

Проект “Розробка веб-порталу”

Виконали студенти групи БІ-1

Антоненко Вероніка,

Плакидюк Влад,

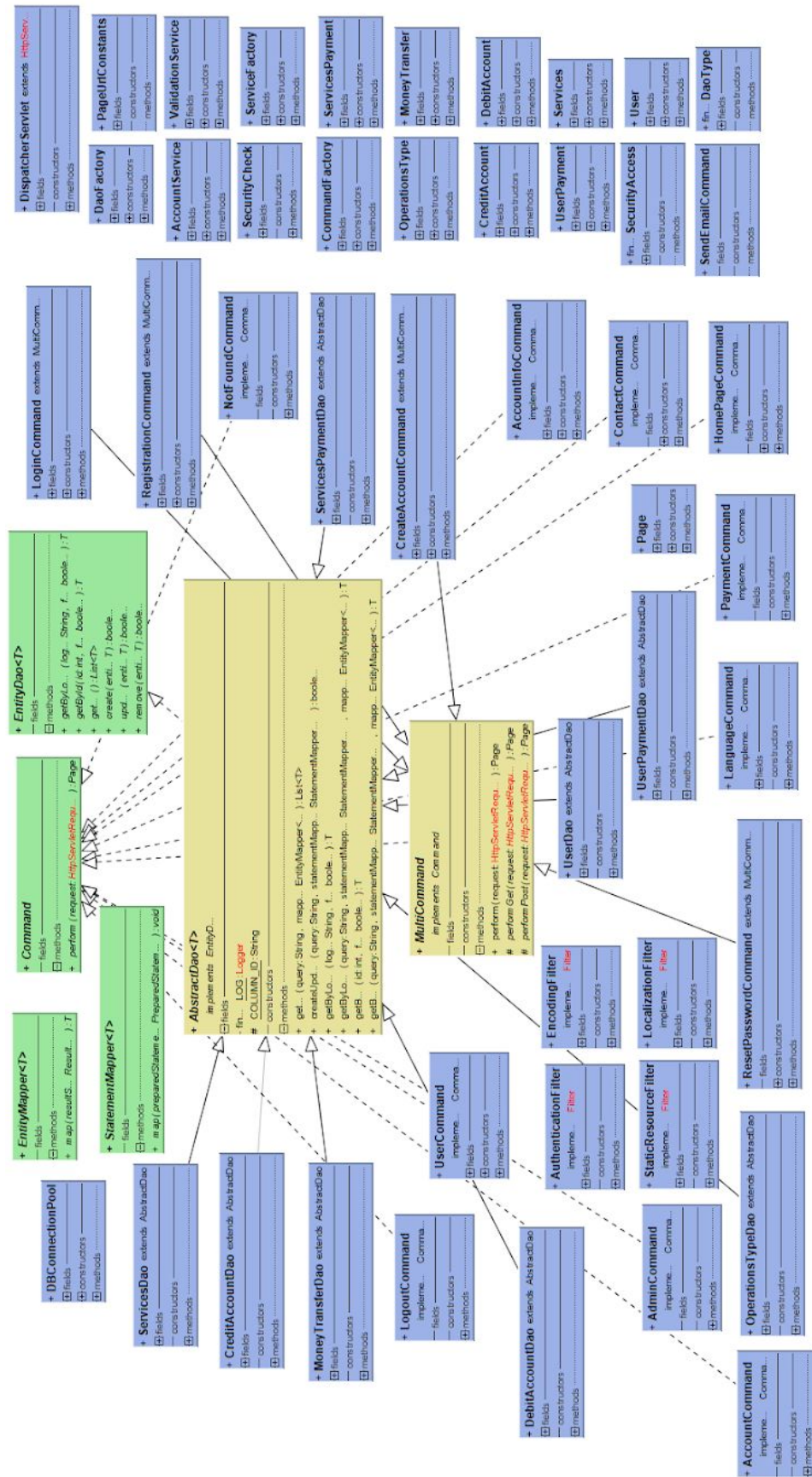
Франчук Іван

1. Встановлення та специфікація вимог

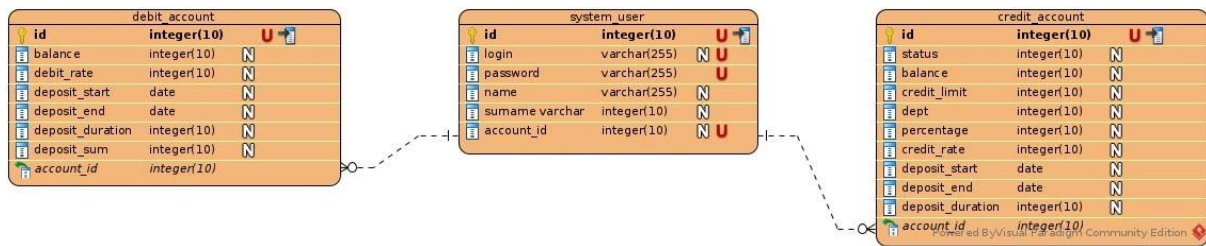
У якості предметної області була обрана розробка веб-порталу для банківського сервісу. Клієнти (користувачі), користуючись ресурсом мають змогу дистанційно відкривати кредитні та депозитні рахунки, переглядати дані про них. Відкриття депозитного рахунку відбувається миттєво, проте для відкриття кредитного акаунта необхідно дочекатися схвалення від адміна. Користувач повинен мати змогу редагувати свої депозитні акаунти та видаляти їх якщо потреба в них зникне. Веб-ресурс повинен мати певний рівень захисту та надавати доступ до конфіденційної інформації - як-то дані рахунків користувачів або перегляд їх акаунтів лише а) зареєстрованим користувачам і б) користувачам, яким було надане право доступу до подібних даних. На даному етапі розробки повними правами володіє лише один адміністратор. Дані про рахунки та клієнтів зберігаються у базі даних SQL, оновлюються у режимі реального часу.

2. Проектування архітектури.

Для наочного представлення основи майбутнього ресурсу було розроблено діаграму класів, представлену на малюнку нижче.



Діаграма Entity-Relationship демонструє базу даних.



3. Релізація

Проект було реалізовано мовою Java. Веб-додаток розроблено з використанням Servlets, Tomcat, візуальна частина виконана за допомогою JSP.

За зв'язок з базою даних відповідає клас BDConnectionPool. Для розробки програми можна використовувати пули різного типу об'єктів, наприклад пул потоків Thread Pool, але у даній роботі було використано пул підключень Connection Pool. Для отримання джерела даних (DataSource) використовується механізм JNDI.

Для ізолювання прикладного рівня програми від основного джерела даних було використано структурний шаблон Data Access Object (DAO) з використанням абстрактного API. Функціональність полягає в тому, щоб приховати від додатка всі процеси, пов'язані з виконанням операцій CRUD. Подібний підхід дозволяє дозволяє обом рівням розвиватися окремо, нічого не знаючи один про одного.

```
public class DBConnectionPool {
    private static final Logger LOG = Logger.getLogger(DBConnectionPool.class);
