L A M P I R A N

user controller.py

```
from fastapi import status,
HTTPException
from sqlalchemy.orm import
Session
from libs import hash

from models import UserModel
from schemas import
user_schema
from libs import auth
from controllers import
pond_controller


async def
store(request:user_schema.Crea
te, db:Session):
    to_add =
UserModel(name=request.name,
email=request.email,
password=hash.bcrypt(request.p
assword))
    if
to_add.check_email_has_taken:
        raise HTTPException(

status_code=status.HTTP_409_CO
NFLICT,
            detail="No. Telpon
Sudah Digunakan"
        )
    try:
        db.add(to_add)
        db.commit()

    except:
        raise HTTPException(
            status_code=400,
detail="Ada masalah"
        )
    await
pond_controller.store(to_add.i
d, request.pond, db)
    db.refresh(to_add)
    return to_add

async def update(id:int,
request:user_schema.Update,
db:Session):
    to_update =
db.query(UserModel).filter(Use
rModel.id==id)
    user      =
to_update.first()
    if user is None:
        raise
HTTPException(status_code=404,
detail="data tidak ditemukan")
    check_user =
db.query(UserModel).filter(Use
rModel.id!=user.id,
UserModel.email==request.email
).first()
    if check_user:
        raise HTTPException(

status_code=status.HTTP_409_CO
NFLICT,
            detail="email
sudah ada"
        )

    request.password =
hash.bcrypt(request.password)

to_update.update(request.dict(
))
    db.commit()
    db.refresh(user)
    return user

async def
get_user(username:str, db:
Session):
    user =
db.query(UserModel).filter(Use
rModel.email==username).first(
)
    db.close()
    return user

async def
login_for_access_token(request
, db:Session):
    user = await
get_user(request.username, db)
    if user is None or not
hash.verify(request.password,
user.password):
        raise
HTTPException(detail="password
atau email salah",
status_code=401)
    access_token = await
auth.create_access_token({"sub
" : user.email})
    return {
        "access_token" :
access_token,
```

```
        "token_type"    :
"bearer"
    }
```

title controller.py

```
from sqlalchemy.orm import
Session

from fastapi import
HTTPException
from schemas import
title_schema
from models import TitleModel
from controllers import
data_controller


async def reads(pond_id:int,
db:Session):
    titles =
db.query(TitleModel).filter(Ti
tleModel.pond_id==pond_id).all
()
    data   = await
data_controller.read_by_title(
titles, db)
    if not titles:
        raise
HTTPException(status_code=400,
detail="tambak tidak meiliki
title")
    return data

async def create():
    pass

async def store(pond_id:int,
requests:list[title_schema.Cre
ate], db:Session):
    count_title =
db.query(TitleModel).filter(Ti
tleModel.pond_id==pond_id).cou
nt()
    try:
        for i in
range(len(requests)):
            var =
f"X{i+1+count_title}"
            to_add =
TitleModel(name=requests[i].na
me, var=var, pond_id=pond_id)
            db.add(to_add)
        db.commit()
    except Exception:
        raise HTTPException(
            status_code=400,
detail=requests[0].name
        )
    return {
        "message" : "title
berhasil ditambahkan"
    }
async def edit(title_id:int,
pond_id:int, db:Session,
user):
    title =
db.query(TitleModel).filter(Ti
tleModel.id==title_id,
TitleModel.pond_id==pond_id).f
irst()
    if title is None:
        raise
HTTPException(status_code=404,
detail="title tidak
ditemukan")
    return title

async def update(title_id:int,
pond_id,
request:title_schema.Create,
db:Session):
    to_update =
db.query(TitleModel).filter(Ti
tleModel.id==title_id,
TitleModel.pond_id == pond_id)
    if to_update.first() is
None:
        raise
HTTPException(status_code=404,
detail="title tidak
ditemukan")

to_update.update({"name":reque
st.name})
    db.commit()
    return {"message" : "title
berhasil diubah"}

async def
destroy(title_id:int,
pond_id:int, db:Session):
    to_destroy =
db.query(TitleModel).filter(Ti
tleModel.id==title_id,
TitleModel.pond_id==pond_id)
    if to_destroy.first() is
None:
        raise
HTTPException(status_code=404,
detail="gagal menghapus
title")
```

```
    await
data_controller.destroy_with_t
itle(title_id, db)
    to_destroy.delete()

    to_updates =
db.query(TitleModel).where(Tit
leModel.pond_id==pond_id).all(
)
    for i in
range(len(to_updates)):
        to_updates[i].var =
f"X{i+1}"
    db.commit()
    return {
        "message" : "title
berhasil di hapus"
    }

async def
destroy_with_pond(pond_id:int,
db:Session):
    to_destroys =
db.query(TitleModel).filter(Ti
tleModel.pond_id==pond_id)
    if not to_destroys.all():
        return False
    to_destroys.delete()
    return True

async def
get_title(pond_id:int,
db:Session):
    titles =
db.query(TitleModel).filter(Ti
tleModel.pond_id==pond_id).all
()
    return titles
```

