4. IMPLEMENTASI SISTEM

Pada bab ini akan dibahas mengenai implementasi dari desain sistem yang telah dibuat pada bab sebelumnya. Implementasi sistem berupa penggunaan perangkat lunak yang digunakan, *library, syntax, data cleaning & preprocessing*, *training* dan *testing* model *Linear Programming*. Serta penerapan pembuatan sistem berupa *Graphical User Interface* (*GUI*) yang digunakan untuk menjalankan program optimasi pakan ayam yang dapat digunakan *user* untuk memilih jenis pakan ayam, memilih bahan makanan yang dipakai, dan hasil formulasi pakan ayam dari jenis dan bahan makanan yang sudah dipilih.

4.1. Implementasi Perangkat Lunak yang Digunakan

Penerapan *Machine Learning* pada sistem di penelitian ini berupa *Graphical User Interface* (*GUI*). *GUI* dibuat dengan bahasa pemrograman *Python* versi 3.11.8. dengan bantuan *React*. *Python* juga digunakan dalam proses pengambilan data, perhitungan hingga dengan melatih model *Linear Programming*.

4.2. Library yang Digunakan

Penggunaan *library* dalam penelitian ini sangat penting karena dapat menyediakan *library* dan *tools* yang diperlukan untuk menyelesaikan berbagai program formulasi pakan ayam.

a. Library dan Tools

Library adalah kumpulan kode yang dapat digunakan kembali oleh program lain. Library biasanya berisi fungsi, kelas, atau metode yang telah diprogram sebelumnya dan dapat digunakan untuk menyelesaikan tugas-tugas tertentu. Sedangkan *Tools* adalah aplikasi atau perangkat lunak yang membantu pengembang dalam proses pengembangan, pengujian, dan pemeliharaan perangkat lunak. *Tools* dapat bervariasi dari alat pengembangan kode hingga alat manajemen projek. Berikut ini merupakan Library dan Tools yang digunakan dalam program optimasi pakan ayam:

Tabel 4.1. Tabel Library dan Tools

Tabel yang berisi tentang Library dan Tools beserta dengan kegunaanya

Kategori	Nama Library	Kegunaan
Built-in Libraries	*itertools*	Iterator konstruksi dari *APL*, *Haskell*, dan *SML* diubah menjadi bentuk *Python*.

	requests	Melakukan request berupa seperti *GET, POST, PUT, DELETE,* dan lain-lain.
	json	Melakukan formatting untuk pertukaran data agar lebih ringan dan mudah dibaca dan dioperasikan.
	sys	Menyediakan akses ke beberapa variabel untuk mengirimkan argumen dari backend *API* ke *file Formulate*.
Third-party Libraries	*pulp*	Menyediakan fungsi untuk memodelkan dan menyelesaikan masalah optimasi dengan metode *Linear Programming* dalam integer.
	ortools.linear_solver	*Library* yang disediakan oleh *Google OR-Tools* untuk menyelesaikan masalah optimasi dengan metode *Linear Programming* dalam integer.
	cors	Memungkinkan *resource* di suatu *website* untuk diminta dari domain yang berbeda dari domain asal.
	express	*Framework web* untuk *Node.js* yang dirancang untuk membangun aplikasi *website* dan *API* dengan cepat dan mudah.
	body-parser	*Middleware* untuk *Express* yang digunakan untuk mem-parsing (mengurai) *body* dari permintaan *HTTP.*
	child_process	Menjalankan file *Python* dalam *environment node js.*
	material-tailwind	*UI framework* yang yang berasal dari *Tailwind CSS* yang menyediakan kumpulan *User Interface* yang siap untuk digunakan.
	heroicons	Kumpulan ikon *open-source* yang dirancang khusus untuk digunakan dalam pengembangan *website* dan aplikasi.

	react-router-dom	*Library* untuk React yang digunakan untuk menangani *routing* (pengaturan halaman) dalam aplikasi *website*.
	axios	*Library HTTP client* yang digunakan untuk membuat permintaan *HTTP* dari *browser* atau dari *Node.js.*
	tailwindcss	*Framework CSS* yang memungkinkan user untuk membangun sebuah *User Interface* dengan cepat dan mudah dengan menggunakan kelas-kelas utilitas.
	concurrently	*Package NPM* yang digunakan untuk menjalankan beberapa perintah dalam satu waktu di terminal.
	autoprefixer	*Tools* yang digunakan untuk menambahkan awalan vendor ke properti *CSS* yang memerlukannya yang dilakukan secara otomatis
	postcss	*Tools* untuk pengolahan *CSS* yang memungkinkan *user* untuk melakukan transformasi terhadap kode *CSS* dengan menggunakan berbagai macam *plugin*.
Pulp Specific Libraries	*LpVariable*	Library class yang digunakan untuk metode *Linear Programming* untuk mewakili variabel dalam model matematis.
	LpProblem	*Library* class yang digunakan untuk metode *Linear Programming* untuk merepresentasikan masalah optimasi linier dalam sebuah model matematis.
	LpMinimize	*Library class* yang digunakan untuk metode *Linear Programming*
	lpSum	*Library class* yang digunakan untuk metode *Linear Programming* untuk menentukan solusi optimal dari suatu masalah dengan

		tujuan meminimumkan fungsi objektif, yang biasanya berupa kombinasi linear dari variabel-variabel dalam masalah tersebut.
	value	Nama variabel untuk menyimpan nilai dari suatu ekspresi atau hasil operasi.
Ortools Specific Libraries	*pywraplp*	Modul yang merupakan bagian dari *Google OR-Tools (Operations Research Tools)* untuk *Python* yang berfungsi untuk menyelesaikan masalah optimasi.
database	*mysql*	Sistem manajemen basis data (*database*) yang menggunakan bahasa *SQL (Structured Query Language)* untuk mengakses, mengelola, dan memanipulasi data yang disimpan dalam *database*.

b. Segmen Program

Segmen program adalah bagian dari kode atau instruksi yang membentuk sebuah program komputer. Segmen-segmen ini berfungsi sebagai blok-blok yang lebih kecil dan spesifik yang bersama-sama menyusun keseluruhan program. Berikut ini adalah Tabel segmen program beserta dengan keterangannya.

Tabel 4.2. Tabel Segmen Program

Tabel segmen-segmen program yang digunakan dalam menyusun program optimasi pakan ayam

Segmen Program	Nama Segmen Program
Segmen Program 4.3	Membuat semua kemungkinan persamaan
Segmen Program 4.4	Mengubah hasil persamaan kedalam bentuk *file.txt*
Segmen Program 4.5	Mengambil semua data bahan makananan

Segmen Program 4.6	Mengambil parameter nutrisi (max dan min) berdasarkan id
Segmen Program 4.7	*Initializes solver*
Segmen Program 4.8	Membuat variabel untuk jumlah bahan makanan yang digunakan
Segmen Program 4.9	Membuat *function* untuk meminimalisir harga
Segmen Program 4.10	Menambahkan batasan untuk minimum parameter nutrisi
Segmen Program 4.11	Menambahkan batasan untuk maksimum parameter nutrisi
Segmen Program 4.12	Menambahkan batasan untuk minimum dan maksimum untuk setiap bahan makanan
Segmen Program 4.13	Menambahkan batasan total bahan makanan maksimum 1500 kg
Segmen Program 4.14	Menampilkan hasil bahan makanan yang digunakan
Segmen Program 4.15	Menghitung total nutrisi untuk setiap parameter
Segmen Program 4.16	Menampilkan hasil nutrisi setiap bahan
Segmen Program 4.17	Inisialisasi koneksi ke *database MySQL*
Segmen Program 4.18	Menjalankan file *Python* untuk formulasi pakan ayam
Segmen Program 4.19	Mengambil semua bahan makanan
Segmen Program 4.20	Mengambil semua bahan makanan yang berstatus aktif di *database*
Segmen Program 4.21	Mengganti status dari setiap bahan makanan berdasarkan id yang dipilih
Segmen Program 4.22	Mengganti stok dari setiap bahan makanan berdasarkan id yang dipilih

Segmen Program 4.23	Mengganti minimal dari setiap bahan makanan berdasarkan id yang dipilih
Segmen Program 4.24	Mengganti maksimal dari setiap bahan makanan berdasarkan id yang dipilih
Segmen Program 4.25	Mengambil data parameter minimal berdasarkan id yang dipilih
Segmen Program 4.26	Mengambil semua data parameter minimal yang ada
Segmen Program 4.27	Mengambil data parameter maksimal berdasarkan id yang dipilih
Segmen Program 4.28	Mengambil semua data parameter maksimal yang ada
Segmen Program 4.29	Menghapus parameter makanan berdasarkan id yang dipilih
Segmen Program 4.30	Menambah minimal parameter nutrisi yang baru
Segmen Program 4.31	Menambah maksimal parameter nutrisi yang baru
Segmen Program 4.32	*Formulate*
Segmen Program 4.33	Menambahkan hasil formulasi ke tabel *history*
Segmen Program 4.34	Mengambil semua data *history*
Segmen Program 4.35	Mengambil data *history* berdasarkan id yang dipilih
Segmen Program 4.36	Implemantasi Program
Segmen Program 4.37	*Home*
Segmen Program 4.38	*Footer*
Segmen Program 4.39	*Navbar*
Segmen Program 4.40	*Ingredients*
Segmen Program 4.41	*Nutritions*
Segmen Program 4.42	*Formulate*
Segmen Program 4.43	*Phase*

4.3. Membuat semua kemungkinan persamaan

Pertama-tama, identifikasi dan kelompokan kategori ke dalam *array* untuk mengetahui ada berapa kategori untuk mengelompokkan semua bahan makanan ke dalam *array* kategori tersebut.

