III.A.11. DOKUMEN RULE VALIDASI

Sistem Informasi Penilaian Pegawai Terbaik (SIPIA)

BPS KABUPATEN KUANTAN SINGINGI

2021

**III.A.11 RULE VALIDASI SISTEM INFORMASI PENILAIAN PEGAWAI TERBAIK (SIPIA)**

1. **Deskripsi Singkat SIstem**

Sistem Informasi Penilaian Pegawai Terbaik (SIPIA) adalah sistem penilaian pegawai terbaik dengan cara menilai penerapan nilai-nilai PIA pegawai berbasis *web*. SIPIA memiliki fitur-fitur sebagai berikut:

* Form penilaian untuk seluruh pegawai di-*generate* secara otomatis untuk setiap periodenya.
* Form penilaian terdiri dari penilaian profesionalitas, integritas, dan keamanahan pegawai.
* Hanya pengguna yang telah masuk kedalam sistem yang dapat mengakses form penilaian.
* Memiliki tampilan status penilaian pegawai terhadap pegawai lain apakah sudah atau belum lengkap.
* Memiliki fitur untuk memantau status kelengkapan penilaian seluruh pegawai.
* Dapat menampilkan hasil penilaian sementara secara *realtime*.
* Dapat menyimpan *track record* penilaian pegawai kedalam *database*.

Penjelasan dari gambar 1 adalah sebagai berikut:

1. Form penilaian untuk seluruh pegawai di-*generate* secara otomatis oleh sistem untuk setiap periodenya.
2. Pegawai wajib menilai seluruh pegawai kecuali dirinya sendiri.
3. Kasubbag umum memberikan informasi *progress* penilaian ke Kepala BPS.
4. Kepala BPS memilih pegawai terbaik dari 5 kandidat pegawai dengan nilai tertinggi.



