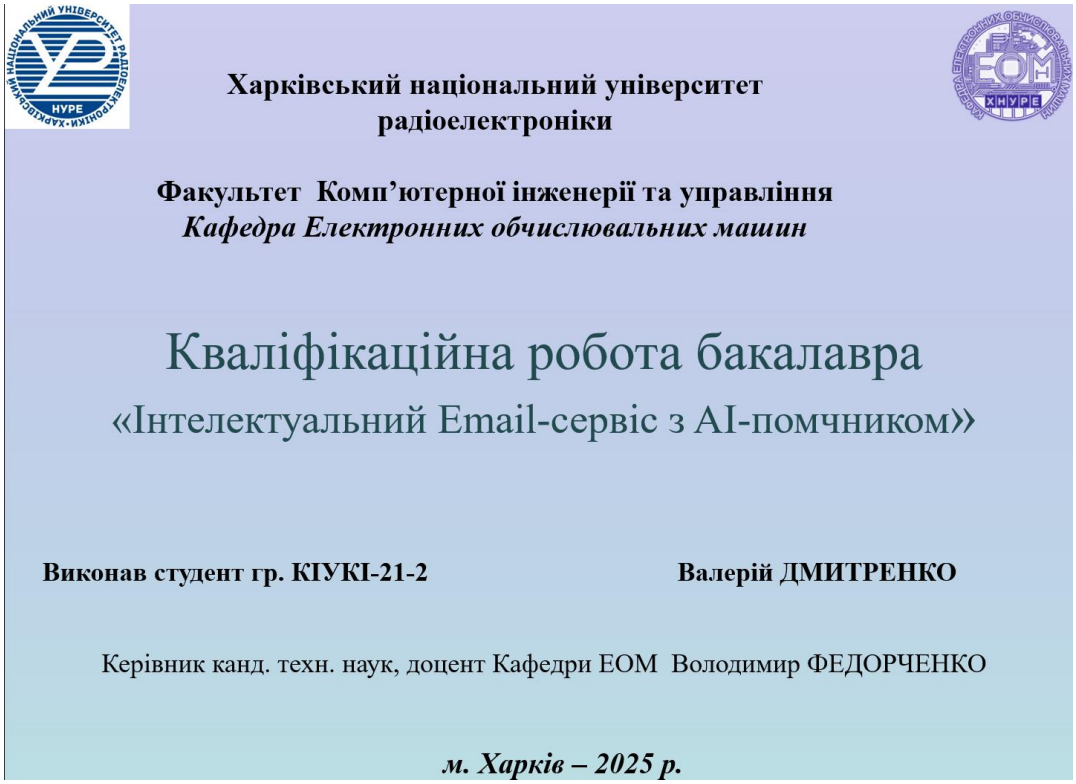
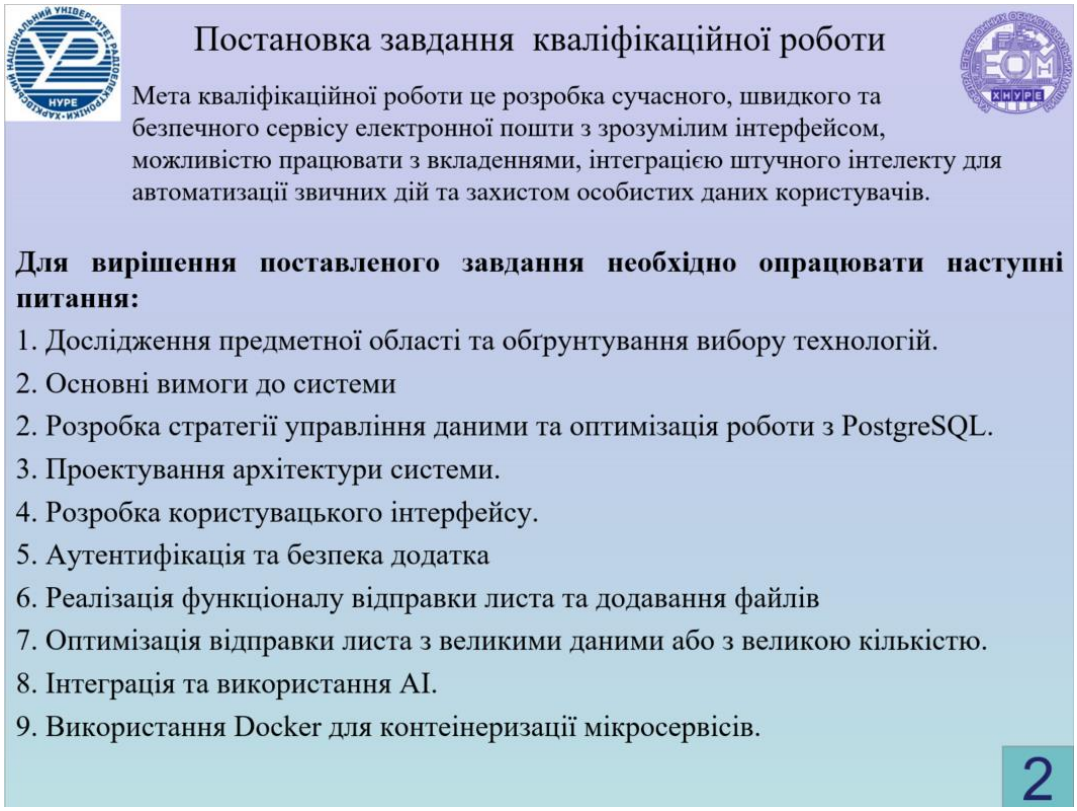
ДОДАТОК А