Segmen Program 4.1. Memasukan bahan makanan kedalam kategori

```
categories = [CAT2, CAT3, CAT4, CAT5, CAT6, CAT7, CAT8, CAT9,
CAT10, CAT11, CAT12, CAT13]
items = [list(cat.keys()) for cat in categories]
all_combinations = product(*items)
```

Kemudian data didalam *array* tersebut dimasukkan ke dalam *list comprehension* untuk membuat daftar baru bernama *items*. Untuk setiap data dalam *categories, cat.keys()* akan mengembalikan semua data-data yang ada didalamnya dari data kategori tersebut dalam bentuk daftar, kemudian setiap daftar ini ditambahkan ke dalam daftar *items*. Selanjutnya, program ini menggunakan fungsi *product* dari modul *itertools* untuk menghasilkan semua kombinasi kemungkinan dari data-data yang ada dalam data kategori. Tanda * digunakan untuk menyusun kembali daftar *items* menjadi argumen-argumen terpisah untuk fungsi *product*.

4.4. Mengubah hasil persamaan kedalam bentuk file.txt

Pada tahap ini, program tersebut akan membuka sebuah *file* bernama *persamaan.txt* untuk diisi dan di tuliskan semua kombinasi dari item-item yang ada di dalam kategori tersebut dalam ke dalam bentuk persamaan ke dalam *file* tersebut. Kemudian program akan mengiterasi semua kombinasi yang dihasilkan oleh *all_combinations*. Fungsi *enumerate* digunakan untuk mendapatkan indeks idx dari setiap kombinasi, dimulai dari 1. Jadi, setiap kombinasi adalah salah satu dari kombinasi dari data-data yang ada didalam data kategori yang dihasilkan sebelumnya.

Segmen program 4.2. Menuliskan semua kemungkinan persamaan kedalam file.txt

```
with open('persamaan.txt', 'w') as f:
    for idx, combination in enumerate(all_combinations,
start=1):
        equation = ' + '.join([f"{categories[i][item]}{item}"
for i, item in enumerate(combination)])
        f.write(f"{idx}. {equation}\n")
```

Pada tahap selanjutnya, Baris kode tersebut membentuk sebuah string bernama *equation*, yang merupakan persamaan dari data-data dalam kombinasi yang sedang diproses. Proses ini dimulai dengan mengiterasi setiap data dalam kombinasi menggunakan *enumerate(combination),* yang memberikan setiap data bersama dengan indeksnya i. Kemudian, program akan menuliskan semua hasil persamaan yang telah dibentuk ke dalam *file persamaan.txt*. Setiap persamaan diawali dengan nomor indeksnya *idx*, diikuti oleh persamaan itu sendiri, dan diakhiri dengan karakter *newline* *n* untuk memulai baris baru di *file.txt.*

4.5. Mengambil semua data bahan makananan

Pada tahap ini, program mengirim permintaan *HTTP GET* ke *URL http://localhost:3000/api/ingredients/active* untuk mendapatkan data bahan-bahan makanan yang berstatus aktif dari database yang berjalan di *port* 3000. Dengan menggunakan *HTTP requests*, hasilnya akan disimpan ke dalam variabel *get_ingredients.*

Segmen Program 4.3. Mengambil semua data dari database melalui API

```
get_ingredients =
requests.get('http://localhost:3000/api/ingredients/active')
ingredients = get_ingredients.json()
```

Selanjutnya, metode *json()* dipanggil pada *get_ingredients* untuk mengubah *response* dari *database* ke dalam bentuk *JSON*, dan hasilnya disimpan di dalam variabel *ingredients*. Dengan demikian, *ingredients* akan berisi data bahan-bahan yang berstatus aktif yang diambil dari *API* tersebut.

4.6. Mengambil parameter nutrisi (max dan min) berdasarkan id

Pada tahap ini, dilakukan pengambilan id berdasarkan indeks yang dipilih. Jadi nilai tersebut akan disimpan dalam variabel id.

Segmen Program 4.4. Mengambil Parameter Nutrisi melalui API

```
get_max_param =
requests.get(f'http://localhost:3000/api/parameters/max/{id}')
constraints_max = get_max_param.json()


get_min_param =
requests.get(f'http://localhost:3000/api/parameters/min/{id}')
constraints_min = get_min_param.json()
```

Nilai parameter maksimum dan minimum dari database akan diambil melalui *API* berdasarkan id. Kemudian, id akan didapatkan dari langkah sebelumnya untuk melakukan *request* ke *HTTP GET* lalu dikirim ke *URL http://localhost:3000/api/parameters/max/{id}* menggunakan *HTTP requests* untuk mendapatkan nilai parameter maksimum berdasarkan id yang dipilih tersebut, dan hasilnya disimpan dalam variabel *constraints_max* lalu variabel *get_max_param* akan dikonversikan ke dalam bentuk *JSON*. Selanjutnya, request yang sama dikirimkan ke *URL http://localhost:3000/api/parameters/min/{id}* untuk mendapatkan nilai parameter minimum, dan hasilnya disimpan dalam variabel *constraints_min*. Kemudian dengan cara yang sama, variabel *get_min_param* dikonversikan ke dalam bentuk *JSON*. Dengan demikian, tahap ini digunakan untuk mendapatkan nilai batasan maksimum dan minimum untuk parameter formulasi pakan ayam dari *API* berdasarkan id yang dipilih.

4.7. Inisialisasi solver

Pada tahap ini, *Solver* berfungsi untuk membuat sebuah objek pemecah masalah menggunakan library pywraplp dari *Google OR-Tools*, yang merupakan alat untuk optimasi untuk metode L*inear* P*rogramming* dengan campuran integer didalammnya *(MILP/LP).* Dengan metode ini, *pywraplp.Solver.CreateSolver('SCIP')* akan memanggil metode CreateSolver dengan argumen *'SCIP'*, yang menunjukkan bahwa *solver* yang akan dibuat adalah *SCIP (Solving Constraint Integer Programs),* yang membuat *solver* menjadi salah satu metode yang sangat efisien untuk memecahkan masalah di dalam program optimasi.

Segmen Program 4.5. Initializes solver

```
solver = pywraplp.Solver.CreateSolver('SCIP')
```

Objek *solver* yang dihasilkan disimpan dalam variabel *solver*, dan hal ini memudahkan *user* untuk mendefinisikan variabel-variabel, masalah, dan tujuan untuk program optimasi yang ingin diselesaikan menggunakan *SCIP* sebagai backend program optimasi ini. *SCIP* dikenal karena kemampuannya menangani masalah di dalam program optimasi yang sangat kompleks, sehingga metode ini biasanya digunakan untuk pemecahan masalah yang memerlukan solusi efisien dan efektif.

4.8. Membuat variabel untuk jumlah bahan makanan yang digunakan

Pada tahap ini, akan dibuat *dictionary 'food_vars'* yang berisi variabel keputusan untuk setiap bahan makanan yang diambil dari daftar *ingredients['ingredients'].* Tujuan langkah ini untuk mengiterasi setiap item *food* dalam *ingredients['ingredients'].* Untuk setiap *food*, akan membuat variabel keputusan menggunakan metode *solver.NumVar(0, solver.infinity(), food),* yang membuat variabel dengan batas bawah 0 hingga batas atas tak terhingga. Kemudian, *food_vars* akan diubah menjadi nama bahan makanan, dan nilainya merupakan variabel keputusan yang dapat digunakan dalam program optimasi pakan ayam.

Segmen Program 4.6. Membuat Nama Bahan Makanan dan Variabel Keputusan

```
food_vars = {food: solver.NumVar(0, solver.infinity(), food)
for food in ingredients['ingredients']}
```

4.9. Membuat function untuk meminimalisir harga

Pada tahap ini, program ini bertujuan untuk membuat program optimasi pakan ayam dengan memenuhi kebutuhan nutrisi dengan harga yang paling murah. Jadi tujuan objektif dari program optimasi adalah meminimalisir harga. Pertama, membuat objek baru yang dibuat menggunakan algoritma *solver.Objective().* Kemudian akan terjadi sebuah iterasi melalui setiap pasangan di dalam *dictionary food_vars* menggunakan *looping*. Di setiap iterasi, metode *SetCoefficient(var, ingredients['ingredients'][food]['Harga'])* dipanggil pada objek objektif yang dibuat di awal tadi, yang digunakan untuk menetapkan koefisien untuk variabel keputusan var sebesar harga bahan makanan tersebut yang diambil dari daftar *ingredients['ingredients'][food]['Harga'].*

Segmen Program 4.7. Function untuk meminimalisir total harga bahan makanan

```
objective = solver.Objective()
for food, var in food_vars.items():
    objective.SetCoefficient(var,
ingredients['ingredients'][food]['Harga'])
objective.SetMinimization()
```

Fungsi koefisien pada variabel keputusan dalam objektif adalah melihat dan mengambil nilai harga bahan makanan tersebut. Setelah semua koefisien ditetapkan, *objective.SetMinimization()* berfungsi untuk menetapkan tujuan minimisasi, yang berarti model akan mencoba untuk meminimalkan atau mencari jumlah nilai objektif yang paling kecil.

