POLITECHNIKA WROCŁAWSKA
WYDZIAŁ INFORMATYKI I TELEKOMUNIKACJI

PROJEKT Z BAZ DANYCH

Temat projektu

Termin zajęć: Czwartek, 13:15 - 15:00

AUTOR/AUTORZY:
Opis słowny:
Krzysztof Pawlaczek,
Yaroslav Starzhynskyi,
Jakub Wójcik

PROWADZĄCY ZAJĘCIA:
dr inż. Konrad Kluwak, W4N/K28

Wymagania:
Igor Adamowicz,
Wojciech Nachaj,
Ziemowit Popławski,
Yuliya Sodal

Modele bazy:
Olha Denysiuk,
Karol Gębski,
Paweł Gryszczyk,
Natan Lazar

Projekt aplikacji:
Kamil Dyrek,
Dawid Pimpicki,
Vitalii Protsiuk,
Oleg Yaroshevich

Wrocław, 2025 r.

Spis treści:

1. Wstęp

1.1. Cel projektu

System rekrutacyjny dla uczelni, umożliwiający zarządzanie danymi kandydatów, przetwarzanie dokumentów oraz generowanie raportów.

1.2. Zakres projektu

Obejmuje przechowywanie informacji o kandydatach, filtrowanie, generowanie raportów oraz automatyzację procesów rekrutacyjnych, w tym zarządzanie dokumentami i statusem kandydatów

2. Analiza wymagań

Wybór i opracowanie wstępnych założeń dotyczących wybranych tematów projektów.

2.1. Opis działania i schemat logiczny systemu

System rekrutacyjny składa się z kilku procesów, w tym składania dokumentów przez kandydatów, przyjmowania ich na kierunki, ustalania progów rekrutacyjnych oraz tworzenia raportów i obliczania wolnych miejsc

2.2. Wymagania funkcjonalne

- Baza danych studentów rekrutowanych
- Archiwizacja
- Filtrowanie kandydatów wg kryteriów
- Ustawianie priorytetów i wyświetlanie kierunków w kolejności ustalonych priorytetów
- Generowanie raportu liczby zakwalifikowanych w formie odpowiedniej dla dziekana
- Obliczenie miejsc wolnych i stworzenie drugiej tury
- Generowanie podpisów cyfrowych na przesyłanych dokumentach
- Status płatności, czy kandydat zapłacił czy nie
- Odpowiednie zabezpieczenia. Automatycznie tworzenie kopii zapasowych.
- Termin przekazania dokumentów
- Obsługa błędów

2.3. Wymagania niefunkcjonalne

- Baza danych osób rekrutowanych osób rekrutowanych MySQL
- Interfejs aplikacji (frontend) o Python, Javascript
- Wewnętrzny system obsługi aplikacji (backend) oparty na Django framework

2.3.1. Wykorzystywane technologie i narzędzia

Baza danych MySQL, Frontend: Python, JS, Backend: Django framework

2.3.2. Wymagania dotyczące rozmiaru bazy danych

Brak.

2.3.3. Wymagania dotyczące bezpieczeństwa systemu

Generowanie podpisów cyfrowych, automatyczne tworzenie kopii zapasowych, szyfrowanie dokumentów, bezpieczne przechowywanie haseł w bazie danych (np. z użyciem bcrypt), bezpieczna autentykacja.

