**III.A.9 RULE VALIDASI SISTEM INFORMASI PENILAIAN CKP PEGAWAI (SICAKEP)**

1. **Deskripsi Singkat SIstem**

Sistem Informasi Penilaian CKP Pegawai (SICAKEP) adalah sistem penilaian CKP *online* dengan menerapkan konsep baru serta menerapkan sistem berbasis *web application*. SICAKEP membuat basis data yang menyimpan seluruh data CKP setiap pegawai serta basis data butir kegiatan fungsional beserta besaran angka kreditnya. Sistem ini berbasis web sehingga setiap pegawai dapat melakukan entri data capaian hasil pekerjaan yang telah diselesaikan secara langsung dimana saja dan kapan saja. Kemudian atasan bersangkutan dapat memberikan persetujuan dan penilaian hasil pekerjaan bawahannya dimana saja dan kapan saja. Dengan adanya sistem tersebut, para pegawai dapat melihat *progress* capaian kinerjanya dari waktu ke waktu dan seorang atasan dapat melakukan evaluasi capaian kinerja bawahannya sesuai dengan periode yang ditentukan.

Gambaran umum SICAKEP digambarkan pada proses berikut:



**Gambar 1**. Gambaran umum SICAKEP

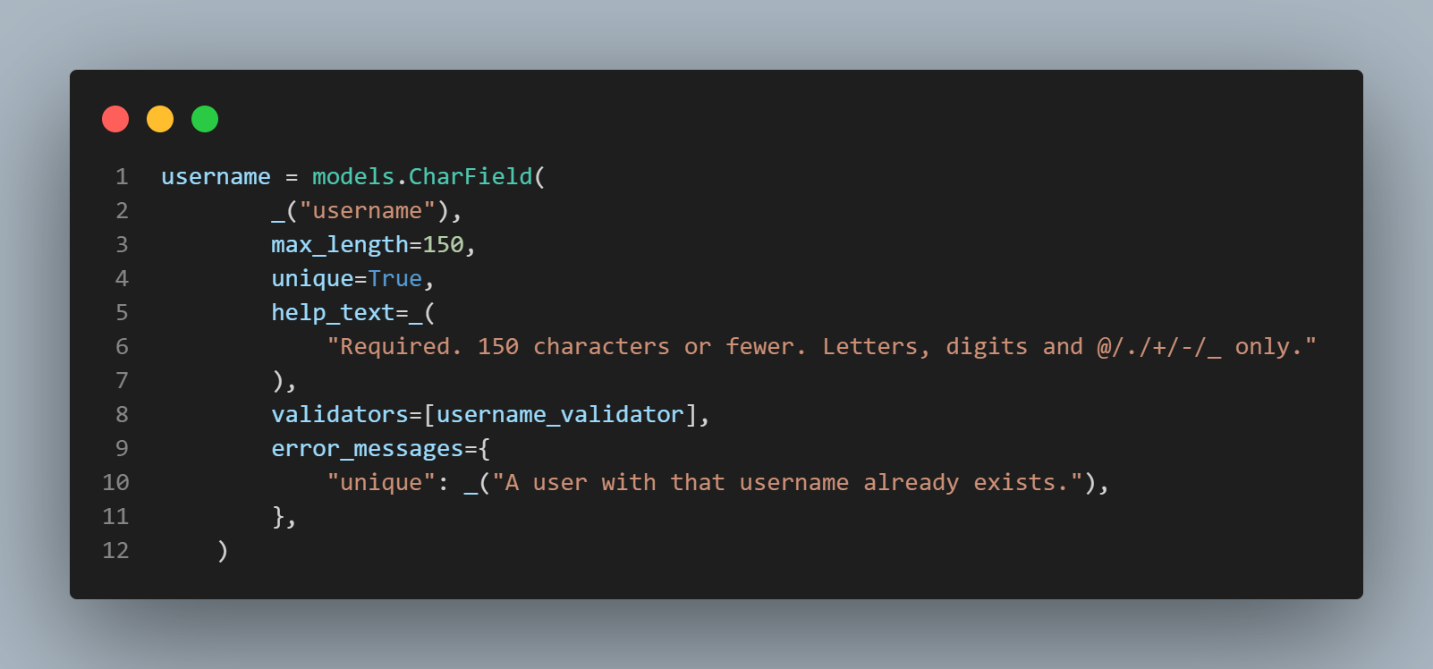
1. Kepala BPS pertama kali harus melakukan pembagian kerja pegawai sesuai tim kerja, untuk melakukan hal tersebut kepala BPS membutuhkan informasi bobot dari setiap kegiatan.
2. Ketua tim menentukan target pekerjaan setiap kegiatan kepada pegawai, dengan mempertimbangkan bobot dari setiap kegiatan.
3. Pegawai BPS berusaha memenuhi target yang telah diberikan.
4. Pegawai BPS menyusun laporan CKP sejalan dengan pemenuhan target pekerjaan, pemenuhan target pekerjaan dipantau oleh ketua tim.
5. Subbagian umum memantau proses penyusunan CKP pegawai.
6. Ketua tim melaporkan capaian target kinerja pegawai ke Kepala BPS.
7. Kepala BPS meminta laporan CKP dari pegawai ke Kasubbag Umum untuk menilai CKP pegawai berdasarkan informasi yang telah didapat dari ketua tim.
8. Kepala BPS memberikan hasil penilaian CKP ke subbag umum untuk diinput ke sistem sebagai dasar penilaian tunjangan kinerja.