4.10. Menambahkan batasan untuk minimum parameter nutrisi

Pada tahap ini, proses *looping* dilakukan untuk menetapkan batasan parameter minimum untuk setiap nutrisi yang diperlukan dalam model optimasi. Dalam *looping for*, setiap pasangan *key value* dalam *dictionary constraints_min['constraints_min']* akan dijalankan, di mana *nutrient* adalah nama nutrisi dan *min_value* adalah nilai minimum yang diperlukan untuk memenuhi nutrisi tersebut. Jika nilai minimum tersebut lebih dari 0 yang artinya ada batasan minimum yang harus dipenuhi, maka sebuah objek batasan baru dibuat menggunakan *solver.Constraint(min_value, solver.infinity()),* dengan batas bawah yang ditetapkan sebesar *min_value* hingga batas atas yang tak terhingga.

Segmen Program 4.8. Membuat parameter nutrisi minimum

```
for nutrient, min_value in
constraints_min['constraints_min'].items():
    if min_value != 0:
        expr_min = solver.Constraint(min_value,
solver.infinity())
        for food, var in food_vars.items():
            expr_min.SetCoefficient(var,
ingredients['ingredients'][food][nutrient])
```

Kemudian, dilakukan iterasi melalui setiap pasangan *key value* dalam *dictionary food_vars* menggunakan looping for. Di setiap iterasi yang dijalankan, koefisien untuk variabel keputusan *var* yang terkait dengan bahan makanan tersebut ditetapkan dalam batasan yang sesuai menggunakan metode *expr_min.SetCoefficient(var,*

ingredients['ingredients'][food][nutrient]). Hal ini, koefisien variabel keputusan dalam batasan adalah jumlah nutrisi yang terkandung dalam bahan makanan tersebut, yang diambil dari daftar *ingredients['ingredients'][food][nutrient].* Dengan metode ini, setiap batasan minimum nutrisi akan ditetapkan untuk memastikan bahwa nilai dari parameter minimum sudah melebihi dari nilai yang diminta dalam program optimasi pakan ayam.

4.11. Menambahkan batasan untuk maksimum parameter nutrisi

Pada tahap ini, langkah kerjanya sama dengan cara diatas seperti langkah dalam membuat parameter nutrisi minimum. Pertama-tama dilakukan looping for untuk setiap pasangan *key value* dalam *dictionary constraints_max['constraints_max']* di iterasikan, di mana nutrient adalah nama nutrisi dan *max_value* adalah nilai maksimum yang digunakan untuk nutrisi tersebut. Jika nilai maksimum tersebut bukan 0 yang artinya ada batasan maksimum yang harus dipenuhi, maka sebuah objek batasan baru dibuat menggunakan *solver.Constraint(0, max_value),* dengan batas bawah 0 dan batas atas yang ditetapkan sebesar *max_value.* Kemudian, terjadi iterasi melalui setiap pasangan *key value* dalam *dictionary food_vars* menggunakan *looping for.*

Segmen Program 4.9. Membuat parameter nutrisi maksimum

```
for nutrient, max_value in
constraints_max['constraints_max'].items():
    if max_value != 0:
        expr_max = solver.Constraint(0, max_value)
        for food, var in food_vars.items():
            expr_max.SetCoefficient(var,
ingredients['ingredients'][food][nutrient])
```

Di setiap iterasi yang dijalankan, nilai koefisien pada variabel keputusan var yang ada dalam bahan makanan tersebut akan ditetapkan dalam batasan yang sesuai menggunakan metode *expr_max.SetCoefficient(var, ingredients['ingredients'][food][nutrient]).* Hal ini berarti bahwa koefisien variabel keputusan dalam batasan adalah jumlah nutrisi yang terkandung dalam bahan makanan tersebut, yang diambil dari daftar *ingredients['ingredients'][food][nutrient].* Dengan cara ini, setiap batasan maksimum nutrisi akan ditetapkan untuk mengecek bahwa apakah semua nilai parameter nutrisi maksimum tidak melebihi nilai yang diminta dalam program optimasi pakan ayam.

4.12. Menambahkan batasan untuk minimum dan maksimum untuk setiap bahan makanan

Proses *looping* yang pertama digunakan untuk mengambil nilai minimum dan maksimum yang dipakai untuk setiap bahan makanan dalam program optimasi pakan ayam. Jadi di dalam *looping for*, setiap pasangan *key* vialue dalam *dictionary food_vars* akan di iterasikan, di mana *food* adalah nama bahan makanan dan var adalah variabel keputusan. Di setiap iterasi, nilai minimum dan maksimum yang diizinkan untuk bahan makanan tersebut diambil dari daftar *ingredients['ingredients'][food]['Min']* dan *ingredients['ingredients'][food]['Max'].* Kemudian nilai tersebut akan disimpan ke dalam *variabel min_value_food* dan *max_value_food.*

Segemen Program 4.10. Menambahkan nilai minimum dan maksimum di setiap bahan makanan

```
for food, var in food_vars.items():
    min_value_food = ingredients['ingredients'][food]['Min']
    max_value_food = ingredients['ingredients'][food]['Max']
    if min_value_food != 0:
        min_value_food = min_value_food / 100 * 1500
        food_min_expr = solver.Constraint(min_value_food,
solver.infinity())
        food_min_expr.SetCoefficient(var, 1)
    if max_value_food != 0:
        max_value_food = max_value_food / 100 * 1500
        food_max_expr = solver.Constraint(-solver.infinity(),
max_value_food)
        food_max_expr.SetCoefficient(var, 1)
```

Setelah proses *looping* untuk memasukan nilai minimal dan maksimal setiap bahan makanan, perlu ditetapkan batasan minimum untuk jumlah bahan makanan yang harus dipilih dalam program optimasi pakan ayam, dengan memperhitungkan persentase tambahan dari nilai minimum yang sudah dimasukkan. Dalam kondisi *if*, jika nilai minimum untuk bahan makanan tersebut tidak sama dengan 0, maka nilai minimum tersebut akan diubah dengan menghitung persentase dengan jumlah total bahan makanan yang dipakai selama proses formulasi. Setelah nilai minimum yang telah dihitung, *min_value_food,* akan dibuat dengan menggunakan *solver.Constraint(min_value_food, solver.infinity()).* Kemudian, koefisien untuk variabel

keputusan var yang berhubungan dengan nilai bahan makanan tersebut ditetapkan dalam batasan yang sesuai menggunakan metode *food_min_expr.SetCoefficient(var, 1)* yang bertujuan untuk merubah koefisien variabel keputusan dalam batasan menjadi 1, untuk membuat jumlah bahan makanan yang dipilih setidaknya lebih besar atau sama dengan nilai minimum yang telah dimodifikasi.

Selanjutnya proses yang dilakukan untuk nilai maksimum hampir sama dengan diatas yaitu, konsidi *if*, akan dilakukan dengan tujuan untuk memeriksa apakah nilai maksimum untuk bahan makanan tersebut tidak sama dengan 0. Kemudian nilai maksimum tersebut akan diubah dengan menghitung persentase dengan jumlah total bahan makanan yang dipakai selama proses formulasi. Setelah nilai maksimum dihitung, *max_value_food,* akan membuat objek batasan baru dengan menggunakan metode *solver.Constraint(-solver.infinity(), max_value_food),* dengan batas bawah yang tak terhingga negatif dan batas atas yang ditetapkan sebesar nilai maksimum. Kemudian, koefisien untuk variabel keputusan *var* yang terkait dengan bahan makanan tersebut ditetapkan dalam batasan yang sesuai menggunakan metode *food_max_expr.SetCoefficient(var, 1)* yang bertujuan untuk merubah koefisien variabel keputusan dalam batasan menjadi 1, untuk membuat jumlah bahan makanan yang dipilih setidaknya lebih besar atau sama dengan nilai maksimum yang telah dimodifikasi.

4.13. Menambahkan batasan total bahan makanan maksimum 1500 kg

Pada tahap ini, sebuah objek batasan baru dibuat menggunakan *solver.Constraint*(1500, 1500), dengan batas bawah dan batas atas yang sama, yaitu 1500. Ini berarti bahwa total jumlah bahan makanan yang dipilih harus tepat 1500 kg. Kemudian, dilakukan iterasi melalui semua variabel keputusan untuk bahan makanan ke dalam *looping for*. Di setiap iterasi, koefisien untuk variabel keputusan *var* yang terkait dengan bahan makanan tersebut ditetapkan dalam batasan total yang telah dibuat menggunakan metode *total_food_expr.SetCoefficient(var, 1).*

Segmen Program 4.11. Membuat batasan total bahan makan yang digunakan

```
total_food_expr = solver.Constraint(1500, 1500)
for var in food_vars.values():
    total_food_expr.SetCoefficient(var, 1)
```

4.14. Menampilkan hasil bahan makanan yang digunakan

Pada tahap ini, dilakukan pengecekan apakah status hasil optimasi. Jika status hasil optimasi adalah optimal, maka kode akan mencetak message bahwa solusi telah ditemukan menggunakan print('Solusi ditemukan:'). Kemudian, harga total bahan makanan yang ditemukan sebagai hasil optimasi juga di *print* dengan menggunakan print(f'Harga total makanan: *{solver.Objective().Value():.2f}').* Metode .*Value()* digunakan untuk mendapatkan nilai objektif optimasi, yaitu harga total bahan makanan.

