

**PERANCANGAN ARSITEKTUR APLIKASI BUDIDAYA
PERIKANAN MODERN PADA BACKEND YANG
BERTANGGUNG JAWAB DALAM MELAYANI TRANSAKSI
QUERY WEBSERVICE DENGAN MENGGUNAKAN
TEKNOLOGI FLASK MICROSERVICE**

Skripsi

**Disusun untuk memenuhi salah satu syarat
memperoleh gelar Sarjana Komputer**



*Mencerdaskan dan
Memartabatkan Bangsa*

**Andri Rahmanto
1313618007**

**PROGRAM STUDI ILMU KOMPUTER
FAKULTAS MATEMATIKA DAN ILMU PENGETAHUAN ALAM
UNIVERSITAS NEGERI JAKARTA**

2023

LEMBAR PENGESAHAN

Dengan ini saya mahasiswa Fakultas Matematika dan Ilmu Pengetahuan
Alam, Universitas Negeri Jakarta

Nama : Andri Rahmanto

No. Registrasi : 1313618007

Program Studi : Ilmu Komputer

Judul : Perancangan Aritekturn Aplikasi Budidaya Perikanan
Modern Pada Backend yang Bertanggung Jawab
Dalam Melayani Transaksi Query Webservice Dengan
Menggunakan Teknologi Flask Micorservice

Menyatakan bahwa proposal skripsi ini telah siap diajukan untuk seminar pra skripsi.

Menyetujui,

Dosen Pembimbing I



Muhammad Eka Survana, M.Kom.

NIP. 19851223 201212 1 002

Dosen Pembimbing II



Med Irzal, M.Kom.

NIP. 19770615 200312 1 001

Mengetahui,

Koordinator Program Studi Ilmu Komputer



Ir. Fariani Hermin Indiyah, M.T

NIP. 19600211 198703 2 001

LEMBAR PERNYATAAN

Saya menyatakan dengan sungguh-sungguh bahwa skripsi dengan judul **“Perancangan Aritektur Aplikasi Budidaya Perikanan Modern Pada Backend yang Bertanggung Jawab Dalam Melayani Transaksi Query Webservice Dengan Menggunakan Teknologi Flask Micorservice”** yang telah saya susun sebagai syarat untuk memperoleh gelar Sarjana Komputer dari Program Studi Ilmu Komputer Universitas Negeri Jakarta adalah karya ilmiah saya dengan arahan dari dosen pembimbing.

Seluruh bahan dan data yang didapatkan dari penulis terdahulu yang sudah terpublikasikan yang tercantum dalam teks skripsi ini, telah tercantum di dalam Daftar Pustaka sesuai dengan etika, norma, dan kaidah penulisan ilmiah.

Apabila dikemudian hari diketemukan sebagian isi skripsi ini bukan hasil karya saya sendiri dalam beberapa bagian tertentu, saya bersedia mendapatkan sanksi pencabutan gelar akademik yang saya sanding dan sanksi-sanksi lainnya sesuai dengan peraturan perundang-undangan yang diberlakukan.

Jakarta, 20 Juli 2023

Andri Rahmanto

HALAMAN PERSEMBAHAN

Untuk Keluargaku dan Diriku Sendiri.

ABSTRAK

ANDRI RAHMANTO. Perancangan Aritektur Aplikasi Budidaya Perikanan Modern Pada *Backend* yang Bertanggung Jawab Dalam Melayani Transaksi *Query Webservice* Dengan Menggunakan Teknologi *Flask Micorservice*. Skripsi. Fakultas Matematika dan Ilmu Pengetahuan Alam, Universitas Negeri Jakarta. 2023. Di bawah bimbingan Muhammad Eka Suryana, M.Kom dan Med Irzal, M.Kom.

Budidaya ikan air tawar merupakan salah satu sumber perikanan negara Indonesia. Pencatatan indikator seperti jumlah ikan, suhu air, ph air, pemberikan pakan wajib dilakukan dalam pembudidayaan ikan air tawar, baik untuk menentukan tindakan terhadap keadaan kolam maupun menentukan harga jual. Pada umumnya pencatatan indikator - indikator pada budidaya ikan dilakukan secara konvensional yang rawan akan kesalahan dalam perhitungan. Penelitian ini bertujuan untuk membuat arsitektur *backend web service API* untuk aplikasi budidaya perikanan modern. Jenis Penelitian ini adalah Pengembangan/*Research and Development*. Data diambil dari hasil diskusi bersama pembudidaya ikan air tawar JFT (*J Farm Technology*) dan studi literatur dengan membaca jurnal-jurnal yang terkait dengan topik penelitian. Diskusi menghasilkan suatu *user requirement* yang menjadi acuan fitur yang akan diterapkan pada arsitektur *backend web service API*. Proses pengembangan sistem ini menggunakan metode *Scrum* dan seluruh program yang dibuat menggunakan bahasa pemrograman *Python*. Hasil akhir dari arsitektur ini mencakup struktur *project*, rancangan diagram, *web service* berupa *REST API*, serta dokumentasi lengkap *API*.

Kata kunci: *web service, REST API, budidaya ikan, perikanan modern, backend, scrum*

ABSTRACT

ANDRI RAHMANTO. Design of Modern Fish Farming Application Architecture on Backend Responsible for Serving Transaction Query Webservice Using Flask Microservice Technology. Thesis. Faculty of Mathematics and Natural Sciences, Jakarta State University. 2023. Under the guidance of Muhammad Eka Suryana, M.Cs and Med Irzal, M.Cs.

Freshwater fish farming is one of the sources of fisheries in Indonesia. Recording indicators such as the number of fish, water temperature, water pH, and feed provision are mandatory in freshwater fish farming, both to determine actions against pond conditions and to determine selling prices. In general, the recording of indicators in fish farming is carried out conventionally, which is prone to errors in calculations. This study aims to create a backend web service API architecture for modern fish farming applications. This research is of the Development/Research and Development type. Data was collected from discussions with freshwater fish farmers from JFT (J Farm Technology) and literature studies by reading journals related to the research topic. The discussion resulted in user requirements that become the reference for the features to be implemented in the backend web service API architecture. The development process of this system used the Scrum method, and all programs were developed using the Python programming language. The final result of this architecture includes project structure, diagram designs, web service in the form of REST API, and comprehensive API documentation.

Keywords: web service, REST API, fish farming, modern fisheries, backend, scrum

KATA PENGANTAR

Puji syukur penulis panjatkan ke hadirat Allah SWT, karena dengan rahmat dan karunia-Nya, penulis dapat menyelesaikan skripsi yang berjudul **“Perancangan Aritektur Aplikasi Budidaya Perikanan Modern Pada Backend yang Bertanggung Jawab Dalam Melayani Transaksi Query Webservice Dengan Menggunakan Teknologi Flask Micorservice”**.