data controller.py

```
from sqlalchemy.orm import
Session

from models import DataModel,
TitleModel
from schemas import
data_schema

async def
reads(production_id:int,
db:Session):
    data =
db.query(DataModel).filter(Dat
aModel.production_id==producti
on_id).all()
    return data

async def
create(production_id:int,
pond_id:int,
requests:list[data_schema.Crea
te], db:Session):
    try:
        for item in requests:
            title =
db.query(TitleModel).filter(Ti
tleModel.id==item.title_id,
TitleModel.pond_id==pond_id).f
irst()
            if title:
                to_add =
DataModel(production_id=produc
tion_id, **item.dict())
                db.add(to_add)
        db.commit()
    except:
        return False
    return True

async def
edit(production_id:int,
db:Session):
    data =
db.query(DataModel).filter(Dat
aModel.production_id==producti
on_id).all()
    if len(data) == 0:
        return False
    return data

async def
update(production_id:int,
requests:list[data_schema.Upda
te], db:Session):
    for item in requests:
        to_update =
db.query(DataModel).filter(Dat
aModel.id==item.id,
DataModel.production_id==produ
ction_id)
        if to_update.first()
is None:
            return False

to_update.update({"value":item
.value})
    return True

async def
destroy(production_id:int,
db:Session):
```

```
    to_delete =
db.query(DataModel).filter(Dat
aModel.production_id==producti
on_id)
    if int(to_delete.count())
== 0:
        return False
    to_delete.delete()
    return True

async def
destroy_with_title(title_id:in
t, db:Session):
    to_deletes =
db.query(DataModel).filter(Dat
aModel.title_id==title_id)
    if not to_deletes.all():
        return True
    to_deletes.delete()
    return True

async def
read_by_title(titles:list,
db:Session):
    for x in titles:
        data =
db.query(DataModel).filter(Dat
aModel.title_id==x.id).all()
        x.data = data
    return titles
```