Графічний матеріал кваліфікаційної роботи



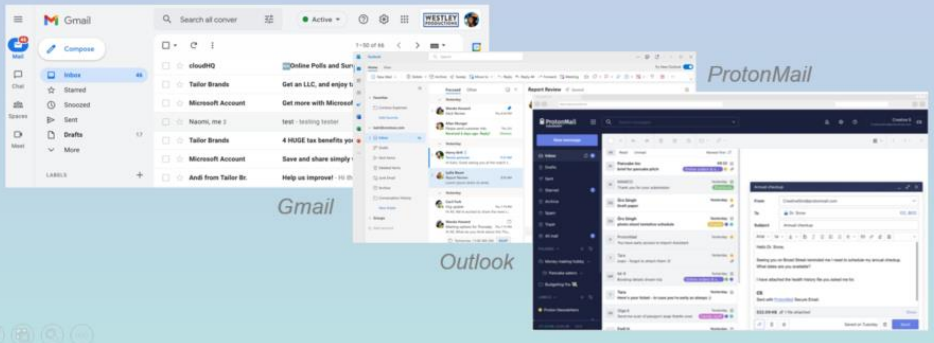


3

ДОСЛІДЖЕННЯ ПРЕДМЕТНОЇ ОБЛАСТІ

Існуючі аналоги

- *Gmail – наймасовіший сервіс (понад 1,8 млрд користувачів),не має наскрізного шифрування й дуже залежить від хмарної екосистеми Google*
- *Outlook – гігант корпоративного сегменту, де фокус зроблено на інтеграції з календарем і офісними застосунками, AI-функції з'явилися лише останні два роки, інтерфейс залишається перевантаженим.*
- *ProtonMail – один із лідерів з безпеки та конфіденційності (E2E-шифрування default), проте клієнти для десктопа перебувають у beta, AI-можливостей (пошук, автозаповнення) нема.*



4

ОБГРУНТУВАННЯ ВИБОРУ ТЕХНОЛОГІЇ

Середа розробки та мова програмування

- *Поєднання мови програмування Java (версії 24) та інтегроване середовище розробки як IntelliJ IDEA, надає потужну основу для створення програмного забезпечення.*
- *Використання фреймворку Spring Boot 3 з gRPC + Protobuf та JavaFX значно спрощує розробку бекенду, надаючи готові конфігурації та інструменти для швидкого створення потужних мікросервісів.*

Мова програмування

Середовище розробки

Середовище SceneBuilder для генерації UI елементів для JavaFX

5

ОСНОВНІ ВИМОГИ ДО СИСТЕМИ

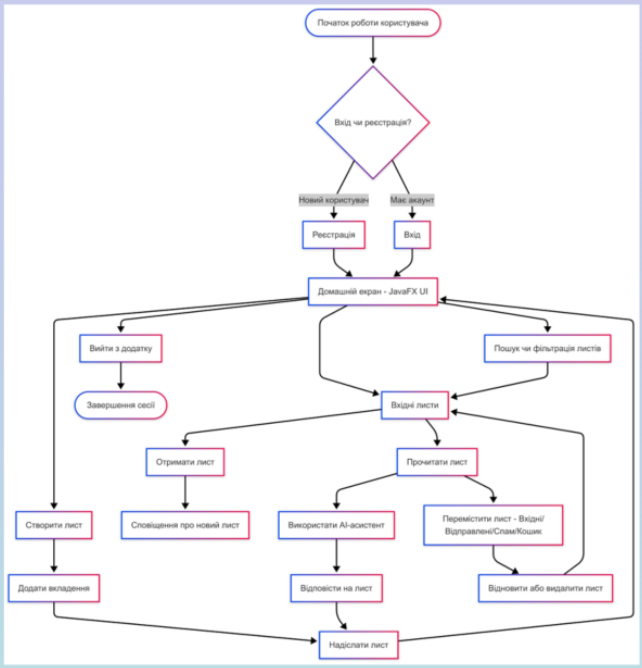
Java 24

Windows 11

Linux

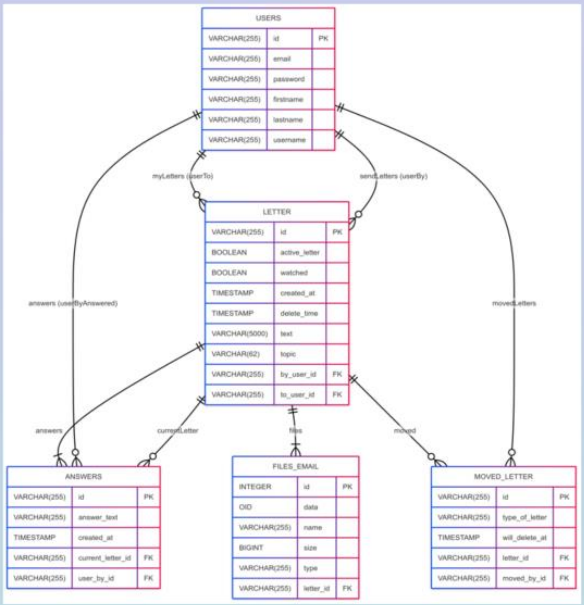
Windows 10

1 GB



6

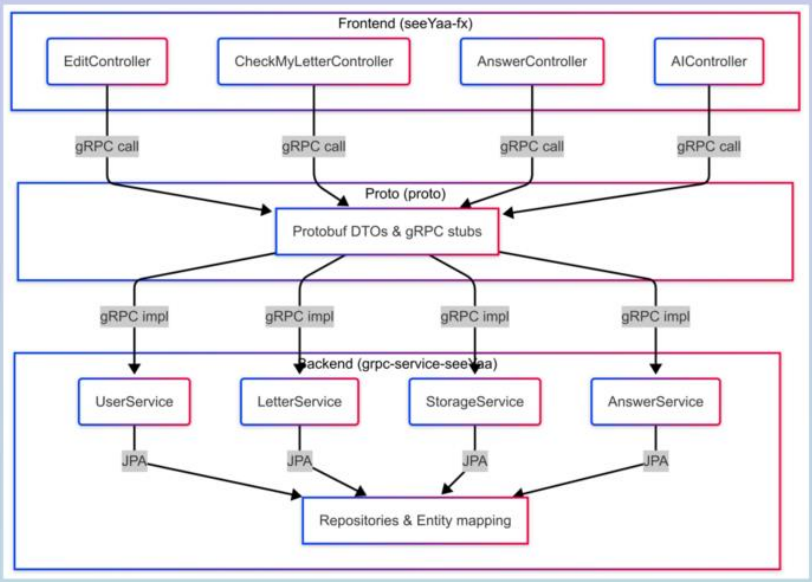
РОЗРОБКА СТРАТЕГІЇ УПРАВЛІННЯ ДАНИМИ ТА ОПТИМІЗАЦІЯ РОБОТИ З PostgreSQL