Keberhasilan dalam menyusun skripsi ini tidak lepas dari bantuan berbagai pihak yang mana dengan tulus dan ikhlas memberikan masukan guna sempurnanya skripsi ini. Oleh karena itu dalam kesempatan ini, dengan kerendahan hati penulis mengucapkan banyak terima kasih kepada:

1. Yth. Para petinggi di lingkungan FMIPA Universitas Negeri Jakarta,
2. Yth. Ibu Dr. Ria Arafiyah, M.Si. selaku Koordinator Program Studi Ilmu Komputer,
3. Bapak Muhammad Eka Suryana M.Kom selaku Dosen Pembimbing I yang telah memberikan arahan dan bimbingan dalam penulisan proposal skripsi ini,
4. Bapak Med Irzal M.Kom selaku Dosen Pembimbing II yang telah memberikan arahan dan bimbingan dalam penulisan proposal skripsi ini,
5. Kedua orang tua penulis dan keluarga yang selama ini telah senantiasa sabar mendorong, memberikan semangat, dan mendoakan penulis,
6. Teman-teman Program Studi Ilmu Komputer 2018 yang telah mendukung dan membantu skripsi ini,
7. Teman-teman dalam grup "SPS victory?", "El Matador", "Acts of Service" dan "BJ Pasca Pandemic" yang dalam beberapa kesempatan hadir dalam kehidupan

penulis sebagai penyemangat dan suport penulis,

8. Sahabat-sahabat penulis Ilham Ramanda, Zaidan Pratama, Hendi nur Ibrahim, Ninda Mardiana Usman, Kurnia Hotmaida Sianturi, Divya Regina Adha, Shifa Chairunisa, Olivia Desvina Putri, Fara Saila, Agatha Marcella Setiawan, Fathimah Az Zahra, dan lainnya yang tidak bisa penulis sebutkan satu persatu, yang merupakan keluarga kedua penulis, dan sangat berarti dalam perjalanan penulis dalam melakukan penelitian.

Penulis menyadari bahwa penyusunan skripsi ini masih jauh dari sempurna karena keterbatasan ilmu dan pengalaman yang dimiliki. Oleh karenanya, kritik dan saran yang bersifat membangun akan penulis terima dengan senang hati. Akhir kata, penulis berharap tugas akhir ini bermanfaat bagi semua pihak khususnya penulis sendiri. Semoga Allah SWT senantiasa membala kebaikan semua pihak yang telah membantu penulis dalam menyelesaikan skripsi ini.

Jakarta, 20 Juli 2023

Andri Rahmanto

DAFTAR ISI

LEMBAR PENGESAHAN	ii
LEMBAR PERNYATAAN	iii
LEMBAR PERSEMBAHAN	iv
ABSTRAK	v
ABSTRACT	vi
KATA PENGANTAR	vi
DAFTAR ISI	viii
DAFTAR GAMBAR	x
DAFTAR TABEL	xii
I PENDAHULUAN	1
A. Latar Belakang Masalah	1
B. Rumusan Masalah	4
C. Pembatasan Masalah	5
D. Tujuan Penelitian	5
E. Manfaat Penelitian	5
II KAJIAN PUSTAKA	7
A. <i>Scrum</i>	7
1. Tim Scrum	8
2. Aktivitas-aktivitas Scrum (Scrum Events)	11

3.	Komponen <i>Scrum</i> (<i>Scrum Artifacts</i>)	15
B.	REST	17
1.	URL Design	18
2.	HTTP Verbs	18
3.	HTTP Response Code	19
4.	Format Response	19
C.	MongoDB	19
D.	Flask Framework	23
1.	Flask routing	23
2.	Dependencies	25
E.	Apache	27
1.	Konfigurasi httpd.conf	28
F.	Unit Testing	29
G.	User Acceptance Test (UAT)	30
III Metodologi Penelitian		31
A.	Pengumpulan Data	32
B.	Analisa Sistem	32
C.	Analisa Kebutuhan	34
D.	Perancangan Sistem	34
1.	Produk Backlog	36
2.	Sprint Backlog	38
3.	Daily Scrum	39
4.	Sprint Review dan Sprint Retrospective	39
E.	Pengujian	40
IV Hasil Dan Pembahasan		44

A.	Perancangan Sistem	44
1.	Sprint 1 Report	44
2.	Sprint 2 Report	55
3.	Sprint 3 Report	56
4.	Sprint 4 Report	69
5.	Sprint 5 Report	92
6.	Sprint 6 Report	101
7.	Sprint 7 Report	108
8.	Sprint 8 Report	134
9.	Sprint 9 Report	152
10.	Sprint 10 Report	199
B.	Uji Sistem	222
1.	Unit Testing	222
2.	Pengujian Web Service	227
3.	Pengujian UAT (User Acceptance Test)	228
V	KESIMPULAN DAN SARAN	230
A.	Kesimpulan	230
B.	Saran	230
LAMPIRAN		232
A	Transkrip Percakapan	232
DAFTAR PUSTAKA		235

DAFTAR GAMBAR

Gambar 2.1	Pemodelan RDBMS	20
Gambar 2.2	Pemodelan NoSQL	21
Gambar 3.1	Tahapan Peneltian	31
Gambar 3.2	<i>Flowchart</i> request	32
Gambar 3.3	<i>Usecase</i> diagram	33
Gambar 3.4	Flowchart Scrum	35
Gambar 4.1	Github Projects Sprint-1	45
Gambar 4.2	ERD Database Sprint-1	46
Gambar 4.3	Class Diagram Sprint-1	48
Gambar 4.4	<i>Request Get</i> pemberian pakan <i>Postman</i>	53
Gambar 4.5	View pakan bulanan	55
Gambar 4.6	View pakan harian	56
Gambar 4.7	ERD Database Sprint-3	57
Gambar 4.8	View detail kolam	68
Gambar 4.9	ERD Database Sprint-4	70
Gambar 4.10	View status kolam	91
Gambar 4.11	View list musim budidaya per kolam	91
Gambar 4.12	View detail musim budidaya	92
Gambar 4.13	ERD Database Sprint-5	94
Gambar 4.14	View list kematian perbulan	108
Gambar 4.15	ERD Database Sprint-7	110
Gambar 4.16	ERD Database Sprint-8	135
Gambar 4.17	ERD Database Sprint-9	154
Gambar 4.18	View list pencatatan kolam harian	198

Gambar 4.19 View list pencatatan kolam mingguan	199
Gambar 4.20 ERD Database Sprint-10	201
Gambar 4.21 View list treatment kolam perbulan	222
Gambar 4.22 Tampilan Aqua Breeding berbasis android	228