pond_controller.py

```
from fastapi import status,
HTTPException
from sqlalchemy.orm import
Session
from schemas import
pond_schema
from models import PondModel,
DataModel, ProductionModel
from controllers import
title_contoller,
production_controller
from libs.regression import
Regression


async def read(pond_id:int,
user_id: int, db:Session):
    pond =
db.query(PondModel).filter(Pon
dModel.user_id==user_id,\

PondModel.id==pond_id).first()
    if not pond:
        raise HTTPException(

status_code=status.HTTP_404_NO
T_FOUND,
            detail="gagal
membuat data tambak"
        )
    productions = await
production_controller.reads(po
nd_id, db)
    titles = await
title_contoller.get_title(pond
_id, db)
    pond.productions =
productions
    pond.title = titles

    regression = Regression()
    for title in titles:
        setattr(regression,
title.var.upper(),
db.query(DataModel).filter(Dat
aModel.title_id==title.id).all
())
    setattr(regression, "Y",
db.query(ProductionModel).filt
er(ProductionModel.pond_id==po
nd_id).all())

    pond.regression =
regression.get_result(["A",
"A1", "A2", "A3", "A4"])
    return pond


async def reads(user_id:int,
db:Session):
    ponds  =
db.query(PondModel).filter(Pon
dModel.user_id==user_id).all()
    for pond in ponds:
        data = await
title_contoller.reads(pond.id,
db)
        production = await
production_controller.reads(po
nd.id, db)
        pond.factor_production
= data
        pond.production =
production
    return ponds

async def create():
    pass
```

```
async def store(user_id:int,
request:pond_schema.Create,
db:Session):
    to_add =
PondModel(location=request.loc
ation,
land_area=request.land_area,
user_id=user_id)
    try:
        db.add(to_add)
        db.commit()
    except:
        raise HTTPException(

status_code=status.HTTP_406_NO
T_ACCEPTABLE,
            detail="gagal
membuat tambak"
        )
    await
title_contoller.store(to_add.i
d, request.titles, db)
    db.refresh(to_add)
    return to_add

async def edit(pond_id:int,
user_id:int, db:Session):
    pond =
db.query(PondModel).filter(Pon
dModel.id==pond_id,
PondModel.user_id==user_id).fi
rst()
    if pond is None:
        raise
HTTPException(status_code=404,
detail="Tambak tidak
ditemukan")
    return pond

async def update(pond_id:int,
user_id:int,
request:pond_schema.Update,
db:Session):
    to_update =
db.query(PondModel).filter(Pon
dModel.id==pond_id,
PondModel.user_id==user_id)
    if to_update.first() is
None:
        raise
HTTPException(status_code=400,
detail="gagal mengupdate
tambak")
    try:
to_update.update(request.dict(
))
        db.commit()
    except:
        raise
HTTPException(status_code=400,
detail="gagal mengupdate
tambak")
    return {"message" :
"tambak berhasil di update"}

async def destroy(pond_id:int,
user_id:int, db:Session):
    err_handler =
HTTPException(status_code=404,
detail="gagal menghapus
tambak")
    to_delete =
db.query(PondModel).filter(Pon
dModel.id==pond_id,
PondModel.user_id==user_id).fi
rst()
    if to_delete is None:
        return err_handler
    del_production = await
production_controller.destroy_
with_pond(pond_id, db)
    if not del_production:
        return err_handler
    del_title = await
title_contoller.destroy_with_p
ond(pond_id, db)
    if not del_title:
        return err_handler
    db.delete(to_delete)
    db.commit()
    return {
        "message" : "tambak
berhasil dihapus"
    }

async def
get_pond(pond_id:int,
user_id:int, db:Session):
    pond =
db.query(PondModel).filter(Pon
dModel.user_id==user_id,\

PondModel.id==pond_id).first()
    return pond


async def
get_ponds(user_id:int,
db:Session):
```

```
    ponds  =
db.query(PondModel).filter(Pon
dModel.user_id==user_id).all()
    return ponds
```

