Долоо хоног 9: Гадаад сангуудтай ажиллах ба алдааны мэдээлэл (Error Handling)

- **Сэдэв:** Гадаад сангууд ашиглах, алдаа засварлах
- **Агуулга:**
    - Төсөлдөө шаардлагатай гадаад пакет (npm, pip, Maven) нэмж суурилуулах.
    - Алдааны мэдээлэл (Exceptions) илрүүлэх, боловсруулах.
    - Алдааны мэдээллийг хэрэглэгчдэд ойлгомжтойгоор харуулах.

Долоо хоног 9: Гадаад сангуудтай ажиллах ба алдааны мэдээлэл

Төсөлдөө шаардлагатай гадаад пакет нэмж суурилуулах

Python пакетүүдийг менежмент хийх

requirements.txt файлыг бэлтгэх:

txt

requirements.txt

Flask==2.3.3

requests==2.31.0

SQLAlchemy==2.0.21

Werkzeug==2.3.7

Jinja2==3.1.2

python-dotenv==1.0.0

pytest==7.4.2

pytest-cov==4.1.0

Flask-SQLAlchemy==3.0.5

Flask-Login==0.6.2

Dev requirements (хөгжүүлэлтийн орчинд):

txt

requirements-dev.txt

-r requirements.txt

black==23.9.1

flake8==6.1.0

pylint==2.17.5

autopep8==2.0.4

Пакетуудыг суурилуулах:

bash

 # Үндсэн пакетууд

pip install -r requirements.txt

 # Хөгжүүлэлтийн пакетууд

pip install -r requirements-dev.txt

 # Тодорхой пакет суурилуулах

pip install requests

 # Пакетын тодорхой хувилбар

pip install "flask>=2.0,<3.0"

Node.js (npm) пакет менежмент

json

// package.json

```
{
  "name": "my-project",
  "version": "1.0.0",
  "dependencies": {
    "express": "^4.18.2",
    "axios": "^1.5.0",
    "cors": "^2.8.5",
    "dotenv": "^16.3.1"
  },
  "devDependencies": {
```

```
    "nodemon": "^3.0.1",
    "eslint": "^8.48.0",
    "jest": "^29.6.4"
  }
}
```

Алдааны мэдээлэл барьж авах, боловсруулах

Үндсэн алдаа барьж авах

python

```
# exceptions/base.py
class AppError(Exception):
    """Үндсэн аппликейшн алдааны класс"""
    def __init__(self, message, status_code=500, details=None):
        super().__init__(message)
        self.message = message
        self.status_code = status_code
        self.details = details or {}

    def to_dict(self):
        return {
            error: self.message,
            status_code: self.status_code,
            details: self.details
        }

class ValidationError(AppError):
    """Баталгаажуулалтын алдаа"""
    def __init__(self, message="Баталгаажуулалтын алдаа", details=None):
        super().__init__(message, 400, details)
```

```
class NotFoundError(AppError):
    """Олдсонгүй алдаа"""
    def __init__(self, resource_name, resource_id):
        message = f"{resource_name} олдсонгүй: {resource_id}"
        super().__init__(message, 404)


class DatabaseError(AppError):
    """Өгөгдлийн сангийн алдаа"""
    def __init__(self, message="Өгөгдлийн сангийн алдаа", details=None):
        super().__init__(message, 500, details)
```

Алдаа барьж авах декоратор

```
python
# utils/error_handler.py
from functools import wraps
from flask import jsonify
from exceptions.base import AppError


def handle_errors(func):
    """Функц доторх алдааг барьж авах декоратор"""
    @wraps(func)
    def wrapper(*args, **kwargs):
        try:
            return func(*args, **kwargs)
        except AppError as e:
            # Өөрийн алдаанууд
            return jsonify(e.to_dict()), e.status_code
        except ValueError as e:
```

```
            # Утгын алдаа
            return jsonify({
                error:   Буруу утга,
                message: str(e),
                status_code: 400
            }), 400
        except Exception as e:
            # Бусад бүх алдаанууд
            return jsonify({
                error:   Дотоод серверийн алдаа,
                message:   Алдаа гарлаа. Дараа дахин оролдоно уу.,
                status_code: 500
            }), 500
    return wrapper
```