Setelah itu, jumlah masing-masing bahan makanan yang dipilih dalam solusi yang optimal akan didapatkan melalui *looping for*, setiap pasangan *key value* dalam *dictionary food_vars* yang di iterasi. Dalam setiap iterasi, akan dilakukan pengecekkan apakah nilai variabel keputusan *var* lebih besar dari 0, yang memiliki arti bahwa bahan makanan tersebut dipilih dalam solusi bahan makanan yang optimal. Jika nilainya lebih besar dari 0, maka program akan mencetak juga jumlah bahan makanan yang dipilih tersebut.

Segmen Program 4.12. Menampilkan hasil bahan makanan yang sudah di formulate

```
if status == pywraplp.Solver.OPTIMAL:
    print('Solusi ditemukan:')
    print(f'Harga total makanan: Rp.
{round(solver.Objective().Value()):,.2f}')
    print('Jumlah masing-masing bahan makanan:')
    for food, var in food_vars.items():
        if var.solution_value() > 0:
           print(f"{ingredients['ingredients'][food]['Name']}:
{var.solution_value():.2f} kg")
           total_ingredients += var.solution_value()
```

Dalam setiap iterasi, *ingredients['ingredients'][food]['Name']* digunakan untuk mengambil nama bahan makanan yang termasuk didalam key value food. Kemudian, nilai variabel keputusan var diakses menggunakan metode *.solution_value(),* yang mengembalikan nilai jumlah bahan makanan yang dipilih dalam solusi pakan ayam yang optimal. Selain hasil bahan makan yang optimal tersebut di *print*, jumlah bahan makanan yang dipilih tersebut ke dalam variabel *total_ingredients* juga ditampilkan disebelah kanan nama dari bahan makanan yang terpilih. Variabel *total_ingredients* akan diinisialisasi sebelum *loop* dimulai, dan

akan digunakan untuk menghitung total jumlah semua bahan makanan yang dipilih dalam solusi optimal.

4.15. Menghitung total nutrisi untuk setiap parameter

Pada tahap ini, dengan menggunakan *looping for*, setiap nutrisi yang diperlukan akan diiterasi. Di setiap iterasi, kode membuat entri baru untuk perhitungan pada *dictionary total_nutrition.* Kemudian, jumlah total kontribusi nutrisi dari semua bahan makanan yang dipilih. Untuk setiap bahan makanan yang dipilih akan dihitung menggunakan metode *.solution_value().* Jumlah bahan makanan tersebut kemudian dikalikan dengan jumlah nutrisi yang terkandung dalam bahan makanan tersebut, yang diambil dari *ingredients['ingredients'][food][nutrient].* Setiap kandungan nutrisi dari bahan makanan yang dipilih kemudian dijumlahkan menggunakan fungsi *sum().*

Segmen Program 4.13. Menghitung total nutrisi

```
total_nutrition = {}
    for nutrient in constraints_min['constraints_min'].keys():
        total_nutrition[nutrient] =
round(sum(var.solution_value() *
ingredients['ingredients'][food][nutrient] for food, var in
food_vars.items() if var.solution_value() > 0), 2)
```

Hasil perhitungan ini kemudian dibulatkan menjadi desimal dengan 2 angka dibelakang koma menggunakan fungsi *round()* dan disimpan sebagai nilai dalam *dictionary total_nutrition.*

4.16. Menampilkan hasil nutrisi setiap bahan

Pada langkah terakhir, dengan menggunakan *looping for*, setiap pasangan *key value* dalam *dictionary* `total_nutrition` diiterasi, nama nutrisi dan total jumlah kandungan nutrisi tersebut akan diprint di bawah teks “Total bahan makanan”

Segmen Program 4.14. Menampilkan total nutrisi dan total bahan makanan

```
print('\nTotal Nutrisi:')
    for nutrient, value in total_nutrition.items():
        print(f'{nutrient}: {value:.2f}')
    print(f'Total bahan makanan: {total_ingredients:.2f}')
else:
    print('Tidak ada solusi yang ditemukan')
```

Setelah selesai mencetak total nutrisi untuk setiap nutrisi yang diperlukan, jumlah total semua bahan makanan yang dipilih dalam solusi optimal juga di *print* dengan teks "Total bahan makanan:" dan dilanjutkan dengan nilai *`total_ingredients`,* yang berarti total jumlah semua bahan makanan yang dipilih. Jika tidak ada solusi yang ditemukan, maka kode akan mencetak *message* "Tidak ada solusi yang ditemukan".

4.17. Inisialisasi koneksi ke database MySQL

Pada tahap ini, menghubungkan aplikasi *Node.js* ke *database MySQL* menggunakan modul *mysql*. Modul *mysql* di *import* dengan *require('mysql').* dan *mysql.createConnection()* dipanggil untuk menerima sebuah objek konfigurasi. Objek konfigurasi ini berisi detail koneksi seperti host *(localhost)*, user (*root'*), *password*, dan *database* (*database* yang digunakan bernama 'contoh_pakan').

Segmen Program 4.15. Import database untuk program optimasi

```
const mysql = require('mysql')
const connection = mysql.createConnection({
    host: 'localhost',
    user: 'root',
    password: '',
    database: 'contoh_pakan'
  })
  connection.connect()
  module.exports = connection
```

Setelah koneksi dibuat, metode *connection.connect()* dipanggil untuk menginisialisasi koneksi ke *database*. *Objek connection* di *export* menggunakan *module.exports* sehingga dapat digunakan di file lain dari aplikasi *Node.js.*

4.18. Menjalankan file Python untuk formulasi pakan ayam

Pada tahap ini, fungsi *JavaScript asinkron executePython* digunakan untuk menjalankan kode *Python* dari *file* lain. Fungsi ini menerima dua parameter yaitu *script*, merupakan *path* yang berisi direktori *file* yang akan dijalankan. Selanjutnya *args*, adalah *array* yang berisi argumen-argumen tambahan yang akan diteruskan ke kode tersebut. Di dalam fungsi, modul *spawn* dari *child_process* digunakan untuk memulai proses *Python* dengan menjalankan perintah *Python* bersama dengan kode dan argumen yang diberikan.

Segmen Program 4.16. Menjalankan file Python untuk program optimasi

```
const executePython = async (script, args = []) => {
    const py = spawn('python', [script, ...args]);
    const result = await new Promise((resolve, reject) => {
        let output;
        py.stdout.on('data', (data) => {
            output = data.toString();
        });
        py.stderr.on('data', (data) => {
            reject(new Error(`Python error: ${data}`));
            console.log(`stderr: ${data}`);
        });
```

```
py.on('close', (code) => {
            console.log(`child process close all stdio with
code ${code}`);
            resolve(output);
        });
    }).catch((error) => {
        console.error(`Error running Python script:
${error}`);
        throw error;
    });
    return result;
}
```

Data yang dihasilkan oleh *Python* (*stdout*) diterima dan diubah menjadi bentuk *string*, dan apabila terjadi *error* (*stderr*) akan di *catch promise*. Setelah selesai, *promise* akan diselesaikan dengan output dari kode *Python*.

4.19. Mengambil semua bahan makanan

Tahap pertama pada kode ini adalah melakukan *query* pada *database* untuk mengambil semua data bahan makanan dengan *field* yang ditentukan. Hasil *query* tersebut akan disimpan pada variabel *ingredients* yang merupakan sebuah array. *HTTP request* ini bertipe *GET* dengan *URL /api/ingredients. Response* akan memiliki status bernilai 200 dan apabila respon mengirimkan hasil *error*, maka data tidak akan ditampilkan karena *message error* akan ditampilkan melalui terminal.

Segmen Program 4.17. Mengambil data makananan dari database

```
app.get('/api/ingredients', async function (req, res, next) {
    db.query('select bm.id, var, name_product, stock,harga,
min, max, me, crude_protein, true_protein, ee, cf, ca,
total_p, avail_p, cap, na, cl, choline, folate, dlys, dmet,
dtsaa, dthr, dtrp, darg, dval, status from bahan_makanan bm
join kategori k on k.id = bm.bahan_kategori order by
NAME_PRODUCT, k.id', function (error, results, fields) {
        if (error) throw error;
        const ingredients = { 'ingredients': {} };
        results.forEach(result => {
            ingredients['ingredients'][result.var] = {
                'ID': result.id,
                'Name': result.name_product,
                'Harga': result.harga,
                'Min': result.min,
                'Max': result.max,
                'Stock': result.stock,
                'ME': result.me,
                'Crude_Protein': result.crude_protein,
                …
                'dVAL': result.dval,
                'Status': result.status
            };
        });
        return res.status(200).send(ingredients);
    });
});
```

Setiap baris hasil *query* yang disimpan kedalam *result* akan di iterasi menggunakan metode *forEach.* Sebuah entri baru akan ditambahkan ke objek *ingredients['ingredients']* dengan *key* sebagai nilai var dari hasil *query*. Nilai untuk setiap *key* adalah objek yang berisi berbagai atribut bahan makanan, termasuk *ID, Name, Harga, Min, Max, Stock,* dan berbagai atribut nutrisi lainnya seperti *ME, Crude_Protein, True_Protein, EE, CF, Ca, Total_P, Avail_P, CaP, Na, Cl, Choline, Folate, dLYS, dMET, dTSAA, dTHR, dTRP, dARG, dVAL*, dan Status.