DAFTAR TABEL

Tabel 2.1	<i>Komparasi Database RDBMS dan NoSQL</i>	22
Tabel 3.1	<i>Product Backlog</i>	37
Tabel 3.2	<i>Sprint-1 backlog</i>	39
Tabel 3.3	Skenario unit testing.	40
Tabel 3.4	Tabel <i>User Acceptance Test</i>	43
Tabel 4.1	Unit Testing Sprint 1	54
Tabel 4.2	<i>Sprint-2 backlog</i>	55
Tabel 4.3	<i>Sprint-3 backlog</i>	57
Tabel 4.4	<i>Sprint-4 backlog</i>	69
Tabel 4.5	Form entry musim budidaya kolam.	74
Tabel 4.6	Form entry panen musim budidaya kolam.	78
Tabel 4.7	<i>Sprint-5 backlog</i>	93
Tabel 4.8	Form entry kematian ikan.	98
Tabel 4.9	Form edit kematian ikan.	100
Tabel 4.10	<i>Sprint-6 backlog</i>	102
Tabel 4.11	<i>Sprint-7 backlog</i>	109
Tabel 4.12	Form entry kematian ikan.	115
Tabel 4.13	<i>Sprint-8 backlog</i>	134
Tabel 4.14	Form entry kematian ikan.	139
Tabel 4.15	Sprint-09 backlog.	152
Tabel 4.16	Form entry pencatatan kolam harian.	158
Tabel 4.17	Form entry pencatatan kolam mingguan.	175
Tabel 4.18	<i>Sprint-10 backlog</i>	200
Tabel 4.19	Form entry treatment kolam.	206

Tabel 4.20	Unit testing fitur pemberian pakan.	223
Tabel 4.21	Unit testing fitur registrasi kolam.	223
Tabel 4.22	Unit testing fitur musim budidaya kolam.	224
Tabel 4.23	Unit testing fitur kematian ikan.	224
Tabel 4.24	Unit testing fitur perpindahan ikan antar kolam.	225
Tabel 4.25	Unit testing fitur grading berat ikan.	225
Tabel 4.26	Unit testing fitur pencatatan kualitas air harian.	226
Tabel 4.27	Unit testing fitur pencatatan kualitas air mingguan.	226
Tabel 4.28	Unit testing fitur pencatatan treatment.	227
Tabel 4.29	Daftar pengujian UAT.	229

BAB I

PENDAHULUAN

A. Latar Belakang Masalah

Indonesia memiliki beberapa sumber perikanan salah satunya adalah budidaya ikan air tawar. Budidaya ikan air tawar memerlukan beberapa modal yaitu lahan, prasarana kolam, Pakan, dan Pengetahuan atau skill dalam berbudidaya ikan. Umumnya budidaya ikan air tawar memelihara banyak jenis ikan atau yang disebut pemeliharaan campuran hal ini disebabkan karena terdapatnya berbagai macam makanan untuk berbagai jenis ikan air tawar, walaupun lebih baik memperhatikan mana jenis ikan yang akan dijadikan peliharaan pokok dan mana yang akan dijadikan peliharaan sampingan (infishta, 2019).

Dalam pembudidayaan ikan air tawar diperlukannya pencatatan beberapa indikator yang berguna untuk menilai kelayakan habitat, memantau perkembangan ikan, dan memantau pemberian pakan sehingga bisa memutuskan treatment apa yang akan dilakukan kedepannya untuk mencapai panen yang baik. Beberapa indikator yang perlu di perhatikan adalah kadar oksigen dalam air, pengukuran kadar ph dalam air, DO (Dissolved Oxygen), Suhu, dan Ammonia. Indikator air tersebut berpengaruh terhadap kelayakan habitat ikan air tawar (Pramleonita dkk., 2018). Di sisi lain pencatatan terhadap kolam, perlakuan kolam, penaburan ikan dan pemberian pakan juga merupakan faktor yang penting untuk dicatat. Perlunya pencatatan terhadap banyak indikator secara berulang dan intensif menjadi faktor diperlukannya sistem modern yang membantu mempermudah pekerjaan tersebut.

Di Indonesia terdapat beberapa penelitian sistem modern terkait perikanan yang berkontribusi pada budidaya perikanan oleh (Supriyati, 2018) melakukan

penelitian yang menghasilkan sistem akuntansi budidaya perikanan berbasis android. Penelitian ini dilakukan untuk memecahkan suatu masalah pada pencatatan secara tradisional yang dilakukan oleh petani perikanan yang menyebabkan kurang maksimalnya income yang didapat. Sistem yang dibuat akan digunakan oleh beberapa stakeholders diantaranya adalah: Ketua dari perikanan tersebut, Sekertaris, Bendahara, Seksi sarana produksi, Seksi pengadaan, dan Seksi pemasaran. Sistem ini menerapkan beberapa fitur yaitu Transaksi jual-beli, kas keluar-masuk, dan pembukuan. (Dhita Widhiastika, 2021) melakukan penelitian yang berjudul "Perancangan Aplikasi Jual Beli Produk Perikanan Berbasis Mobile Android". Aplikasi tersebut memungkinkan pengguna dalam melakukan transaksi jual beli perikanan. Aplikasi juga menyediakan wadah untuk diskusi dan informasi gizi dari suatu produk perikanan. Dari dua penelitian sebelumnya didapatkan sistem yang dibuat membantu petani perikanan dalam kegiatan pembukuan transaksi, namun tidak dengan kegiatan aktivitas budidaya.

Penelitian pada sektor IoT pemantauan kualitas air secara real-time dilakukan oleh (Yudhis Thiro Kabul Yunior, 2019) yang mana menghasilkan IoT monitoring kualitas air terintegrasi oleh database server. Indikator yang dicatat adalah Ph, Dissolved Oxygen (DO), Suhu, dan Turbidity. IoT dibuat menggunakan microcontroller Arduino, yang mana perannya adalah mengirim data dari sensor dengan protokol GSM ke database server. Pada server juga terdapat halaman yang menampilkan monitoring data berdasarkan database. Penelitian (Ahmad Rifa'i, 2021) menghasilkan IoT untuk Pemantauan dan Kontrol otomatis kualitas air berbasis IoT menggunakan Node-Red untuk budidaya udang. Dalam penerapannya indikator yang diukur sama dengan apa yang diukur dalam penelitian (Yudhis Thiro Kabul Yunior, 2019) namun dengan tambahan pengukuran ketinggian air. Nantinya setiap sensor terhubung dengan microcontroller arduino, namun pada penelitian

(Ahmad Rifa'i, 2021) memilih menggunakan NodeMCU (modul wifi) untuk terhubung ke internet. Server yang dibangun menggunakan Node-Red yang merupakan tools yang mempermudah membangun sistem berbasis IoT. Saat pengiriman data ke server arduino menggunakan protokol MQTT yang menerapkan konsep publish dan subscribe. Pada alat kontrol aktuator terdiri dari NodeMCU dan relay dan berguna sebagai pengontrol pompa air dan aerator. Dari dua penelitian diatas terdapat beberapa masalah terkait device IoT terutama pada sensor Ph dan DO. Keakuratan sensor Ph dan DO didasarkan pada grade sensor, semakin tinggi grade sensor semakin mahal sensor tersebut. Sensor Ph dan Do juga mengalami penurunan kualitas seiring waktu berjalan, maka dari itu diperlukannya replacement berkala selambat-lambatnya 1 bulan sekali.