Використання FlywayDB для мігрування

7

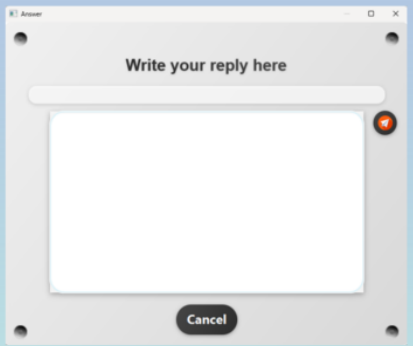
ПРОЕКТУВАННЯ АХІТЕКТУРИ СИСТЕМИ



8

РОЗРОБКА КОРИСТУВАЦЬКОГО ІНТЕРФЕЙСУ

Створення інтерфейсу за допомогою програми SceneBuilder у JavaFX

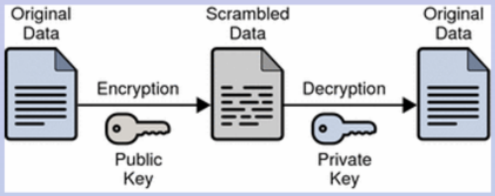


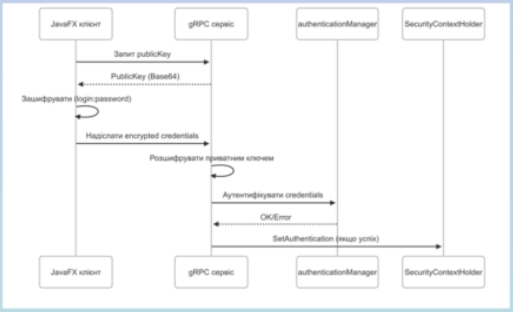
9

АУТЕНТИФІКАЦІЯ ТА БЕЗПЕКА ДОДАТКУ

- *Клієнт запитує публічний ключ у сервера.*
- *Клієнт шифрує логін та пароль, відправляє їх у зашифрованому вигляді за gRPC.*
- *Сервер розшифровує дані та здійснює звичайну аутентифікацію.*
- *У разі успіху зберігається статус сесії користувача.*

RSA encryption ma decryption





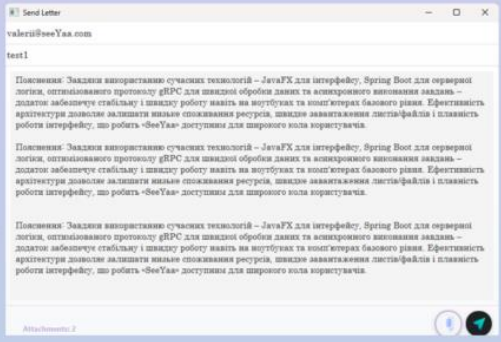
** RSA - криптографічний ключ, що використовується в алгоритмі RSA, який є основою асиметричної криптографії. Він складається з публічного (відкритого) ключа, який можна поширювати.*

10

РОЗРОБКА ФУНКЦІОНАЛУ ВІДПРАВКИ ЛИСТА

- *створення файлу .proto;*
- *генерація gRPC-коду і інтеграція у проект;*
- *імплементація згенерованого коду для серверної логіки (@GrpcService);*
- *використання JPA у сервісі для збереження у базу даних*

Відправка листа



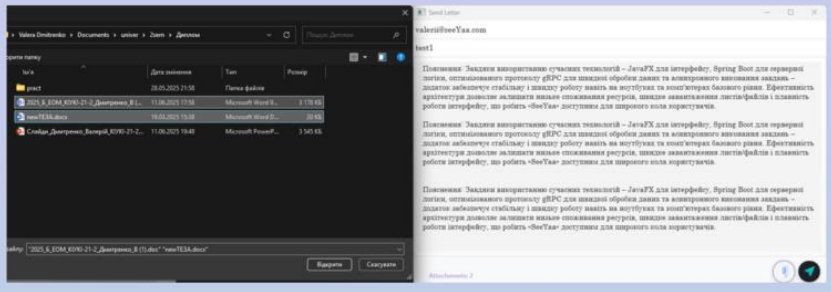
letter_service.proto

```
message LetterRequest {
  string user_by_email = 1;
  string user_to_email = 2;
  string topic         = 3;
  string text          = 4;
}

service LetterService {
  rpc SendLetter           (LetterRequest)          returns (email.model
  rpc SetLetterToSpam      (LetterIdEmailRequest)   returns (google.prot
  rpc SetLetterToGarbage   (LetterIdEmailRequest)   returns (google.prot

  rpc FindById             (LetterIdRequest)        returns (email.model
  rpc FindAllSentByTopic   (TopicSearchRequestBy)   returns (LetterList)
  rpc FindAllInboxByTopic  (TopicSearchRequestTo)   returns (LetterList)

  rpc DeleteLetterById     (LetterIdRequest)        returns (google.prot

  rpc CountOfInboxesLetter (LetterByEmail)  returns (LetterCount);
}
```

** gRPC - є сучасною високопродуктивною платформою віддаленого виклику процедур (RPC) з відкритим кодом, яка може працювати в будь-якому середовищі.*