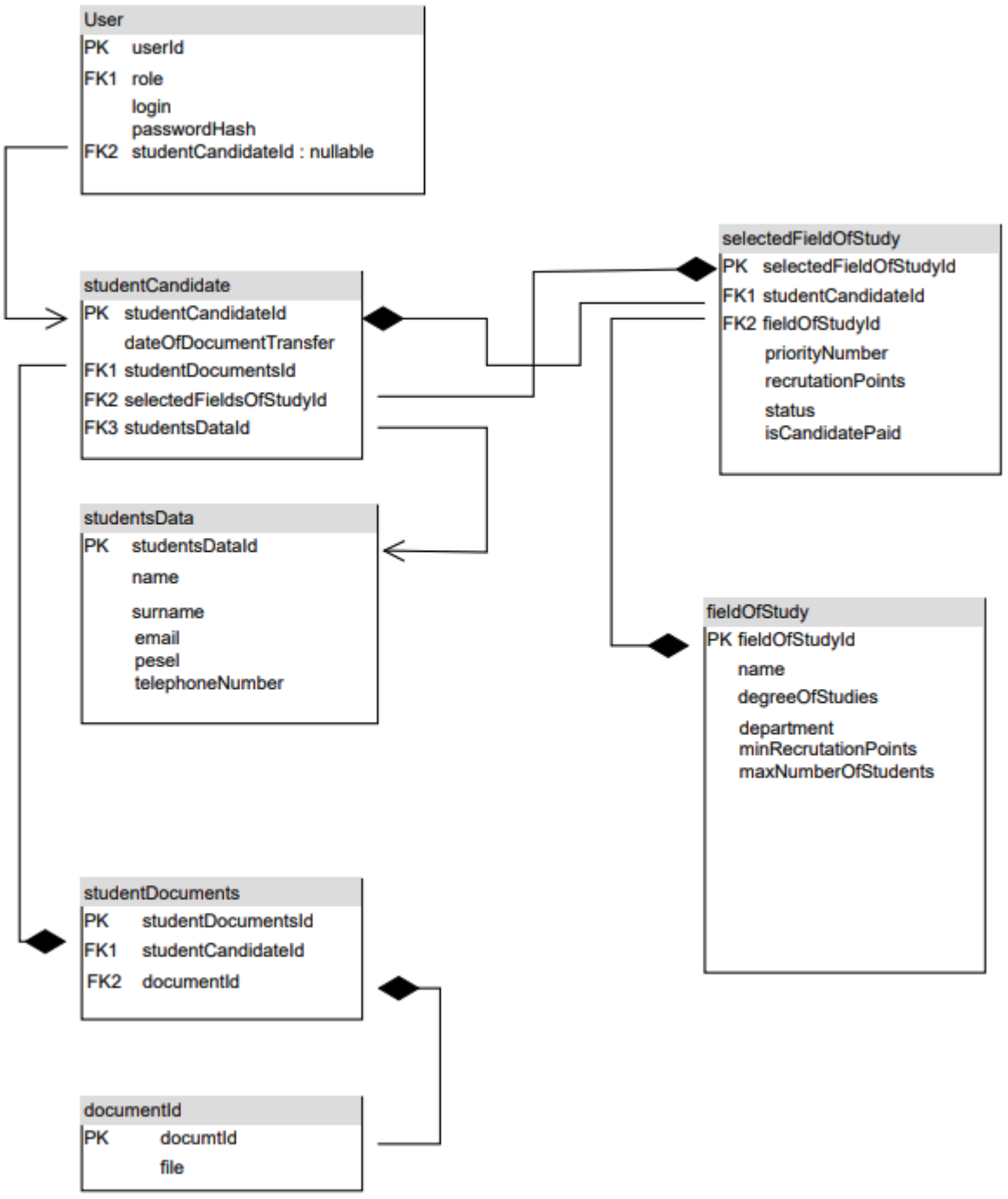
2.4. Przyjęte założenia projektowe

3. Projekt systemu

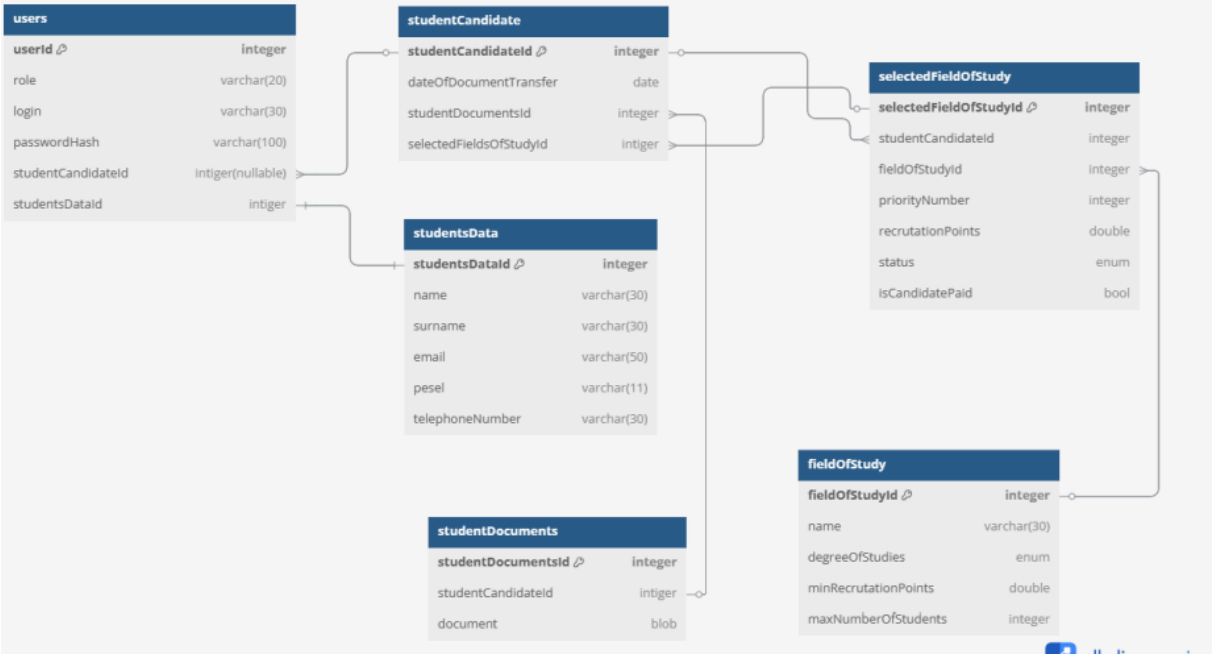
Projekt i struktury bazy danych, mechanizmów zapewniania poprawności przechowywanych informacji, oraz kontroli dostępu do danych.

3.1. Projekt bazy danych

3.1.1. Analiza rzeczywistości i uproszczony model konceptualny



3.1.2. Model logiczny i normalizacja



3.1.3. Model fizyczny i ograniczenia integralności danych

```
CREATE TABLE users (
   userId INT AUTO_INCREMENT PRIMARY KEY,
   role VARCHAR(20) NOT NULL,
   login VARCHAR(30) NOT NULL UNIQUE,
   passwordHash VARCHAR(100) NOT NULL,
   studentCandidateId INT,
   studentsDataId INT NOT NULL,
   FOREIGN KEY (studentCandidateId) REFERENCES studentCandidate(studentCandidateId),
   FOREIGN KEY (studentsDataId) REFERENCES studentsData(studentsDataId)
);
```

Tabela users przechowuje dane logowania oraz rolę użytkownika w systemie, wiążąc go z danymi osobowymi i statusem rekrutacyjnym.

```
