Міністерство освіти і науки України
Національний технічний університет України
“Київський політехнічний інститут імені Ігоря Сікорського”
Факультет інформатики та обчислювальної техніки
Кафедра автоматики та управління в технічних системах

Лабораторна робота №2

Технології розроблення програмного забезпечення

Діаграма варінтів використання. Сценарії варінтів використання. Діаграми UML. Діаграми класів. Концептуальна модель системи

Варіант 10(VCS all-in-one)

Виконав
студент групи ІА-13:
Луценко Юлія

Перевірив:
Драган Михайло

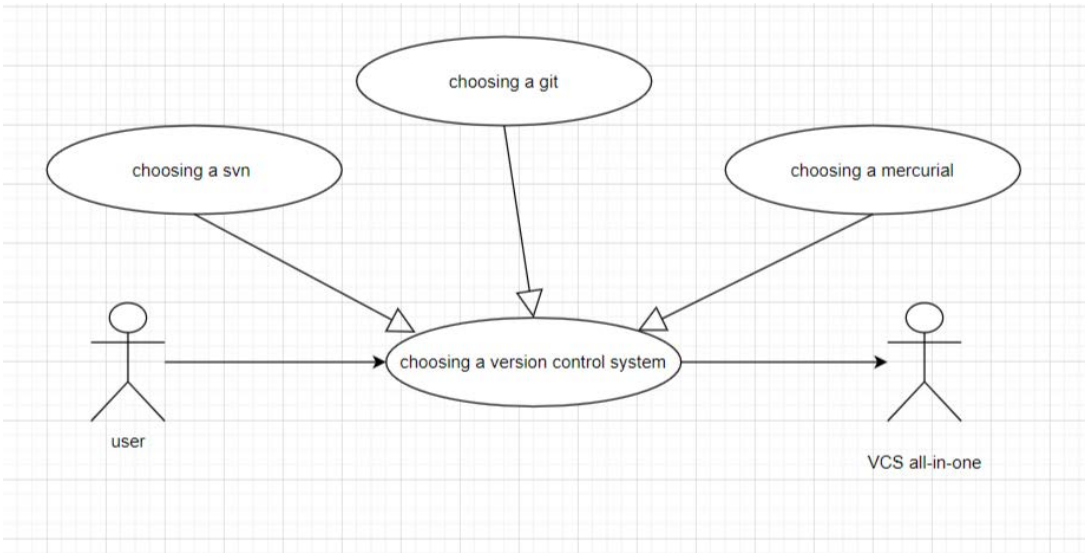
Київ 2023

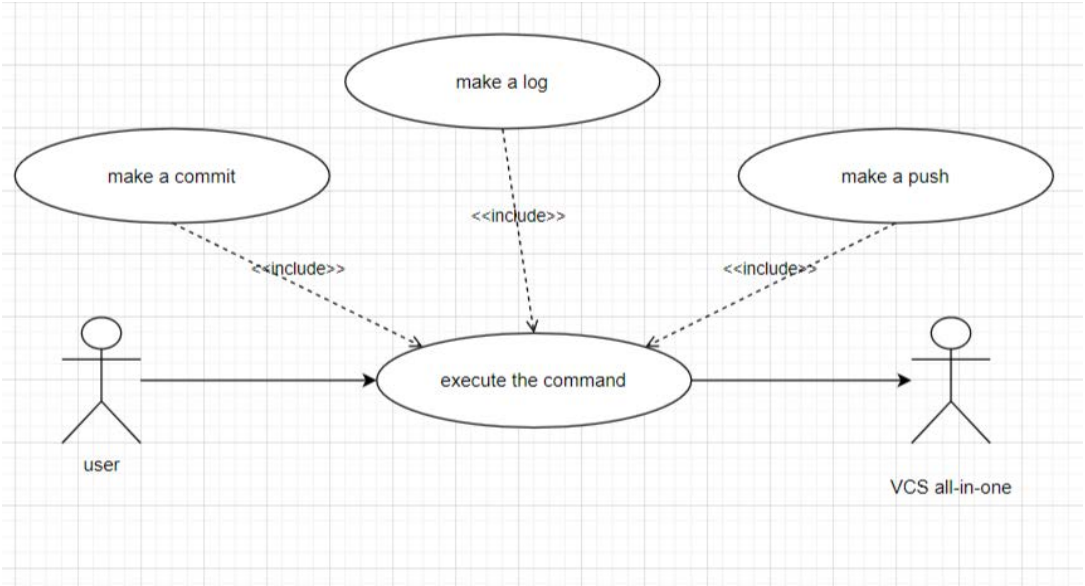
Тема: Діаграма варінтів використання. Сценарії варінтів використання. Діаграми UML. Діаграми класів. Концептуальна модель системи.