Алдааны мэдээллийг хэрэглэгчдэд ойлгомжтойгоор харуулах

Flask аппликейшнд алдаа боловсруулах

python

```
# app/error_handlers.py
from flask import jsonify, render_template, request
from exceptions.base import AppError, ValidationError, NotFoundError

def register_error_handlers(app):
    """Алдаа боловсруулагч функцыг бүртгэх"""

    @app.errorhandler(ValidationError)
    def handle_validation_error(error):
        if request.path.startswith(/api/):
            return jsonify(error.to_dict()), error.status_code
```

```
    else:
        return render_template(errors/400.html, error=error), 400

@app.errorhandler(NotFoundError)
def handle_not_found_error(error):
    if request.path.startswith(/api/):
        return jsonify(error.to_dict()), error.status_code
    else:
        return render_template(errors/404.html, error=error), 404

@app.errorhandler(AppError)
def handle_app_error(error):
    if request.path.startswith(/api/):
        return jsonify(error.to_dict()), error.status_code
    else:
        return render_template(errors/500.html, error=error), 500

@app.errorhandler(404)
def handle_404(error):
    if request.path.startswith(/api/):
        return jsonify({
            error: API endpoint    олдсонгүй,
            status_code: 404
        }), 404
    else:
        return render_template(errors/404.html, error=error), 404
    @app.errorhandler(500)
def handle_500(error):
```

```
    # Production орчинд мэдээллийг хязгаарлах
    if app.config[DEBUG]:
        message = str(error)
    else:
        message =  Дотоод серверийн алдаа


    if request.path.startswith(/api/):
        return jsonify({
            error:   Дотоод серверийн алдаа,
            message: message,
            status_code: 500
        }), 500
    else:
        return render_template(errors/500.html, error=message), 500
```

Алдааны HTML template

```
html
<!-- templates/errors/404.html -->
<!DOCTYPE html>
<html>
<head>
    <title>404 - Олдсонгүй</title>
    <style>
        .error-container {
            text-align: center;
            padding: 50px;
            font-family: Arial, sans-serif;
        }
        .error-code {
```

```
            font-size: 72px;
            color: #dc3545;
        }
        .error-message {
            font-size: 24px;
            margin: 20px 0;
        }
        .home-link {
            color: #007bff;
            text-decoration: none;
        }
    </style>
</head>
<body>
    <div class="error-container">
        <div class="error-code">404</div>
        <div class="error-message">Хуудас олдсонгүй</div>
        <p>Уучлаарай, таны хайсан хуудас олдсонгүй.</p>
        <a href="/" class="home-link">Нүүр хуудас руу буцах</a>
    </div>
</body>
</html>
html
<!-- templates/errors/500.html -->
<!DOCTYPE html>
<html>
<head>
    <title>500 - Серверийн алдаа</title>
```

```
    <style>
        .error-container {
            text-align: center;
            padding: 50px;
            font-family: Arial, sans-serif;
        }
        .error-code {
            font-size: 72px;
            color: #dc3545;
        }
    </style>
</head>
<body>
    <div class="error-container">
        <div class="error-code">500</div>
        <div class="error-message">Серверийн алдаа</div>
        <p>Системд алдаа гарлаа. Дараа дахин оролдоно уу.</p>
        {% if config.DEBUG %}
        <div class="error-details">
            <pre>{{ error }}</pre>
        </div>
        {% endif %}
        <a href="/" class="home-link">Нүүр хуудас руу буцах</a>
    </div>
</body>
</html>
```