    private static DataSource dataSource;

    static {
        try {
            Context initContext = new InitialContext();
            dataSource = (DataSource)
            initContext.lookup("java:comp/env/jdbc/bank_system");
        } catch (NamingException e) {
            LOG.error("Could not find DataSource JNDI", e);
        }
    }

    private DBConnectionPool() {
    }

    public static Connection getConnection() {

        Connection connection = null;
```

```

        try {
            connection = dataSource.getConnection();
            LOG.debug("Connection received " + connection);
        } catch (SQLException e) {
            LOG.error("Some problem was occurred while getting connection to
Database", e);
        }
        return connection;
    }

    public static PreparedStatement getPreparedStatement(String query) throws
SQLException {
        return getConnection().prepareStatement(query);
    }
}

```

Таким чином для основних класів - User, CreditAccount, DebitAccount було розроблено відповідні DAO-класи. Було створено інтерфейс EntityDao з оголошеними основними методами CRUD операцій:

```

public interface EntityDao<T> {

    T getByLogin(String login, boolean full);

    T getById(int id, boolean full);

    List<T> getAll();

    boolean create(T entity);

    boolean update(T entity);

    boolean remove(T entity);

}

```

Даний інтерфейс був імплементований абстрактним класом AbstractDao<T> з універсальним параметром - Generics. Таким чином, усі наступні DAO-класи будуть наслідуватись від нього, незважаючи на свій тип.

```

public abstract class AbstractDao<T> implements EntityDao<T> {
    private static final Logger LOG = Logger.getLogger(AbstractDao.class);
    protected static String COLUMN_ID = "id";

    public List<T> getAll(String query, EntityMapper<T> mapper) {
        List<T> result = new ArrayList<>();
        try (PreparedStatement preparedStatement =
DBConnectionPool.getPreparedStatement(query);
            ResultSet resultSet = preparedStatement.executeQuery()) {

            while (resultSet.next()) {
                T entity = mapper.map(resultSet);
                result.add(entity);
            }

        } catch (SQLException e) {
            LOG.error("Error occurred while getting all entities.");
        }
    }
}

