**MANUAL BOOK**

**BUKU MANUAL**



**CREATE BY : RKS-308 TEAM**

**DIBUAT OLEH : TIM RKS-308**

**2025**

**A. Informasi Umum *Website***

“Website Borang RKS-308” merupakan platform digital yang memfasilitasi proses peminjaman ruangan dan barang di lingkungan sekolah secara efisien dan terstruktur. Dirancang dengan dua peran utama, yaitu admin (guru) dan pengguna (siswa), website ini memungkinkan guru untuk mengelola inventaris melalui operasi CRUD (Create, Read, Update, Delete) serta meninjau dan memutuskan permohonan peminjaman dari siswa. Sementara itu, siswa dapat dengan mudah melihat daftar ruangan dan barang yang tersedia, mengajukan permohonan peminjaman, serta memantau status permohonan mereka.

Fitur utama meliputi sistem autentikasi yang aman, kalender ketersediaan untuk menghindari konflik peminjaman, notifikasi otomatis mengenai status permohonan, dan dashboard yang intuitif untuk memantau aktivitas peminjaman. Dengan antarmuka yang user-friendly dan keamanan data yang terjamin, “Website Borang RKS-308” bertujuan meningkatkan efisiensi operasional, transparansi, dan kemudahan akses dalam pengelolaan sumber daya sekolah.

**B. User Stories atau FR/NFR**

Analisis kebutuhan merupakan langkah krusial dalam pengembangan perangkat lunak karena kesalahan dalam menggambarkan kebutuhan sistem dapat menyebabkan kegagalan dalam proses pengembangan. Kebutuhan sistem dibagi menjadi dua kategori utama, yaitu kebutuhan fungsional (FR) dan kebutuhan non-fungsional (NFR).

Website peminjaman ruangan dan barang ini dirancang untuk dioperasikan oleh tiga jenis pengguna utama, yaitu **Admin (Superuser)**, **Admin (Guru)**, dan **User (Siswa)**. **Admin Superuser** memiliki hak akses penuh termasuk mengelola pengguna, **Admin Guru** bertanggung jawab dalam mengelola inventaris dan memproses permohonan peminjaman, serta **User Siswa** dapat melihat daftar ruangan dan mengajukan permohonan peminjaman.

Berikut ini adalah deskripsi kebutuhan umum dari perspektif masing-masing pengguna, serta definisi yang menggunakan **Functional Requirements (FR)** dan **Non-Functional Requirements (NFR)** yang dikelompokkan berdasarkan jenis pengguna/aktor.

Tentu! Berikut adalah **User Stories**, **Functional Requirements (FR)**, dan **Non-Functional Requirements (NFR)** untuk website peminjaman ruangan dan barang di lingkungan sekolah Anda dengan tiga peran pengguna: **Admin Superuser**, **Admin Guru**, dan **User Siswa**. Kebutuhan-kebutuhan ini disajikan dalam bentuk tabel untuk memudahkan pemahaman dan pengelolaan.

#### **Tabel 1. Kebutuhan Fungsional (Admin Superuser)**

| **ID** | **Deskripsi** |
| --- | --- |
| KFA-01 | Admin Superuser dapat melakukan login menggunakan akun superuser. |
| KFA-02 | Admin Superuser dapat menambah akun pengguna (Admin Guru dan User Siswa) dengan mengisi form yang diperlukan. |
| KFA-03 | Admin Superuser dapat mengedit informasi akun pengguna yang ada. |
| KFA-04 | Admin Superuser dapat menghapus akun pengguna yang tidak diperlukan. |
| KFA-05 | Admin Superuser dapat menambahkan ruangan/barang baru dengan detail seperti nama, deskripsi, kategori, gambar, kapasitas, kondisi, dan lokasi. |
| KFA-06 | Admin Superuser dapat mengedit informasi ruangan/barang yang ada. |
| KFA-07 | Admin Superuser dapat menghapus ruangan/barang yang tidak diperlukan. |
| KFA-08 | Admin Superuser dapat melihat semua permintaan peminjaman dari siswa. |
| KFA-09 | Admin Superuser dapat menerima atau menolak permintaan peminjaman dari siswa. |
| KFA-10 | Admin Superuser dapat melihat histori peminjaman. |
| KFA-11 | Admin Superuser dapat mengirim notifikasi kepada siswa mengenai status permohonan peminjaman. |
| KFA-12 | Admin Superuser dapat melihat statistik peminjaman, permintaan terbaru, dan notifikasi di dashboard. |
| KFA-13 | Sistem mencatat aktivitas pengguna untuk audit log. |

#### **Tabel 2. Kebutuhan Fungsional (Admin Guru)**

| **ID** | **Deskripsi** |
| --- | --- |
| KFG-01 | Admin Guru dapat melakukan login menggunakan akun guru. |
| KFG-02 | Admin Guru dapat menambahkan ruangan/barang baru dengan detail seperti nama, deskripsi, kategori, gambar, kapasitas, kondisi, dan lokasi. |
| KFG-03 | Admin Guru dapat mengedit informasi ruangan/barang yang ada. |
| KFG-04 | Admin Guru dapat menghapus ruangan/barang yang tidak diperlukan. |
| KFG-05 | Admin Guru dapat melihat semua permintaan peminjaman dari siswa. |
| KFG-06 | Admin Guru dapat menerima atau menolak permintaan peminjaman dari siswa. |
| KFG-07 | Admin Guru dapat melihat histori peminjaman. |
| KFG-08 | Admin Guru dapat mengirim notifikasi kepada siswa mengenai status permohonan peminjaman. |
| KFG-09 | Admin Guru dapat melihat statistik peminjaman, permintaan terbaru, dan notifikasi di dashboard. |
| KFG-10 | Sistem mencatat aktivitas pengguna untuk audit log. |

#### **Tabel 3. Kebutuhan Fungsional (User Siswa)**

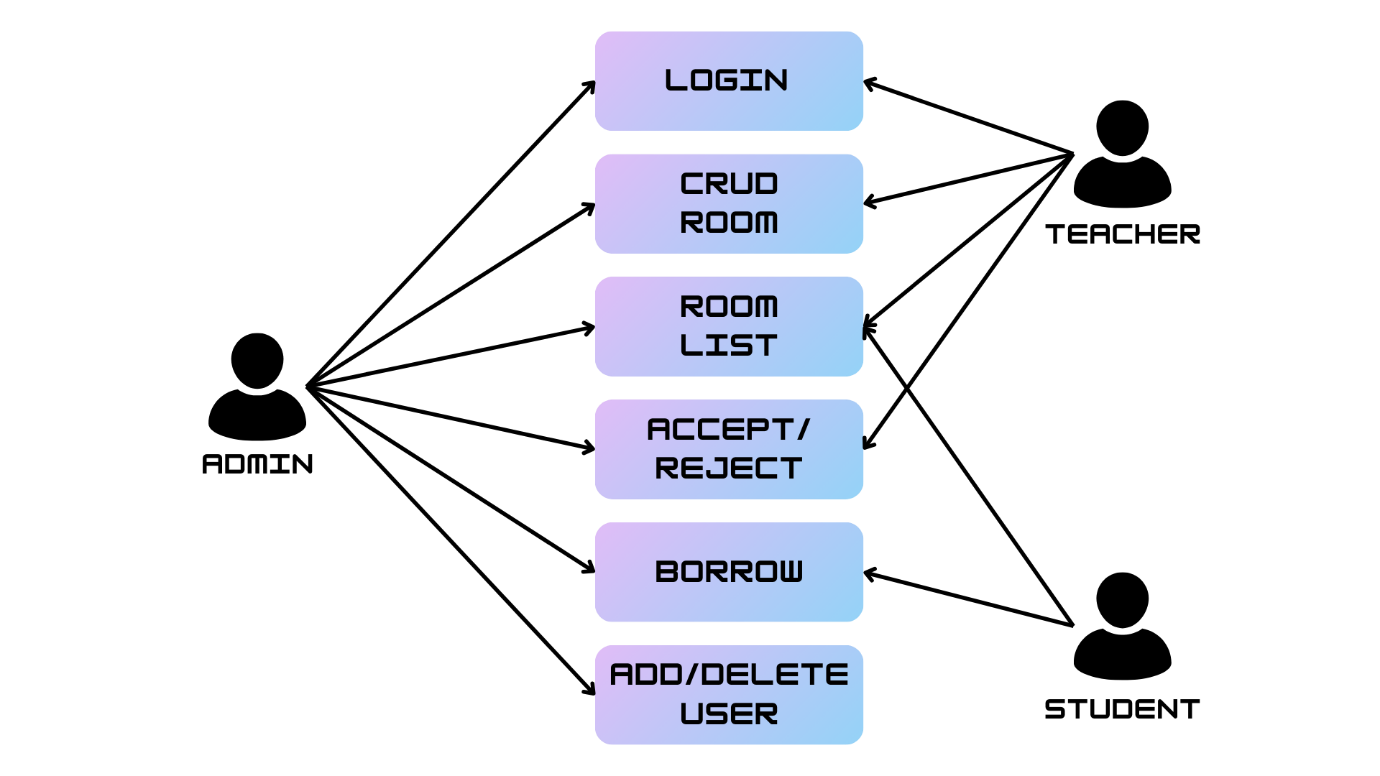
| **ID** | **Deskripsi** |
| --- | --- |
| KFS-01 | Siswa dapat melakukan login menggunakan akun siswa. |
| KFS-02 | Siswa dapat melihat daftar ruangan dan barang yang tersedia untuk dipinjam. |
| KFS-03 | Siswa dapat melihat detail informasi setiap ruangan/barang. |
| KFS-04 | Siswa dapat mengajukan permohonan peminjaman dengan mengisi formulir yang mencakup tanggal, waktu, durasi, dan tujuan. |
| KFS-05 | Siswa dapat melihat status permohonan peminjaman (diterima, ditolak, pending). |
| KFS-06 | Siswa dapat melihat riwayat peminjaman yang pernah diajukan. |
| KFS-07 | Siswa dapat melihat daftar ruangan/barang dan status permintaan terbaru di dashboard mereka. |
| KFS-08 | Siswa menerima notifikasi mengenai status permohonan mereka. |