production controller.py

```
from fastapi import
HTTPException
from sqlalchemy.orm import
Session

from models import
ProductionModel, TitleModel
from schemas import
production_schema
from controllers import
data_controller,
pond_controller

async def reads(pond_id:int,
db:Session):
    productions =
db.query(ProductionModel).filt
er(ProductionModel.pond_id==po
nd_id).all()
    for production in
productions:
        data = await
data_controller.reads(producti
on.id, db)
        production.data = data
    return productions

async def create(pond_id:int,
db:Session):
    titles =
db.query(TitleModel).filter(Ti
tleModel.pond_id==pond_id).all
()
    if not titles:
        raise
HTTPException(status_code=404,
detail="title tidak
ditemukan")
    return titles

async def store(pond_id:int,
user_id:int,
request:production_schema.Crea
te, db:Session):
    to_add =
ProductionModel(value=request.
value, pond_id=pond_id)
    try:
        db.add(to_add)
        db.commit()
    except:
        raise
HTTPException(status_code=400,
detail="gagal menambahakan
data")
    data = await
data_controller.create(to_add.
id, pond_id, request.data, db)
    if not data:
        db.delete(to_add)
        raise
HTTPException(status_code=400,
detail="gagal menambahakan
data")
    items = await
pond_controller.read(pond_id,
user_id, db)
    return {
        "detail" : "data
berhasil ditambahkan",
        "items" : items
    }

async def
edit(production_id:int,
pond_id:int, db:Session):
    productions =
db.query(ProductionModel).wher
e(ProductionModel.id==producti
on_id,\

ProductionModel.pond_id==pond_
id).first()
    if productions is None:
        raise
HTTPException(status_code=400,
detail="gagal mengambil data")
    return productions


async def
update(production_id:int,
pond_id:int, user_id:int,
request:production_schema.Upda
te, db:Session):
    to_update =
db.query(ProductionModel).filt
er(ProductionModel.id==product
ion_id,\

ProductionModel.pond_id==pond_
id)
    if to_update.first() is
None:
```

```
        raise
HTTPException(status_code=404,
detail="gagal mengubah data")

to_update.update({"value":requ
est.value})
    data_update = await
data_controller.update(product
ion_id, request.data, db)
    if not data_update:
        raise
HTTPException(status_code=404,
detail="gagal mengupdate
data")
    db.commit()
    items = await
pond_controller.read(pond_id,
user_id, db)
    return {
        "detail" : "data
berhasil ditambahkan",
        "items" : items
    }

async def
destroy(production_id:int,
pond_id:int, user_id:int,
db:Session):
    credentials_exception =
HTTPException(status_code=400,
detail="gagal menghapus data")
    delete_data = await
data_controller.destroy(produc
tion_id, db)
    if not delete_data:
        return
credentials_exception
    to_delete =
db.query(ProductionModel).filt
er(ProductionModel.id==product
ion_id,\

ProductionModel.pond_id==pond_
id)
    if to_delete.first() is
None:
        db.rollback()
        return
credentials_exception
    to_delete.delete()
    db.commit()
    items = await
pond_controller.read(pond_id,
user_id, db)
    return {
        "detail" : "data
berhasil dihapus",
        "items" : items
    }

async def
destroy_with_pond(pond_id:int,
db:Session):
    to_deletes =
db.query(ProductionModel).filt
er(ProductionModel.pond_id==po
nd_id)
    if not to_deletes:
        return False
    for to_delete in
to_deletes.all():
        delete_data = await
data_controller.destroy(to_del
ete.id, db)
        if not delete_data:
            return False
    to_deletes.delete()
    return True


# async def edit(id:int,
pond_id:int, db:Session):
#     productions =
db.query(ProductionModel).wher
e(ProductionModel.id==id).firs
t()
#     data        = await
data_controller.edit(id, db)
#     if productions is None
or not data:
#         raise
HTTPException(status_code=400,
detail="gagal mengambil data")
#     titles     = await
create(pond_id, db)
#     return {
#         "productions" :
{"id":productions.id,"value_pr
oduction":productions.value,
"data" : data},
#         "title"       :
titles
#     }
```

auth.py

```
import os
from dotenv import load_dotenv
from fastapi import Depends,
HTTPException
```

```
from fastapi.responses import
JSONResponse
# from fastapi.security import
OAuth2PasswordBearer
from libs.oauth2 import
OAuth2PasswordBearer
from jose import jwt, JWTError
from datetime import datetime,
timedelta

from controllers import
user_controller,
pond_controller
from schemas.user_schema
import User as UserSchema,
Show as UserShow
from database import get_db

# to get a string like this
run:
# openssl rand -hex 32
load_dotenv()
SECRET_KEY     =
str(os.environ.get("SECRET_KEY
"))
ALGORITHM      = "HS256"
ACCESS_TOKEN_EXPIRE_DAYS = 30
oauth2_schema =
OAuth2PasswordBearer(tokenUrl=
"login")


async def
create_access_token(data:dict)
:
    to_encode = data.copy()
    expire    =
datetime.utcnow() +
timedelta(days=ACCESS_TOKEN_EX
PIRE_DAYS)
    to_encode.update({
        "exp" : expire
    })
    encoded_jwt =
jwt.encode(to_encode,SECRET_KE
Y, algorithm=ALGORITHM)
    return encoded_jwt

async def
get_current_user(token:str =
Depends(oauth2_schema), db =
Depends(get_db)):
    credentials_exception =
HTTPException(status_code=401,
detail= "token anda
kedaluarsa, silahkan login
ulang", headers={"WWW-
Authenticate": "Bearer", "set-
cookie" :
"access_token=deleted;path=/;e
xpires=Thu, 01 Jan 1970
00:00:00 GMT"})
    try:
        payload     =
jwt.decode(token, SECRET_KEY,
algorithms=[ALGORITHM])
        username    =
payload.get("sub")
        if username is None:
            raise
credentials_exception
    except JWTError:
        raise
credentials_exception
    user  = await
user_controller.get_user(usern
ame, db)
    return UserSchema(
        id=user.id,
email=user.email,
name=user.name
    )

async def
get_current_user_and_pond(pond
_id:int, db = Depends(get_db),
user:UserSchema =
Depends(get_current_user)):
    pond = await
pond_controller.get_pond(pond_
id, user.id, db)
    if pond is None:
        raise
HTTPException(status_code=401,
message="tambak salah")
    return
UserShow(name=user.name,
id=user.id, email=user.email,
pond_id=pond.id,\

location_pond=pond.location,
land_area=pond.land_area)
```