```

```

        }

        return result;
    }

    public boolean createUpdate(String query, StatementMapper<T> statementMapper) {
        try (PreparedStatement preparedStatement =
            DBConnectionPool.getPreparedStatement(query);) {
            statementMapper.map(preparedStatement);

            int result = preparedStatement.executeUpdate();

            return result == 1;
        } catch (SQLException e) {
            LOG.error("Could not create entity!!", e);
        }

        return false;
    }

    @Override
    public T getById(int id, boolean full) {
        return null;
    }

    public T getById(String query, StatementMapper<T> statementMapper,
        EntityManager<T> mapper) {
        T result = null;

        try (PreparedStatement preparedStatement =
            DBConnectionPool.getPreparedStatement(query);) {
            statementMapper.map(preparedStatement);
            try (ResultSet resultSet = preparedStatement.executeQuery()) {
                while (resultSet.next()) {
                    result = mapper.map(resultSet);
                }
            }
        } catch (SQLException e) {
            LOG.error("Exception while getting all entities", e);
        }

        return result;
    }
}

```

Наступним рівнем розробки було створення інтерфейсу веб-додатку. Інтерфейс можна побачити в додатку А разом з демонстрацією роботи веб-порталу.

Після цього необхідно було об'єднати прикладний рівень із зовнішнім, для цього було використано сервлети. Було створено `DispatcherServlet` який відповідає за отримання запиту від користувача та подальше направлення на відповідний контролер з командами, у якому й буде відбуватися вся обробка даних.

Команда являє собою наступний інтерфейс:

```

public interface Command {

```

```

    Page perform(HttpServletRequest request);
}

```

Дата виводиться на екран у вигляді сторінки - Page. Сторінка містить у собі актуальне посилання, на яке необхідно перейти та boolean параметр redirect - в залежності від того чи необхідно буде зробити перенаправлення чи ні.

Усі команди за замовчуванням виконують лише запити типу GET. Для розширення можливості команди до обробки обох запитів і GET і POST було створено інтерфейс MultiCommand.

```

public abstract class MultiCommand implements Command {

    @Override
    public Page perform(HttpServletRequest request) {
        String type = request.getMethod();

        return "GET".equals(type)
            ? performGet(request)
            : performPost(request);
    }

    protected abstract Page performGet(HttpServletRequest request);

    protected abstract Page performPost(HttpServletRequest request);
}

```

Усі подальші команди імплементують один з двох відповідних видів команд в залежності від того чи сторінка буде лише показувати дані, чи й обробляти їх, взаємодіяти з користувачем.

Таким чином було налаштовано зв'язок між внутрішньою та зовнішньою частинами додатку.

У фінальній частині було розроблено системи аутентифікації та верифікації, виконано Error Handling.

Сервіс для валідації користувача як зареєстрованого, такого який має доступ до конфіденційних ресурсів:

```

public class ValidationService {
    private static final Logger LOG = Logger.getLogger(ValidationService.class);
    private EntityDao<User> userDao;

    public ValidationService() {
        this.userDao = DaoFactory.getEntityDao(DaoType.USER);
    }

    public Optional<User> validateUser(String login, String password) {
        List<User> all = userDao.getAll();

        return all.stream()
            .filter(u -> u.getLogin().equals(login)
                && u.getPassword().equals(password))
            .findFirst();
    }

    public boolean validateLogin(String login) {

```

```

        List<User> users = userDao.getAll();
        for (User user : users) {
            if (user.getLogin().equals(login)) {
                LOG.debug("Login not validated: " + user.getLogin());
                return false;
            }
        }
        return true;
    }

    public boolean isExist(String login) {
        List<User> all = userDao.getAll();
        return all.stream()
            .anyMatch(u -> u.getLogin().equals(login));
    }

    public User registrationUser(String login, String password, String name, String
surname) {
        Random random = new Random();
        int accountId = random.nextInt(900) + 1000;
        User newUser = new User(login, password, name, surname, accountId,
SecurityAccess.USER);
        userDao.create(newUser);
        return newUser;
    }
}

```

З погляду безпеки, користувачі існують двох видів - звичайний користувач та адмін. Права доступу до ресурсів в них мають відрізнятися.

```

public enum SecurityAccess {
    ADMIN, USER
}