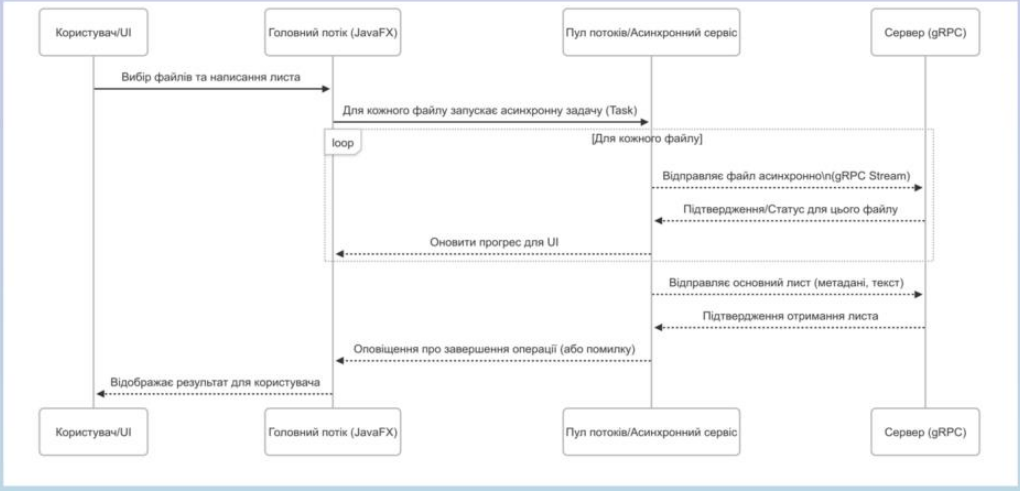
11

РОЗРОБКА ФУНКЦІОНАЛУ ДОДАВАННЯ ТА ВІДПРАВКИ ФАЙЛІВ



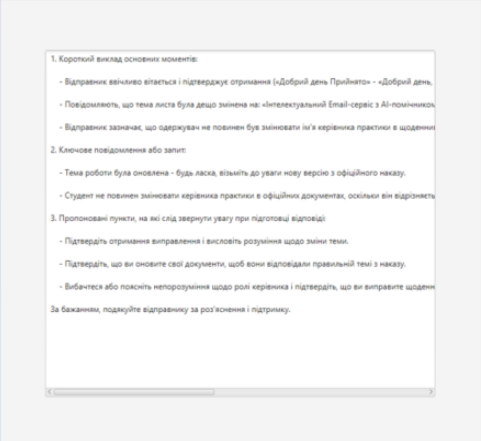
12

ОПТИМІЗАЦІЯ ВІДПРАВКИ ЛИСТА З ВЕЛИКИМИ ДАНИМИ АБО З ВЕЛИКОЮ КІЛЬКІСТЮ ФАЙЛІВ



13

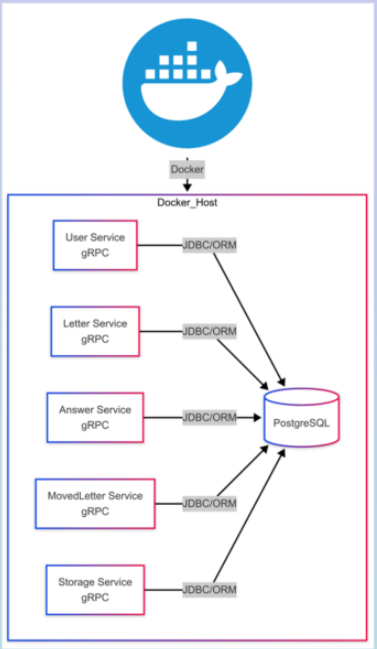
ВИКОРИСТАННЯ AI-АСИСТЕНТА



- *При запуску сервісу створюється клієнт Vertex AI з параметрами проєкту*
- *Перед відправленням запиту до Vertex AI налаштовуються, максимальна кількість токенів та температура*
- *Від системи формується текстовий промпт*
- *Усі результати генерації передаються у відповідне текстове поле інтерфейсу*

14

ВИКОРИСТАННЯ DOCKER ДЛЯ КОНТЕЙНЕРИЗАЦІЇ МІКРОСЕРВІСІВ



- **Кожен мікросервіс** – це окремий контейнер у Docker.
- **Усі мікросервіси** взаємодіють з єдиною базою даних PostgreSQL
- **Контейнери** об'єднані у одну Docker-мережу, що гарантує зв'язність і безпеку між ними.
- Будь-який сервіс (наприклад, Letter або Storage) може бути розгорнутий у кількох екземплярах

15

ВИСНОВОК

У основі проекту створена модульна структура з чітким розділенням, відповідальності між клієнтом та сервером. Пильну увагу було звернено на створення гарної серверної інфраструктури з використанням Spring Boot та gRPC, інтеграції безпечної системи аутентифікації, а також організації зберігання даних у реляційній базі PostgreSQL. Використання JavaFX дозволило зробити простий та функціональний користувацький інтерфейс що підвищує зручність і продуктивність для користувачів.

Апробація

Дмитренко Валерій, Федорченко В.М. // Методи використання Spring Boot у хмарних обчислень // Матеріали п'ятнадцятої міжнародної науково-технічної конференції «Сучасні напрями розвитку інформаційно-комунікаційних технологій та засобів управління». Том 3: тези доповідей, 24-25 квітня 2025 р. – Харків: ХНУРЕ, 2025 – С. 55 - 56.