#### **Tabel 4. Kebutuhan Non-Fungsional**

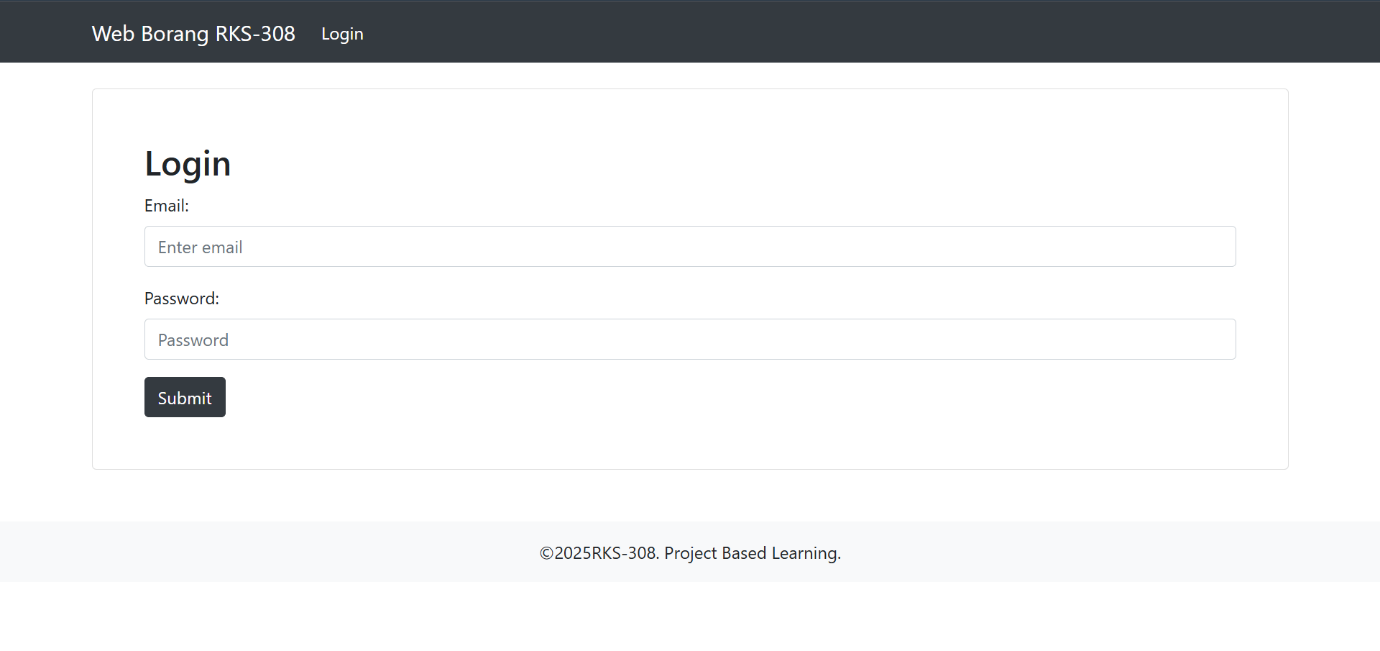
| **ID** | **Deskripsi** |
| --- | --- |
| KNF-01 | Website memiliki tampilan (UI/UX) yang mudah dipahami. |
| KNF-02 | Website menggunakan Bahasa Indonesia. |
| KNF-03 | Sistem dapat memproses permintaan CRUD dalam waktu kurang dari 1 detik. |
| KNF-04 | Website memuat halaman dalam waktu kurang dari 3 detik untuk pengalaman pengguna yang baik. |
| KNF-05 | Data pengguna dan peminjaman disimpan dengan aman menggunakan enkripsi. |
| KNF-6 | Sistem memiliki proteksi terhadap serangan seperti SQL Injection, XSS, dan CSRF. |
| KNF-7 | Hak akses terjamin sehingga pengguna hanya dapat mengakses fitur sesuai dengan peran mereka. |
| KNF-8 | Antarmuka pengguna intuitif dan mudah digunakan oleh admin guru, dan siswa tanpa memerlukan pelatihan khusus. |
| KNF-9 | Kode sumber terstruktur dengan baik dan mengikuti standar pengkodean untuk memudahkan pemeliharaan dan pengembangan di masa depan. |