Мета: навчитися створювати діаграму варіантів використання, сценарії варіантів використання, UML діаграми, діаграми класів, концептуальну модель системи.

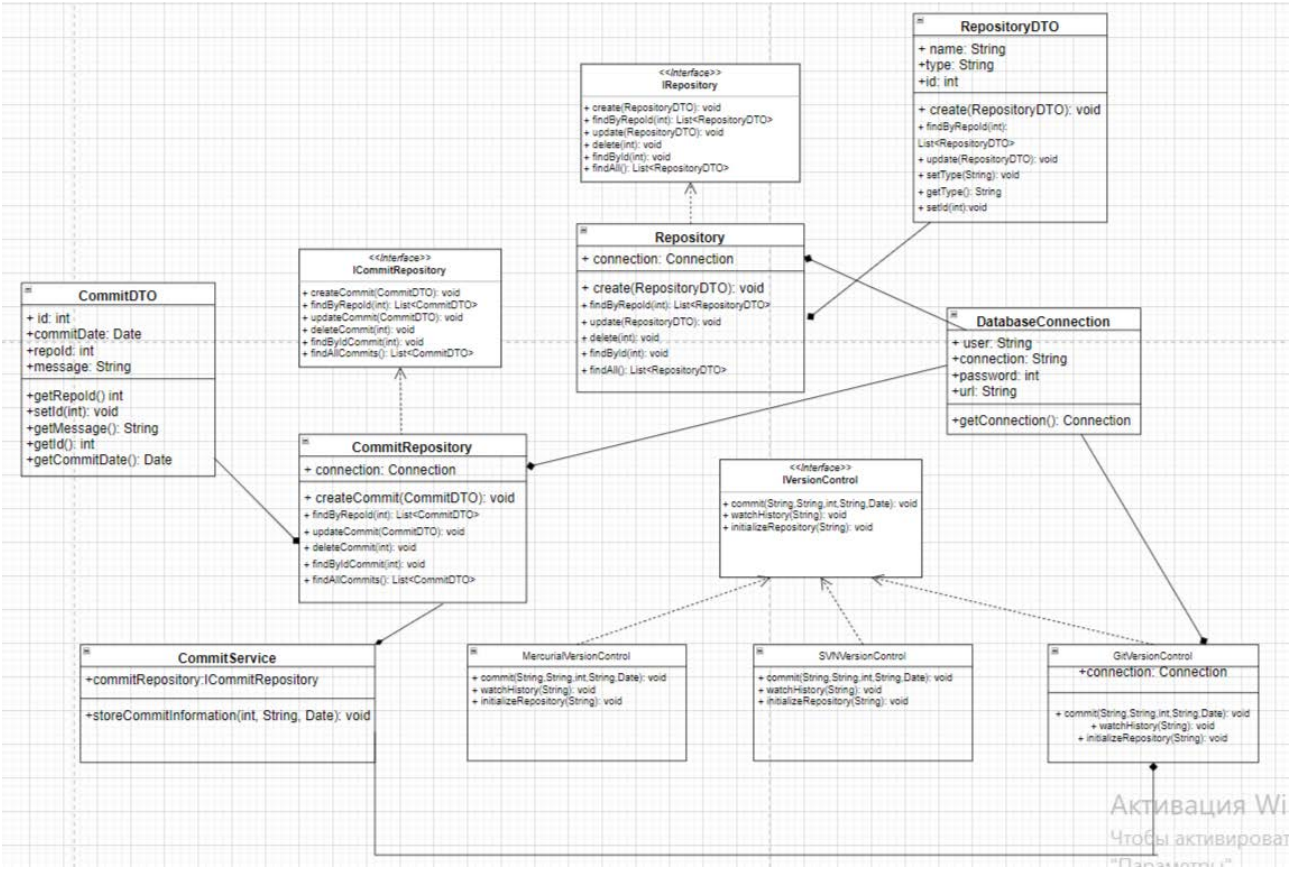
Хід роботи:

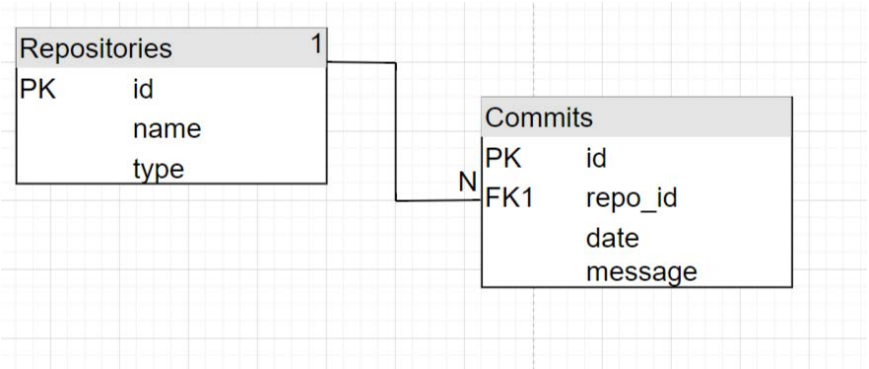
1. Схема прецедентів






user
keep a register of repositories
VCS all-in-one


RepositoryDTO
+ name: String
+type: String
+id: int
+ create(RepositoryDTO): void
+ findByRepoId(int): List<RepositoryDTO>
+ update(RepositoryDTO): void
+ setType(String): void
+ getType(): String
+ setId(int):void
<<Interface>>
IRepository
+ create(RepositoryDTO): void
+ findByRepoId(int): List<RepositoryDTO>
+ update(RepositoryDTO): void
+ delete(int): void
+ findById(int): void
+ findAll(): List<RepositoryDTO>
Repository
+ connection: Connection
+ create(RepositoryDTO): void
+ findByRepoId(int): List<RepositoryDTO>
+ update(RepositoryDTO): void
+ delete(int): void
+ findById(int): void
+ findAll(): List<RepositoryDTO>
<<Interface>>
ICommitRepository
+ createCommit(CommitDTO): void
+ findByRepoId(int): List<CommitDTO>
+ updateCommit(CommitDTO): void
+ deleteCommit(int): void
+ findByIdCommit(int): void
+ findAllCommits(): List<CommitDTO>
CommitDTO
+ id: int
+commitDate: Date
+repoId: int
+message: String
+getRepoId() int
+setId(int): void
+getMessage(): String
+getId(): int
+getCommitDate(): Date
DatabaseConnection
+ user: String
+connection: String
+password: int
+url: String
+getConnection(): Connection
CommitRepository
+ connection: Connection
+ createCommit(CommitDTO): void
+ findByRepoId(int): List<CommitDTO>
+ updateCommit(CommitDTO): void
+ deleteCommit(int): void
+ findByIdCommit(int): void
+ findAllCommits(): List<CommitDTO>
<<Interface>>
IVersionControl
+ commit(String,String,int,String,Date): void
+ watchHistory(String): void
+ initializeRepository(String): void
CommitService
+commitRepository:ICommitRepository
+storeCommitInformation(int, String, Date): void
MercurialVersionControl
+ commit(String,String,int,String,Date): void
+ watchHistory(String): void
+ initializeRepository(String): void
SVNVersionControl
+ commit(String,String,int,String,Date): void
+ watchHistory(String): void
+ initializeRepository(String): void
GitVersionControl
+connection: Connection
+ commit(String,String,int,String,Date): void
+ watchHistory(String): void
+ initializeRepository(String): void