ДОДАТОК Б

Програмування

Б.1 Сервіси додатку

Б.1.1 User сервіс

```
@Slf4j
@GrpcService
@RequiredArgsConstructor
public class UserService extends
UsersServiceGrpc.UsersServiceImplBase {
    private final PasswordEncoder passwordEncoder;
    private final UserRepository userRepository;
    private final SecurityService securityService;
    private final GrpcValidatorService grpcValidatorService;

    @Override
    public void signUp(SignUpRequest request,
StreamObserver<Empty> responseObserver) {
        try {
        grpcValidatorService.validSignUp(new SignUpRequestDto(
                request.getEmail(), request.getUsername(),
request.getPassword(),
                request.getFirstname(), request.getLastname()));
        final var savedUser =
org.parent.grpcserviceseeyaa.entity.Users.newBuilder()
                .setEmail(request.getEmail())

.setPassword(passwordEncoder.encode(request.getPassword()))
                .setFirstname(request.getFirstname())
                .setLastname(request.getLastname())
                .setUsername(request.getUsername())
                .build();

            userRepository.save(savedUser);
        } catch (ConstraintViolationException e) {
            String msg = e.getConstraintViolations().stream()
                    .map(v -> v.getPropertyPath() + ": " +
v.getMessage())
                    .reduce((a, b) -> a + "; " + b)
                    .orElse(e.getMessage());


responseObserver.onError(io.grpc.Status.INVALID_ARGUMENT
                    .withDescription(msg)
```

```
                    .asRuntimeException());
        } catch (Exception ex) {
            responseObserver.onError(io.grpc.Status.INTERNAL
                    .withDescription("Internal server error: " +
ex.getMessage())
                    .asRuntimeException());
        }

        responseObserver.onNext(Empty.getDefaultInstance());
        responseObserver.onCompleted();
    }

    @Override
    @Transactional(readOnly = true)
    public void getCurrentUser(Empty request,
StreamObserver<Users> responseObserver) {
        final var user =
userRepository.findByEmail(securityService.getCurrentUserEmail()
)
                .orElseThrow(() -> Status.NOT_FOUND
                        .withDescription("You are not logged
in")
                        .asRuntimeException());


responseObserver.onNext(UserMapper.INSTANCE.toUserProto(user));
        responseObserver.onCompleted();
    }

    @Override
    @Transactional(readOnly = true)
    public void getCurrentUserExtended(Empty request,
StreamObserver<UserWithLetters> responseObserver) {
        final var user =
userRepository.findByEmail(securityService.getCurrentUserEmail()
)
                .orElseThrow(() -> Status.NOT_FOUND
                        .withDescription("You are not logged
in")
                        .asRuntimeException());


responseObserver.onNext(UserMapper.INSTANCE.toUserWithLettersPro
to(user));
        responseObserver.onCompleted();
    }

    @Override
    public void editProfile(EditProfileRequest request,
StreamObserver<Empty> responseObserver) {
        grpcValidatorService.validEditProfile(new
EditRequestDto(
                request.getFirstname(), request.getLastname(),
```

```
request.getUsername(), request.getPassword()));
        final var users =
userRepository.findByEmail(securityService.getCurrentUserEmail()
)
                .orElseThrow(() -> Status.NOT_FOUND
                        .withDescription("You are not logged
in")
                        .asRuntimeException());

        if
(!request.getFirstname().equals(users.getFirstname()))
users.setFirstname(request.getFirstname());
        if (!request.getPassword().equals("N") &&
!passwordEncoder.matches(users.getPassword(),
request.getPassword()))

users.setPassword(passwordEncoder.encode(request.getPassword()))
;
        if (!request.getLastname().equals(users.getLastname()))
users.setLastname(request.getLastname());
        if (!request.getUsername().equals(users.getUsername()))
users.setUsername(request.getUsername());
        userRepository.save(users);

        responseObserver.onNext(Empty.getDefaultInstance());
        responseObserver.onCompleted();
    }
}
```

Б.1.2 Сервіс для зберігання файлу

```
@GrpcService
@RequiredArgsConstructor
@Slf4j
public class StorageService extends
StorageServiceGrpc.StorageServiceImplBase {
    private final FilesRepository filesRepository;
    private final LetterRepository letterRepository;


    @Override
    public void uploadFile(UploadFileRequest request,
StreamObserver<Empty> responseObserver) {
        try {
            final var letter =
letterRepository.findById(request.getLetterId())
                    .orElseThrow(() -> new
NotFoundException("Letter not found with id: " +
request.getLetterId()));
```

```
filesRepository.save(org.parent.grpcserviceseeyaa.entity.Files.b
uilder()
                    .name(request.getName())
                    .type(request.getType())
                    .size(request.getSize())
                    .data(request.getData().toByteArray())
                    .letter(letter)
                    .build());


responseObserver.onNext(com.google.protobuf.Empty.getDefaultInst
ance());
            responseObserver.onCompleted();
        } catch (Exception ex) {
            log.error("Error uploading file", ex);
            responseObserver.onError(
                    Status.INTERNAL
                            .withDescription(ex.getMessage())
                            .withCause(ex)
                            .asRuntimeException()
            );;
        }
    }

    @Override
    @Transactional
    public void downloadFile(FileIdRequest request,
StreamObserver<DownloadFileResponse> responseObserver) {
        try {
            org.parent.grpcserviceseeyaa.entity.Files file =
filesRepository.findById(request.getFileId())
                    .orElseThrow(() ->
                            Status.NOT_FOUND
                                    .withDescription("File not
found id=" + request.getFileId())
                                    .asRuntimeException());

            DownloadFileResponse out =
DownloadFileResponse.newBuilder()

.setData(ByteString.copyFrom(file.getData()))
                    .build();

            responseObserver.onNext(out);
            responseObserver.onCompleted();
        } catch (StatusRuntimeException sre) {
            responseObserver.onError(sre);
        } catch (Exception ex) {
            responseObserver.onError(Status.INTERNAL
                    .withDescription("Download failed")
                    .withCause(ex)
                    .asRuntimeException());
        }
```

```
    }

    @Override
    @Transactional
    public void getFilesByLetterId(LetterIdRequest request,
StreamObserver<FilesList> responseObserver) {
        final var allByLetterId =
filesRepository.findAllByLetterId(request.getLetterId())
                .stream()
                .map(FilesMapper.INSTANCE::toFilesProto)
                .toList();


responseObserver.onNext(FilesList.newBuilder().addAllFiles(allBy
LetterId).build());
        responseObserver.onCompleted();
    }

    @Override
    @Transactional
    public void getFileMetadataByLetterId(LetterIdRequest
request, StreamObserver<FileMetadataList> responseObserver) {
        final var rows =
filesRepository.findAllByLetterId(request.getLetterId())
                .stream()
                .map(FilesMapper.INSTANCE::toMetaProto)
                .toList();

        responseObserver.onNext(FileMetadataList.newBuilder()
                .addAllMetadata(rows)
                .build());
        responseObserver.onCompleted();
    }

    @Override
    @Transactional
    public void getFileById(FileIdRequest request,
StreamObserver<Files> responseObserver) {
        try {
            final var entity =
filesRepository.findById(request.getFileId())
                    .orElseThrow(() ->
                            Status.NOT_FOUND
                                    .withDescription("File not
found id=" + request.getFileId())
                                    .asRuntimeException());


responseObserver.onNext(FilesMapper.INSTANCE.toFilesProto(entity
));
            responseObserver.onCompleted();
        } catch (StatusRuntimeException sre) {
            responseObserver.onError(sre);
```

```
        }
    }
}
```