CREATE TABLE studentCandidate (
   studentCandidateId INT AUTO_INCREMENT PRIMARY KEY,
   dateOfDocumentTransfer DATE,
   studentDocumentsId INT NOT NULL,
   selectedFieldsOfStudyId INT,
   FOREIGN KEY (studentCandidateId) REFERENCES studentDocuments(studentCandidateId)
);
```

Tabela studentCandidate przechowuje dane kandydata w procesie rekrutacyjnym, w tym datę złożenia dokumentów i wybory kierunków.

```
CREATE TABLE studentsData (
   studentsDataId INT AUTO_INCREMENT PRIMARY KEY,
   name VARCHAR(30) NOT NULL,
   surname VARCHAR(30) NOT NULL,
   email VARCHAR(50) NOT NULL,
   pesel VARCHAR(11),
   telephoneNumber VARCHAR(30) NOT NULL
);
```

Tabela studentsData zawiera podstawowe dane osobowe kandydatów, takie jak imię, nazwisko, email i numer telefonu.

```
CREATE TABLE studentDocuments (
   studentDocumentsId INT AUTO_INCREMENT PRIMARY KEY,
   studentCandidateId INT NOT NULL,
   document BLOB NOT NULL,
   FOREIGN KEY (studentCandidateId) REFERENCES studentCandidate(studentCandidateId)
);
```