Repositories
1
PK
id
name
type
N
Commits
PK
id
FK1
repo_id
date
message

```java
package Entities;


import java.util.Date;

public class CommitDTO {
    private int id;
    private int repoId;
    private String message;
    private final Date commitDate;

    public CommitDTO(int id, int repoId, String message, Date commitDate) {
        this.id = id;
        this.repoId = repoId;
        this.message = message;
        this.commitDate = commitDate;
    }
```

```java
package Entities;
public class RepositoryDTO {
    private int id;
    private String name;
    private String type;

    public RepositoryDTO(int id, String name, String type) {
        this.id = id;
        this.name = name;
        this.type = type;
    }

    public int getId() { return id; }

    public void setId(int id) { this.id = id; }

    public String getName() { return name; }

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

```java
package Repositories;

import ...

public interface ICommitRepository {
    CommitDTO findByIdCommit(int id);
    List<CommitDTO> findByRepoId(int repoId);
    void createCommit(CommitDTO commit);
    void updateCommit(CommitDTO commit);
    void deleteCommit(int id);
    List<CommitDTO> findAllCommits();
}
```

```java
package Services;

import ...

public class CommitService {
    private final ICommitRepository commitRepository;

    public CommitService(ICommitRepository commitRepository) {
        this.commitRepository = commitRepository;
    }

    public void storeCommitInformation(int repoId, String message, Date commitDate) {
        CommitDTO commit = new CommitDTO( id: 0,repoId,message,commitDate);
        commitRepository.createCommit(commit);
    }

    public List<CommitDTO> findCommits() { return commitRepository.findAllCommits(); }
```

```java
package Repositories;
import ...

public class Repository implements IRepository<RepositoryDTO> {
    private Connection connection;

    public Repository(Connection connection) { this.connection = connection; }

    @Override
    public RepositoryDTO findById(int id) {
        try {
            String sql = "SELECT * FROM repositories WHERE id = ?";
            PreparedStatement pstmt = connection.prepareStatement(sql);
            pstmt.setInt( parameterIndex: 1, id);
            ResultSet resultSet = pstmt.executeQuery();

            if (resultSet.next()) {
                int repoId = resultSet.getInt( columnLabel: "id");
                String name = resultSet.getString( columnLabel: "name");
                String type = resultSet.getString( columnLabel: "type");

                return new RepositoryDTO(repoId, name, type);
            }
        } catch (SQLException e) {
            e.printStackTrace();
        }

        return null;
    }
```

```java
@Override
public List<RepositoryDTO> findAll() {
    List<RepositoryDTO> repositories = new ArrayList<>();

    try {
        String sql = "SELECT * FROM repositories";
        PreparedStatement pstmt = connection.prepareStatement(sql);
        ResultSet resultSet = pstmt.executeQuery();

        while (resultSet.next()) {
            int repoId = resultSet.getInt( columnLabel: "id");
            String name = resultSet.getString( columnLabel: "name");
            String type = resultSet.getString( columnLabel: "type");

            repositories.add(new RepositoryDTO(repoId, name, type));
        }
    } catch (SQLException e) {
        e.printStackTrace();
    }

    return repositories;
}

@Override
public void create(RepositoryDTO entity) {
    try {
        String sql = "INSERT INTO repositories (name, type) VALUES (?, ?)";
```

```java
@Override
public void update(RepositoryDTO entity) {
    try {
        String sql = "UPDATE repositories SET name = ?, type = ? WHERE id = ?";
        PreparedStatement pstmt = connection.prepareStatement(sql);
        pstmt.setString( parameterIndex: 1, entity.getName());
        pstmt.setString( parameterIndex: 2, entity.getType());
        pstmt.setInt( parameterIndex: 3, entity.getId());
        pstmt.executeUpdate();
    } catch (SQLException e) {
        e.printStackTrace();
    }
}