SICAKEP termasuk dalam sistem aplikasi kompleks karena memilki lebih dari 5 subsistem, diantaranya login, home, session, CRUD (*create*, *read*, *update*, *delete*) data, export data, manajemen pengguna, 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",

            )

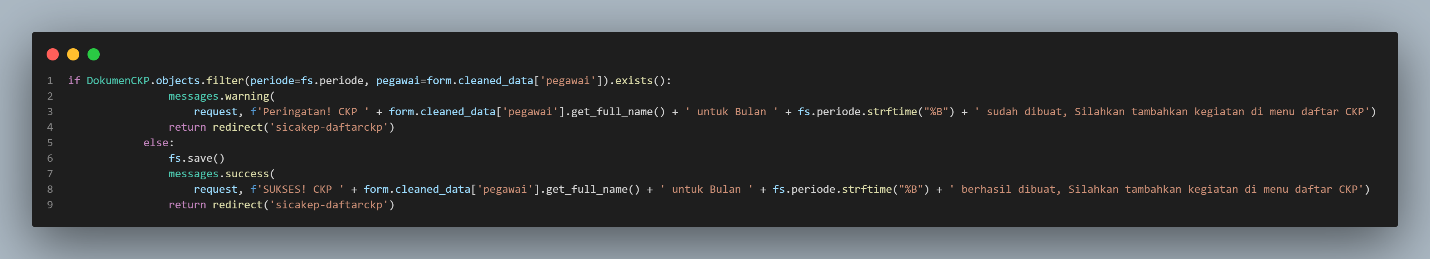
    def get\_help\_text(self):

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

**Gambar 3**. Rule validasi password

**Buat Dokumen CKP Pegawai**

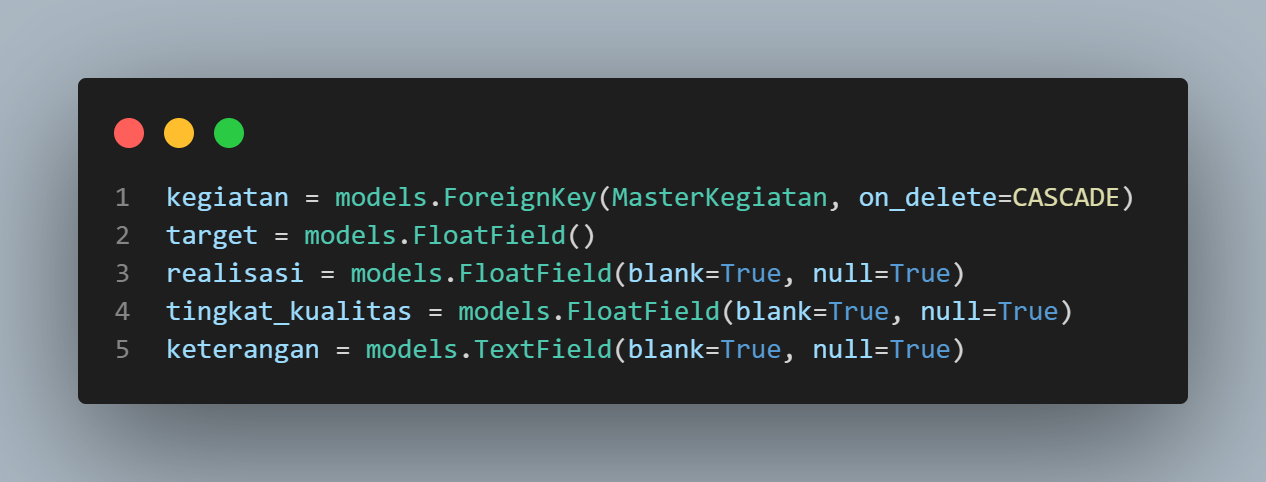
* Dokumen CKP yang sudah pernah dibuat tidak bisa dibuat kembali.



**Gambar 4**. Rule validasi buat CKP pegawai

**Input Penilaian Kegiatan**

* Atribut target, realisasi, dan tingkat kualitas kegiatan hanya dapat diisi dengan angka.



**Gambar 5**. Rule validasi penilaian kegiatan