regression.py

```
from scipy import stats
import numpy as np

class Regression():
    K = 4
    def sum(self, var:str):
        result = 0
```

```
        for item in
getattr(self, var.upper()):
            result +=
float(item.value)
        return result

    def sumSquared2(self,
var:str):
        result = 0
        for item in
getattr(self, var.upper()):
            result +=
float(item.value)**2
        return result

    def sumMultiple(self,
var1:str, var2:str):
        result = 0
        for x in
range(len(getattr(self,
var1.upper()))):
            result =
(float(getattr(self,
var1.upper())[x].value) *
float(getattr(self,
var2.upper())[x].value)) +
result
        return result

    def detMatriks(self,
type:str|None = None):
        matriks = np.matrix([
            [len(self.Y),
self.sum("x1"),
self.sum("x2"),
self.sum("x3")],
            [self.sum("x1"),
self.sumSquared2("x1"),
self.sumMultiple("x1", "x2"),
self.sumMultiple("x1", "x3")],
            [self.sum("x2"),
self.sumMultiple("x1", "x2"),
self.sumSquared2("x2"),
self.sumMultiple("x2", "x3")],
            [self.sum("x3"),
self.sumMultiple("x1", "x3"),
self.sumMultiple("x2", "x3"),
self.sumSquared2("x3")],
        ])
        y = self.sum("y")
        x1y =
self.sumMultiple("x1", "y")
        x2y =
self.sumMultiple("x2", "y")
        x3y =
self.sumMultiple("x3", "y")
        number = [number for
number in type.split("A") if
number.isnumeric()]
        if(len(number) > 0):
            number =
int(number[0]) - 1

matriks.itemset((0,number), y)

matriks.itemset((1,number),
x1y)

matriks.itemset((2,number),
x2y)

matriks.itemset((3,number),
x3y)
        return
np.linalg.det(matriks)


    def get_result(self,
matriks_name:list):
        determinans = {}
        for matriks in
matriks_name:

determinans[matriks] =
(self.detMatriks(matriks))

        if (len([determinan
for determinan in
determinans.values() if
abs(determinan) == 0.0]) > 0)
or self.K >= len(self.Y):
            return {
                "status" :
False
            }

        determinan_A =
determinans["A"]
        determinans.pop("A")
        koefisiens = {}
        for keys,value in
determinans.items():
            [number] = [number
for number in list(keys) if
number.isnumeric()]
            koefisien = "a" if
int(number) == 1 else
f"b{int(number)-1}"

koefisiens[koefisien] =
```

```
round((value / determinan_A),
3)
            setattr(self,
f"KOEFISIEN_{koefisien.upper()
}", (value / determinan_A))

        mean_square =
self.mean_square()
        koefisien_determinasi
= self.koefisien_determinasi()
        return { "status" :
True, **koefisiens,
**mean_square,
**koefisien_determinasi}



    def Y_prediksi(self,
i:int):
        a, b1, b2, b3 =
float(self.KOEFISIEN_A),
float(self.KOEFISIEN_B1),
float(self.KOEFISIEN_B2),
float(self.KOEFISIEN_B3)
        X1, X2, X3 = self.X1,
self.X2, self.X3

        Y_prediksi = a +
b1*float(X1[i].value) +
b2*float(X2[i].value) +
b3*float(X3[i].value)
        return  Y_prediksi

    def sum_of_square(self) ->
dict:
        SSR = SSE = 0
        Y_AKTUAL = self.Y
        n = len(Y_AKTUAL)
        Y_mean =
self.mean("y")
        for i in range(n):
            Y_prediksi =
self.Y_prediksi(i)
            SSR += (Y_prediksi
- Y_mean)**2
            SSE +=
(float(Y_AKTUAL[i].value) -
Y_prediksi)**2
        df = self.df(n)
        return {
            "SSR" : SSR, "SSE"
: SSE, **df
        }

    def df(self, n):
        # K is many variable
        K = self.K
        df_SSR = K - 1
        df_SSE = n - K
        return {
            "df_SSR":df_SSR,
"df_SSE": df_SSE
        }

    def mean_square(self):
        ss =
self.sum_of_square()
        SSR, SSE, df_SSR,
df_SSE = ss.values()
        MSR = SSR / df_SSR
        MSE = SSE / df_SSE
        return {
            "MSR" : round(MSR,
3), "MSE" : round(MSE, 3),
"F_HITUNG" : round(MSR/MSE,
3), "F_TABLE":
stats.f.ppf(q=1-.05,
dfn=df_SSR, dfd=df_SSE),
            "SSR" : round(SSR,
3), "SSE" : round(SSE, 3),
"df_SSR" : df_SSR, "df_SSE" :
df_SSE
        }


    def
koefisien_determinasi(self):
        Y_AKTUAL, X1_AKTUAL,
X2_AKTUAL, X3_AKTUAL =
self.Y,self.X1,self.X2,self.X3
        Y_MEAN, X1_MEAN,
X2_MEAN, X3_MEAN =
self.mean("y"),self.mean("x1")
,self.mean("x2"),self.mean("x3
")
        b1, b2, b3 =
float(self.KOEFISIEN_B1),
float(self.KOEFISIEN_B2),
float(self.KOEFISIEN_B3)

        x1y = x2y = x3y =
y_square = 0
        for i in
range(len(Y_AKTUAL)):
            y, x1, x2, x3 =
float(Y_AKTUAL[i].value)  -
Y_MEAN,
float(X1_AKTUAL[i].value) -
X1_MEAN,
float(X2_AKTUAL[i].value) -
X2_MEAN,
```

```
float(X3_AKTUAL[i].value) -
X3_MEAN
            x1y += x1*y
            x2y += x2*y
            x3y += x3*y
            y_square += y**2
        koefisien_determinasi
= (b1*x1y + b2*x2y + b3*x3y) /
y_square
        return {
            "x1y" : x1y,
"x2y":x2y, "x3y":x3y,
"y_square":y_square,
"koefisien_determinasi":round(
koefisien_determinasi, 9)
        }

    def mean(self, var:str) ->
float:
        n = len(getattr(self,
var.upper()))
        return
float(self.sum(var) / n)
```