Penelitian IoT pemberian pakan otomatis dilakukan oleh (Aep Setiawan, 2022) yang menghasilkan suatu alat pemberian pakan otomatis. Alat pemberian pakan dapat bekerja secara otomatis berdasarkan waktu yang telah di jadwalkan. IoT yang diterapkan menggunakan NodeMCU sebagai modul yang bisa mengakses jaringan wifi dan blynk sebagai middleware untuk memonitoring pemberian pakan, persediaan pakan,pengaturan jadwal dan takaran pakan. Dalam kasus IoT pemberian pakan otomatis terdapat beberapa kendala yaitu resiko tidak meratanya pemberian pakan karena terbatasnya jangkauan pelemparan pakan. Selanjutnya adalah pelemparan pakan otomatis pada praktiknya ditetapkan oleh jadwal sedangkan dosis pakan bisa berubah tergantung dengan keadaan di lapangan.

Dari beberapa penelitian diatas, penelitian ini bertujuan untuk memberikan solusi dari setiap permasalahan yang ada pada perikanan modern. Fokus penelitian ini adalah membangun arsitektur aplikasi budidaya perikanan modern pada backend yang dapat melayani transaksi query dari berbagai platform. Penelitian ini selanjutnya akan terus dikembangkan dari berbagai sisi antara lain adalah frontend,

AI, dll. Berikut beberapa penelitian yang telah dilakukan untuk berkontribusi pada perikanan modern ini.

Penelitian yang terkait dalam penelitian ini adalah penelitian bidang Monitoring data sensing pada budidaya ikan air tawar sudah dilakukan oleh Fadhilah Perwira Hadi dalam penelitian yang berjudul "Rancang Bangun Web Service dan Website sebagai Storage Engine dan Monitoring Data Sensing". Penelitian tersebut menghasilkan suatu sistem web service yang dapat menerima dan memonitoring data yang dikirimkan oleh embedded device, dengan menerapkan konsep IoT (Hadi, 2021). Pada dasarnya web service yang dirancang oleh (Hadi, 2021) dikhkususkan untuk tersambung kepada embedded device IoT. Embedded device sebagai perangkat sensor terhadap beberapa data-data indikator unsur dalam kolam serta mengirim data tersebut ke web service. Penelitian lainnya yang terkait adalah penelitian yang dilakukan oleh (Nugraha, 2022) yang berjudul "Ekstraksi Latar Depan pada Citra Ikan dengan Metode GrabCut yang Diautomasi Menggunakan Saliency Map" yang bertujuan untuk membangun sistem yang memisahkan antara latar depan dan latar belakang pada citra ikan. Penelitian selanjutnya adalah penelitian dengan judul "Fish Movement Tracking dengan Menggunakan Metode GMM dan Kalman Filter" yang dilakukan oleh (Alim, 2022) yang bertujuan untuk membangun sebuah sistem yang dapat melakukan pelacakan pergerakan ikan yang diharapkan nantinya dapat dikembangkan kembali untuk sistem penghitungan ikan. Ke 3 penelitian tersebut merupakan kontribusi yang kedepannya akan diterapkan bersamaan dalam penelitian ini kedalam sistem induk.

B. Rumusan Masalah

Berdasarkan latar belakang penelitian ini, maka perumusan masalah pada penelitian ini adalah "Bagaimana merancang Arsitektur Aplikasi Budidaya

Perikanan Modern pada Backend yang Bertanggung Jawab dalam Melayani Transaksi Query Webservice dengan Menggunakan Teknologi Flask Microservice”.

C. Pembatasan Masalah

Adapun beberapa pembatasan masalah yang bertujuan agar penelitian ini lebih terarah dan sesuai dengan tujuan penelitian:

1. Webservice dikembangkan khusus untuk 1 mitra, dalam hal ini UD JFarm Teknologi. Hanya untuk 1 user.
2. Pengembangan web service menggunakan Framework Flask.

D. Tujuan Penelitian

Penelitian yang dilakukan bertujuan untuk merancang arsitektur aplikasi budidaya perikanan modern pada backend yang bertanggung jawab dalam melayani transaksi query webservice.

E. Manfaat Penelitian

1. Bagi sektor perikanan

Hasil perancangan arsitektur backend server ini dapat memberikan kontribusi terhadap sistem perikanan modern dalam bentuk web service yang dapat terhubung dengan multi platform.

2. Bagi penulis

Menambah pengetahuan dibidang pengembangan web service khususnya pengembangan Arsitektur BackEnd REST API, mengasah kemampuan *programming*, dan memperoleh gelar sarjana dibidang Ilmu Komputer. Selain

itu, penulisan ini juga merupakan media bagi penulis untuk mengaplikasikan ilmu yang didapat di kampus ke kehidupan masyarakat.

3. Bagi Universitas Negeri Jakarta

Menjadi pertimbangan dan evaluasi akademik khususnya Program Studi Ilmu Komputer dalam penyusunan skripsi sehingga dapat meningkatkan kualitas akademik di program studi Ilmu Komputer Universitas Negeri Jakarta serta meningkatkan kualitas lulusannya.

BAB II

KAJIAN PUSTAKA

A. *Scrum*

Scrum merupakan kerangka kerja ringan yang dapat menghasilkan solusi yang adaptif untuk masalah yang kompleks bagi orang, tim, dan organisasi. Kerangka kerja scrum dibiarkan tidak lengkap, hanya mendefinisikan bagian-bagian yang diperlukan untuk mengimplementasi teori *scrum*, selebihnya kecerdasan kolektif orang-orang yang menggunakan *scrum* akan membangun *scrum* itu sendiri. *Scrum* tidak memberikan instruksi yang terperinci kepada setiap anggota tim, melainkan memandu hubungan dan interaksi mereka.

Scrum terus berkembang dengan adanya pengalaman dan pemikiran, lalu menghasilkan suatu keputusan berdasarkan pengamatan. Menggunakan proses iterasi dan perlahan menambahkan sesuatu, untuk mengendalikan risiko dan membuat sesuatu dapat diprediksi.

Scrum menggunakan prinsip pendekatan *Agile* untuk dapat mengatasi segala macam masalah secara kreatif dan adaptif. Berbagai proses, teknik dan metode dapat digunakan dalam kerangka kerja.

Scrum menggabungkan empat aktivitas formal untuk memeriksa dan mengadaptasi menjadi satu aktivitas konten, *Sprint*. Kegiatan ini dapat berhasil jika menerapkan pilar empiris transparansi, inspeksi, dan adaptasi *Scrum*.