4.20. Mengambil semua bahan makanan yang berstatus aktif di database

Tahap ini memiliki kode yang hampir sama dengan kode pada Segmen Program 4.19. Setiap baris hasil *query* yang disimpan kedalam result akan di iterasi menggunakan metode *forEach*. Entri baru akan ditambahkan ke objek *ingredients['ingredients']* dengan *key* sebagai nilai var dari hasil *query*. Nilai untuk setiap *key* adalah objek yang berisi berbagai atribut bahan makanan, termasuk *ID*, *Name*, *Harga*, *Min*, *Max*, *Stock*, dan berbagai atribut nutrisi lainnya seperti *ME, Crude_Protein, True_Protein, EE, CF, Ca, Total_P, Avail_P, CaP, Na, Cl, Choline, Folate, dLYS, dMET, dTSAA, dTHR, dTRP, dARG, dVAL,* dan Status. Selanjutnya, *query* pada *database* tetap dilakukan untuk mengambil semua data bahan makanan. Namun pada kode kali ini, data bahan makanan yang diambil melalui *query* hanya data yang berstatus aktif atau bernilai 1.

Segmen Program 4.18. Mengambil data makananan dari database yang berstatus aktif

```
app.get('/api/ingredients/active', async function (req, res,
next) {

    db.query('select var, name_product, harga, min, max, me,
crude_protein, true_protein, ee, cf, ca, total_p, avail_p,
cap, na, cl, choline, folate, dlys, dmet, dtsaa, dthr, dtrp,
darg, dval from bahan_makanan bm join kategori k on k.id =
bm.bahan_kategori where status = 1 order by NAME_PRODUCT,
k.id', function (error, results, fields) {

        if (error) throw error;

        const ingredients = { 'ingredients': {} };

        results.forEach(result => {

            ingredients['ingredients'][result.var] = {

                'Name': result.name_product,

                'Harga': result.harga,

                'Min': result.min,

                'Max': result.max,

                'Stock': result.stock,

                'ME': result.me,

                'Crude_Protein': result.crude_protein,

                …

                'dVAL': result.dval,

            };

        });

        return res.status(200).send(ingredients);

    });

});
```

Hasil *query* tersebut akan disimpan pada variabel *ingredients* yang merupakan sebuah *array*. *HTTP request* ini bertipe *GET* dengan *URL/api/ingredients. Response* akan memiliki status bernilai 200 dan apabila *response* mengirimkan hasil *error*, maka data tidak akan ditampilkan karena *message error* akan ditampilkan melalui terminal.

4.21. Mengganti status dari setiap bahan makanan berdasarkan id yang dipilih

Pada tahap kali ini, sebuah *endpoint API* dalam aplikasi *Express.js* yang merespons permintaan *PUT* ke *path /api/ingredients/status/:id* digunakan untuk memperbarui status dari

bahan makanan berdasarkan ID yang dipilih. Ketika *endpoint* diakses, fungsi *asinkron* dijalankan yang menggunakan *query SQL* untuk memperbarui data di *database*. *Query SQL* ini mengubah nilai status dalam tabel bahan_makanan sesuai dengan nilai yang dikirimkan dari *req.body.status*. ID dari bahan makanan yang akan di *update*, diambil dari parameter *req.params.id.*

Segmen Program 4.19. Mengganti status bahan makanan berdasarkan id

```
app.put('/api/ingredients/status/:id', async function (req,
res, next) {

    db.query('update bahan_makanan set status = ? where id =
?', [req.body.status, req.params.id], function (error,
results, fields) {

        if (error) throw error;

        return res.status(200).send({ message: 'Updated' });

    });

});
```

Query ini dijalankan dengan menggunakan *callback function* yang menerima tiga parameter: *error*, *results*, dan *fields*. Jika terjadi kesalahan selama eksekusi *query*, maka kesalahan tersebut akan dilempar, menyebabkan aplikasi berhenti. Jika *query* berhasil, respon akan memiliki status bernilai 200 dan *message JSON* yang berisi *{ message: 'Updated' }.*

4.22. Mengganti stok dari setiap bahan makanan berdasarkan id yang dipilih

Pada tahap ini, sebuah *endpoint API* dalam aplikasi *Express.js* akan merespons permintaan *PUT* ke *path /api/ingredients/stock/:id. Endpoint* ini digunakan untuk memperbarui jumlah stok dari bahan makanan berdasarkan ID yang dipilih. Ketika *endpoint* diakses, fungsi *asinkron* dijalankan yang menggunakan *query SQL* untuk mengupdate data pada *database*. *Query SQL* ini mengubah nilai kolom stok dalam tabel bahan_makanan sesuai dengan nilai yang dikirimkan melalui req.body.stock. ID dari bahan makanan yang akan di update oleh *req.params.id*.

Segmen Program 4.20. Mengganti stok bahan makanan berdasarkan id

```
app.put('/api/ingredients/stock/:id', async function (req,
res, next) {
    db.query('update bahan_makanan set stock = ? where id =
?', [req.body.stock, req.params.id], function (error, results,
fields) {
        if (error) throw error;
        return res.status(200).send({ message: 'Updated' });
    });
});
```

Query ini dijalankan dengan menggunakan *callback function* yang menerima tiga parameter: *error*, *results*, dan *fields*. Jika terjadi kesalahan selama eksekusi *query,* maka kesalahan tersebut akan dilempar, menyebabkan aplikasi berhenti. Jika *query* berhasil, respon akan memiliki status bernilai 200 dan *message JSON* yang berisi *{ message: 'Updated' }.*

4.23. Mengganti minimal dari setiap bahan makanan berdasarkan id yang dipilih

Pada tahap ini, sebuah *endpoint API* dalam aplikasi *Express.js* akan merespons permintaan *PUT* ke *path /api/ingredients/min/:id. Endpoint* ini digunakan untuk memperbarui nilai minimum (min) dari bahan makanan berdasarkan ID yang dipilih. Ketika *endpoint* diakses, fungsi *asinkron* dijalankan yang menggunakan *query SQL* untuk *mengupdate* data pada *database*. *Query SQL* ini mengubah nilai kolom min dalam tabel bahan_makanan sesuai dengan nilai yang dikirimkan melalui *req.body.stock*. ID dari bahan makanan yang akan di *update* oleh *req.params.id.*

Segmen Program 4.21. Mengganti minimal bahan makanan berdasarkan id

```
app.put('/api/ingredients/min/:id', async function (req, res,
next) {
    db.query('update bahan_makanan set min = ? where id = ?',
[req.body.min, req.params.id], function (error, results,
fields) {
        if (error) throw error;
        return res.status(200).send({ message: 'Updated' });
    });
});
```

Query ini dijalankan dengan menggunakan *callback function* yang menerima tiga parameter: *error*, *results*, dan *fields*. Jika terjadi kesalahan selama eksekusi *query*, maka kesalahan tersebut akan dilempar, menyebabkan aplikasi berhenti. Jika berhasil, respon akan memiliki status bernilai 200 dan *message JSON* yang berisi { *message: 'Updated' }.*

4.24. Mengganti maksimal dari setiap bahan makanan berdasarkan id yang dipilih

Pada tahap ini, sebuah *endpoint API* dalam aplikasi *Express.js* akan merespons permintaan *PUT* ke *path /api/ingredients/max/:id*. *Endpoint* ini digunakan untuk memperbarui nilai minimum (min) dari bahan makanan berdasarkan ID yang dipilih. Ketika *endpoint* diakses, fungsi *asinkron* dijalankan yang menggunakan *query SQL* untuk mengupdate data pada *database*. *Query SQL* ini mengubah nilai kolom max dalam tabel bahan_makanan sesuai dengan nilai yang dikirimkan melalui *req.body.stock.* ID dari bahan makanan yang akan di *update* oleh *req.params.id.*

Segmen Program 4.22. Mengganti maksimal bahan makanan berdasarkan id

```
app.put('/api/ingredients/max/:id', async function (req, res,
next) {

    db.query('update bahan_makanan set max = ? where id = ?',
[req.body.max, req.params.id], function (error, results,
fields) {

        if (error) throw error;

        return res.status(200).send({ message: 'Updated' });

    });

});
```

Query ini dijalankan dengan menggunakan *callback function* yang menerima tiga parameter: *error*, *results,* dan *fields*. Jika terjadi kesalahan selama eksekusi *query*, maka kesalahan tersebut akan dilempar, menyebabkan aplikasi berhenti. Jika *query* berhasil, respon akan memiliki status bernilai 200 dan *message JSON* yang berisi *{ message: 'Updated' }.*

4.25. Mengambil data parameter minimal berdasarkan id yang dipilih

Pada tahap ini, sebuah *endpoint API* dalam aplikasi *Express.js* akan merespons permintaan *GET* ke *path /api/ingredients/min/:id. Endpoint* ini digunakan untuk memperbarui nilai minimum (min) dari bahan makanan berdasarkan ID yang dipilih. Ketika *endpoint* diakses, fungsi *asinkron* dijalankan yang menggunakan *query SQL* untuk mengambil data dari tabel *layer_phase_min*.