```

Тому користувачі класифікуються за цим розподілом, а сервіс перевірки необхідного рівня безпеки для доступу певного ресурсу блокує, наприклад, спробу користувача відкрити панель адміна, підібравши URL та схвалити власний запит на створення кредитного акаунта.

```

public class SecurityCheck {
    private static final Map<SecurityAccess, List<String>> securityPages = new
HashMap<>();

    static {
        securityPages.put(SecurityAccess.ADMIN, Arrays.asList("/account_info",
"/create_account", "/admin_account_info"));
        securityPages.put(SecurityAccess.USER, Arrays.asList("/account_info",
"/create_account", "/manage_accounts"));
    }

    public static boolean isSecurePage(String page) {
        return securityPages.values().stream()
            .anyMatch(list -> list.stream()
                .anyMatch(pageValue -> pageValue.equals(page)));
    }

    public static boolean hasPermission(String page, SecurityAccess securityAccess) {
        return securityPages.getOrDefault(securityAccess, Collections.EMPTY_LIST)
            .stream()
            .anyMatch(securePage -> securePage.equals(page));
    }

    public static boolean hasPermission(HttpServletRequest request, SecurityAccess
role) {

```



```

        User currentUser = getCurrentUser(request);
        return currentUser != null && currentUser.getSecurityAccess().equals(role);
    }

    public static User getCurrentUser(HttpServletRequest request) {
        return (User) request.getSession().getAttribute("user");
    }
}

```

Для аутентифікації користувачів, створення локалізації та зміни кодування було застосовано Intercepting Filter Pattern. Intercepting Filters - це фільтри, які виконують певні дії до або після обробки вхідного запиту. Фільтри перехоплення є централізовані компоненти в веб-додатку, загальні для всіх запитів і розгортаються, не зачіпаючи існуючі методи обробки. Було створено три фільтри: Localization Filter (для двох локалей - російської та англійської з можливістю зміни на кожній сторінці.

```

public class LocalizationFilter implements Filter {
    private static final Logger LOG = Logger.getLogger(LocalizationFilter.class);
    private static final String LOCALE = "locale";
    private static final String BUNDLE = "bundle";

    private String defaultLocale;
    private String defaultBundle;

    @Override
    public void init(FilterConfig filterConfig) throws ServletException {
        this.defaultLocale = filterConfig.getInitParameter(LOCALE);
        this.defaultBundle = filterConfig.getInitParameter(BUNDLE);
    }

    @Override
    public void doFilter(ServletRequest request, ServletResponse response,
        FilterChain chain)
        throws IOException, ServletException {
        HttpServletRequest httpRequest = (HttpServletRequest) request;
        HttpServletResponse httpResponse = (HttpServletResponse) response;
        HttpSession session = httpRequest.getSession();

        setLocale(session);
        setBundle(session);

        chain.doFilter(request, response);
    }

    private void setLocale(HttpSession session) {
        String locale = (String) session.getAttribute(LOCALE);
        if (locale == null) {
            LOG.debug("Set locale to session");
            LOG.debug("Set locale to session");
            session.setAttribute(LOCALE, defaultLocale);
        }
    }

    private void setBundle(HttpSession session) {
        String bundle = (String) session.getAttribute(BUNDLE);
        if (bundle == null) {
            LOG.debug("Set bundle to session");
            session.setAttribute(BUNDLE, defaultBundle);
        }
    }
}

```


Encoding Filter

```
public class EncodingFilter implements Filter {

    private static final String ENCODING_UTF_8 = "UTF-8";
    private static final String DEFAULT_CONTENT_TYPE = "text/html; charset=UTF-8";
    private static final String REQUEST_ENCODING = "requestEncoding";
    private String defaultEncoding;

    @Override
    public void init(FilterConfig config) throws ServletException {
        defaultEncoding = config.getInitParameter(REQUEST_ENCODING);
        if (defaultEncoding == null) {
            defaultEncoding = ENCODING_UTF_8;
        }
    }

    @Override
    public void doFilter(ServletRequest request, ServletResponse response,
        FilterChain chain)
        throws IOException, ServletException {
        if (request.getCharacterEncoding() == null) {
            request.setCharacterEncoding(defaultEncoding);
        }

        response.setContentType(DEFAULT_CONTENT_TYPE);
        response.setCharacterEncoding(defaultEncoding);

        chain.doFilter(request, response);
    }
}
```