1. Transparansi

Proses dan pekerjaan yang muncul harus dapat dilihat oleh mereka yang melakukan pekerjaan dan mereka yang menerimanya. Untuk *Scrum*, sangat penting bahwa keputusan didasarkan pada kondisi tiga komponen formal, dan

komponen *Scrum* dengan transparansi rendah mengurangi nilai dan meningkatkan risiko.

2. Inspeksi

Artefak *scrum* dan kemajuan menuju tujuan yang disepakati harus dirombak secara berkala untuk mendeteksi kemungkinan perbedaan atau masalah yang tidak terduga. Untuk membantu inspeksi, *Scrum* menyediakan ritme dalam bentuk lima acaranya.

Centang untuk memungkinkan adaptasi. Inspeksi tanpa adaptasi dianggap tidak berguna. Aktivitas *scrum* dirancang untuk menginspirasi perubahan.

3. Adaptasi

Jika ada aspek proses yang menyimpang dari batas yang dapat diterima, atau jika produk yang dihasilkan tidak dapat diterima, proses yang diterapkan atau bahan yang dihasilkan harus disesuaikan. Penyesuaian harus dilakukan sesegera mungkin untuk meminimalkan penyimpangan lebih lanjut.

Adaptasi menjadi lebih sulit ketika orang-orang yang terlibat tidak diberdayakan atau dikelola sendiri. Tim *scrum* diharapkan untuk menyesuaikan diri saat mereka mempelajari hal-hal baru melalui inspeksi.

1. Tim Scrum

Unit dasar *Scrum* adalah tim kecil. *Scrum Team* terdiri dari *Scrum Master*, *Product Owner*, dan *Developer*. Dalam tim *Scrum*, tidak ada sub-tim atau hierarki. Tim *scrum* bersifat lintas fungsi, yang berarti bahwa anggota memiliki semua keterampilan yang mereka butuhkan untuk menciptakan nilai di setiap *Sprint*. Tim scrum cukup kecil untuk tetap lincah dan cukup besar untuk menyelesaikan pekerjaan penting dalam sprint, biasanya 10 orang atau kurang.

Tim *Scrum* bertanggung jawab atas semua aktivitas terkait produk, termasuk kolaborasi pemangku kepentingan, validasi, pemeliharaan, operasi, eksperimen, penelitian dan pengembangan, dan aktivitas lain apa pun yang mungkin diperlukan.

Seluruh Tim *Scrum* bertanggung jawab untuk menciptakan peningkatan yang berharga dan berguna di setiap *Sprint*. *Scrum* mendefinisikan tiga tanggung jawab khusus dalam Tim *Scrum*: Pengembang, Pemilik Produk, dan *Scrum Master*.

1. Pengembang (*Developers*)

Pengembang adalah seseorang di tim *Scrum* yang bekerja untuk membuat aspek apa pun dari *Increment* yang dapat digunakan oleh setiap Sprint.

Keterampilan khusus yang dibutuhkan oleh pengembang sering kali luas dan bervariasi menurut bidang pekerjaan. Namun, pengembang selalu bertanggung jawab untuk:

- a. Mengembangkan rencana untuk *Sprint*, *Sprint Backlog*;
- b. Menanamkan kualitas dengan tetap berpegang pada definisi selesai;
- c. Menyesuaikan rencana harian mereka untuk mencapai *Sprint Goals*; dan,
- d. Sebagai profesional, kami memiliki tanggung jawab satu sama lain.

2. Pemilik Produk (*Product Owner*)

Pemilik Produk bertanggung jawab untuk memaksimalkan nilai produk yang dikembangkan oleh Tim *Scrum*. Bagaimana hal ini dicapai akan bervariasi antara perusahaan, tim, dan individu. *Product Owner* juga bertanggung jawab untuk mengelola *Product Backlog* dengan baik, termasuk:

- a. Mengembangkan dan mengkomunikasikan tujuan produk dengan jelas;
- b. Membuat dan mengkomunikasikan item backlog produk dengan jelas;

- c. Memesan backlog produk; dan,
- d. Pastikan *Product Backlog* transparan, terlihat, dan mudah dipahami.

Pemilik Produk dapat melakukan pekerjaan yang dijelaskan di atas atau mendelegasikan tanggung jawab kepada orang lain. Seluruh organisasi harus menghormati keputusan mereka. Keputusan ini tercermin dalam konten dan urutan *Product Backlog*, dan melalui peningkatan yang dapat diperiksa di *Sprint Review*.

Pemilik Produk adalah orang, bukan panitia. Pemilik Produk dapat mewakili kebutuhan banyak pemangku kepentingan dalam *Product Backlog*. Siapapun anggota tim yang ingin mengubah *Product Backlog* dapat melakukannya dengan mencoba meyakinkan Pemilik Produk.

3. *Scrum Master*

Scrum Master bertanggung jawab untuk membangun *Scrum* sebagaimana didefinisikan dalam Panduan *Scrum*. Mereka melakukan ini dengan membantu semua orang di tim dan organisasi *Scrum* memahami teori dan praktik *Scrum*.

Scrum Master bertanggung jawab atas efektivitas Tim *Scrum*. Mereka melakukan ini dengan meminta tim *Scrum* meningkatkan praktik mereka dalam kerangka kerja *Scrum*.

Scrum Masters adalah pemimpin sejati yang melayani Tim *Scrum* dan organisasi yang lebih besar.

Scrum Master melayani Tim Scrum dengan cara, seperti:

- a. Melatih anggota tim dalam manajemen diri dan manajemen berbagai bidang;

- b. Bantu tim *Scrum* fokus pada penciptaan peningkatan bernilai tinggi yang memenuhi definisi selesai;
- c. Buka blokir kemajuan Tim *Scrum*; dan,
- d. Pastikan semua acara *Scrum* terjadi, positif, produktif, dan sesuai jadwal.

Scrum Master melayani Pemilik Produk dengan cara, seperti:

- a. Membantu menemukan teknik untuk definisi tujuan produk yang efektif dan manajemen simpanan produk;
- b. Membantu tim *Scrum* memahami kebutuhan akan item *Product Backlog* yang jelas dan ringkas;
- c. Membantu membangun program produk empiris untuk lingkungan yang kompleks; dan,
- d. Memfasilitasi kolaborasi pemangku kepentingan sesuai kebutuhan atau kebutuhan.

Scrum Master melayani organisasi dengan cara, seperti:

- a. Memimpin, melatih dan melatih organisasi dalam implementasi *Scrum*;
- b. Merencanakan dan merekomendasikan penerapan *Scrum* dalam organisasi;
- c. Membantu karyawan dan pemangku kepentingan memahami dan menerapkan metode empiris untuk pekerjaan yang kompleks; dan,
- d. Menghilangkan hambatan antara *stakeholder* dan tim *Scrum*.

2. Aktivitas-aktivitas Scrum (Scrum Events)

Sprint adalah wadah untuk semua acara lainnya. Setiap acara di *Scrum* adalah kesempatan formal untuk meninjau dan menyesuaikan artefak *Scrum*.