Query ini memilih semua kolom dari tabel *layer_phase_min* di mana nilai kolom ID sesuai dengan ID yang dipilih dalam *req.params.id.*

Segmen Program 4.23. Mengambil data parameter minimal berdasarkan id

```
app.get('/api/parameters/min/:id', async function (req, res,
next) {
    db.query('select * from layer_phase_min where ID = ?',
[req.params.id], function (error, results, fields) {
        if (error) throw error;
        return res.status(200).send({
            constraints_min: {
                'ME': results[0].ME,
                'Crude_Protein': results[0].Crude_Protein,
                …
                'dVAL': results[0].dVAL
            }
        });
    });
});
```

Query ini dijalankan dengan menggunakan *callback function* yang menerima tiga parameter: *error, results,* dan *fields.* Jika terjadi sebuah *error* selama *query* dijalankan, error tersebut akan *throw* dan program akan berhenti. Jika *query* berhasil, hasil *query (results[0])* akan digunakan untuk membentuk *objek JSON* yang berisi nilai parameter minimum seperti *ME, Crude_Protein, EE, CF, Ca, Total_P, Avail_P, Na, Cl, Choline, dLYS, dMET, dTSAA, dTHR, dTRP, dARG,* dan *dVAL.*

4.26. Mengambil semua data parameter minimal yang ada

Pada tahap ini, sebuah *endpoint API* dalam aplikasi *Express.js* akan merespons permintaan *GET* ke *path /api/ingredients/min/:id*. *Endpoint* ini digunakan untuk memperbarui nilai minimum (min) dari bahan makanan berdasarkan ID yang dipilih. Ketika *endpoint* diakses, fungsi *asinkron* dijalankan yang menggunakan *query SQL* untuk mengambil data dari tabel

layer_phase_min. Query ini akan memilih kolom id, *Phase, ME, Crude_Protein, True_Protein, EE, CF, Ca, Total_P, Avail_P, CaP, Na, Cl, Choline, Folate, dLYS, dMET, dTSAA, dTHR, dTRP, dARG,* dan *dVAL* dari tabel tersebut.

Segmen Program 4.24. Mengambil semua data parameter minimal

```
app.get('/api/parameters/min', async function (req, res, next)
{
    db.query('select id, Phase, ME, Crude_Protein,
True_Protein, EE, CF, Ca, Total_P, Avail_P, CaP, Na, Cl,
Choline, Folate, dLYS, dMET, dTSAA, dTHR, dTRP, dARG, dVAL
from layer_phase_min', function (error, results, fields) {
        if (error) throw error;
        if (results.length > 0) {
            return res.status(200).send({ constraints_min:
results });
        } else {
            return res.status(404).send({ message: 'No results
found' });
        }
    });
});
```

Query ini dijalankan dengan menggunakan *callback function* yang menerima tiga parameter: *error*, *results*, dan *fields*. Jika terjadi sebuah *error* selama *query* dijalankan, *error* tersebut akan *throw* dan program akan berhenti. Jika *query* berhasil, akan mengambil semua data parameter minimal. Kemudian, data hasil *query* akan dikirim sebagai respon dengan status kode 200.

4.27. Mengambil data parameter maksimal berdasarkan id yang dipilih

Pada tahap ini, sebuah *endpoint API* dalam aplikasi *Express.js* akan merespons permintaan *GET* ke *path /api/ingredients/max/:id. Endpoint* ini digunakan untuk memperbarui nilai maksimum (max) dari bahan makanan berdasarkan ID yang dipilih. Ketika *endpoint* diakses, fungsi *asinkron* dijalankan yang menggunakan *query SQL* untuk mengambil data dari tabel *layer_phase_max*. *Query* ini memilih semua kolom dari tabel *layer_phase_max* di mana nilai kolom ID sesuai dengan ID yang dipilih dalam *req.params.id.*

Segmen Program 4.25. Mengambil data parameter maksimal berdasarkan id

```
app.get('/api/parameters/max/:id', async function (req, res,
next) {
    db.query('select * from layer_phase_max where ID = ?',
[req.params.id], function (error, results, fields) {
        if (error) throw error;
        if (results.length > 0) {
            return res.status(200).send({
                constraints_max: {
                    'ME': results[0].ME,
                    'Crude_Protein': results[0].Crude_Protein,
                    …
                    'dVAL': results[0].dVAL
                }
            });
        } else {
            return res.status(404).send({ message: 'No results
found' });
        }
    });
});
```

Query ini dijalankan dengan menggunakan *callback function* yang menerima tiga parameter: *error, results,* dan *fields*. Jika terjadi sebuah *error* selama *query* dijalankan, *error* tersebut akan *throw* dan program akan berhenti. Jika *query* berhasil, hasil *query (results[0])* akan digunakan untuk membentuk objek *JSON* yang berisi nilai parameter maksimum seperti *ME, Crude_Protein, EE, CF, Ca, Total_P, Avail_P, Na, Cl, Choline, dLYS, dMET, dTSAA, dTHR, dTRP, dARG,* dan *dVAL.*

4.28. Mengambil semua data parameter maksimal yang ada

Pada tahap ini, sebuah *endpoint API* dalam aplikasi *Express.js* akan merespons permintaan *GET* ke *path /api/ingredients/max/:id. Endpoint* ini digunakan untuk memperbarui nilai maksimum(max) dari bahan makanan berdasarkan ID yang dipilih. Ketika *endpoint* diakses,

fungsi *asinkron* dijalankan yang menggunakan *query SQL* untuk mengambil data dari tabel *layer_phase_max*. *Query* ini akan memilih kolom id, *Phase, ME, Crude_Protein, True_Protein, EE, CF, Ca, Total_P, Avail_P, CaP, Na, Cl, Choline, Folate, dLYS, dMET, dTSAA, dTHR, dTRP, dARG,* dan *dVAL* dari tabel tersebut.

Segmen Program 4.26. Mengambil semua data parameter maksimal

```
app.get('/api/parameters/max', async function (req, res, next)
{
    db.query('select id, Phase, ME, Crude_Protein,
True_Protein, EE, CF, Ca, Total_P, Avail_P, CaP, Na, Cl,
Choline, Folate, dLYS, dMET, dTSAA, dTHR, dTRP, dARG, dVAL
from layer_phase_max', function (error, results, fields) {
        if (error) throw error;
        if (results.length > 0) {
            return res.status(200).send({ constraints_max:
results });
        } else {
            return res.status(404).send({ message: 'No results
found' });
        }
    });
});
```

Query ini dijalankan dengan menggunakan *callback function* yang menerima tiga parameter: *error, results,* dan *fields*. Jika terjadi sebuah *error* selama *query* dijalankan, *error* tersebut akan *throw* dan program akan berhenti. Jika *query* berhasil, akan mengambil semua data parameter maksimal. Kemudian, data hasil *query* akan dikirim sebagai respon dengan status kode 200.

4.29. Menghapus parameter makanan berdasarkan id yang dipilih

Pada tahap ini, sebuah *endpoint API* dalam aplikasi *Express.js* akan merespons permintaan *DELETE* ke *path /api/parameters/:id. Endpoint* ini digunakan untuk menghapus parameter dengan ID yang dipilih dari dua tabel dalam *database*: *layer_phase_max* dan *layer_phase_min.* Ketika endpoint ini diakses, dua *query SQL* dijalankan secara berurutan untuk menghapus data dari tabel *layer_phase_max* dan *layer_phase_min.* Kedua *query* ini menggunakan ID yang dipilih dari parameter *req.params.id.*

Segmen Program 4.27. Menghapus parameter makanan berdasarkan id

```
app.delete('/api/parameters/:id', function (req, res, next) {


    db.query('delete from layer_phase_max where ID = ?',
[req.params.id], function (error, results, fields) {
        if (error) throw(error);
    });
    db.query('delete from layer_phase_min where ID = ?',
[req.params.id], function (error, results, fields) {
        if (error) throw(error);
    });
    return res.status(200).send({ message: 'Deleted' });
});
```

Query pertama akan menghapus baris dari tabel *layer_phase_max* di mana kolom ID sama dengan ID yang dipilih. *Query* akan dijalankan dengan menggunakan *callback function* yang menerima tiga parameter: *error, results,* dan *fields*. ika terjadi sebuah *error* selama *query* dijalankan, *error* tersebut akan *throw* dan program akan berhenti. *Query* kedua menghapus baris dari tabel *layer_phase_min* di mana kolom ID juga cocok dengan ID yang diberikan. Sama seperti *query* yang pertama, *callback function* digunakan untuk menangani kemungkinan terjadinya *error*. Setelah kedua *query* dijalankan, response akan dikirim dengan status kode 200 dan sebuah *message JSON* yang berisi *{ message: 'Deleted' },* yang menandakan bahwa data telah dihapus.

4.30. Menambah minimal parameter nutrisi yang baru

Pada tahap ini, sebuah *endpoint API* dalam aplikasi *Express.js* akan merespons permintaan *POST* ke *path /api/ingredients/min/:id. Endpoint* ini digunakan untuk memperbarui nilai minimal(min) dari bahan makanan berdasarkan ID yang dipilih. Ketika *endpoint* ini diakses, fungsi *asinkron* dijalankan yang menggunakan *query SQL* untuk menambahkan data baru ke tabel *layer_phase_min. Query* ini menggunakan pernyataan *INSERT INTO* untuk memasukkan nilai ke dalam kolom-kolom yang sesuai. Nilai untuk setiap kolom diambil dari *req.body.*

Segmen Program 4.28. Menambah minimal parameter nutrisi yang baru

```
app.post('/api/parameters/min', async function (req, res,
next) {

    let sql = `INSERT INTO layer_phase_min (Phase, DM, ME,
Crude_Protein, True_Protein, EE, CF, Ca, Total_P, Avail_P,
CaP, Na, Cl, Choline, Folate, dLYS, dMET, dTSAA, dTHR, dTRP,
dARG, dVAL) VALUES
(?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?)`;

    db.query(sql, [req.body.Phase, req.body.DM, req.body.ME,
req.body.Crude_Protein, req.body.True_Protein, req.body.EE,
req.body.CF, req.body.Ca, req.body.Total_P, req.body.Avail_P,
req.body.CaP, req.body.Na, req.body.Cl, req.body.Choline,
req.body.Folate, req.body.dLYS, req.body.dMET, req.body.dTSAA,
req.body.dTHR, req.body.dTRP, req.body.dARG, req.body.dVAL],
function (err, result) {

        if (err) throw err;

    });

});
```

Query ini dijalankan dengan menggunakan *callback function* yang menerima tiga parameter: *error, results,* dan *fields.* Jika terjadi sebuah *error* selama *query* dijalankan, *error* tersebut akan *throw* dan program akan berhenti. *Endpoint* ini memungkinkan user untuk menambahkan data baru ke tabel *layer_phase_min* dalam *database* dengan mengirimkan permintaan *POST* yang mencakup nilai untuk setiap kolom yang diperlukan.