@Override
public void delete(int id) {
    try {
        String sql = "DELETE FROM repositories WHERE id = ?";
        PreparedStatement pstmt = connection.prepareStatement(sql);
        pstmt.setInt( parameterIndex: 1, id);
        pstmt.executeUpdate();
    } catch (SQLException e) {
        e.printStackTrace();
    }
```

```java
package GitVersionControl;

import java.util.Date;

public interface IVersionControlSystem {
    void commit(String repoName,String fileName, int repoId, String message, Date commitDate);
    void watchHistory(String repoName);
    void initializeRepository(String repoDirectory);
}
```

```java
package GitVersionControl;

import ...

public class GitVersionControl implements IVersionControlSystem {
    private final String gitExecutablePath;
    private Connection connection;

    public GitVersionControl(Connection connection, String gitExecutablePath) {
        this.connection = connection;
        this.gitExecutablePath = gitExecutablePath;
    }

    @Override
    public void commit(String repoName, String fileName, int repoId, String message, Date commitDate) {
        String repoDirectory = "D:\\ТПР3\\" + repoName;

        try {
            // Initialize the Git repository using JGit
            Repository repository = new FileRepositoryBuilder()
                    .setGitDir(new File( pathname: repoDirectory + "/.git"))
                    .readEnvironment() // Consider environment variables
                    .findGitDir()
                    .build();

            Git git = new Git(repository);
            git.add().addFilepattern(fileName).call();
            git.commit().setMessage(message).call();
            git.close();
```



```java
package GitVersionControl;

import java.util.Date;

public class SVNVersionControl implements IVersionControlSystem {
    @Override
    public void commit(String repoName,String fileName, int repoId, String message, Date commitDate) {
        // Implement SVN commit logic
    }

    @Override
    public void watchHistory(String repoName) {
        // Implement SVN history display logic
    }

    @Override
    public void initializeRepository(String repoDirectory) {
    }
}
```

```java
package GitVersionControl;

import java.util.Date;

public class MercurialVersionControl implements IVersionControlSystem {
    @Override
    public void commit(String repoName,String fileName, int repoId, String message, Date commitDate) {
        // Implement Git commit logic
    }

    @Override
    public void watchHistory(String repoName) {
        // Implement Git history display logic
    }
    @Override
    public void initializeRepository(String repoDirectory) {
    }
}
```

```java
package com.company;

import ...

public class Main {

        public static void main(String[] args) {
            try {
                Connection dbConnection = DatabaseConnection.getConnection();
                String repoDirectory = "/D:/ТПР3/myRepo";
                String gitExecutablePath = "C:\\Program Files\\Git\\cmd\\git.exe";
//              SVNVersionControl svn = new SVNVersionControl();
                GitVersionControl git = new GitVersionControl(dbConnection,gitExecutablePath);
//              MercurialVersionControl mercurial = new MercurialVersionControl();
                git.initializeRepository(repoDirectory);
                git.commit( repoName: "myRepo",  fileName: "jj.txt", repoId: 0,  message: "Git commit2",new Date());
                git.watchHistory( repoName: "myRepo");
            } catch (Exception e) {
                e.printStackTrace();
            }
        }
    }
```

```java
package DB;

import ...

public class DatabaseConnection {
    private static final String url = "jdbc:postgresql://localhost:5432/repositories";
    private static final String user = "postgres";
    private static final String password = "postgres";
    private static Connection connection;

    public static Connection getConnection() throws SQLException {
        if (connection == null || connection.isClosed()) {
            connection = DriverManager.getConnection(url, user, password);
        }
        if (connection != null) {
            System.out.println("Connected to the database successfully.");
        }
        return connection;
    }
}
```

Висновок: : я намалювала діаграму класів для реалізованої частини системи, вибрала прецеденти і написала на їх основі прецеденти, розробила основні класи і структуру системи баз даних.