Acara-acara ini secara khusus dirancang untuk mencapai transparansi yang diperlukan. Kegagalan untuk mengoperasikan acara apapun seperti yang ditentukan menghasilkan kesempatan yang hilang untuk peninjauan dan adaptasi. Acara digunakan di *Scrum* untuk menciptakan ketertiban dan meminimalkan kebutuhan akan rapat yang tidak ditentukan dalam *Scrum*. Idealnya, semua acara diadakan pada waktu dan tempat yang sama untuk mengurangi kerumitan.

1. *Sprint*

Sprint adalah jantung dari *Scrum*, di mana ide-ide diubah menjadi nilai. Mereka adalah acara berdurasi tetap satu bulan atau kurang untuk menciptakan konsistensi. *Sprint* baru dimulai segera setelah *Sprint* sebelumnya berakhir.

Semua pekerjaan yang diperlukan untuk mencapai Tujuan Produk, termasuk Perencanaan *Sprint*, *Scrum* Harian, Ulasan *Sprint*, dan Retrospektif *Sprint*, terjadi selama:

- a. tidak membuat perubahan apa pun yang akan membahayakan *Sprint Goal*;
- b. Kualitas tidak berkurang;
- c. *Product Backlog* sedetail yang dibutuhkan; dan,
- d. Semakin banyak yang dipelajari, ruang lingkup dapat diklarifikasi dan dinegosiasikan ulang dengan Pemilik Produk.

Sprint memungkinkan prediktabilitas dengan memastikan bahwa kemajuan terhadap sasaran produk diperiksa dan diakomodasi setidaknya setiap bulan kalender. Ketika Cakrawala Sprint terlalu panjang, Tujuan Sprint mungkin gagal, kompleksitas dapat meningkat, dan risiko dapat meningkat. *Sprint* yang lebih pendek dapat digunakan untuk menghasilkan lebih banyak siklus

pembelajaran dan membatasi risiko biaya dan upaya untuk kerangka waktu yang lebih pendek. Setiap *Sprint* dapat dianggap sebagai proyek pendek.

Berbagai praktik ada untuk memprediksi kemajuan, seperti kelelahan, kelelahan, atau aliran kumulatif. Meskipun terbukti bermanfaat, ini tidak menggantikan pentingnya empirisme. Dalam lingkungan yang kompleks, apa yang terjadi tidak diketahui. Hanya apa yang telah terjadi yang dapat digunakan untuk keputusan berwawasan ke depan. *Sprint* dapat dibatalkan jika *Sprint Goals* sudah kedaluwarsa. Hanya Pemilik Produk yang berhak membatalkan *Sprint*.

2. Perencanaan *Sprint*

Perencanaan *Sprint* memulai *Sprint* dengan meletakkan pekerjaan yang harus dilakukan untuk *Sprint*. Rencana akhir dibuat secara kolaboratif oleh seluruh tim *Scrum*.

Product Owner memastikan bahwa peserta siap untuk mendiskusikan item *Product Backlog* yang paling penting dan bagaimana mereka memetakan ke tujuan produk. Tim *Scrum* juga dapat mengundang orang lain untuk berpartisipasi dalam Perencanaan *Sprint* untuk memberikan saran.

3. *Scrum* harian (*Daily Scrum*)

Tujuan dari *Daily Scrum* adalah untuk memeriksa kemajuan terhadap *Sprint Goals* dan menyesuaikan *Sprint Backlog* sesuai kebutuhan untuk menyesuaikan rencana kerja di masa mendatang.

Daily Scrum adalah aktivitas 15 menit untuk pengembang di tim *Scrum*. Untuk mengurangi kompleksitas, ini dilakukan pada waktu dan tempat yang sama setiap hari kerja *Sprint*. Jika *Product Owner* atau *Scrum Master* aktif mengerjakan item di *Sprint Backlog*, mereka berpartisipasi sebagai *developer*.

Pengembang dapat memilih struktur dan teknik apa pun yang mereka inginkan, selama *Scrum* harian mereka berfokus pada kemajuan Tujuan *Sprint* dan memiliki rencana yang dapat diterapkan untuk hari kerja berikutnya. Ini menciptakan fokus dan meningkatkan manajemen diri.

Daily Scrum dapat meningkatkan komunikasi, mengidentifikasi hambatan, dan memfasilitasi pengambilan keputusan yang cepat, menghilangkan kebutuhan untuk rapat ulang.

Daily Scrum bukan satu-satunya waktu pengembang menyesuaikan rencana mereka. Tim sering bertemu sepanjang hari untuk berdiskusi lebih rinci tentang cara menyesuaikan atau merencanakan ulang sisa *print*.

4. Pembahasan *Sprint* (*Sprint Review*)

Tujuan dari *Sprint Review* adalah untuk memeriksa hasil *Sprint* dan mengidentifikasi penyesuaian di masa depan. Tim scrum mempresentasikan hasil kerja mereka kepada pemangku kepentingan utama dan mendiskusikan kemajuan menuju tujuan produk.

Selama acara, Tim *Scrum* dan pemangku kepentingan meninjau apa yang telah dicapai dalam *Sprint* dan bagaimana lingkungan mereka telah berubah. Berdasarkan informasi ini, peserta berkolaborasi tentang apa yang harus dilakukan selanjutnya. *Product Backlog* juga dapat disesuaikan untuk memenuhi peluang baru. Tinjauan *Sprint* adalah rapat kerja dan Tim Scrum harus menghindari membatasinya pada presentasi.

Sprint Review adalah kegiatan *Sprint* kedua dari belakang, dan *Sprint* sebulan tidak melebihi maksimal empat jam. Acara biasanya lebih pendek untuk *Sprint* yang lebih pendek.

5. *Sprint Retrospektif*

Tujuan dari *Sprint Retrospective* adalah untuk merencanakan cara-cara untuk meningkatkan kualitas dan efektivitas.

Tim *Scrum* memeriksa bagaimana *Sprint* terakhir berjalan sehubungan dengan individu, interaksi, proses, alat, dan Definisi Selesai. Elemen yang diperiksa seringkali berbeda dengan domain pekerjaan. Asumsi yang menyesatkan mereka diidentifikasi dan asal-usulnya dieksplorasi. Tim *Scrum* mendiskusikan apa yang berjalan dengan baik selama *Sprint*, masalah apa yang dihadapi, dan bagaimana masalah tersebut (atau tidak) diselesaikan.

Tim *Scrum* mengidentifikasi perubahan yang paling membantu untuk meningkatkan efektivitasnya. Perbaikan yang paling berdampak ditangani sesegera mungkin. Mereka bahkan dapat ditambahkan ke *Sprint Backlog* untuk *Sprint* berikutnya.

Sprint Retrospective mengakhiri *Sprint*. Ini dibatasi waktu hingga maksimum tiga jam untuk *Sprint* satu bulan. Untuk *Sprint* yang lebih pendek, acaranya biasanya lebih pendek.