Tabela studentDocuments przechowuje elektroniczne wersje dokumentów dostarczonych przez kandydatów w formacie binarnym (BLOB).

```
CREATE TABLE selectedFieldOfStudy (
   selectedFieldOfStudyId INT AUTO_INCREMENT PRIMARY KEY,
   studentCandidateId INT NOT NULL,
   fieldOfStudyId INT NOT NULL,
   priorityNumber INT NOT NULL,
   recrutationPoints DOUBLE NOT NULL,
   status ENUM('kandydat', 'oczekujący', 'odrzucony', 'zakwalifikowany', 'niezakwalifikowany')
NOT NULL,
   isCandidatePaid BOOLEAN NOT NULL,
   FOREIGN KEY (studentCandidateId) REFERENCES studentCandidate(studentCandidateId),
   FOREIGN KEY (fieldOfStudyId) REFERENCES fieldOfStudy(fieldOfStudyId)
);
```

Tabela selectedFieldOfStudy przechowuje informacje o wybranych przez kandydata kierunkach, ich priorytetach, punktacji i statusie w procesie rekrutacji.

```
CREATE TABLE fieldOfStudy (
   fieldOfStudyId INT AUTO_INCREMENT PRIMARY KEY,
   name VARCHAR(30) NOT NULL,
   degreeOfStudies ENUM('inz', 'mgr ', 'dr') NOT NULL,
   minRecrutationPoints DOUBLE NOT NULL,
   maxNumberOfStudents INT NOT NULL
);
```

Tabela fieldOfStudy definiuje dostępne kierunki studiów, ich nazwę, stopień oraz progi punktowe i limity miejsc.

3.1.4. Inne elementy schematu – mechanizmy przetwarzania danych

Dodawanie danych do tabel

INSERT INTO users (role, login, passwordHash, studentCandidateId, studentsDataId) VALUES (?);

INSERT INTO studentCandidate (dateOfDocumentTransfer, studentDocumentsId, selectedFieldsOfStudyId) VALUES (?);

INSERT INTO studentsData (name, surname, email, pesel, telephoneNumber) VALUES (?);

INSERT INTO studentDocuments (studentCandidateId, document) VALUES (?);

INSERT INTO selectedFieldOfStudy (studentCandidateId, fieldOfStudyId, priorityNumber, recrutationPoints, status, isCandidatePaid) VALUES (?);

INSERT INTO fieldOfStudy (name, degreeOfStudies, minRecrutationPoints, maxNumberOfStudents) VALUES (?);

● Archiwizacja za wyborem kandydata

(w tym przedstawiono stworzenie analogicznych tabelek tylko archiwalnych po czym jeżeli student chce żeby jego dane były skopiowane do archiwum odznacza to, i program każdy dzień o 2:00 sprawdza co jest odznaczone haczykiem i przenosze to do archiwum po czym haczyk jest usuwany)

CREATE TABLE studentsData_archive LIKE studentsData;

CREATE TABLE studentCandidate_archive LIKE studentCandidate;

CREATE TABLE selectedFieldOfStudy_archive LIKE selectedFieldOfStudy;

CREATE TABLE studentDocuments_archive LIKE studentDocuments;

(stworzenie kolumny z zaznaczeniem flagi dla archiwizacji)

ALTER TABLE studentCandidate

ADD COLUMN archiveRequested TINYINT(1) NOT NULL DEFAULT 0;

(procedura kopiowania)

DELIMITER $$

CREATE PROCEDURE archiveRequestedCandidates()

BEGIN

//dane osobowe

INSERT IGNORE INTO studentsData_archive

SELECT sd.*

FROM studentsData AS sd

JOIN studentCandidate AS sc

```
ON sd.studentsDataId = sc.studentsDataId
WHERE sc.archiveRequested = 1;
//wiersz kandydata
INSERT IGNORE INTO studentCandidate_archive
SELECT *
FROM studentCandidate
WHERE archiveRequested = 1;
//
INSERT IGNORE INTO selectedFieldOfStudy_archive
SELECT *
FROM selectedFieldOfStudy
WHERE studentCandidateId IN (
SELECT studentCandidateId
FROM studentCandidate
WHERE archiveRequested = 1
);
//Dokumenty
INSERT IGNORE INTO studentDocuments_archive
SELECT *
FROM studentDocuments
WHERE studentCandidateId IN (
SELECT studentCandidateId
FROM studentCandidate
WHERE archiveRequested = 1
);
// restart flagi
UPDATE studentCandidate
SET archiveRequested = 0
WHERE archiveRequested = 1;
END$$
DELIMITER ;
(sprawdzenie kto zaznaczył flagę do archiwizacji codziennie o 2:00)
DELIMITER $$
CREATE EVENT ev_auto_archive
ON SCHEDULE
EVERY 1 DAY
```