Б.1.3 Сервіс для переміщення листу

```
@Slf4j
@Component
@RequiredArgsConstructor
public class MovedLetterConfigurationImpl extends
MovedLetterConfigurationGrpc.MovedLetterConfigurationImplBase {
    private final LetterRepository letterRepository;
    private final MovedLetterRepository movedLetterRepository;
    private final UserRepository userRepository;

    @Override
    public void setLetterType(SetLetterTypeRequest request,
StreamObserver<Empty> responseObserver) {
            final var user =
userRepository.findByEmail(request.getEmail())
                    .orElseThrow(() -> new
NotFoundException("User not found"));

            final var movedLetterEntity =
movedLetterRepository.findByLetterId(request.getLetterId())
                    .map(movedLetter -> {

movedLetterRepository.delete(movedLetter);

letterRepository.updateActiveLetterById(request.getLetterId(),
true);
                        return movedLetter;
                    }).orElseGet(() -> {

letterRepository.updateActiveLetterById(request.getLetterId(),
false);
                        final var letter =
letterRepository.findById(request.getLetterId())
                                .orElseThrow(() -> new
NotFoundException("Letter not found"));

                        final var movedLetterBuild =
MovedLetter.builder()
                                .letter(letter)
                                .typeOfLetter(request.getType())
                                .movedBy(user)
                                .build();
                        return
movedLetterRepository.save(movedLetterBuild);
                    });
```

```
            log.info("Moved letter with id {} to {}",
movedLetterEntity.getLetter().getId(), request.getType());
            responseObserver.onNext(Empty.getDefaultInstance());
            responseObserver.onCompleted();
    }
}
```

Б.1.4 Сервіс для відправлення листа

```
@Slf4j
@GrpcService
@RequiredArgsConstructor
public class LetterService extends
LetterServiceGrpc.LetterServiceImplBase {
    private final MovedLetterConfigurationImpl
movedLetterConfiguration;
    private final LetterRepository letterRepository;
    private final UserRepository userRepository;
    private final GrpcValidatorService grpcValidatorService;

    @Override
    public void sendLetter(LetterRequest request,
StreamObserver<Letter> responseObserver) {
        grpcValidatorService.validateLetter(new
LetterRequestDto(
                request.getText(), request.getTopic(),
request.getUserToEmail(), request.getUserByEmail()));
        final var usersBy =
userRepository.findByEmail(request.getUserByEmail())
                .orElseThrow(() -> new NotFoundException("User
not found"));
        final var letter =
userRepository.findByEmail(request.getUserToEmail())
                .map(userTo -> letterRepository.save(

org.parent.grpcserviceseeyaa.entity.Letter.builder()
                                .userBy(usersBy)
                                .userTo(userTo)
                                .text(request.getText())
                                .topic(request.getTopic())
                                .activeLetter(Boolean.TRUE)
                                .createdAt(LocalDateTime.now())
                                .deleteTime(LocalDateTime.now())
                                .build())
                ).orElseThrow(() -> new NotFoundException("User
not found"));

        final var letterProto =
LetterMapper.INSTANCE.toLetterProto(letter);
        responseObserver.onNext(letterProto);
        responseObserver.onCompleted();
```

```
    }

    @Override
    @Transactional
    public void setLetterToSpam(LetterIdEmailRequest request,
StreamObserver<Empty> responseObserver) {

movedLetterConfiguration.setLetterType(SetLetterTypeRequest.newB
uilder()
                .setLetterId(request.getLetterId())
                .setEmail(request.getEmail())
                .setType(TypeOfLetter.SPAM)
                .build(), responseObserver);

        responseObserver.onNext(Empty.getDefaultInstance());
        responseObserver.onCompleted();
    }

    @Override
    @Transactional
    public void setLetterToGarbage(LetterIdEmailRequest request,
StreamObserver<Empty> responseObserver) {

movedLetterConfiguration.setLetterType(SetLetterTypeRequest.newB
uilder()
                .setLetterId(request.getLetterId())
                .setEmail(request.getEmail())
                .setType(TypeOfLetter.GARBAGE)
                .build(), responseObserver);

        responseObserver.onNext(Empty.getDefaultInstance());
        responseObserver.onCompleted();
    }

    @Override
    @Transactional(readOnly = true)
    public void findById(LetterIdRequest request,
StreamObserver<Letter> responseObserver) {
        try {
            final var letter =
letterRepository.findById(request.getId())
                    .orElseThrow(() -> new NotFoundException("No
letter found with id: " + request.getId()));


responseObserver.onNext(LetterMapper.INSTANCE.toLetterProto(lett
er));
            responseObserver.onCompleted();
        } catch (Exception e) {
            responseObserver.onError(
                    io.grpc.Status.INTERNAL
                            .withDescription("Unexpected server
error ")
```

```
                                .withCause(e)
                                .asRuntimeException());
        }

    }

    @Override
    @Transactional(readOnly = true)
    public void findAllSentByTopic(TopicSearchRequestBy request,
StreamObserver<LetterList> responseObserver) {
        final var allByTopicContainingAndUserBy =
letterRepository.findAllByTopicContainingAndUserBy(
                        request.getTopic(),

userRepository.findByEmail(request.getUserByEmail()).orElse(null
))
                .stream()
                .map(LetterMapper.INSTANCE::toLetterProto)
                .toList();

        responseObserver.onNext(LetterList.newBuilder()
                .addAllLetters(allByTopicContainingAndUserBy)
                .build());
        responseObserver.onCompleted();
    }

    @Override
    @Transactional(readOnly = true)
    public void findAllInboxByTopic(TopicSearchRequestTo
request, StreamObserver<LetterList> responseObserver) {
        final var allByTopicContainingAndUserBy =
letterRepository.findAllByTopicContainingAndUserTo(
                        request.getTopic(),

userRepository.findByEmail(request.getUserToEmail()).orElse(null
))
                .stream()
                .map(LetterMapper.INSTANCE::toLetterProto)
                .toList();

        responseObserver.onNext(LetterList.newBuilder()
                .addAllLetters(allByTopicContainingAndUserBy)
                .build());
        responseObserver.onCompleted();
    }

    @Override
    @Transactional
    public void deleteLetterById(LetterIdRequest request,
StreamObserver<Empty> responseObserver) {

letterRepository.findById(request.getId()).ifPresent(letterRepos
itory::delete);
```

```
        responseObserver.onNext(Empty.getDefaultInstance());
        responseObserver.onCompleted();
    }
}
```