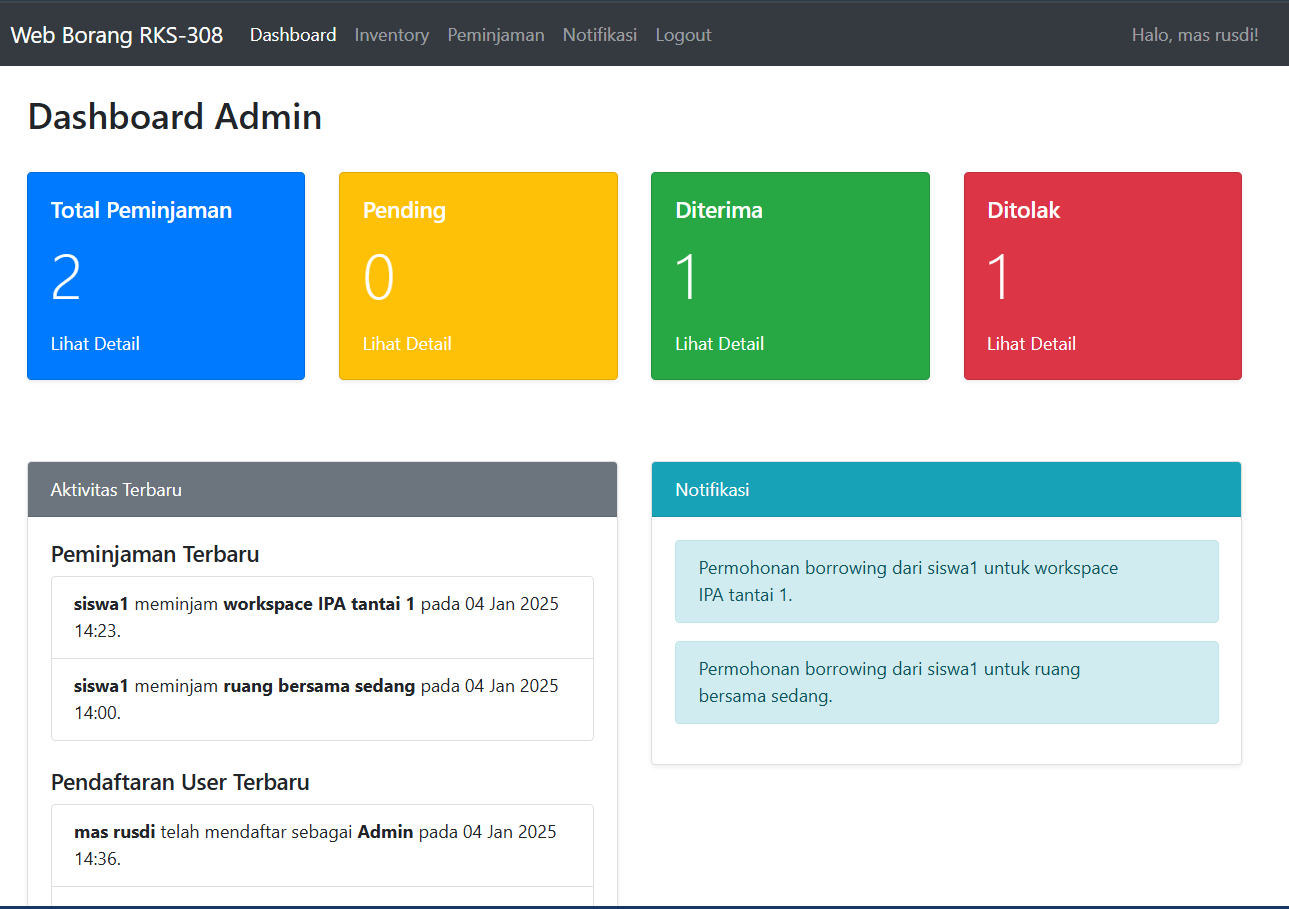
**D. Usecase Diagram**

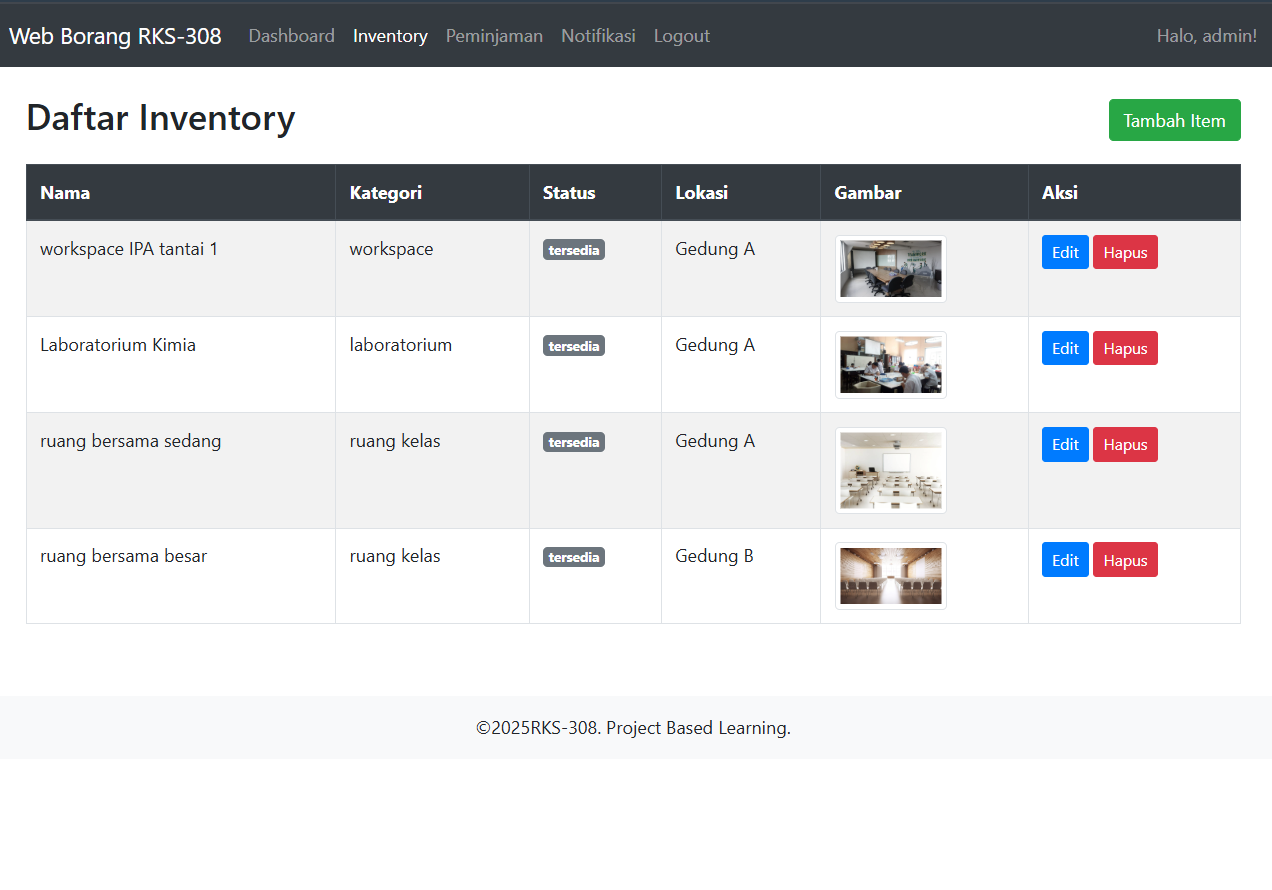
Usecase Diagram merupakan suatu bentuk diagram yang menggambarkan fungsi-fungsi yang diharapkan dari sebuah sistem yang dikembangkan. Dalam usecase diagram akan merepresentasikan sebuah interaksi antara pelaku atau aktor dengan sistem yang akan terjadi kedepannya. Use Case Diagram adalah diagram yang digunakan untuk menggambarkan interaksi antara sistem dan pengguna atau entitas lainnya. Diagram ini membantu dalam memvisualisasikan fungsi-fungsi utama sistem dan skenario penggunaannya.