STARTS CONCAT(CURDATE(), ' 02:00:00')

DO

CALL archiveRequestedCandidates();

$$

DELIMITER ;

● Automatyczne tworzenie kopii zapasowych

(Kopie zapasowa jako root każdego 1 dnia każdego miesiąca o 2:00)

0 2 1 * * /usr/bin/mysqldump -u root -p ”hasłoroot” RekrutacjaDB \ >

/backup/rekrutacja_$(date +\%F).sql

● Filtrowanie kandydatów wg kryteriów

SELECT studentsData.name, studentsData.surname,

selectedFieldOfStudy.isCandidatedPaid FROM (((studentsData INNER JOIN users ON

studentsData.studentsDataId = users.studentsDataId) INNER JOIN studentCandidate ON

users.studentCandidateId = studentCandidate.studentCandidateId) INNER JOIN

selectedFieldOfStudy ON studentCandidate.selectedFieldOfStudyId =

selectedFieldOfStudy.selectedFieldOfStudyId) WHERE

selectedFieldOfStudy.isCandidatedPaid = 0;

● Ustawianie priorytetów i wyświetlanie kierunków w kolejności

ustalonych priorytetów

SELECT selectedFieldOfStudy.isCandidatedPaid FROM ((((studentsData INNER JOIN

users ON studentsData.studentsDataId = users.studentsDataId) INNER JOIN

studentCandidate ON users.studentCandidateId = studentCandidate.studentCandidateId)

INNER JOIN selectedFieldOfStudy ON studentCandidate.selectedFieldOfStudyId =

selectedFieldOfStudy.selectedFieldOfStudyId) INNER JOIN fieldOfStudy ON

selectedFieldOfStudy.fieldOfStudyId = fieldOfStudy.fieldOfStudyId) WHERE users.login = ?

ORDER BY selectedFieldOfStudy.priorityNumber DESC;

● Generowanie raportu liczby zakwalifikowanych w formie

odpowiedniej dla dziekana

SELECT studentData.name, studentData.surname FROM ((((studentsData INNER JOIN

users ON studentsData.studentsDataId = users.studentsDataId) INNER JOIN

studentCandidate ON users.studentCandidateId = studentCandidate.studentCandidateId)

INNER JOIN selectedFieldOfStudy ON studentCandidate.selectedFieldOfStudyId =

selectedFieldOfStudy.selectedFieldOfStudyId) INNER JOIN fieldOfStudy ON

selectedFieldOfStudy.fieldOfStudyId = fieldOfStudy.fieldOfStudyId) WHERE

selectedFieldOfStudy.status = ‘zakwalifikowany’ ORDER BY

selectedFieldOfStudy.recrutationPoints DESC;

3.1.5. Projekt mechanizmów bezpieczeństwa na poziomie bazy danych

Mechanizmy bezpieczeństwa bazy danych obejmują:

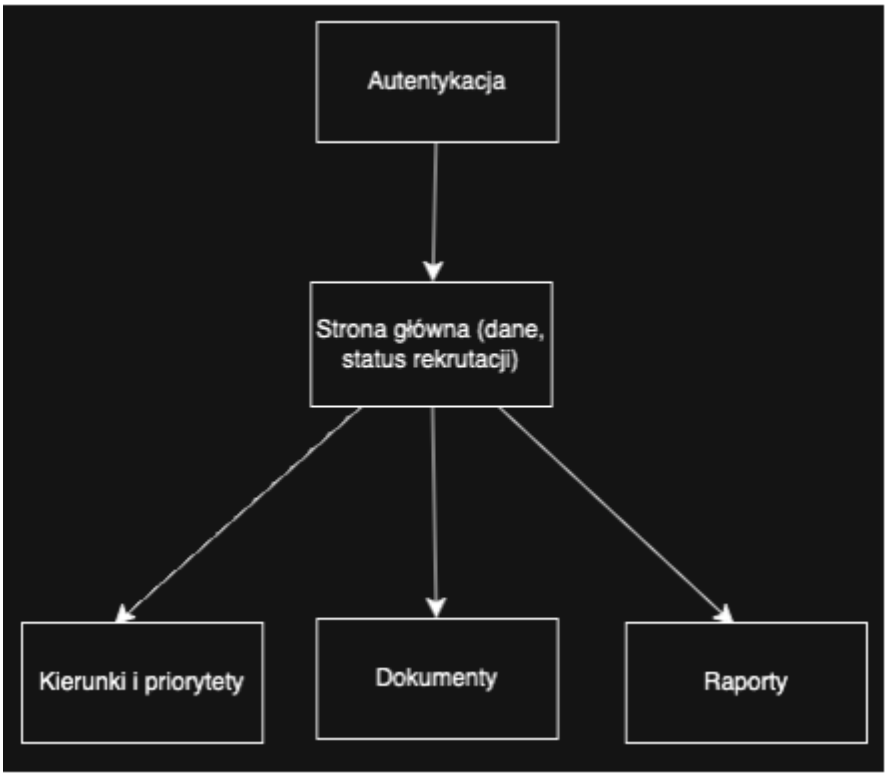
- generowanie podpisów cyfrowych dokumentów,
- automatyczne tworzenie kopii zapasowych (cron + mysqldump),
- archiwizację danych zgodnie z flagą użytkownika,
- bezpieczne przechowywanie haseł (passwordHash), co sugeruje użycie funkcji hashujących (np. bcrypt),
- kontrolę dostępu przez role użytkownika (role w tabeli users),
- obsługę błędów i unikanie duplikatów danych poprzez INSERT IGNORE.

3.2. Projekt aplikacji użytkownika

3.2.1. Architektura aplikacji i diagramy projektowe

Architektura trójwarstwowa (MCV):

- Frontend (View): aplikacja webowa napisana w Pythonie i JavaScript, zapewnia dostęp do funkcji systemu (formularze, listy, zarządzanie danymi).
- Backend (Controller): Django (Python), realizuje operacje na danych, autoryzację, obsługę żądań użytkowników, generowanie raportów.
- Baza danych (Model): MySQL – przechowuje dane kandydatów, kierunków studiów, statusów itd.



3.2.2. Interfejs graficzny i struktura menu

Link do projektu graficznego:

https://www.figma.com/proto/gLlKSuxoh5VHcOjdfgigSd/Projekt?node-id=5-379&t=2AcdaJ

3.2.3. Projekt wybranych funkcji systemu

Tworzenie bazy danych i tabel

System rekrutacyjny zakłada utworzenie relacyjnej bazy danych w MySQL, zawierającej znormalizowane tabele: użytkowników, kandydatów, danych osobowych, dokumentów, wybranych kierunków oraz kierunków studiów. Każda tabela zawiera klucze główne oraz powiązania za pomocą kluczy obcych, umożliwiające spójne przechowywanie danych.

Dodawanie danych do systemu

W aplikacji możliwe jest dodawanie danych kandydatów, takich jak dane osobowe, wybrane kierunki wraz z priorytetami, status płatności oraz przesłane dokumenty. Dane

te są rejestrowane w odpowiednich tabelach przy pomocy formularzy frontendowych, a następnie przesyłane do bazy danych poprzez Django.

Archiwizacja danych kandydatów
System umożliwia kandydatowi zaznaczenie opcji archiwizacji jego danych. Codziennie o godzinie 2:00 uruchamiana jest procedura, która automatycznie kopiuje dane wybranych kandydatów do odpowiednich tabel archiwalnych i resetuje znacznik archiwizacji.

Automatyczne tworzenie kopii zapasowych
W celu zapewnienia bezpieczeństwa danych, każdego 1. dnia miesiąca o godzinie 2:00 wykonywana jest automatyczna kopia zapasowa całej bazy danych przy użyciu narzędzia mysqldump. Kopia jest zapisywana w zdefiniowanym katalogu na serwerze.