KARTU MONITORING BIMBINGAN

MAHASISWA PROGRAM STUDI TEKNIK INFORMATIKA
FAKULTAS TEKNIK
UNIVERSITAS MUHAMMADIYAH PAREPARE

PROPOSAL

Mahasiswa : Muh. Syawal	**Pembimbing I : Masnur, ST., M.Kom.**
NIM : 218280175	**Pembimbing II : Muhammad Ismail, S.Kom., M.Kom.**
Judul Skripsi : EVALUASI HASIL PRODUKSI UDANG VANAME DENGAN METODE REGRESI LINEAR BERGANDA	

ARAHAN PEMBIMBING I	HARI/TGL & PARAF PEMBIMBING	ARAHAN PEMBIMBING II	HARI/TGL & PARAF PEMBIMBING
Konsultasi 1 - Format penulisan.	10/6-2022	Konsultasi 1 Format penulisan	
Konsultasi 2 - Bahasa Arab kecil miring	12/6-2022	Konsultasi 2 Konsultasi selesai	
Konsultasi 3 - Daftar pustaka dimbalkan	17/6-2022	Konsultasi 3 Desain Sistem	
Konsultasi 4 - Acc.	18/6-2022	Konsultasi 4	
Konsultasi 5		Konsultasi 5	

Lanjut ke halaman sebelah...