3. Komponen *Scrum* (*Scrum Artifacts*)

Artefak *scrum* mewakili pekerjaan atau nilai. Mereka dirancang untuk memaksimalkan transparansi informasi penting. Jadi setiap orang yang mengajinya memiliki dasar yang sama untuk adaptasi.

Setiap artefak berisi komitmen untuk memastikan bahwa artefak tersebut memberikan informasi yang meningkatkan transparansi dan fokus serta kemajuan yang dapat diukur:

- a. Untuk *Product Backlog*, itu adalah tujuan produk.
- b. Untuk *Sprint Backlog*, itu adalah *Sprint Goal*.

- c. Untuk *Increment*, itu adalah definisi penyelesaian.

Komitmen ini dirancang untuk memperkuat empirisme dan nilai-nilai *Scrum* bagi tim *Scrum* dan pemangku kepentingannya.

- a. *Product Backlog*

Product Backlog adalah daftar berurutan yang muncul dari apa yang dibutuhkan untuk meningkatkan suatu produk. Ini adalah satu-satunya sumber pekerjaan yang dilakukan oleh tim *Scrum*.

Item *Product Backlog* yang dapat dikerjakan oleh *Scrum Team* dalam sebuah *Sprint* dianggap siap untuk diseleksi dalam aktivitas *Sprint Planning*. Mereka biasanya mendapatkan transparansi ini setelah kegiatan pemurnian. Penyempurnaan *Product Backlog* adalah tindakan memecah dan mendefinisikan lebih lanjut item *Product Backlog* menjadi item yang lebih kecil dan lebih tepat. Menambahkan detail seperti deskripsi, urutan, dan dimensi adalah aktivitas yang berkelanjutan. Atribut sering bervariasi menurut bidang pekerjaan.

Pengembang yang akan melakukan pekerjaan bertanggung jawab atas ukurannya. Pemilik produk dapat mempengaruhi pengembang dengan membantu mereka memahami dan memilih pertukaran.

- b. *Sprint Backlog*

Sprint Backlog mencakup *Sprint Goal* (Mengapa), satu set Item *Product Backlog* yang dipilih untuk *Sprint* (Apa), dan rencana yang layak untuk memberikan *Increment* (Bagaimana).

Sprint Backlog adalah rencana yang dibuat oleh pengembang. Ini adalah gambaran *real-time* yang sangat visual tentang apa yang akan dicapai pengembang selama *Sprint* untuk mencapai Tujuan *Sprint*. Oleh karena itu, *Sprint Backlog* diperbarui

sepanjang *Sprint* karena lebih banyak yang dipelajari. Itu harus memiliki detail yang cukup sehingga mereka dapat memeriksa kemajuan mereka di *Scrum* harian.

c. *Increment* (Peningkatan)

Increment adalah batu loncatan konkret untuk mencapai tujuan produk. Setiap kenaikan melengkapi semua kenaikan sebelumnya dan divalidasi secara menyeluruh untuk memastikan bahwa semua kenaikan bekerja sama. Untuk memberikan nilai, peningkatan harus tersedia.

Mungkin ada beberapa peningkatan dalam *Sprint*. Jumlah kenaikan yang disediakan dalam *Sprint Review* mendukung empirisme. Namun, delta dapat dikirimkan ke pemangku kepentingan sebelum akhir *sprint*. Ulasan *sprint* tidak boleh dilihat sebagai cara untuk membuka nilai.

Pekerjaan tidak dapat dianggap sebagai bagian dari kenaikan kecuali jika memenuhi definisi selesai.

B. REST

REST adalah arsitektur yang digunakan untuk pemrograman web, dan beroperasi menggunakan protokol HTTP. Arsitektur REST berfokus pada sumber daya, yang masing-masing merupakan komponen terpisah. Sumber daya diakses melalui tautan menggunakan metode HTTP standar, dan setiap klien dan server dalam sistem REST bekerja secara berbeda, klien mengakses dan berinteraksi dengan sumber daya, dan server hanya menyediakan akses ke sumber daya.

Arsitektur REST adalah arsitektur pengembangan API yang menggunakan hyperlink, header, dan kode status untuk berkomunikasi. Layanan web RESTful adalah layanan web yang dikembangkan menggunakan arsitektur REST. Setiap sumber daya di REST diberi URI atau ID universal. Layanan web RESTful

menggunakan metode HTTP untuk menyediakan representasi sumber daya, seperti yang didefinisikan oleh Uniform Resource Identifier. Layanan web menggunakan metode HTTP untuk mengimplementasikan arsitektur REST.

1. URL Design

Akses RESTful API menggunakan protokol HTTP. Penamaan dan struktur URL yang konsisten akan menghasilkan API yang baik yang mudah dipahami oleh pengembang. URL API sering disebut sebagai titik akhir dalam pemanggilannya.

2. HTTP Verbs

Setiap permintaan yang dibuat di sana menggunakan metode sehingga server tahu apa yang diminta klien:

a. GET

GET adalah metode permintaan HTTP paling sederhana yang digunakan untuk membaca atau mendapatkan data dari suatu sumber.

b. POST

POST adalah metode permintaan HTTP yang digunakan untuk membuat data baru dengan memasukkan data ke dalam badan saat membuat permintaan.

c. PUT

PUT adalah metode permintaan HTTP yang biasanya digunakan untuk memperbarui data sumber daya.

d. DELETE

DELETE adalah metode permintaan HTTP yang digunakan untuk menghapus data dari sumber daya.

3. HTTP Response Code

Kode respons HTTP adalah kode standar yang memberitahu klien tentang hasil permintaan. Secara umum, ada 3 kode grup yang sering dijumpai di RESTful API, yaitu:

- a. 2XX : adalah kode respon yang menunjukkan bahwa permintaan berhasil.
- b. 4XX : adalah kode respon yang menunjukkan bahwa permintaan mengalami kesalahan di sisi klien.
- c. 5XX: adalah kode respon, yang menunjukkan bahwa permintaan mengalami kesalahan di sisi server.

4. Format Response

Setiap permintaan yang dibuat, klien akan menerima data respons dari server, biasanya dalam bentuk data XML atau JSON. Setelah klien memiliki data respons, klien dapat menggunakannya dengan menguraikan data dan memprosesnya sesuai kebutuhan.

C. MongoDB

MongoDB adalah sistem manajemen basis data yang dirancang untuk internet dan aplikasi berbasis web. MongoDB adalah database NoSQL berbasis dokumen. Model data dokumen MongoDB membuatnya mudah untuk dibuat aktif, karena memiliki dukungan untuk data tidak terstruktur. Dokumen MongoDB dikodekan dalam format seperti JSON, yang disebut BSON. BSON sangat cocok untuk objek modern metodologi pemrograman berorientasi dan ringan dan cepat (Hema Krishnan, 2016).