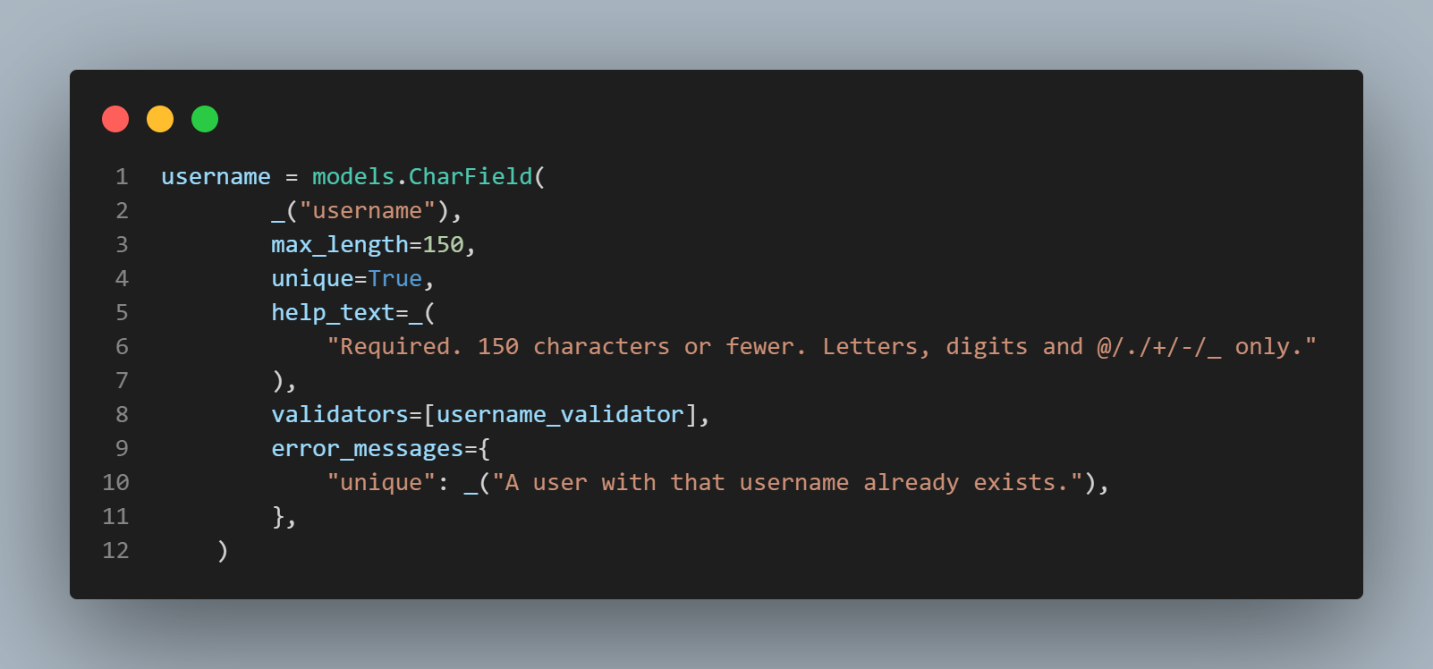
**Gambar 1**. Gambaran umum SIPIA

SIPIA termasuk dalam sistem aplikasi kompleks karena memilki lebih dari 5 subsistem, diantaranya login, home, session, CRUD (*create*, *read*, *update*, *delete*) data, dan monitoring.

1. **Daftar *Rule* Validasi**

**Login dan Registrasi**

* *Field* *username* dan password tidak boleh kosong.
* Atribut username panjang maksimal 150 karakter dan harus unik.
* Atribut password minimal 8 karakter, tidak boleh telalu mirip dengan informasi pribadi, tidak boleh angka semua. dam tidak boleh berupa sandi yang umum digunakan, seperti abcd, 123456, dll.



**Gambar 2**. Rule validasi *username*

def validate\_password(password, user=None, password\_validators=None):

    """

    Validate that the password meets all validator requirements.

    If the password is valid, return ``None``.

    If the password is invalid, raise ValidationError with all error messages.

    """

    errors = []

    if password\_validators is None:

        password\_validators = get\_default\_password\_validators()

    for validator in password\_validators:

        try:

            validator.validate(password, user)

        except ValidationError as error:

            errors.append(error)

    if errors:

        raise ValidationError(errors)

def password\_changed(password, user=None, password\_validators=None):

    """

    Inform all validators that have implemented a password\_changed() method

    that the password has been changed.

    """

    if password\_validators is None:

        password\_validators = get\_default\_password\_validators()

    for validator in password\_validators:

        password\_changed = getattr(validator, "password\_changed", lambda \*a: None)

        password\_changed(password, user)

def password\_validators\_help\_texts(password\_validators=None):

    """

    Return a list of all help texts of all configured validators.

    """

    help\_texts = []

    if password\_validators is None:

        password\_validators = get\_default\_password\_validators()

    for validator in password\_validators:

        help\_texts.append(validator.get\_help\_text())

    return help\_texts

def \_password\_validators\_help\_text\_html(password\_validators=None):

    """

    Return an HTML string with all help texts of all configured validators

    in an <ul>.

    """

    help\_texts = password\_validators\_help\_texts(password\_validators)

    help\_items = format\_html\_join(

        "", "<li>{}</li>", ((help\_text,) for help\_text in help\_texts)

    )

    return format\_html("<ul>{}</ul>", help\_items) if help\_items else ""

password\_validators\_help\_text\_html = lazy(\_password\_validators\_help\_text\_html, str)

class MinimumLengthValidator:

    """

    Validate that the password is of a minimum length.

    """

    def \_\_init\_\_(self, min\_length=8):

        self.min\_length = min\_length

    def validate(self, password, user=None):

        if len(password) < self.min\_length:

            raise ValidationError(

                ngettext(

                    "This password is too short. It must contain at least "

                    "%(min\_length)d character.",

                    "This password is too short. It must contain at least "

                    "%(min\_length)d characters.",

                    self.min\_length,

                ),

                code="password\_too\_short",

                params={"min\_length": self.min\_length},

            )

    def get\_help\_text(self):

        return ngettext(

            "Your password must contain at least %(min\_length)d character.",

            "Your password must contain at least %(min\_length)d characters.",

            self.min\_length,

        ) % {"min\_length": self.min\_length}

def exceeds\_maximum\_length\_ratio(password, max\_similarity, value):

    """

    Test that value is within a reasonable range of password.

    The following ratio calculations are based on testing SequenceMatcher like

    this:

    for i in range(0,6):

      print(10\*\*i, SequenceMatcher(a='A', b='A'\*(10\*\*i)).quick\_ratio())

    which yields:

    1 1.0

    10 0.18181818181818182

    100 0.019801980198019802

    1000 0.001998001998001998

    10000 0.00019998000199980003

    100000 1.999980000199998e-05

    This means a length\_ratio of 10 should never yield a similarity higher than

    0.2, for 100 this is down to 0.02 and for 1000 it is 0.002. This can be

    calculated via 2 / length\_ratio. As a result we avoid the potentially

    expensive sequence matching.