4.31. Menambah maksimal parameter nutrisi yang baru

Pada tahap ini, sebuah *endpoint API* dalam aplikasi *Express.js* akan merespons permintaan *POST* ke *path /api/ingredients/max/:id.* Endpoint ini digunakan untuk memperbarui nilai maksimum (max) dari bahan makanan berdasarkan ID yang dipilih. Ketika *endpoint* ini diakses, fungsi asinkron dijalankan yang menggunakan *query SQL* untuk menambahkan data baru ke tabel *layer_phase_max. Query* ini menggunakan pernyataan *INSERT INTO* untuk memasukkan nilai ke dalam kolom-kolom yang sesuai. Nilai untuk setiap kolom diambil dari *req.body.*

Segmen Program 4.29. Menambah maksimal parameter nutrisi yang baru

```
app.post('/api/parameters/max', async function (req, res,
next) {

    let sql = `INSERT INTO layer_phase_max (Phase, DM, ME,
Crude_Protein, True_Protein, EE, CF, Ca, Total_P, Avail_P,
CaP, Na, Cl, Choline, Folate, dLYS, dMET, dTSAA, dTHR, dTRP,
dARG, dVAL) VALUES
(?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?)`;

    db.query(sql, [req.body.Phase, req.body.DM, req.body.ME,
req.body.Crude_Protein, req.body.True_Protein, req.body.EE,
req.body.CF, req.body.Ca, req.body.Total_P, req.body.Avail_P,
req.body.CaP, req.body.Na, req.body.Cl, req.body.Choline,
req.body.Folate, req.body.dLYS, req.body.dMET, req.body.dTSAA,
req.body.dTHR, req.body.dTRP, req.body.dARG, req.body.dVAL],
function (err, result) {

        if (err) throw err;

    });

});
```

Query ini dijalankan dengan menggunakan *callback function* yang menerima tiga parameter: *error*, *results*, dan *fields*. Jika terjadi sebuah *error* selama *query* dijalankan, *error* tersebut akan *throw* dan program akan berhenti. *Endpoint* ini memungkinkan *user* untuk menambahkan data baru ke tabel *layer_phase_max* dalam *database* dengan mengirimkan permintaan *POST* yang mencakup nilai untuk setiap kolom yang diperlukan.

4.32. Formulate

Pada tahap ini, sebuah *endpoint API* dalam aplikasi *Express.js* akan merespons permintaan *GET* ke *path /api/formulate/:id. Endpoint* ini bertujuan untuk memformulasikan bahan makanan berdasarkan ID yang dipilih atau bahan yang berstatus aktif, dengan menggunakan *Python* yang berada di dalam file yang bernama *formulate.py.* Ketika *endpoint* ini diakses, fungsi *asinkron* dijalankan untuk mengeksekusi kode *Python formulate.py* dengan menggunakan fungsi *executePython*, yang mengambil ID sebagai argumen. Hasil dari eksekusi kode *Python* disimpan dalam variablel *result*.

Segmen Program 4.30. Mendapatkan hasil formulasi dari file Python

```
app.get('/api/formulate/:id', async function (req, res, next)
{
    try {
        const id = req.params.id;
        const result = await
executePython('./Python/formulate.py', [id]);
        let startStr = "Jumlah masing-masing bahan makanan:";
        let endStr = "Total Nutrisi:";
        let startIndex = result.indexOf(startStr);
        let endIndex = result.indexOf(endStr);
        if (startIndex !== -1 && endIndex !== -1) {
            let desiredData = result.substring(startIndex +
startStr.length, endIndex).trim();
```

```
const lines = desiredData.split('\r\n');
            const ingredients = [];
            for (const line of lines) {
                const parts = line.split(':');
                if (parts.length === 2) {
                    const name = parts[0].trim();
                    const value = parseFloat(parts[1]);
                    if (!isNaN(value)) {
                        ingredients.push({
                            name: name,
                            value: value
                        });
                    }
                }
            }
```

Kemudian, kode akan melakukan pemrosesan pada hasil *Python* untuk mengekstrak data yang diperlukan. Setelah itu, kode akan mencari *startIndex* dan *endIndex* dari bagian hasil yang berisi informasi tentang jumlah bahan makanan dan total nutrisi. Kemudian, kode mengekstrak data di antara kedua indeks tersebut dan memisahkan baris-barisnya. Setiap baris kemudian

dipisahkan berdasarkan tanda : (titik dua). Jika sebuah baris terdiri dari dua bagian setelah dipisahkan, yang menandakan bahwa data bahan makanan valid (ada solusi), maka nama bahan makanan dan nilainya akan diproses. Nilai-nilai valid diambil dan dimasukkan ke dalam sebuah *array* objek yang berisi nama dan nilai masing-masing bahan makanan.

Setelah mendapatkan data bahan makanan dari hasil formulasi, Kemudian, program akan melakukan iterasi melalui setiap bahan makanan dalam *array ingredients*. Untuk setiap bahan makanan, program akan membangun dan menjalankan *query SQL* untuk mengurangi stok bahan makanan yang sesuai di *database*. *Query* ini menggunakan pernyataan *UPDATE* untuk memperbarui nilai kolom Stok dari tabel bahan_makanan, dengan mengurangkan nilai *ingredient.value* dari stok saat ini.

Segmen Program 4.31. Update stok setelah formulasi

```
for (const ingredient of ingredients) {
                let sql = `UPDATE bahan_makanan SET Stock =
Stock - ? WHERE NAME_PRODUCT = ?`;
                db.query(sql, [ingredient.value,
ingredient.name], function (err, result) {
                    if (err) throw err;
                });
            }
        } else {
            console.log("Strings not found.");
        }
        return res.status(200).send(
            result);
    } catch (error) {
        return res.status(500).send(error);
    }
});
```

Setelah iterasi selesai, program akan memberikan respon dengan status 200, yang berisi hasil formulasi yang diperoleh dari sebelumnya. Jika terjadi kesalahan selama proses formulasi atau pembaruan stok, respon dengan status 500 akan dikirimkan, yang berisi *error message*.

Endpoint ini memungkinkan *user* untuk memperoleh hasil formulasi dan mengurangi stok bahan makanan yang dibutuhkan secara otomatis dalam *database*.

4.33. Menambahkan hasil formulasi ke tabel histori

Pada tahap ini, sebuah *endpoint API* dalam aplikasi *Express.js* akan merespons permintaan *POST ke path /api/history. Endpoint* bertujuan untuk menyimpan *history* formulasi yang dilakukan oleh *user*. Kemudian, program melakukan *checking* apakah data yang diperlukan sudah tersedia dalam *request body* atau belum. Jika tidak ada nilai yang diberikan untuk parameter *Phase* (parameter minimum dan maksimum), maka respons dengan status 400 akan dikirimkan, yang berisi message bahwa *Phase* diperlukan untuk melakukan penyimpanan *history* formulasi.

Segmen Program 4.32. Menambahkan hasil formulasi ke tabel histori

```
app.post('/api/history', async function (req, res, next) {
    if (!req.body.Phase) {
        return res.status(400).send({ message: 'Phase is
required' });
    }
    let sql = `INSERT INTO history_formulate (tanggal, params,
details) VALUES (?, ?, ?)`;
    const date = new Date().toISOString().slice(0,
19).replace('T', ' ');
    db.query(sql, [date, req.body.Phase, req.body.details],
function (err, result) {
        if (err) {
            console.error(err);
            return res.status(500).send({ message: 'Internal
server error' });
        }
        return res.status(200).send({ message: 'Success' });
    });
});
```

Setelah melakukan *checking*, *Query SQL* akan dijalankan untuk menyimpan data riwayat formulasi ke dalam tabel *history_formulate*. Query ini menggunakan pernyataan *INSERT INTO* untuk memasukkan nilai ke dalam kolom-kolom yang sesuai, yaitu tanggal, params, dan *details*.

Nilai untuk tanggal diambil dari waktu saat ini menggunakan objek *Date*, kemudian diformat menjadi *string* yang sesuai dengan format *datetime MySQL*. Nilai untuk params dan details diambil dari *request body*.

Setelah *query* dijalankan, *callback function* digunakan untuk menangani kemungkinan *error* selama *query* dijalankan. Jika terjadi *error*, *error message* akan dicetak dalam *console* dan respon dengan status 500 dikirimkan, yang menandakan adanya kesalahan pada server internal. Jika *query* berhasil dijalankan tanpa *error*, respon dengan status 200 dikirimkan, yang berisi *message* bahwa penyimpanan data *history* formulasi berhasil dijalankan.

4.34. Mengambil semua data history

Pada tahap ini, sebuah *endpoint API* dalam aplikasi *Express.js* akan merespons permintaan *GET* ke *path /api/history*. *Endpoint* ini bertujuan untuk mengambil *history* formulasi yang telah disimpan sebelumnya. Ketika *endpoint* dijalankan, fungsi *callback* dijalankan untuk menjalankan *query SQL* yang mengambil data dari tabel *history_formulate*. *Query* ini memilih kolom id, tanggal, params, dan *details* dari tabel tersebut.