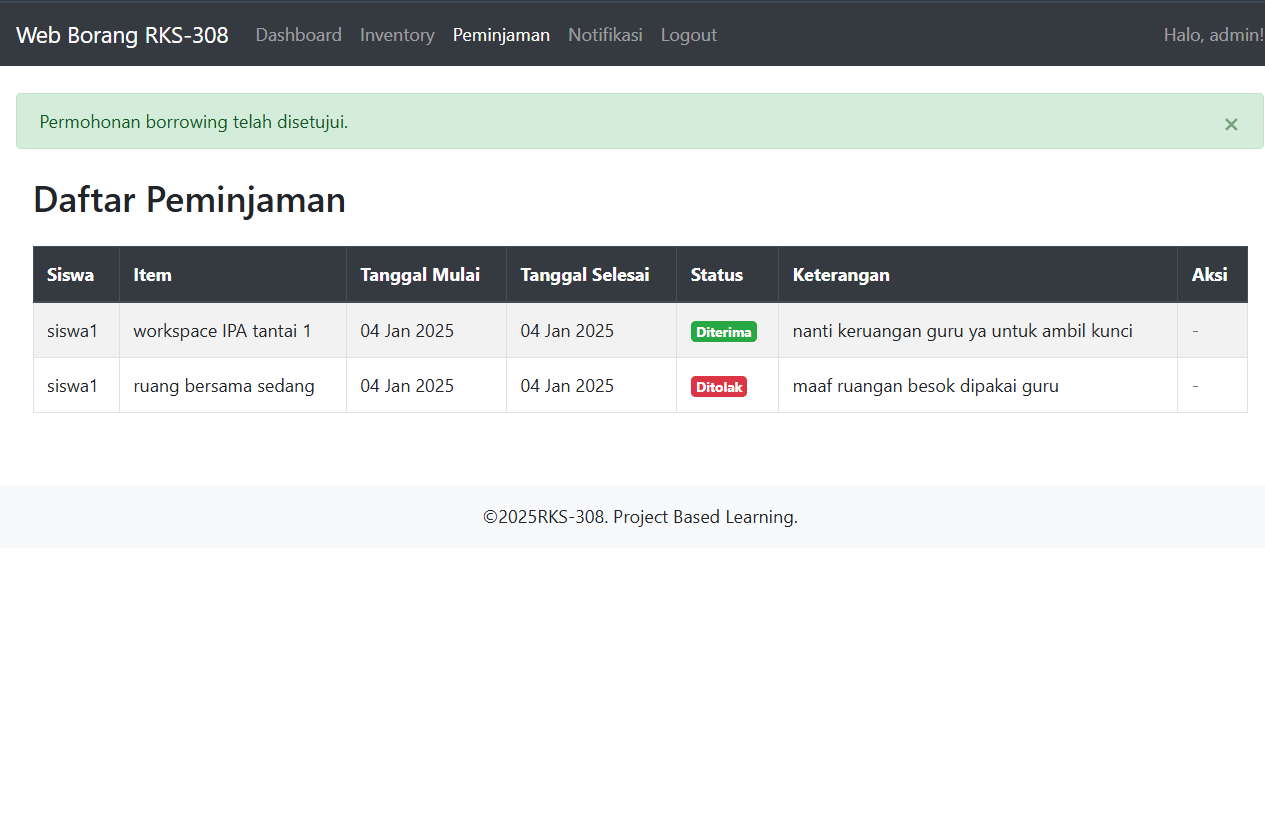


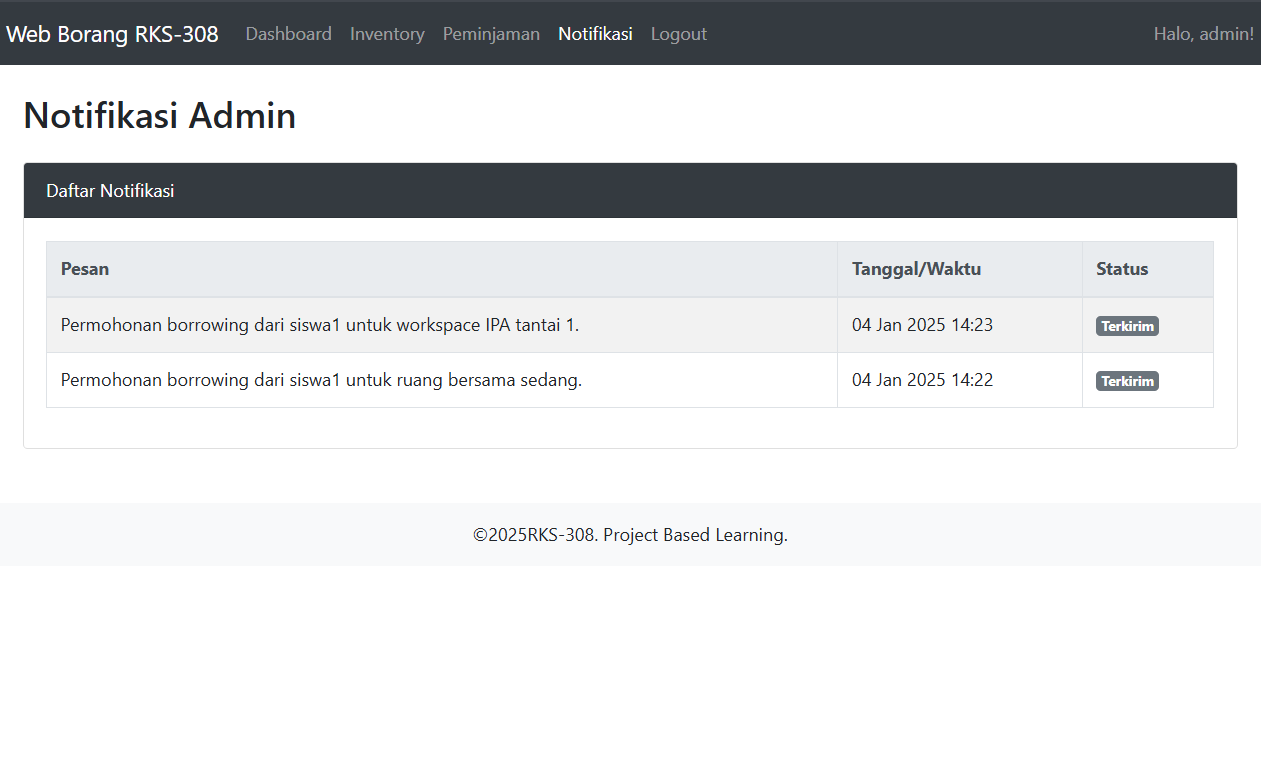
**E. Desain Antarmuka**

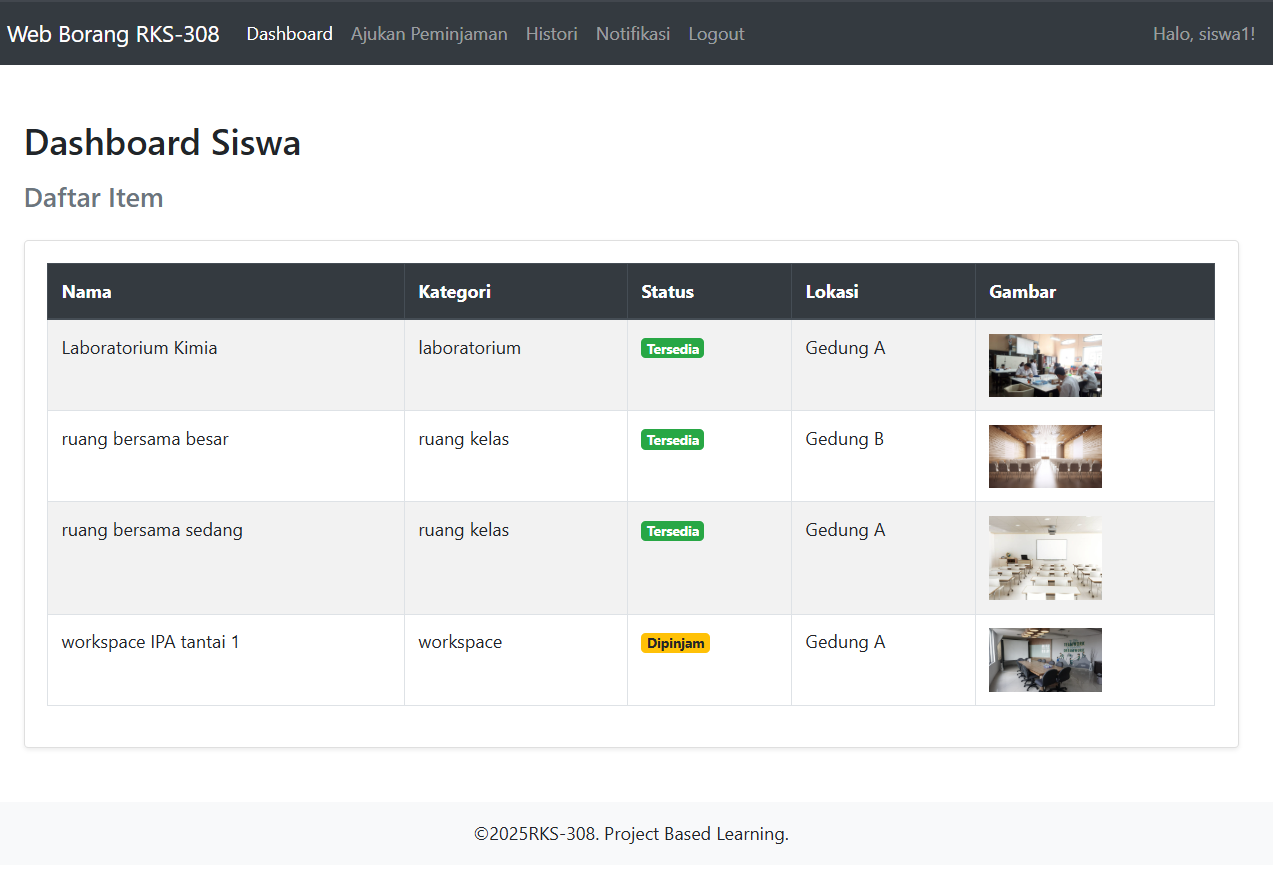


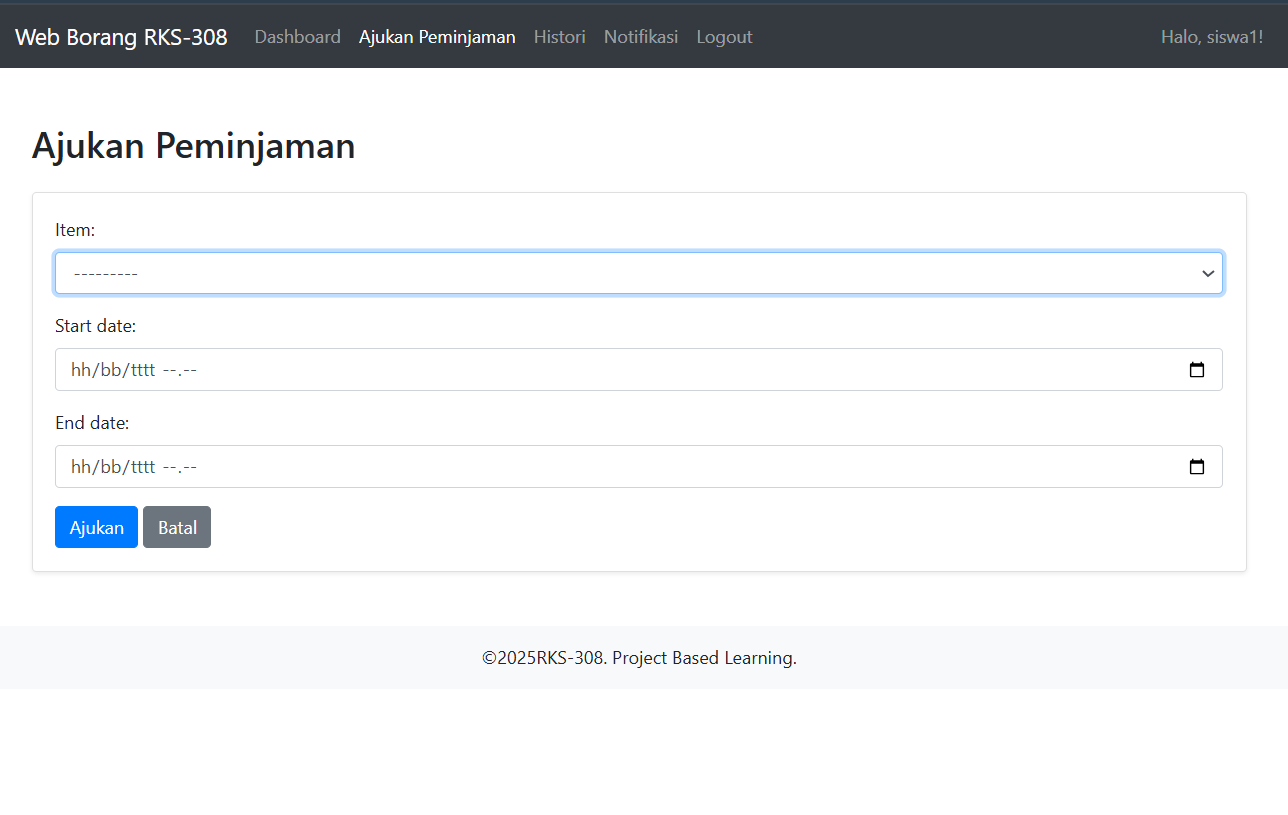


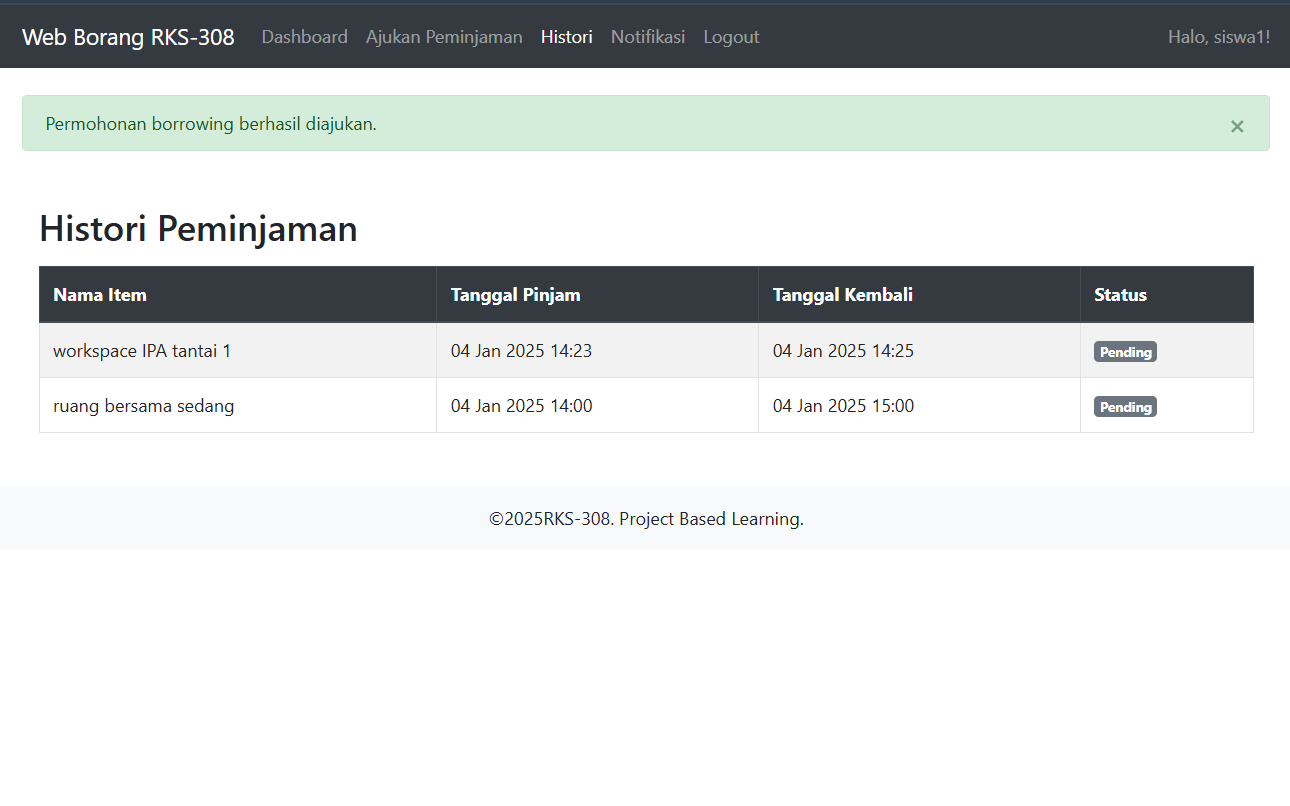


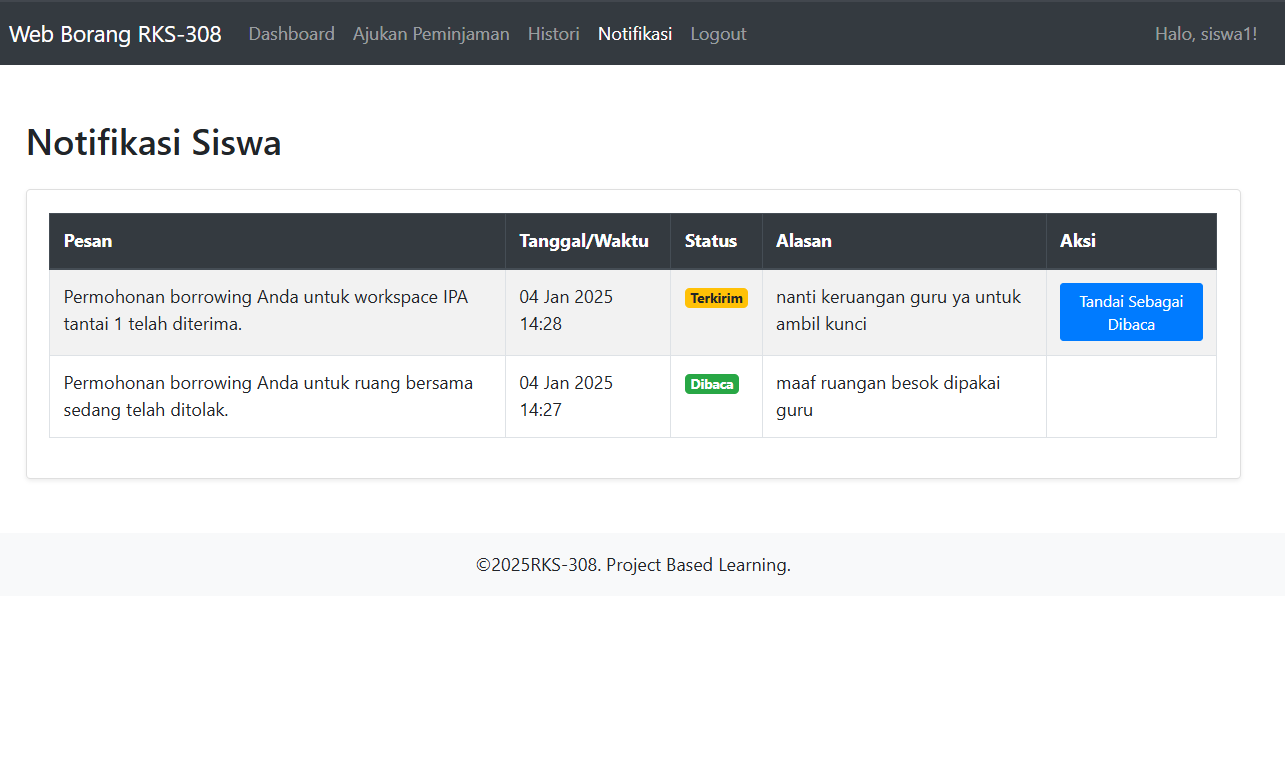


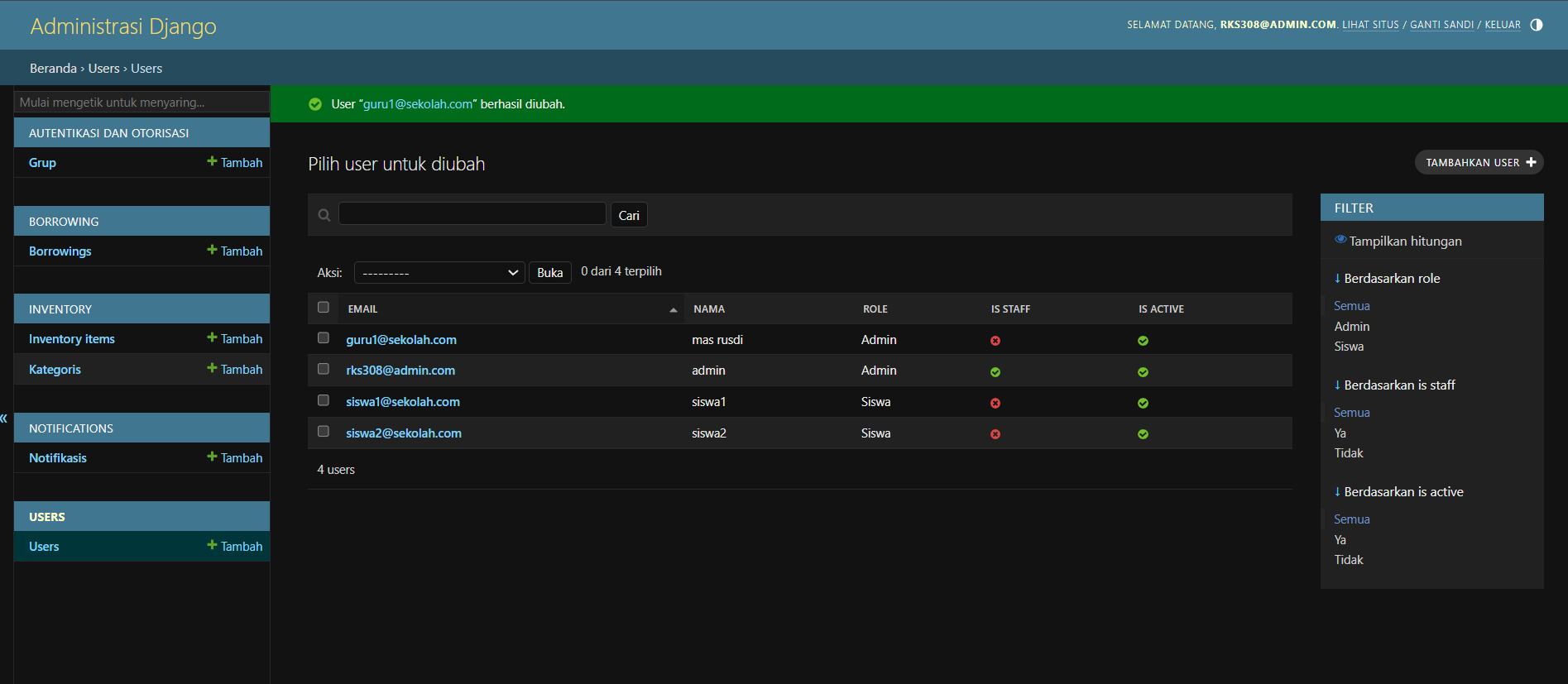












**F. Panduan instalasi**

### **Prasyarat (Prerequisites)**

Sebelum memulai, pastikan Anda memiliki:

1. **Akses ke terminal** di Linux/Ubuntu.
2. **Git** terinstal.
3. **Python 3.8+** terinstal.
4. **pip** (Package Installer for Python) terinstal.
5. **MySQL** terinstal dan berjalan.
6. **MySQL Client** terinstal untuk menghubungkan Django dengan MySQL.

### **Langkah 1: Update Sistem dan Instal Dependensi**

Pertama, perbarui daftar paket dan instal dependensi yang diperlukan.

**sudo apt update**

**sudo apt upgrade -y**

**sudo apt install git python3-pip python3-venv mysql-server mysql-client libmysqlclient-dev -y**

### **Langkah 2: Clone Repository dari GitHub**

Clone repository proyek Anda ke direktori lokal.