Authentication Filter.

```
@WebFilter("/*")
public class AuthenticationFilter implements Filter {
    private static final Logger LOG = Logger.getLogger(AuthenticationFilter.class);

    @Override
    public void init(FilterConfig filterConfig) throws ServletException {
    }

    @Override
    public void doFilter(ServletRequest request, ServletResponse response,
        FilterChain chain)
        throws IOException, ServletException {
        HttpServletRequest httpRequest = (HttpServletRequest) request;
        HttpServletResponse httpResponse = (HttpServletResponse) response;

        String path = getPath(httpServletRequest);

        if (!SecurityCheck.isSecurePage(path)) {
            LOG.info("Page is not secured: " + path);
            chain.doFilter(request, response);
            return;
        }
        String contextPath = httpRequest.getContextPath();

        HttpSession session = httpRequest.getSession();
        User user = (User) session.getAttribute("user");
        if (user == null) {
            LOG.info("User not logged");
            httpResponse.sendRedirect(contextPath + LOGIN_PAGE);
            return;
        }
    }
}
```

```

    }

    boolean hasPermission = SecurityCheck.hasPermission(path,
user.getSecurityAccess());

    if (!hasPermission) {
        LOG.info("User does not have permission : " + user + " , " + path);
        httpServletResponse.sendRedirect(contextPath + NO_ACCESS_PAGE);
        return;
    }

    LOG.info("User has permission. Continue");
    chain.doFilter(request, response);
}

```

Порядок їх виклику зазначено у файлі web.xml

```

<filter>
    <filter-name>localizationFilter</filter-name>
    <filter-class>com.web.filter.LocalizationFilter</filter-class>
    <init-param>
        <param-name>locale</param-name>
        <param-value>en</param-value>
    </init-param>
    <init-param>
        <param-name>bundle</param-name>
        <param-value>messages</param-value>
    </init-param>
</filter>

<filter-mapping>
    <filter-name>localizationFilter</filter-name>
    <url-pattern>/*</url-pattern>
</filter-mapping>

<filter>
    <filter-name>encodingFilter</filter-name>
    <filter-class>com.web.filter.EncodingFilter</filter-class>
    <init-param>
        <param-name>requestEncoding</param-name>
        <param-value>UTF-8</param-value>
    </init-param>
</filter>

<filter-mapping>
    <filter-name>encodingFilter</filter-name>
    <url-pattern>/*</url-pattern>
</filter-mapping>

<filter>
    <filter-name>authenticationFilter</filter-name>
    <filter-class>com.web.filter.AuthenticationFilter</filter-class>
</filter>
<filter-mapping>
    <filter-name>authenticationFilter</filter-name>
    <url-pattern>/*</url-pattern>
</filter-mapping>

```

4. Демонстрація роботи веб-додатку

Головна сторінка

BANK SYSTEM


HOMECONTACTLOG IN

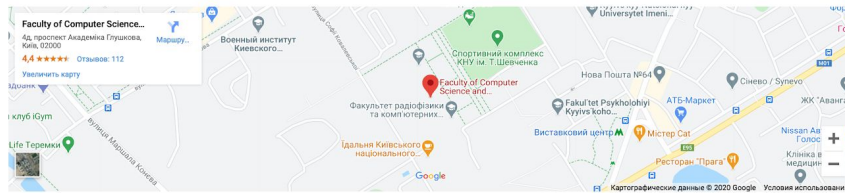


PROFESSIONAL TEAMOUR MAIN FOCUS IS YOUWORLDWIDE PARTNERSHIP

Сторінка контактів

BANK SYSTEM

HOMECONTACTLOG IN



Address 1

Факс:1-22-5555
Telephone:1-22-5555
bank_system@gmail.co

Address 2

Факс:1-22-5555
Telephone:1-22-4444
bank_system2@gmail.co

Contact Us

Name	<input type="text"/>
E-Mail	<input type="text"/>
Subject	<input type="text"/>
<input type="button" value="SUBMIT"/>	

Реєстрація нового користувача

Registration

☐ I Agree To The Terms & Conditions

SIGNUP

Already have an Account? [Login](#)

Вхід для вже зареєстрованих

Login Into Your Account

LOGIN

Don't have an account? [Create one now.](#)

Приклад - реєстрація та подальша робота з акаунтом

Registration

☒ I Agree To The Terms & Conditions

SIGNUP

Already have an Account? [Login](#)

BANK SYSTEM

[HOME](#)

[CONTACT](#)

[TEST ACCOUNT](#)

[LOGOUT](#)