Б.1.5 Сервіс для завантаження файлу з листа

```
@Service
@RequiredArgsConstructor
public class FileDownloadServiceImpl implements
FileDownloadService {
    private final StorageService storageService;

    @Override
    public Task<Files> createFileLoadTask(int fileId) {
        return new Task<>() {
            @Override
            protected Files call() {
                return storageService.getFileById(fileId);
            }
        };
    }

    @Override
    public Task<Void> createDownloadTask(Files completeFile,
Stage stage, Runnable onDone, Consumer<Exception> onError) {
        return new Task<>() {
            @Override
            protected Void call() {
                File saveFile = getSaveFileFromUser(stage,
completeFile.getName());
                if (saveFile != null) {
                    try {
                        copyToFile(completeFile.getData(),
saveFile);
                        if (onDone != null)
Platform.runLater(onDone);
                    } catch (Exception ex) {
                        if (onError != null)
Platform.runLater(() -> onError.accept(ex));
                    }
                }
                return null;
            }
        };
    }

    private File getSaveFileFromUser(Stage stage, String
fileName) {
        final File[] selectedFile = new File[1];
```

```
        CountDownLatch latch = new CountDownLatch(1);

        Platform.runLater(() -> {
            FileChooser fileChooser = new FileChooser();
            fileChooser.setTitle("Save File");
            fileChooser.setInitialFileName(fileName);
            selectedFile[0] = fileChooser.showSaveDialog(stage);
            latch.countDown();
        });

        try {
            latch.await();
        } catch (InterruptedException e) {
            Thread.currentThread().interrupt();
            return null;
        }
        return selectedFile[0];
    }

    private void copyToFile(byte[] data, File outFile) throws
IOException {
        try (InputStream is = new ByteArrayInputStream(data);
             FileOutputStream fos = new
FileOutputStream(outFile)) {
            final byte[] buffer = new byte[1024 * 1024];
            int bytesRead;
            while ((bytesRead = is.read(buffer)) != -1)
                fos.write(buffer, 0, bytesRead);
        }
    }
}
```

Б.2 Логіка створення файлів у вигляді піраміди з ScrollPane

```
@Service
@Slf4j
public class FileActionImpl implements FileAction {
    private final StorageService storageService;

    public FileActionImpl(StorageService storageService) {
        this.storageService = storageService;
    }

    @Override
    public HBox createFillRow(Files files, Stage stage) {
        final var hbox = new HBox();
        hbox.getStyleClass().add("file-row");
        hbox.setSpacing(10);
        hbox.setAlignment(Pos.CENTER_LEFT);
        hbox.setMinHeight(40);
```

```
        final var fileNameLabel = new Label(files.getName());
        fileNameLabel.getStyleClass().add("file-name-label");
        fileNameLabel.setWrapText(true);
        HBox.setHgrow(fileNameLabel, Priority.ALWAYS);

        final var fileSizeLabel = new
Label(formatFileSize(files.getSize()));
        fileSizeLabel.getStyleClass().add("file-size-label");

        final var fileInfoBox = new VBox(5);
        fileInfoBox.getChildren().addAll(fileNameLabel,
fileSizeLabel);
        HBox.setHgrow(fileInfoBox, Priority.ALWAYS);

        final Trio<HBox, Button, ProgressIndicator> object =
createObjects();
        final var downloadButton = object.second();
        final var progressIndicator = object.third();
        final var buttonBox = object.first();

        downloadButton.setOnAction(e -> {
            progressIndicator.setVisible(true);
            downloadButton.setVisible(false);

            Task<Void> downloadTask = new Task<>() {
                @Override
                protected Void call() {
                    downloadFile(files, stage);
                    return null;
                }
            };

            downloadTask.setOnSucceeded(event ->
                    Platform.runLater(() -> {
                        progressIndicator.setVisible(false);
                        downloadButton.setVisible(true);
                    }));

            downloadTask.setOnFailed(event ->
                    Platform.runLater(() -> {
                        progressIndicator.setVisible(false);
                        downloadButton.setVisible(true);
                        showAlert(Alert.AlertType.ERROR,
"Download Failed",
                                "Error: " +
downloadTask.getException().getMessage());
                    }));

            new Thread(downloadTask).start();
        });

        hbox.setOnMouseEntered(e -> {
            final var pt = new ParallelTransition();
```

```
            final var scale = new
ScaleTransition(Duration.millis(200), hbox);
            scale.setToX(1.02);
            scale.setToY(1.02);

            final var fade = new
FadeTransition(Duration.millis(200), downloadButton);
            fade.setToValue(0.8);

            pt.getChildren().addAll(scale, fade);
            pt.play();
        });

        hbox.setOnMouseExited(e -> {
            final var pt = new ParallelTransition();

            final var scale = new
ScaleTransition(Duration.millis(200), hbox);
            scale.setToX(1);
            scale.setToY(1);

            final var fade = new
FadeTransition(Duration.millis(200), downloadButton);
            fade.setToValue(1);

            pt.getChildren().addAll(scale, fade);
            pt.play();
        });

        hbox.getChildren().addAll(fileInfoBox, buttonBox);
        return hbox;
    }

    private String formatFileSize(long size) {
        if (size < 1024) return size + " B";
        else if (size < 1024 * 1024) return String.format("%.2f
KB", size / 1024.0);
        else if (size < 1024 * 1024 * 1024) return
String.format("%.2f MB", size / (1024.0 * 1024));
        else return String.format("%.2f GB", size / (1024.0 *
1024 * 1024));
    }

    @Override
    public void downloadButton(Files files, Stage stage) {
        downloadFile(files, stage);
    }

    @Override
    public Trio<HBox, Button, ProgressIndicator> createObjects()
{
        final var downloadButton = new Button("↓");
```

```
        downloadButton.getStyleClass().add("download-button");

        final var progressIndicator = new ProgressIndicator();
        progressIndicator.setMaxSize(24, 24);
        progressIndicator.setVisible(false);

        final var buttonBox = new HBox(5);
        buttonBox.setAlignment(Pos.CENTER_RIGHT);
        buttonBox.getChildren().addAll(progressIndicator,
downloadButton);
        return new Trio<>(buttonBox, downloadButton,
progressIndicator);
    }

    private void downloadFile(Files files, Stage stage) {
        log.info("Starting download for file: {}",
files.getName());

        final var fileChooser = new FileChooser();
        fileChooser.setTitle("Save file");
        fileChooser.setInitialFileName(files.getName());
        final var fileToSave =
fileChooser.showSaveDialog(stage);
        if (fileToSave == null) return;

        try (InputStream inputStream =
storageService.downloadFile(files.getId().toString());
             FileOutputStream outputStream = new
FileOutputStream(fileToSave)) {
            byte[] buffer = new byte[1024 * 1024];
            int bytesRead;
            while ((bytesRead = inputStream.read(buffer)) != -1)
outputStream.write(buffer, 0, bytesRead);
            showAlert(Alert.AlertType.CONFIRMATION, "Download
Complete", "File downloaded successfully!");
        } catch (IOException e) {
            showAlert(Alert.AlertType.ERROR, "Download Failed",
"Error: " + e.getMessage());
        }
    }
}
```