**git clone https://github.com/zulpicher/coba.git**

Navigasi ke direktori proyek:

**cd coba**

### **Langkah 3: Buat dan Aktifkan Virtual Environment**

Buat virtual environment menggunakan venv dan aktifkan.

**python3 -m venv venv**

**source venv/bin/activate**

Setelah diaktifkan, terminal akan berubah menandakan bahwa virtual environment aktif.

### **Langkah 4: Instal Paket-Paket Python yang Diperlukan**

Pastikan Anda berada di dalam virtual environment, lalu instal semua dependensi yang tercantum di requirements.txt.

**pip install --upgrade pip**

**pip install -r requirements.txt**

### **Langkah 5: Konfigurasi Database MySQL**

#### **a. Buat Database dan User di MySQL**

Masuk ke MySQL sebagai root:

sudo mysql -u root -p

Setelah berhasil masuk, jalankan perintah berikut untuk membuat database dan user:

CREATE DATABASE borrow\_db;

CREATE USER 'borrow\_user'@'localhost' IDENTIFIED BY 'admin';

GRANT ALL PRIVILEGES ON peminjaman\_db.\* TO 'borrow\_user'@'localhost';

FLUSH PRIVILEGES;

EXIT;

#### **b. Ubah Konfigurasi Database di Django**

Buka file settings.py di proyek Django Anda dan ubah bagian DATABASES menjadi seperti berikut:

DATABASES = {

'default': {

'ENGINE': 'django.db.backends.mysql',

'NAME': 'borrow\_db',

'USER': 'borrow\_user',

'PASSWORD': 'admin',

'HOST': 'localhost',

'PORT': '3306',

}

}

### **Langkah 6: Terapkan Migrations dan Buat Superuser**

#### **a. Terapkan Migrations**

Jalankan perintah berikut untuk membuat tabel-tabel di database:

python manage.py makemigrations

python manage.py migrate

#### **b. Buat Superuser**

Jika Anda perlu membuat superuser untuk mengelola admin, jalankan:

python manage.py createsuperuser

Ikuti prompt untuk memasukkan username, email, dan password.

### **Langkah 7: Kumpulkan File Statis**

Kumpulkan semua file statis ke direktori yang telah ditentukan.

python manage.py collectstatic

Tekan y ketika diminta konfirmasi.

### **Langkah 8: Jalankan Server Pengembangan**

Sekarang, jalankan server Django untuk memastikan semuanya berjalan dengan baik.

python manage.py runserver

Buka browser dan akses <http://127.0.0.1:8000/> untuk melihat aplikasi Anda.

G. Penjelasan Kode  
  
Pada framework Django, kita dapat membuat beberapa aplikasi dalam satu proyek web untuk memisahkan fungsi-fungsinya secara modular. Dalam proyek ini, saya membuat empat aplikasi agar setiap fungsi dapat dikelola secara terpisah dan lebih terstruktur. Saya membuat proyek dengan nama school\_borrowing, yang di dalamnya terdapat aplikasi-aplikasi berikut:

1. **users**: Mengelola autentikasi dan manajemen pengguna.
2. **inventory**: Mengatur inventaris barang yang tersedia.
3. **notifications**: Menangani pemberitahuan dan notifikasi kepada pengguna.
4. **borrowing**: Mengelola proses peminjaman barang oleh pengguna.

Berikut adalah perintah yang digunakan untuk membuat proyek dan aplikasi-aplikasi tersebut:

django-admin startproject school\_borrowing

cd school\_borrowing

django-admin startapp users

django-admin startapp inventory

django-admin startapp notifications

django-admin startapp borrowing

1. school\_borrowing  
school\_borrowing merupakan proyek utama (project) dalam framework Django yang berfungsi sebagai kerangka dasar untuk keseluruhan aplikasi web. Sebagai proyek utama, school\_borrowing mengelola pengaturan global, konfigurasi, dan integrasi antar berbagai aplikasi (apps) yang terdapat di dalamnya.  
(ga terlalu penting sih, yg penting cuma ada di settings.py)  
  
  
2. users  
  
# users/admin.py

from django.contrib import admin

from django.contrib.auth.admin import UserAdmin as BaseUserAdmin

from .models import User

class UserAdmin(BaseUserAdmin):

    # Menampilkan kolom-kolom tertentu di halaman admin

    list\_display = ('email', 'nama', 'role', 'is\_staff', 'is\_active')

    # Menambahkan filter berdasarkan role, status staff, dan status aktif

    list\_filter = ('role', 'is\_staff', 'is\_active')

    # Menentukan layout field di halaman detail pengguna

    fieldsets = (

        (None, {'fields': ('email', 'password')}),

        ('Informasi Pribadi', {'fields': ('nama', 'role')}),

        ('Izin', {'fields': ('is\_staff', 'is\_active')}),

    )

    # Menentukan layout field saat menambahkan pengguna baru

    add\_fieldsets = (

        (None, {

            'classes': ('wide',),

            'fields': ('email', 'nama', 'role', 'password1', 'password2', 'is\_staff', 'is\_active')}

        ),

    )

    # Menentukan field yang dapat dicari di halaman admin

    search\_fields = ('email', 'nama')

    # Menentukan urutan default pengurutan pengguna

    ordering = ('email',)

# Mendaftarkan model User dengan konfigurasi UserAdmin

admin.site.register(User, UserAdmin)

# users/decorators.py

from functools import wraps

from django.core.exceptions import PermissionDenied

def role\_required(required\_role):

    """

    Decorator untuk memeriksa apakah pengguna memiliki peran tertentu.

    Args:

        required\_role (str): Peran yang diperlukan (misalnya, 'admin' atau 'siswa').

    Returns:

        function: Fungsi pembungkus.

    """

    def decorator(view\_func):

        @wraps(view\_func)

        def wrapper(request, \*args, \*\*kwargs):

            if request.user.is\_authenticated and request.user.role == required\_role:

                return view\_func(request, \*args, \*\*kwargs)

            raise PermissionDenied

        return wrapper

    return decorator

# Decorator spesifik untuk admin dan siswa

admin\_required = role\_required('admin')

siswa\_required = role\_required('siswa')

# users/forms.py

from django import forms

from .models import User

class LoginForm(forms.Form):

    email = forms.EmailField(

        label='Email',

        widget=forms.EmailInput(attrs={

            'class': 'form-control',

            'id': 'exampleInputEmail1',

            'aria-describedby': 'emailHelp',

            'placeholder': 'Enter email'

        })

    )

    password = forms.CharField(

        label='Password',

        widget=forms.PasswordInput(attrs={

            'class': 'form-control',

            'id': 'exampleInputPassword1',

            'placeholder': 'Password'

        })

    )

# users/models.py

from django.contrib.auth.models import AbstractBaseUser, PermissionsMixin, BaseUserManager

from django.db import models

from django.utils import timezone

class UserManager(BaseUserManager):

    """

    Manager khusus untuk model User.

    """

    def create\_user(self, email, nama, role, password=None, \*\*extra\_fields):

        if not email:

            raise ValueError('User harus memiliki email')

        email = self.normalize\_email(email)

        user = self.model(email=email, nama=nama, role=role, \*\*extra\_fields)

        user.set\_password(password)

        user.save(using=self.\_db)

        return user

    def create\_superuser(self, email, nama, role, password=None, \*\*extra\_fields):

        """

        Membuat superuser dengan izin penuh.

        """

        extra\_fields.setdefault('is\_staff', True)

        extra\_fields.setdefault('is\_superuser', True)

        return self.create\_user(email, nama, role, password, \*\*extra\_fields)

class User(AbstractBaseUser, PermissionsMixin):

    """

    Model pengguna kustom yang menggantikan model User bawaan Django.

    """

    ROLE\_CHOICES = (

        ('admin', 'Admin'),

        ('siswa', 'Siswa'),

    )

    id = models.AutoField(primary\_key=True)

    nama = models.CharField(max\_length=255)

    email = models.EmailField(unique=True)

    role = models.CharField(max\_length=10, choices=ROLE\_CHOICES)

    is\_active = models.BooleanField(default=True)

    is\_staff = models.BooleanField(default=False)

    date\_joined = models.DateTimeField(default=timezone.now)

    objects = UserManager()

    USERNAME\_FIELD = 'email'

    REQUIRED\_FIELDS = ['nama', 'role']

    def \_\_str\_\_(self):

        return self.email

# users/urls.py

from django.urls import path

from . import views

urlpatterns = [

    path('', views.home, name='home'),  # URL root yang mengarahkan ke halaman utama

    path('login/', views.user\_login\_view, name='login'),  # URL untuk login

    path('logout/', views.user\_logout\_view, name='logout'),  # URL untuk logout

    path('dashboard/admin/', views.admin\_dashboard, name='admin\_dashboard'),  # Dashboard admin

    path('dashboard/siswa/', views.siswa\_dashboard, name='siswa\_dashboard'),  # Dashboard siswa

]

# users/views.py

from django.shortcuts import render, redirect

from django.contrib.auth import authenticate, login, logout

from django.core.exceptions import PermissionDenied

from .forms import LoginForm

from .decorators import admin\_required, siswa\_required

from borrowing.models import Borrowing

from inventory.models import InventoryItem

from notifications.models import Notifikasi

from .models import User

def home(request):

    """

    Halaman utama yang mengarahkan pengguna ke dashboard sesuai peran.