Segmen Program 4.33. Mengambil semua data history

```
app.get('/api/history', async function (req, res, next) {
    db.query('select hf.id, hf.tanggal, hf.params, hf.details
from history_formulate hf', function (error, results, fields)
{
        if (error) throw error;
        return res.status(200).send({ history: results });
    });
});
```

Query ini dijalankan dengan menggunakan *callback function* yang menerima tiga parameter: *error*, *results*, dan *fields*. Jika terjadi sebuah *error* selama *query* dijalankan, *error* tersebut akan *throw* dan program akan berhenti. Jika *query* berhasil, hasilnya akan dikirim sebagai respon dengan status kode 200. Data hasil *query* akan ditampung dalam sebuah objek dengan atribut *history*, yang berisi *array* dari objek-objek yang memiliki nilai setiap entri dalam *history* formulasi. *Endpoint* memungkinkan *user* untuk mengambil dan menampilkan *history* formulasi yang telah tersimpan sebelumnya dari *database* dalam format *JSON*.

4.35. Mengambil data history berdasarkan id yang dipilih

Pada tahap ini, sebuah *endpoint API* dalam aplikasi Express.js akan merespons permintaan *GET ke path /api/history. Endpoint* ini bertujuan untuk mengambil *history* formulasi yang telah disimpan sebelumnya. *Endpoint* digunakan untuk mengambil data *history* formulasi yang spesifik berdasarkan ID yang dipilih. Ketika endpoint diakses, fungsi *callback* dijalankan untuk menjalankan *query SQL* yang mengambil data dari tabel *history_formulate* dengan menggunakan *WHERE* untuk memfilter hasil berdasarkan ID yang dipilih dalam parameter *URL req.params.id.*

Segmen Program 4.34. Mengambil data history berdasarkan id

```
app.get('/api/history/:id', async function (req, res, next) {
    db.query('select hf.id, hf.tanggal, hf.params, hf.details
from history_formulate hf where hf.id = ?', [req.params.id],
function (error, results, fields) {
        if (error) throw error;
        return res.status(200).send({ history: results });
    });
});
```

Query ini dijalankan dengan menggunakan *callback function* yang menerima tiga parameter: *error*, *results*, dan *fields*. Jika terjadi sebuah *error* selama *query* dijalankan, *error* tersebut akan *throw* dan program akan berhenti. Jika *query* berhasil, hasilnya akan dikirim sebagai respon dengan status kode 200. Data hasil *query* akan ditampung dalam sebuah objek dengan atribut *history*, yang berisi *array* dari objek-objek yang memiliki nilai setiap entri dalam *history* formulasi sesuai dengan ID yang dipilih.

4.36. Implemantasi Program

Selain segmen program untuk *back-end* untuk program optimasi pakan ayam, terdapat juga segmen program untuk menjelaskan *front-end* program ini, yaitu:

4.36.1. Footer

Segmen Program 4.35. Footer

```
export const Footer = () => {
  return (
    <footer className="mt-auto flex flex-row flex-wrap
justify-center w-[100%] mx-auto gap-y-6 border-t bg-blue-gray-
600 border-blue-gray-50 py-6 text-center md:justify-between">
      <Typography color="blue-gray" className="text-center mx-
auto font-semibold text-lg text-white">
        &copy; Steven Alexandro - C14200044
      </Typography>
    </footer>
  )
}
```

4.36.2. Navbar

Segmen Program 4.36. Navbar

```
function NavList() {
    return (
        <ul className="my-2 flex flex-col gap-2 lg:mb-0 lg:mt-
0 lg:flex-row lg:items-center lg:gap-6">
            <CustomLink to="/">Home</CustomLink>
            <CustomLink to="/formulate">Formulate</CustomLink>
            <CustomLink
to="/ingredients">Ingredients</CustomLink>
            <CustomLink
to="/nutritions">Nutritions</CustomLink>
            <CustomLink to="/history">History</CustomLink>
        </ul>
    );
}
```

4.36.3. Ingredients

Segmen Program 4.37. Ingredients

```
export function Ingredients() {
  const [data, setData] = useState([]);
  const [currentPage, setCurrentPage] = useState(1);
  useEffect(() => {
    api.get('api/ingredients')
      .then(response => {

setData(Object.entries(response.data.ingredients).map(([key,
value]) => ({ ...value, id: key })));
      })
      .catch(error => {
        console.error('There was an error!', error);
      });
  }, []);
```

```
const TABLE_HEAD = ["Name", "Harga", "ME", "Crude Protein",
"True Protein", "EE", "CF", "Ca", "Total P", "Avail P", "CaP",
"Na", "Cl", "Choline", "Folate", "dLYS", "dMET", "dTSAA",
"dTHR", "dTRP", "dARG", "dVAL"];
const ITEMS_PER_PAGE = 12;
const startIndex = (currentPage - 1) * ITEMS_PER_PAGE;
const currentItems = data.slice(startIndex, startIndex +
ITEMS_PER_PAGE);
const totalPages = Math.ceil(data.length / ITEMS_PER_PAGE);
```

4.36.4. Nutritions

Segmen Program 4.38. Nutritions

```
const [showAlert, setShowAlert] = useState(false);
  const addData = () => {
   …
    for (let input of inputs) {
      if (input === '') {
        setShowAlert(true);
        return;
      }
    }
```

4.36.5. Phase

Segmen Program 4.39. Phase

```
const TABLE_HEAD = ["Phase", "ME", "Crude Protein", "True
Protein", "EE", "CF", "Ca", "Total P", "Avail P", "CaP", "Na",
"Cl", "Choline", "Folate", "dLYS", "dMET", "dTSAA", "dTHR",
"dTRP", "dARG", "dVAL", "Action"];
  useEffect(() => {
    if (selectedOption === '1') {
      api.get('api/parameters/min')
        .then(response => {
          if (Array.isArray(response.data.constraints_min)) {
            setData(response.data.constraints_min.map((item)
=> ({ ...item })));
            console.log(response.data.constraints_min);
          } else {
            console.warn('Expected an array but received',
response.data.constraints_min);
          }
        })
```

4.36.6. Formulate

Segmen Program 4.40. Handle option value change di halaman formulate

```
const handleSelectChange = (value) => {
    setSelectedOption(value);
    phasesData.forEach(data => {
      if (data.id === value) {
        // console.log(data.Phase);
      }
    });
  };
```

```
const handleOpen = () => {
    setOpen(!open);
    getPhasesMax();
    getPhasesMin();
    dataTable();
    fetchFormulateData();
  }
```

Segmen Program 4.41. Menambahkan history pada saat formulasi

```
const addHistory = async () => {
    try {
      const promises = phasesData.map(async (data) => {
        if (data.id === selectedOption && data.Phase) {
          let lineString = "Tidak ada solusi yang ditemukan!";
          if (formulateData) {
            const lines = formulateData.split('\r\n');
            const start = lines.indexOf('Jumlah masing-masing
bahan makanan:') + 1;
            const end = lines.indexOf('Total Nutrisi:');
            const relevantLines = lines.slice(start, end);
            const lineObjects = relevantLines
              .filter(line => line.includes(':'))
              .map(item => {
                const [ingredient, amount] = item.split(' :
');
                return { ingredient };
              });
            if (lineObjects.length > 0) {
              lineString = JSON.stringify(lineObjects);
            }
          }
          const response = await api.post('api/history', {
Phase: data.Phase, details: lineString });
          return response;
        } else {
          return null;
        }
      });
```

Segmen Program 4.42. Formulate

```
const getPhasesMin = () => {
    if (selectedOption) {
      api.get(`api/parameters/min/${selectedOption}`, {
params: { id: selectedOption } })
        .then(response => {
          setPhasesMinData(response.data.constraints_min);
        })
        .catch(error => {
          console.error('There was an error!', error);
        });
    }
  }
```

```
const getPhasesMax = () => {
    if (selectedOption) {
      api.get(`api/parameters/max/${selectedOption}`, {
params: { id: selectedOption } })
        .then(response => {
          setPhasesMaxData(response.data.constraints_max);
        })
        .catch(error => {
          console.error('There was an error!', error);
        });
    }
  }
```

```
const fetchFormulateData = () => {
    if (selectedOption) {
      api.get(`api/formulate/${selectedOption}`)
        .then(response => {
          setFormulateData(response.data);
        })
        .catch(error => {
          console.error('There was an error!', error);
        });
    }
  };
```

4.36.7. History

Segmen Program 4.42. Histroy

```
export const History = () => {
  const [data, setData] = useState([]);
  const [details, setDetails] = useState([]);
  const [open, setOpen] = React.useState(false);
  const [selectedId, setSelectedId] = useState(null);
  const handleOpen = () => {
    setOpen(!open);
  }
  useEffect(() => {
    api.get('api/history')
      .then(response => {
        setData(response.data.history);
      })
      .catch(error => {
        console.error('There was an error!', error);
      });
```

```
if (selectedId !== null) {
      console.log(selectedId);
      api.get(`api/history/${selectedId}`)
        .then(response => {
          const details =
JSON.parse(response.data.history[0].details);
          setDetails(details);
          console.log(details);
        })
        .catch(error => {
          console.error('There was an error!', error);
        });
    }
  }, [selectedId]);
  return (
```