Б.3 Функція пошуку листів різних видів, наприклад вхідні, спам, тощо.

Б.3.1 Інтерфейс вибору

```
public interface Choice {
    List<Letter> addToBox(int index, String email);
```

```
    TypeOfLetter getChoice();
}
```

Б.3.2 Якщо вибором є «сміття»

```
@Component
public class GarbageChoice implements Choice {
    private final
MovedLetterServiceGrpc.MovedLetterServiceBlockingStub
movedLetterService;

    public
GarbageChoice(MovedLetterServiceGrpc.MovedLetterServiceBlockingS
tub movedLetterService) {
        this.movedLetterService = movedLetterService;
    }

    @Override
    @PreAuthorize("hasRole('ROLE_USER')")
    @Transactional(readOnly = true)
    public List<Letter> addToBox(int index, String email) {
        return movedLetterService.getGarbageLetters(

EmailRequest.newBuilder().setEmail(email).build())
                .getLettersList();
    }

    @Override
    public TypeOfLetter getChoice() {
        return TypeOfLetter.GARBAGE;
    }
}
```

Б.3.3 Якщо вибором є «вхідні»

```
@Component
public class InboxesChoice implements Choice {
    private final
MovedLetterServiceGrpc.MovedLetterServiceBlockingStub
movedLetterService;

    public
InboxesChoice(MovedLetterServiceGrpc.MovedLetterServiceBlockingS
tub movedLetterService) {
        this.movedLetterService = movedLetterService;
    }

    @Override
```

```
    @PreAuthorize("hasRole('ROLE_USER')")
    @Transactional(readOnly = true)
    public List<Letter> addToBox(int index, String email) {
        return
movedLetterService.getInboxLetters(EmailRequest.newBuilder().set
Email(email).build()).getLettersList();
    }

    @Override
    public TypeOfLetter getChoice() {
        return TypeOfLetter.INBOXES;
    }
}
```

Б.3.4. Якщо вибором є «відправлені»

```
@Component
public class SentChoice implements Choice {
    private final
MovedLetterServiceGrpc.MovedLetterServiceBlockingStub
movedLetterService;

    public
SentChoice(MovedLetterServiceGrpc.MovedLetterServiceBlockingStub
movedLetterService) {
        this.movedLetterService = movedLetterService;
    }

    @Override
    @PreAuthorize("hasRole('ROLE_USER')")
    @Transactional(readOnly = true)
    public List<Letter> addToBox(int index, String email) {
        return
movedLetterService.getSentLetters(EmailRequest.newBuilder().setE
mail(email).build()).getLettersList();
    }

    @Override
    public TypeOfLetter getChoice() {
        return TypeOfLetter.SENT;
    }
}
```

Б.3.5. Якщо вибором є «спам»

```
@Component
public class SpamChoice implements Choice {
    private final
```

```
MovedLetterServiceGrpc.MovedLetterServiceBlockingStub
movedLetterService;

    public
SpamChoice(MovedLetterServiceGrpc.MovedLetterServiceBlockingStub
movedLetterService) {
        this.movedLetterService = movedLetterService;
    }

    @Override
    @PreAuthorize("hasRole('ROLE_USER')")
    @Transactional(readOnly = true)
    public List<Letter> addToBox(int index, String email) {
        return movedLetterService.getSpamLetters(
                EmailRequest.newBuilder()

.setEmail(email).build()).getLettersList();
    }

    @Override
    public TypeOfLetter getChoice() {
        return TypeOfLetter.SPAM;
    }
}
```