    """

    if request.user.is\_authenticated:

        if request.user.role == 'admin':

            return redirect('admin\_dashboard')

        elif request.user.role == 'siswa':

            return redirect('siswa\_dashboard')

    return redirect('login')

def user\_login\_view(request):

    """

    View untuk menangani proses login pengguna.

    """

    if request.method == 'POST':

        form = LoginForm(request.POST)

        if form.is\_valid():

            email = form.cleaned\_data['email']

            password = form.cleaned\_data['password']

            user = authenticate(request, email=email, password=password)

            if user:

                login(request, user)

                if user.role == 'admin':

                    return redirect('admin\_dashboard')

                else:

                    return redirect('siswa\_dashboard')

            else:

                form.add\_error(None, 'Email atau Password salah')

    else:

        form = LoginForm()

    return render(request, 'users/login.html', {'form': form})

def user\_logout\_view(request):

    """

    View untuk menangani proses logout pengguna.

    """

    logout(request)

    return redirect('login')

@admin\_required

def admin\_dashboard(request):

    """

    Dashboard khusus untuk admin yang menampilkan statistik dan notifikasi.

    """

    # \*\*Statistik Peminjaman\*\*

    total\_borrowings = Borrowing.objects.count()  # Jumlah seluruh peminjaman

    pending\_borrowings = Borrowing.objects.filter(status='pending').count()  # Peminjaman Pending

    approved\_borrowings = Borrowing.objects.filter(status='approved').count()  # Peminjaman Diterima

    rejected\_borrowings = Borrowing.objects.filter(status='rejected').count()  # Peminjaman Ditolak

    # \*\*Aktivitas Terbaru\*\*

    recent\_borrowings = Borrowing.objects.all().order\_by('-start\_date')[:5]  # 5 peminjaman terbaru

    recent\_users = User.objects.all().order\_by('-date\_joined')[:5]  # 5 pendaftaran user terbaru

    # \*\*Notifikasi\*\*

    admin\_notifications = Notifikasi.objects.filter(user\_\_role='admin').order\_by('-tanggal\_waktu')[:5]  # 5 notifikasi terbaru

    # \*\*Menyusun Context\*\*

    context = {

        'total\_borrowings': total\_borrowings,

        'pending\_borrowings': pending\_borrowings,

        'approved\_borrowings': approved\_borrowings,

        'rejected\_borrowings': rejected\_borrowings,

        'recent\_borrowings': recent\_borrowings,

        'recent\_users': recent\_users,

        'admin\_notifications': admin\_notifications,

    }

    return render(request, 'users/admin\_dashboard.html', context)

@siswa\_required

def siswa\_dashboard(request):

    """

    Dashboard khusus untuk siswa yang menampilkan peminjaman dan inventaris.

    """

    # Menampilkan peminjaman terbaru yang dilakukan oleh siswa

    latest\_borrowings = Borrowing.objects.filter(user=request.user).order\_by('-start\_date')[:5]

    # Menampilkan semua item, baik tersedia maupun dipinjam

    all\_items = InventoryItem.objects.all().order\_by('nama')

    context = {

        'latest\_borrowings': latest\_borrowings,

        'all\_items': all\_items,

    }

    return render(request, 'users/siswa\_dashboard.html', context)

2. borrowing

# borrowing/admin.py

from django.contrib import admin

from .models import Borrowing

@admin.register(Borrowing)

class BorrowingAdmin(admin.ModelAdmin):

    """

    Konfigurasi tampilan model Borrowing di halaman admin Django.

    """

    list\_display = ('id', 'user', 'item', 'start\_date', 'end\_date', 'status')

    list\_filter = ('status', 'item', 'user')

    search\_fields = ('user\_\_nama', 'item\_\_nama')

# borrowing/forms.py

from django import forms

from .models import Borrowing

from inventory.models import InventoryItem

class BorrowingForm(forms.ModelForm):

    """

    Formulir untuk mengajukan peminjaman item.

    """

    class Meta:

        model = Borrowing

        fields = ['item', 'start\_date', 'end\_date']

        widgets = {

            'start\_date': forms.DateTimeInput(attrs={'type': 'datetime-local'}),

            'end\_date': forms.DateTimeInput(attrs={'type': 'datetime-local'}),

        }

    def \_\_init\_\_(self, \*args, \*\*kwargs):

        super(BorrowingForm, self).\_\_init\_\_(\*args, \*\*kwargs)

        # Hanya menampilkan item yang tersedia

        self.fields['item'].queryset = InventoryItem.objects.filter(status='tersedia')

class ProcessBorrowingForm(forms.ModelForm):

    """

    Formulir untuk memproses peminjaman (menyetujui atau menolak).

    """

    class Meta:

        model = Borrowing

        fields = ['status', 'reason']

        widgets = {

            'status': forms.Select(choices=Borrowing.STATUS\_CHOICES),

            'reason': forms.Textarea(attrs={'rows': 3}),

        }

    def clean(self):

        """

        Validasi tambahan untuk memastikan alasan diisi jika peminjaman ditolak.

        """

        cleaned\_data = super().clean()

        status = cleaned\_data.get('status')

        reason = cleaned\_data.get('reason')

        if status == 'rejected' and not reason:

            self.add\_error('reason', 'Alasan wajib diisi ketika peminjaman ditolak.')

        return cleaned\_data

# borrowing/models.py

from django.db import models

from users.models import User

from inventory.models import InventoryItem

class Borrowing(models.Model):

    """

    Model yang merepresentasikan peminjaman item oleh pengguna.

    """

    STATUS\_CHOICES = (

        ('pending', 'Pending'),

        ('approved', 'Diterima'),

        ('rejected', 'Ditolak'),

        ('completed', 'Selesai'),# Status tambahan utk peminjaman yang selesai

    )

    id = models.AutoField(primary\_key=True)

    user = models.ForeignKey(User, on\_delete=models.CASCADE, related\_name='borrowings')

    item = models.ForeignKey(InventoryItem, on\_delete=models.CASCADE, related\_name='borrowings')

    start\_date = models.DateTimeField()

    end\_date = models.DateTimeField()

    status = models.CharField(max\_length=10, choices=STATUS\_CHOICES, default='pending')

    reason = models.TextField(null=True, blank=True)  # Alasan penolakan

    def \_\_str\_\_(self):

        """

        Mengembalikan representasi string dari peminjaman.

        """

        return f"{self.user.nama} - {self.item.nama} ({self.status})"

# borrowing/urls.py

from django.urls import path

from . import views

urlpatterns = [

    path('submit/', views.submit\_borrowing, name='submit\_borrowing'),  # URL untuk mengajukan peminjaman

    path('admin/list/', views.borrowing\_list, name='borrowing\_list'),  # URL untuk melihat daftar peminjaman (admin)

    path('admin/process/<int:borrowing\_id>/', views.process\_borrowing, name='process\_borrowing'),  # URL untuk memproses peminjaman (admin)

    path('history/', views.borrowing\_history, name='borrowing\_history'),  # URL untuk melihat riwayat peminjaman (siswa)

]

# borrowing/views.py

from django.shortcuts import render, get\_object\_or\_404, redirect

from django.contrib import messages

from .models import Borrowing

from .forms import BorrowingForm, ProcessBorrowingForm

from users.decorators import admin\_required, siswa\_required

from notifications.models import Notifikasi

from users.models import User

def check\_conflict(item, start, end):

    """

    Memeriksa apakah ada konflik jadwal peminjaman untuk item tertentu.

    """

    conflicts = Borrowing.objects.filter(

        item=item,

        status='approved',

        start\_date\_\_lt=end,

        end\_date\_\_gt=start

    )

    return conflicts.exists()

@siswa\_required

def submit\_borrowing(request):

    """

    View untuk siswa mengajukan peminjaman item.

    """

    if request.method == 'POST':

        form = BorrowingForm(request.POST)

        if form.is\_valid():

            borrowing = form.save(commit=False)

            borrowing.user = request.user

            item = borrowing.item

            # Cek konflik jadwal

            if not check\_conflict(item, borrowing.start\_date, borrowing.end\_date):

                borrowing.save()

                # Kirim notifikasi ke semua admin

                admins = User.objects.filter(role='admin')

                for admin in admins:

                    Notifikasi.objects.create(

                        user=admin,

                        pesan=f"Permohonan borrowing dari {request.user.nama} untuk {item.nama}."

                    )

                messages.success(request, 'Permohonan borrowing berhasil diajukan.')

                return redirect('borrowing\_history')

            else:

                form.add\_error(None, 'Konflik jadwal. Silakan pilih waktu lain.')

    else:

        form = BorrowingForm()

    return render(request, 'borrowing/submit\_borrowing.html', {'form': form})

@admin\_required

def borrowing\_list(request):

    """

    View untuk admin melihat daftar semua peminjaman.

    """

    borrowings = Borrowing.objects.all().order\_by('-start\_date')

    return render(request, 'borrowing/borrowing\_list.html', {'borrowings': borrowings})

@admin\_required

def process\_borrowing(request, borrowing\_id):

    """

    View untuk admin memproses (menyetujui/menolak) peminjaman.

