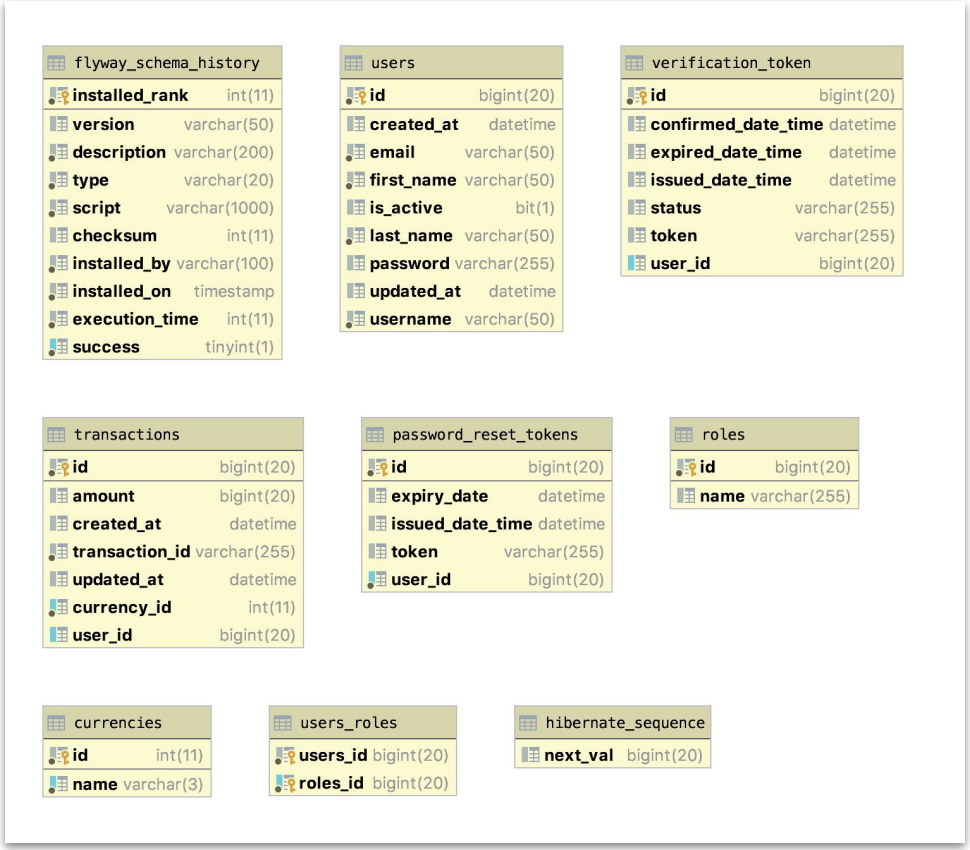
# Password Reset

# Google Account smtp Configuration Changes

# Demo

http://localhost:8080

# Database Visualization



# Password Reset Token

```java
@Entity
@Table(name = "password_reset_tokens")
public class PasswordResetToken {

  private static final int EXPIRATION = 60 * 24;

  @Id
  @GeneratedValue(strategy = GenerationType.AUTO)
  private Long id;

  private String token;

  @OneToOne(targetEntity = User.class, fetch = FetchType.EAGER)
  @JoinColumn(nullable = false, name = "user_id")
  private User user;

  private LocalDateTime issuedDateTime;

  private LocalDateTime expiryDate;

  public PasswordResetToken() {
     this.token = UUID.randomUUID().toString();
     this.issuedDateTime = LocalDateTime.now();
     this.expiryDate = this.issuedDateTime.plusDays(1);
  }

}
```

# PasswordResetTokenRepository

```
package com.nursultanturdaliev.moneytransferapp.repository;

import com.nursultanturdaliev.moneytransferapp.model.PasswordResetToken;
import org.springframework.data.repository.CrudRepository;

public interface PasswordResetTokenRepository extends CrudRepository<PasswordResetToken, Long> {
   public PasswordResetToken findOneByToken(String token);
}
```

# Password Reset Token Service

```
@Service
public class PasswordResetTokenService {

  @Autowired
  private PasswordResetTokenRepository passwordTokenRepository;

  @Autowired
  private UserService userService;

  @Autowired
  private SendingMailService sendingMailService;

  public void createPasswordReset(String userEmail) throws
UserNotFoundException {
      User user = userService.findUserByEmail(userEmail);
      if (user == null) {
         throw new UserNotFoundException();
      }
      PasswordResetToken passwordResetToken = create(user);

      sendingMailService.sendPasswordResetTokenMail(user,
passwordResetToken.getToken());

  }
  public PasswordResetToken create(User user) {
      PasswordResetToken passwordResetToken = new PasswordResetToken();
      passwordResetToken.setUser(user);
      return passwordTokenRepository.save(passwordResetToken);
  }
}
```

# Sending Mail Service

```
@Service
public class SendingMailService {
  private final MailProperties mailProperties;

  private SpringTemplateEngine springTemplateEngine;

  private org.slf4j.Logger logger = LoggerFactory.getLogger(HomeController.class);

  @Autowired
  SendingMailService(MailProperties mailProperties, SpringTemplateEngine
springTemplateEngine) {
      this.mailProperties = mailProperties;
      this.springTemplateEngine = springTemplateEngine;
  }

  public void sendPasswordResetTokenMail(User user, String resetToken) {
      String subject = "Please reset token";
      String body = "";
      try {
          Context context = new Context();
          context.setVariable("resetTokenURL", mailProperties.getResetpasswordapi()
+ resetToken + "&id=" + user.getId());
          context.setVariable("user", user);
          body = springTemplateEngine.process("password-reset-token.html",
context);
      } catch (Exception ex) {
          logger.error(ex.getMessage(), ex);
      }
      sendMail(user.getEmail(), subject, body);
  }
```