Perhatian :

1. Mahasiswa wajib konsultasi minimal 5 kali
2. Kartu ini wajib dibawa oleh mahasiswa disetiap konsultasi dan diisi oleh Pembimbing
3. Kartu ini wajib dilampirkan pada laporan skripsi dan menjadi salah satu persyaratan untuk ikut seminar proposal/ujian skripsi
4. Kartu ini dicetak di atas kertas karton berwarna hijau muda dan dicetak timbal balik

KARTU MONITORING BIMBINGAN

MAHASISWA PROGRAM STUDI TEKNIK INFORMATIKA
FAKULTAS TEKNIK
UNIVERSITAS MUHAMMADIYAH PAREPARE

HASIL

Mahasiswa : MUH. SYAWAL	Pembimbing I : Masnur, ST., M.Kom
NIM : 218280175	Pembimbing II : Marlina, S.Kom,.M.Kom
JUDUL : EVALUASI HASIL PRODUKSI UDANG VANAME DENGAN METODE REGRESI LINEAR BERGANDA	

ARAHAN PEMBIMBING I	HARI/TGL & PARAF PEMBIMBING	ARAHAN PEMBIMBING II	HARI/TGL & PARAF PEMBIMBING
Konsultasi 1 Rumus Regresi Linier berganda dalam Aplikasi.	8/8-2023	Konsultasi 1 Buat Kuesioner Tentang Tambak	25/08/23
Konsultasi 2 Semaikan Activity Diagram → Sequence Diagram → Uji Black Box dan Uji White Box.	14/8-2023	Konsultasi 2 semua Bhs Asing di cetak Miring	28/08/23
Konsultasi 3 Kuisioner Minimal 15 org dan coba di olah Bisa	24/8-2023	Konsultasi 3 Acc Ujian Hasil	
Konsultasi 4 Acc ujian Hasil.	31/8-23	Konsultasi 4	
Konsultasi 5		Konsultasi 5	

Lanjut ke halaman sebelah...

Perhatian :

1. Mahasiswa wajib konsultasi minimal 5 kali
2. Kartu ini wajib dibawa oleh mahasiswa disetiap konsultasi dan diisi oleh Pembimbing
3. Kartu ini wajib dilampirkan pada laporan skripsi dan menjadi salah satu persyaratan untuk ikut seminar proposal/ujian skripsi
4. Kartu ini dicetak di atas kertas karton berwarna hijau muda dan dicetak timbal balik

KARTU MONITORING BIMBINGAN

MAHASISWA PROGRAM STUDI TEKNIK INFORMATIKA
FAKULTAS TEKNIK
UNIVERSITAS MUHAMMADIYAH PAREPARE

SKRIPSI

MAHASISWA : MUH. SYAWAL	**Pembimbing I : Masnur, ST., M.Kom**
NIM : 218280175	**Pembimbing II : Marlina, S.Kom., M.Kom.**
Judul Skripsi : EVALUASI HASIL PRODUKSI UDANG VANAME DENGAN METODE REGRESI LINEAR BERGANDA	

ARAHAN PEMBIMBING I	HARI/TGL & PARAF PEMBIMBING	ARAHAN PEMBIMBING II	HARI/TGL & PARAF PEMBIMBING
Konsultasi 1		Konsultasi 1	
Konsultasi 2		Konsultasi 2	
Konsultasi 3		Konsultasi 3	
Konsultasi 4		Konsultasi 4	
Konsultasi 5		Konsultasi 5	

Acc Ujian Hasil

Acc Ujian Tutup.

Lanjut ke halaman sebelah...

Perhatian :

1. Mahasiswa wajib konsultasi minimal 5 kali
2. Kartu ini wajib dibawa oleh mahasiswa disetiap konsultasi dan diisi oleh Pembimbing
3. Kartu ini wajib dilampirkan pada laporan skripsi dan menjadi salah satu persyaratan untuk ikut seminar proposal/ujian skripsi
4. Kartu ini dicetak di atas kertas karton berwarna hijau muda dan dicetak timbal balik