    """

    borrowing = get\_object\_or\_404(Borrowing, id=borrowing\_id)

    if request.method == 'POST':

        form = ProcessBorrowingForm(request.POST, instance=borrowing)

        if form.is\_valid():

            borrowing = form.save()

            # Update status item berdasarkan keputusan

            if borrowing.status == 'approved':

                borrowing.item.status = 'dipinjam'

                borrowing.item.save()

            elif borrowing.status == 'rejected':

                borrowing.item.status = 'tersedia'  # Asumsikan item menjadi tersedia kembali

            # Kirim notifikasi ke siswa dengan alasan jika ada

            Notifikasi.objects.create(

                user=borrowing.user,

                pesan=f"Permohonan borrowing Anda untuk {borrowing.item.nama} telah {borrowing.get\_status\_display().lower()}.",

                reason=borrowing.reason  # Sertakan alasan

            )

            # Tambahkan pesan berdasarkan status

            if borrowing.status == 'approved':

                messages.success(request, 'Permohonan borrowing telah disetujui.')

            elif borrowing.status == 'rejected':

                messages.warning(request, 'Permohonan borrowing telah ditolak.')

            return redirect('borrowing\_list')

    else:

        form = ProcessBorrowingForm(instance=borrowing)

    return render(request, 'borrowing/process\_borrowing.html', {'form': form, 'borrowing': borrowing})

@siswa\_required

def borrowing\_history(request):

    """

    View untuk siswa melihat riwayat peminjaman mereka.

    """

    borrowings = Borrowing.objects.filter(user=request.user).order\_by('-start\_date')

    # Debugging: Cetak peminjaman untuk pengguna

    print(f"Borrowings for user {request.user.nama}: {borrowings}")

    return render(request, 'borrowing/borrowing\_history.html', {'borrowings': borrowings})

3. inventory

# inventory/admin.py

from django.contrib import admin

from .models import Kategori, InventoryItem

# Mendaftarkan model Kategori ke halaman admin Django

admin.site.register(Kategori)

# Mendaftarkan model InventoryItem ke halaman admin Django

admin.site.register(InventoryItem)

# inventory/forms.py

from django import forms

from .models import InventoryItem

class InventoryForm(forms.ModelForm):

    """

    Formulir untuk menambahkan atau mengedit item inventaris.

    """

    class Meta:

        model = InventoryItem

        fields = ['nama', 'kategori', 'status', 'lokasi', 'gambar', 'kapasitas']

        widgets = {

            'nama': forms.TextInput(attrs={'class': 'form-control'}),

            'kategori': forms.Select(attrs={'class': 'form-control'}),

            'status': forms.Select(attrs={'class': 'form-control'}),

            'lokasi': forms.TextInput(attrs={'class': 'form-control'}),

            'gambar': forms.ClearableFileInput(attrs={'class': 'form-control-file'}),

            'kapasitas': forms.NumberInput(attrs={'class': 'form-control'}),

        }

# inventory/models.py

from django.db import models

class Kategori(models.Model):

    """

    Model yang merepresentasikan kategori item inventaris.

    """

    nama = models.CharField(max\_length=100)

    def \_\_str\_\_(self):

        """

        Mengembalikan representasi string dari kategori.

        """

        return self.nama

class InventoryItem(models.Model):

    """

    Model yang merepresentasikan item inventaris.

    """

    STATUS\_CHOICES = (

        ('tersedia', 'Tersedia'),

        ('dipinjam', 'Dipinjam'),

    )

    TYPING\_CHOICES = (

        ('barang', 'Barang'),

        ('ruangan', 'Ruangan'),

    )

    id = models.AutoField(primary\_key=True)

    nama = models.CharField(max\_length=255)

    kategori = models.ForeignKey(Kategori, on\_delete=models.CASCADE)

    status = models.CharField(max\_length=10, choices=STATUS\_CHOICES, default='tersedia')

    lokasi = models.CharField(max\_length=255)

    gambar = models.ImageField(upload\_to='inventory\_images/')

    kapasitas = models.IntegerField(null=True, blank=True)  # Hanya untuk ruangan

    def \_\_str\_\_(self):

        """

        Mengembalikan representasi string dari item inventaris.

        """

        return self.nama

# inventory/urls.py

from django.urls import path

from . import views

urlpatterns = [

    path('', views.inventory\_list, name='inventory\_list'),  # URL untuk melihat daftar inventaris

    path('add/', views.add\_item, name='add\_item'),  # URL untuk menambahkan item baru

    path('edit/<int:item\_id>/', views.edit\_item, name='edit\_item'),  # URL untuk mengedit item

    path('delete/<int:item\_id>/', views.delete\_item, name='delete\_item'),  # URL untuk menghapus item

]

# inventory/views.py

from django.shortcuts import render, get\_object\_or\_404, redirect

from django.contrib import messages

from .models import InventoryItem

from .forms import InventoryForm

from users.decorators import admin\_required

@admin\_required

def add\_item(request):

    """

    View untuk admin menambahkan item inventaris baru.

    """

    if request.method == 'POST':

        form = InventoryForm(request.POST, request.FILES)

        if form.is\_valid():

            form.save()

            messages.success(request, 'Item berhasil ditambahkan.')

            return redirect('inventory\_list')

        else:

            messages.error(request, 'Terjadi kesalahan saat menambahkan item. Silakan periksa kembali.')

    else:

        form = InventoryForm()

    return render(request, 'inventory/add\_item.html', {'form': form})

@admin\_required

def edit\_item(request, item\_id):

    """

    View untuk admin mengedit item inventaris yang ada.

    """

    item = get\_object\_or\_404(InventoryItem, id=item\_id)

    if request.method == 'POST':

        form = InventoryForm(request.POST, request.FILES, instance=item)

        if form.is\_valid():

            form.save()

            messages.success(request, 'Item berhasil diperbarui.')

            return redirect('inventory\_list')

        else:

            messages.error(request, 'Terjadi kesalahan saat memperbarui item. Silakan periksa kembali.')

    else:

        form = InventoryForm(instance=item)

    return render(request, 'inventory/edit\_item.html', {'form': form, 'item': item})

@admin\_required

def delete\_item(request, item\_id):

    """

    View untuk admin menghapus item inventaris.

    """

    item = get\_object\_or\_404(InventoryItem, id=item\_id)

    if request.method == 'POST':

        item.delete()

        messages.success(request, 'Item berhasil dihapus.')

        return redirect('inventory\_list')

    return render(request, 'inventory/delete\_item.html', {'item': item})

@admin\_required

def inventory\_list(request):

    """

    View untuk admin melihat daftar semua item inventaris.

    """

    items = InventoryItem.objects.all()

    return render(request, 'inventory/inventory\_list.html', {'items': items})

4. notifications

# notifications/admin.py

from django.contrib import admin

from .models import Notifikasi

# Mendaftarkan model Notifikasi ke halaman admin Django

admin.site.register(Notifikasi)

# notifications/models.py

from django.db import models

from users.models import User

class Notifikasi(models.Model):

    """

    Model yang merepresentasikan notifikasi untuk pengguna.

    """

    STATUS\_CHOICES = (

        ('terkirim', 'Terkirim'),

        ('dibaca', 'Dibaca'),

    )

    id = models.AutoField(primary\_key=True)

    user = models.ForeignKey(User, on\_delete=models.CASCADE, related\_name='notifikasis')

    pesan = models.TextField()

    tanggal\_waktu = models.DateTimeField(auto\_now\_add=True)

    status = models.CharField(max\_length=10, choices=STATUS\_CHOICES, default='terkirim')

    reason = models.TextField(null=True, blank=True)  # Field baru untuk alasan penolakan

    def \_\_str\_\_(self):

        """

        Mengembalikan representasi string dari notifikasi.

        """

        return f"Notifikasi untuk {self.user.nama} - {self.get\_status\_display()}"

# notifications/urls.py

from django.urls import path

from . import views

urlpatterns = [

    path('admin/', views.notifikasi\_admin, name='notifikasi\_admin'),  # URL untuk notifikasi admin

    path('siswa/', views.notifikasi\_siswa, name='notifikasi\_siswa'),  # URL untuk notifikasi siswa

    path('mark\_as\_read/<int:notification\_id>/', views.mark\_as\_read, name='mark\_as\_read'),  # URL untuk menandai notifikasi sebagai dibaca

]

# notifications/views.py

from django.shortcuts import render, get\_object\_or\_404, redirect

from django.contrib.auth.decorators import login\_required

from django.contrib import messages

from users.decorators import admin\_required, siswa\_required

from .models import Notifikasi

@admin\_required

def notifikasi\_admin(request):

    """

    View untuk menampilkan notifikasi kepada admin.

    """

    notifications = Notifikasi.objects.filter(user\_\_role='admin').order\_by('-tanggal\_waktu')

    return render(request, 'notifications/notifikasi\_admin.html', {'notifications': notifications})

@siswa\_required

def notifikasi\_siswa(request):

    """

    View untuk menampilkan notifikasi kepada siswa.

    """

    notifications = Notifikasi.objects.filter(user=request.user).order\_by('-tanggal\_waktu')

    return render(request, 'notifications/notifikasi\_siswa.html', {'notifications': notifications})

@login\_required

def mark\_as\_read(request, notification\_id):

    """

    View untuk menandai notifikasi sebagai dibaca.

    """

    notification = get\_object\_or\_404(Notifikasi, id=notification\_id, user=request.user)

    if notification.status != 'dibaca':

        notification.status = 'dibaca'

        notification.save()

        messages.success(request, 'Notifikasi telah ditandai sebagai dibaca.')

    else:

        messages.info(request, 'Notifikasi sudah dalam status dibaca.')

    # Redirect berdasarkan role pengguna

    if request.user.role == 'admin':

        return redirect('notifikasi\_admin')

    else:

        return redirect('notifikasi\_siswa')