# Security Service Password Reset Token Validation

```
@Service
public class SecurityServiceImpl implements SecurityService {
  @Autowired
  private AuthenticationManager authenticationManager;
  @Autowired
  private UserDetailsService userDetailsService;
  @Autowired
  private PasswordResetTokenRepository passwordResetTokenRepository;
  @Override
  public void validatePasswordResetToken(long id, String token) throws
ExpiredTokenException, InvalidTokenException {
      final PasswordResetToken passToken =
passwordResetTokenRepository.findOneByToken(token);
      if ((passToken == null) || (passToken.getUser().getId() != id)) {
        throw new InvalidTokenException();
      }

      final Calendar cal = Calendar.getInstance();
      if (passToken.getExpiryDate().isBefore(LocalDateTime.now())) {
        throw new ExpiredTokenException();
      }

      final User user = passToken.getUser();
      final Authentication auth = new UsernamePasswordAuthenticationToken(user,
null, Arrays.asList(new
SimpleGrantedAuthority("CHANGE_PASSWORD_PRIVILEGE")));
      SecurityContextHolder.getContext().setAuthentication(auth);
  }
}
```

# Password Reset Endpoints Forgot Password

```
@GetMapping("/forgot-password")
public String forgotPassword() {
   return "forgot-password";
}
```

```
<div class="container">
  <div class="row">
     <div class="col-sm-8 col-sm-offset-2">
        <h1 th:text="#{message.resetPassword}">reset</h1>

        <div class="form-group">
           <label th:text="#{label.user.email}">email</label>
           <input id="email" name="email" type="email" value=""
class="form-control" />
        </div>
        <div class="form-group">
           <button type="submit" class="btn btn-info form-control"
onclick="resetPass()"
                   th:text="#{message.resetPassword}">reset</button>
        </div>

        <a th:href="@{/registration}"
th:text="#{label.form.loginSignUp}">
           registration
        </a>
        <a th:href="@{/login}"
th:text="#{label.form.loginLink}">login</a>
     </div>
  </div>
</div>
```

# Reset Password Endpoint

```
@RequestMapping(value = "/reset-password",
      method = RequestMethod.POST)
public ResponseEntity<GenericResponse> resetPassword(
                                        @RequestParam("email") String userEmail)
throws UserNotFoundException {
  passwordResetTokenService.createPasswordReset(userEmail);

  GenericResponse genericResponse = new GenericResponse(
        messageSource.getMessage("message.resetPasswordEmail", null,
              Locale.US));
  return ResponseEntity.status(HttpStatus.CREATED).body(genericResponse);
}

@RequestMapping(value = "/change-password", method = RequestMethod.GET)
public String changePassword(Model model,
                    @RequestParam("id") long id, @RequestParam("token") String token) throws
ExpiredTokenException, InvalidTokenException {
  securityService.validatePasswordResetToken(id, token);

  return "redirect:/update-password";
}
```

# Update Password Endpoint

```
@PreAuthorize("hasAuthority('CHANGE_PASSWORD_PRIVILEGE')")
@RequestMapping("/update-password")
public String updatePassword() {
   return "update-password";
}
```

# Save Password Endpoint

```
@RequestMapping(value = "/save-password", method = RequestMethod.POST)
@ResponseBody
public GenericResponse savePassword(@Valid PasswordDto passwordDto) {
  User user =
        (User) SecurityContextHolder.getContext()
              .getAuthentication().getPrincipal();

  userService.changeUserPassword(user, passwordDto.getNewPassword());
  return new GenericResponse(
        messageSource.getMessage("message.resetPasswordSuc", null,
Locale.US));
}
```

# Password DTO - Data Transfer Object

```
public class PasswordDto {

   @ValidPassword
   private String newPassword;

   public String getNewPassword() {
      return newPassword;
   }

   public void setNewPassword(String newPassword) {
      this.newPassword = newPassword;
   }
}
```

# @ValidPassword Annotation

```java
import java.lang.annotation.Documented;
import java.lang.annotation.Retention;
import java.lang.annotation.Target;

import javax.validation.Constraint;
import javax.validation.Payload;

@Documented
@Constraint(validatedBy = PasswordConstraintValidator.class)
@Target({ TYPE, FIELD, ANNOTATION_TYPE })
@Retention(RUNTIME)
public @interface ValidPassword {

   String message() default "Invalid Password";

   Class<?>[] groups() default {};

   Class<? extends Payload>[] payload() default {};

}
```

# Password Constraint Validator

```
public class PasswordConstraintValidator implements ConstraintValidator<ValidPassword, String> {

  @Override
  public boolean isValid(final String password, final ConstraintValidatorContext context) {
    // @formatter:off
    final PasswordValidator validator = new PasswordValidator(Arrays.asList(
          new LengthRule(8, 30),
          new UppercaseCharacterRule(1),
          new DigitCharacterRule(1),
          new SpecialCharacterRule(1),
          new NumericalSequenceRule(3, false),
          new AlphabeticalSequenceRule(3, false),
          new QwertySequenceRule(3, false),
          new WhitespaceRule()));
    final RuleResult result = validator.validate(new PasswordData(password));
    if (result.isValid()) {
      return true;
    }
    context.disableDefaultConstraintViolation();
    context.buildConstraintViolationWithTemplate(Joiner.on(",").join(validator.getMessages(result))).addConstraintViolation();
    return false;
  }
}
```

# Exercises

1. Add Receiver first name, last name, and phone number
2. Automatically attach transaction creation to the currently logged in user
3. Whenever there is a new transaction send transaction pending email