Filtrowanie kandydatów wg kryteriów
Aplikacja pozwala kadrze rekrutacyjnej na filtrowanie kandydatów na podstawie zadanych kryteriów, takich jak status płatności, status rekrutacyjny czy wybrany kierunek. Ułatwia to analizę danych oraz obsługę procesów administracyjnych.

Ustalanie priorytetów kierunków
Kandydat może przypisać numery priorytetów do wybranych przez siebie kierunków studiów. System rekrutacyjny przechowuje te informacje, umożliwiając ich sortowanie i wyświetlanie w odpowiedniej kolejności podczas oceny aplikacji.

Generowanie raportów dla dziekana
Funkcja ta umożliwia wygenerowanie listy zakwalifikowanych kandydatów do poszczególnych kierunków w formie uporządkowanej według liczby punktów rekrutacyjnych. Raport może być dostarczony dziekanowi jako dokument PDF lub arkusz kalkulacyjny.

Wyliczanie liczby wolnych miejsc i uruchamianie II tury
Po zakończeniu pierwszej tury rekrutacji system analizuje liczbę zakwalifikowanych kandydatów w odniesieniu do maksymalnej liczby miejsc na kierunku. Jeśli są dostępne wolne miejsca, system automatycznie inicjuje drugą turę rekrutacji.

Generowanie podpisów cyfrowych
W celu zapewnienia autentyczności przesyłanych dokumentów kandydatów, system może generować podpisy cyfrowe, np. z użyciem algorytmów hashujących i certyfikatów, co zwiększa bezpieczeństwo operacji elektronicznych.

Status płatności
Każdy kandydat ma przypisany status opłaty rekrutacyjnej (isCandidatePaid). System uwzględnia ten status w procesie dalszej weryfikacji dokumentów i decyzji rekrutacyjnej. Tylko opłacone zgłoszenia są brane pod uwagę w procesie kwalifikacji.

Termin przekazania dokumentów
System umożliwia rejestrowanie daty złożenia dokumentów (dateOfDocumentTransfer). Dane te są używane do weryfikacji, czy kandydat dostarczył wymagane dokumenty w ustalonym terminie.

Obsługa błędów
System implementuje podstawowe mechanizmy obsługi błędów zarówno na froncie (walidacja formularzy), jak i w backendzie (obsługa wyjątków). Dzięki temu użytkownik otrzymuje komunikaty w przypadku nieprawidłowych danych lub błędów aplikacji.

3.2.4. Metoda podłączania do bazy danych – integracja z bazą danych

Django umożliwia w łatwy sposób integrację z serwerem bazodanowym MySQL. W ustawieniach aplikacji Django w pliku settings.py konfigurujemy dostęp. Źródło i dokładny opis: https://www.geeksforgeeks.org/how-to-integrate-mysql-database-with-django/

3.2.5. Projekt zabezpieczeń na poziomie aplikacji

Uwierzytelnianie: logowanie na konta z różnymi rolami (admin, kandydat) – tabela users Hasła: przechowywane jako passwordHash (bcrypt) Uprawnienia: kontrola dostępu w Django przez @login_required, @permission_required Kopia zapasowa: Cron: 0 2 1 * * /usr/bin/mysqldump -u root -p"hasłoroot" > /backup/rekrutacja.sql Walidacja danych w formularzach Szyfrowanie przesyłanych dokumentów Obsługa błędów: np. komunikaty o błędach walidacji lub błędach serwera

4. Implementacja systemu baz danych

Implementacja i testy bazy danych w wybranym systemie zarządzania bazą danych.

4.1. Tworzenie tabel i definiowanie ograniczeń

4.2. Implementacja mechanizmów przetwarzania danych

4.3. Implementacja uprawnień i innych zabezpieczeń

4.4. Testowanie bazy danych na przykładowych danych

5. Implementacja i testy aplikacji

Skrócone sprawozdanie z etapu implementacja i testowania aplikacji.

5.1. Instalacja i konfigurowanie systemu

5.2. Instrukcja użytkowania aplikacji

5.3. Testowanie opracowanych funkcji systemu

5.4. Omówienie wybranych rozwiązań programistycznych

5.4.1. Implementacja interfejsu dostępu do bazy danych

5.4.2. Implementacja wybranych funkcjonalności systemu