Практик жишээ: Бүтээгдэхүүн service

python

```
# services/product_service.py
from exceptions.base import ValidationError, NotFoundError, DatabaseError
from models import Product
from database import db

class ProductService:

    @staticmethod
    def get_product(product_id):
        """Бүтээгдэхүүнийг ID-аар авах"""
        try:
            product = Product.query.get(product_id)
            if not product:
                raise NotFoundError("Бүтээгдэхүүн", product_id)
            return product
        except Exception as e:
            raise DatabaseError("Бүтээгдэхүүн авахад алдаа гарлаа") from e
    @staticmethod
    def create_product(name, price, description=None):
        """Шинэ бүтээгдэхүүн үүсгэх"""
        # Баталгаажуулалт
        if not name or not name.strip():
            raise ValidationError("Бүтээгдэхүүний нэр хоосон байж болохгүй")

        if price <= 0:
            raise ValidationError("Үнэ 0-ээс их байх ёстой")

        try:
```

```
            product = Product(
                name=name.strip(),
                price=price,
                description=description.strip() if description else None
            )
            db.session.add(product)
            db.session.commit()
            return product

        except Exception as e:
            db.session.rollback()
            raise DatabaseError("Бүтээгдэхүүн үүсгэхэд алдаа гарлаа") from e

    @staticmethod
    def update_product(product_id, **kwargs):
        """Бүтээгдэхүүн шинэчлэх"""
        product = ProductService.get_product(product_id)

        if name in kwargs and not kwargs[name].strip():
            raise ValidationError("Бүтээгдэхүүний нэр хоосон байж болохгүй")

        if price in kwargs and kwargs[price] <= 0:
            raise ValidationError("Үнэ 0-ээс их байх ёстой")

        try:
            for key, value in kwargs.items():
                if hasattr(product, key):
                    setattr(product, key, value)
```

```
            db.session.commit()
            return product

        except Exception as e:
            db.session.rollback()
            raise DatabaseError("Бүтээгдэхүүн шинэчлэхэд алдаа гарлаа") from e
```

API endpoint-ууд

```
python
# routes/products.py
from flask import Blueprint, request, jsonify
from services.product_service import ProductService
from utils.error_handler import handle_errors
from exceptions.base import ValidationError

products_bp = Blueprint(products, __name__)

@products_bp.route(/api/products, methods=[POST])
@handle_errors
def create_product():
    """Шинэ бүтээгдэхүүн үүсгэх"""
    data = request.get_json()

    if not data:
        raise ValidationError("JSON өгөгдөл шаардлагатай")

    product = ProductService.create_product(
        name=data.get(name),
```

```
        price=data.get(price),
        description=data.get(description)
    )

    return jsonify({
        message:   Бүтээгдэхүүн амжилттай үүслээ,
        product: {
            id: product.id,
            name: product.name,
            price: product.price,
            description: product.description
        }
    }), 201


@products_bp.route(/api/products/<int:product_id>, methods=[GET])
@handle_errors
def get_product(product_id):
    """Бүтээгдэхүүн авах"""
    product = ProductService.get_product(product_id)
    return jsonify({
        product: {
            id: product.id,
            name: product.name,
            price: product.price,
            description: product.description
        }
    })
@products_bp.route(/api/products/<int:product_id>, methods=[PUT])
```

```
@handle_errors
def update_product(product_id):
    """Бүтээгдэхүүн шинэчлэх"""
    data = request.get_json()

    if not data:
        raise ValidationError("JSON өгөгдөл шаардлагатай")

    product = ProductService.update_product(product_id, **data)
    return jsonify({
        message:   Бүтээгдэхүүн амжилттай шинэчлэгдлээ,
        product: {
            id: product.id,
            name: product.name,
            price: product.price,
            description: product.description
        }
    })
```

Алдааны лог хийх

```
python
# utils/logger.py
import logging
from datetime import datetime

def setup_logger():
    """Логгер тохируулах"""
    logger = logging.getLogger(app)
    logger.setLevel(logging.INFO)
```

```
    # Лог файл үүсгэх
    file_handler = logging.FileHandler(flogs/app_{datetime.now().strftime("%Y%m%d")}.log)
    file_handler.setLevel(logging.ERROR)

    # Консолд хэвлэх
    console_handler = logging.StreamHandler()
    console_handler.setLevel(logging.INFO)

    # Формат тохируулах
    formatter = logging.Formatter(%(asctime)s  - %(name)s - %(levelname)s - %(message)s)
    file_handler.setFormatter(formatter)
    console_handler.setFormatter(formatter)

    logger.addHandler(file_handler)
    logger.addHandler(console_handler)
    return logger
# Глобал логгер
logger = setup_logger()
```

Дүгнэлт

Энэ долоо хоногт бид дараах зүйлсийг сурсан:

1. **Гадаад пакетүүдийг** хэрхэн менежмент хийх, суурилуулах
2. **Алдаа илрүүлэх** төрөл бүрийн аргууд
3. **Custom exception классууд** үүсгэх
4. **Хэрэглэгчид ойлгомжтой** алдааны мэдээлэл харуулах
5. **Алдааны лог** хийх, боловсруулах

Алдааны зөв боловсруулалт нь аппликейшний найдвартай байдал, хэрэглэгчийн туршлагыг сайжруулахад чухал үүрэг гүйцэтгэдэг.