TEST ACCOUNTS

[OPEN A NEW CREDIT ACCOUNT](#)
[OPEN A NEW DEBIT ACCOUNT](#)
[MANAGE ACCOUNTS](#)

INFORMATION

» [Contact](#)

My ACCOUNT

» [Your Account](#)

CONTACT US

+91-123-456789
+00-123-000000

Можна побачити що спочатку в юзера немає ніяких акаунтів. Тепер

створимо запит на створення кредитного рахунку

Open A New Credit Account

Choose Your Action:

Submit a request for a credit account

Натиснувши кнопку відправки ми можемо наочно спостерігати створений акаунт зі статусом REVIEWING - це означає що акаунт знаходиться на розгляді в адміна.



TEST ACCOUNTS

[OPEN A NEW CREDIT ACCOUNT](#)
[OPEN A NEW DEBIT ACCOUNT](#)
[MANAGE ACCOUNTS](#)

CREDIT ACCOUNT: **1245

Status: REVIEWING

Balance: 0\$

Credit Limit: 12\$

Current Debt: 0\$

Current percentage: 5%

Credit rate: 1

Account opened on: 2020-12-18

Total duration(months): 12

Користувач має змогу вносити зміни в свої акаунти, виконувати операції
CRUD

Manage Accounts

Choose Your Action:

☐ Delete account

☒ Update account

Тепер подивимося як виглядає контрольна панель у адміна - список усіх користувачів які подавали запити на створення акаунта з двома виборами під кожним - підтвердити (схвалити) чи з якихось причин відхилити запит

ADMIN REVIEWING ACCOUNTS INFO

NAME: VASYA	NAME: MARIA	NAME: VASYA	NAME: VASYA
SURNAME: PETROV	SURNAME: KRUT	SURNAME: PETROV	SURNAME: PETROV
CREDIT ACCOUNT:: **10	CREDIT ACCOUNT:: **22	CREDIT ACCOUNT:: **10	CREDIT ACCOUNT:: **10
Status: REVIEWING	Status: REVIEWING	Status: REVIEWING	Status: REVIEWING
Balance: 100\$	Balance: 12000\$	Balance: 0\$	Balance: 0\$
Credit Limit: 1000\$	Credit Limit: 25000\$	Credit Limit: 12\$	Credit Limit: 111\$
Current Debt: 90\$	Current Debt: 24999\$	Current Debt: 0\$	Current Debt: 0\$
Current percentage: 10%	Current percentage: 7%	Current percentage: 11%	Current percentage: 1%
Credit rate: 2	Credit rate: 4	Credit rate: 1	Credit rate: 1
Account opened on: 10-12-2019	Account opened on: 09-11-2018	Account opened on: 2020-12-29	Account opened on: 2020-12-31
Total duration(months): 12	Total duration(months): 24	Total duration(months): 11	Total duration(months): 122
<div>APPROVEDECLINE</div>	<div>APPROVEDECLINE</div>	<div>APPROVEDECLINE</div>	<div>APPROVEDECLINE</div>

А ось і створений нами тестовий запит. Підтвердимо його і побачимо що кнопки знизу зникли, адже робота адміна з цим акаунтом на цьому завершена.

NAME: TEST	NAME: TEST
SURNAME: TEST	SURNAME: TEST
CREDIT ACCOUNT:: **1245	CREDIT ACCOUNT:: **1245
Status: APPROVED	Status: APPROVED
Balance: 0\$	Balance: 0\$
Credit Limit: 12\$	Credit Limit: 12\$
Current Debt: 0\$	Current Debt: 0\$
Current percentage: 5%	Current percentage: 5%
Credit rate: 1	Credit rate: 1
Account opened on: 2020-12-18	Account opened on: 2020-12-18
Total duration(months): 12	Total duration(months): 12
<div>APPROVEDECLINE</div>	

На панелі користувача статус запиту також було змінено.

TEST ACCOUNTS

[OPEN A NEW CREDIT ACCOUNT](#)
[OPEN A NEW DEBIT ACCOUNT](#)
[MANAGE ACCOUNTS](#)

CREDIT ACCOUNT: **1245

Status: **APPROVED**

Balance: 0\$

Credit Limit: 12\$

Current Debt: 0\$

Current percentage: 5%

Credit rate: 1

Account opened on: 2020-12-18

Total duration(months): 12

DEPOSIT ACCOUNT: **1245

Balance: 0\$

Debit rate: 12%

Deposited sum: 1200\$

Account opened on: 2020-12-26

Total duration(months): 12