

```java
// com.snapread.dev.domain.entities.User
public class User {
    private final UserId id;
    private final Email email;
    private final Password password;
    private final UserRole role;

    public User(Email email, Password password, UserRole role) {
        this.email = email;
        this.password = password;
        this.role = role;
        this.id = UserId.generate();
    }

    public boolean canAccessAdminPanel() {
        return this.role == UserRole.ADMIN;
    }

    public void changePassword(Password oldPassword, Password newPassword) {
        if (!this.password.matches(oldPassword)) {
            throw new InvalidPasswordException("Old password doesn't match");
        }
        this.password = newPassword;
    }
}

// com.snapread.dev.domain.valueobjects.Email
public class Email {
    private final String value;

    public Email(String email) {
        if (!isValidEmail(email)) {
            throw new InvalidEmailException("Invalid email format");
        }
        this.value = email;
    }

    private boolean isValidEmail(String email) {
        // walidacja email
        return email.matches("^[A-Za-z0-9+_.-]+@[A-Za-z0-9.-]+\\.[A-Za-z]{2,}$");
    }
}
```

Krok 2: Stwórz Application Layer

Interfejsy (Porty):

```java
// com.snapread.dev.application.interfaces.repositories.UserRepository
public interface UserRepository {
    User findById(UserId id);
    User findByEmail(Email email);
    void save(User user);
    boolean existsByEmail(Email email);
}

// com.snapread.dev.application.interfaces.services.EmailService
public interface EmailService {
    void sendWelcomeEmail(Email to, String userName);
    void sendPasswordResetEmail(Email to, String resetToken);
}
```

Use Cases:

java

```java
// com.snapread.dev.application.usecases.user.RegisterUserUseCase
@Component
public class RegisterUserUseCase {
    private final UserRepository userRepository;
    private final EmailService emailService;
    private final PasswordService passwordService;

    public RegisterUserUseCase(UserRepository userRepository,
                               EmailService emailService,
                               PasswordService passwordService) {
        this.userRepository = userRepository;
        this.emailService = emailService;
        this.passwordService = passwordService;
    }

    public void execute(UserRegistrationDTO dto) {
        Email email = new Email(dto.getEmail());

        if (userRepository.existsByEmail(email)) {
            throw new UserAlreadyExistsException("User with this email already exists");
        }

        Password hashedPassword = passwordService.hash(new Password(dto.getPassword()));
        User user = new User(email, hashedPassword, UserRole.USER);

        userRepository.save(user);
        emailService.sendWelcomeEmail(email, dto.getName());
    }
}
```

Krok 3: Implementuj Infrastructure

Adapter dla Repository:

java

```java
// com.snapread.dev.infrastructure.persistence.jpa.adapters.UserRepositoryAdapter
@Repository
public class UserRepositoryAdapter implements UserRepository {
    private final UserJpaRepository jpaRepository;
    private final UserMapper mapper;

    public UserRepositoryAdapter(UserJpaRepository jpaRepository, UserMapper mapper) {
        this.jpaRepository = jpaRepository;
        this.mapper = mapper;
    }

    @Override
    public User findById(UserId id) {
        UserJpaEntity entity = jpaRepository.findById(id.getValue())
            .orElseThrow(() -> new UserNotFoundException("User not found"));
        return mapper.toDomain(entity);
    }

    @Override
    public void save(User user) {
        UserJpaEntity entity = mapper.toJpaEntity(user);
        jpaRepository.save(entity);
    }
}

// com.snapread.dev.infrastructure.persistence.jpa.entities.UserJpaEntity
@Entity
@Table(name = "users")
public class UserJpaEntity {
    @Id
    private String id;
    private String email;
    private String password;
    private String role;
    // JPA annotations i gettery/settery
}
```

Mapper:

```java
// com.snapread.dev.infrastructure.persistence.mappers.UserMapper
@Component
public class UserMapper {
    public User toDomain(UserJpaEntity entity) {
        return new User(
            new UserId(entity.getId()),
            new Email(entity.getEmail()),
            new Password(entity.getPassword()),
            UserRole.valueOf(entity.getRole())
        );
    }

    public UserJpaEntity toJpaEntity(User user) {
        UserJpaEntity entity = new UserJpaEntity();
        entity.setId(user.getId().getValue());
        entity.setEmail(user.getEmail().getValue());
        entity.setPassword(user.getPassword().getValue());
        entity.setRole(user.getRole().name());
        return entity;
    }
}
```

Krok 4: Refaktoryzuj Presentation Layer

Kontroler:

java

```java
// com.snapread.dev.presentation.rest.auth.AuthController
@RestController
@RequestMapping("/api/auth")
public class AuthController {
    private final RegisterUserUseCase registerUserUseCase;
    private final LoginUserUseCase loginUserUseCase;

    public AuthController(RegisterUserUseCase registerUserUseCase,
                         LoginUserUseCase loginUserUseCase) {
        this.registerUserUseCase = registerUserUseCase;
        this.loginUserUseCase = loginUserUseCase;
    }

    @PostMapping("/register")
    public ResponseEntity<String> register(@RequestBody @Valid UserRegistrationDTO dto) {
        registerUserUseCase.execute(dto);
        return ResponseEntity.ok("User registered successfully");
    }

    @PostMapping("/login")
    public ResponseEntity<LoginResponse> login(@RequestBody @Valid LoginRequest request) {
        LoginResponse response = loginUserUseCase.execute(request);
        return ResponseEntity.ok(response);
    }
}
```

Krok 5: Konfiguracja Dependency Injection

java

```java
// com.snapread.dev.infrastructure.config.ApplicationConfig
@Configuration
public class ApplicationConfig {

    @Bean
    public UserRepository userRepository(UserJpaRepository jpaRepository, UserMapper mapper) {
        return new UserRepositoryAdapter(jpaRepository, mapper);
    }

    @Bean
    public EmailService emailService() {
        return new SmtpEmailService();
    }

    @Bean
    public PasswordService passwordService() {
        return new BCryptPasswordService();
    }
}
```

Główne zmiany:

1. **Separacja warstw** - każda warstwa ma własny pakiet
2. **Kierunek zależności** - wszystkie zależności wskazują do środka (Domain)
3. **Logika biznesowa w Domain** - User.canAccessAdminPanel(), Email validation
4. **Interfejsy w Application** - Infrastructure implementuje, nie definiuje
5. **Mapowanie między warstwami** - JPA Entity ≠ Domain Entity
6. **Use Cases zamiast serwisów** - jasno określone przypadki użycia

Korzyści po refaktoryzacji:

- **Testowalność** - Use Cases można testować bez bazy danych
- **Niezależność** - można zmienić bazę danych bez wpływu na logikę
- **Czytelność** - jasny podział odpowiedzialności
- **Rozwój** - łatwe dodawanie nowych funkcji
- **Maintenance** - izolowane zmiany w poszczególnych warstwach