    """

    pwd\_len = len(password)

    length\_bound\_similarity = max\_similarity / 2 \* pwd\_len

    value\_len = len(value)

    return pwd\_len >= 10 \* value\_len and value\_len < length\_bound\_similarity

class UserAttributeSimilarityValidator:

    """

    Validate that the password is sufficiently different from the user's

    attributes.

    If no specific attributes are provided, look at a sensible list of

    defaults. Attributes that don't exist are ignored. Comparison is made to

    not only the full attribute value, but also its components, so that, for

    example, a password is validated against either part of an email address,

    as well as the full address.

    """

    DEFAULT\_USER\_ATTRIBUTES = ("username", "first\_name", "last\_name", "email")

    def \_\_init\_\_(self, user\_attributes=DEFAULT\_USER\_ATTRIBUTES, max\_similarity=0.7):

        self.user\_attributes = user\_attributes

        if max\_similarity < 0.1:

            raise ValueError("max\_similarity must be at least 0.1")

        self.max\_similarity = max\_similarity

    def validate(self, password, user=None):

        if not user:

            return

        password = password.lower()

        for attribute\_name in self.user\_attributes:

            value = getattr(user, attribute\_name, None)

            if not value or not isinstance(value, str):

                continue

            value\_lower = value.lower()

            value\_parts = re.split(r"\W+", value\_lower) + [value\_lower]

            for value\_part in value\_parts:

                if exceeds\_maximum\_length\_ratio(

                    password, self.max\_similarity, value\_part

                ):

                    continue

                if (

                    SequenceMatcher(a=password, b=value\_part).quick\_ratio()

                    >= self.max\_similarity

                ):

                    try:

                        verbose\_name = str(

                            user.\_meta.get\_field(attribute\_name).verbose\_name

                        )

                    except FieldDoesNotExist:

                        verbose\_name = attribute\_name

                    raise ValidationError(

                        \_("The password is too similar to the %(verbose\_name)s."),

                        code="password\_too\_similar",

                        params={"verbose\_name": verbose\_name},

                    )

    def get\_help\_text(self):

        return \_(

            "Your password can’t be too similar to your other personal information."

        )

class CommonPasswordValidator:

    """

    Validate that the password is not a common password.

    The password is rejected if it occurs in a provided list of passwords,

    which may be gzipped. The list Django ships with contains 20000 common

    passwords (lowercased and deduplicated), created by Royce Williams:

    https://gist.github.com/roycewilliams/281ce539915a947a23db17137d91aeb7

    The password list must be lowercased to match the comparison in validate().

    """

    @cached\_property

    def DEFAULT\_PASSWORD\_LIST\_PATH(self):

        return Path(\_\_file\_\_).resolve().parent / "common-passwords.txt.gz"

    def \_\_init\_\_(self, password\_list\_path=DEFAULT\_PASSWORD\_LIST\_PATH):

        if password\_list\_path is CommonPasswordValidator.DEFAULT\_PASSWORD\_LIST\_PATH:

            password\_list\_path = self.DEFAULT\_PASSWORD\_LIST\_PATH

        try:

            with gzip.open(password\_list\_path, "rt", encoding="utf-8") as f:

                self.passwords = {x.strip() for x in f}

        except OSError:

            with open(password\_list\_path) as f:

                self.passwords = {x.strip() for x in f}

    def validate(self, password, user=None):

        if password.lower().strip() in self.passwords:

            raise ValidationError(

                \_("This password is too common."),

                code="password\_too\_common",

            )

    def get\_help\_text(self):

        return \_("Your password can’t be a commonly used password.")

class NumericPasswordValidator:

    """

    Validate that the password is not entirely numeric.

    """

    def validate(self, password, user=None):

        if password.isdigit():

            raise ValidationError(

                \_("This password is entirely numeric."),

                code="password\_entirely\_numeric",

            )

    def get\_help\_text(self):

        return \_("Your password can’t be entirely numeric.")

**Gambar 3**. Rule validasi *password*

**Input Penilaian Kegiatan**

* Atribut profesional, integritas hanya dapat diisi dengan angka.



**Gambar 4**. Rule validasi entri penilaian PIA