Perbedaan utama antara NoSQL dan *Relational Database Management System* (RDBMS) adalah bagaimana data dimodelkan pada basis data. Pemodelan pada RDBMS masih menggunakan tabel-tabel terstruktur, dimana kolom-kolom pada setiap barisnya tetap satu sama lain, sedangkan pemodelan data pada NoSQL khususnya pada MongoDB menggunakan dokumen-dokumen dimana setiap baris memiliki Kolom-kolom yang berbeda dengan baris lainnya. Contoh perbedaan konseptual database antara RDBMS dan NoSQL ditunjukkan pada **Gambar 2.1** (Pemodelan RDBMS) dan **Gambar 2.2** (Pemodelan NoSQL).

Users				
ID	first_name	last_name	cell	city
1	Leslie	Yepp	8125552344	Pawnee

Hobbies		
ID	user_id	hobby
10	1	scrapbooking
11	1	eating waffles
12	1	working

Gambar 2.1: Pemodelan RDBMS

Sumber: <https://www.mongodb.com/nosql-explained>

```
{  
    "_id": 1,  
    "first_name": "Leslie",  
    "last_name": "Yep",  
    "cell": "8125552344",  
    "city": "Pawnee",  
    "hobbies": ["scrapbooking", "eating waffles", "working"]  
}
```

Gambar 2.2: Pemodelan NoSQL

Sumber: <https://www.mongodb.com/nosql-explained>

Menurut Gambar 2.1, untuk menyimpan data *user* dan data *hobby*, diperlukan dua tabel terpisah, sedangkan dalam pemodelan NoSQL Gambar 2.2, data *user* dan *hobby* dalam satu dokumen yang sama. Dengan NoSQL, ketika ingin mengambil dua bagian data secara bersamaan, hanya memerlukan satu dokumen tanpa gabungan, yang menghasilkan kueri lebih cepat daripada RDBMS.

Meskipun NoSQL, desain database masih harus dilakukan. Desain basis data memungkinkan untuk mengonfirmasi dan mendokumentasikan pemahaman tentang basis data dan memastikan bahwa orang-orang melihat lanskap informasi dengan cara yang sama. Dengan kata lain, desain database adalah alat komunikasi. Berikut merupakan komparasi dari RDBMS dan NoSQL **Table 2.1** (Kunda dan Phiri, 2017).

Tabel 2.1: Komparasi Database RDBMS dan NoSQL

No	Kriteria	RDBMS	NoSQL
1	Variety	Terdapat dalam jenis open source dan close platform	Kebanyakan dalam bentuk open source
2	Scalability	Meningkatkan performa dengan meningkatkan hardware dalam server	Meningkatkan dengan cara horizontal atau menambah server
3	Cost	Lebih mahal mengingat harusnya penambahan hardware	Lebih murah mengingat murahnya upgrade secara horizontal
4	Volume of Data	Menangani data yang terbatas	Menangani data yang lebih besar
5	Performance	Lebih lambat karena perlunya memproses informasi	Memiliki query performance lebih baik
6	Complexity	Lebih kompleks karena memerlukan normalisasi hubungan antara tabel	lebih flexible dapat menyimpan struktur yang berbeda dalam satu collections
7	Consistency	Dengan skema yang kompleks menjadikannya lebih konsisten	Kurang konsisten dikarenakan skema yang bermacam-macam
8	Security	Memiliki mekanisme keamanan yang baik untuk melindungi data	Keamanan ditangani oleh middleware dan bukan bagian dari database

D. Flask Framework

Framework adalah kerangka kerja untuk mengembangkan aplikasi berbasis web dan desktop. Kerangka kerja di sini sangat membantu pengembang untuk menulis hal-hal yang lebih terstruktur dan rapi.

Framework ini dibuat untuk menyederhanakan kinerja bagi pemrogram. Jadi programmer tidak perlu menulis kode berulang-ulang. Karena hanya perlu mengkompilasi komponen pemrograman itu sendiri (Adani, 2020).

Flask Framework merupakan kerangka kerja yang berbasis web. Flask dibangun diatas bahasa pemrograman python dengan menggunakan dependensi Werkzeug dan Jinja2. Kerangka kerja Flask bersifat mikro karena tidak membutuhkan alat-alat tertentu atau pustaka. Flask mendukung ekstensi yang dapat menambahkan fitur aplikasi seolah-olah mereka diimplementasikan dalam Flask itu sendiri.

1. Flask routing

Routing adalah modul dalam aplikasi yang mengatur pengoperasian aplikasi berbasis web. Route dapat menangani semua perintah yang telah dideklarasikan. Routing bisa dianalogikan sebagai route diagram penjelas tentang cara menavigasi aplikasi yang sedang dibangun.

App Routing berarti memetakan URL ke fungsi tertentu yang akan menangani logika untuk URL tersebut. Berikut adalah contoh routing di framework flask:

```

1 from flask import Flask
2
3 app = Flask(__name__)
4
5 # Pass the required route to the decorator.
6 @app.route("/hello")
7 def hello():

```

```

8     return "Hello, Welcome to GeeksForGeeks"
9
10 @app.route("/")
11 def index():
12     return "Homepage of GeeksForGeeks"
13
14 if __name__ == "__main__":
15     app.run(debug=True)

```

Seperti potongan code di atas pemetaan URL ke fungsi menggunakan decorator sebelum fungsi ditulis. Dan contoh diatas merupakan contoh yang sederhana pada routing. Berikut adalah beberapa hal yang bisa dilakukan flask routing:

a. Route Methods

Pada dasarnya saat melakukan routing, secara default metode yang dipakai adalah GET. Untuk membuat agar URL dapat menggunakan metode lain seperti POST diharuskan mendefinisikan metodenya pada argumen route.

```

1 @app.route('/home', methods=['POST', 'GET'])
2 def home():
3     return '<h1>Anda berada di halaman beranda!</h1>'

```

b. Route Variables

Meneruskan variables merupakan sebuah fitur dasar yang memungkinkan pengguna mengirimkan informasi melalui URL. Untuk mengimplementasikan fitur ini route harus mendefinisikan variabel tersebut dalam URL dan menjadikannya argumen pada fungsinya (Rasouli, 2020).

```

1 @app.route('/home/<firstname>', methods=['POST', 'GET'])
2 def home(firstname):
3     return '<h1>Halo {}, Anda berada di halaman beranda!</h1>'.format(firstname)

```

2. Dependencies

Dependencies merupakan ketergantungan suatu sistem kepada sistem lainnya. Dalam halnya framework, ketergantungannya adalah terhadap sebuah package atau extension. Package atau extension berisi sebuah resource suatu logic yang bisa diimplementasikan dengan mudah dengan hanya memanggil package tersebut yang biasanya berbentuk object class.

a. Flask-REStful

Flask-REStful merupakan extension untuk framework Flask untuk membantu membangun REST API dengan cepat. Flask-REStful mengedepankan peraktek terbaik dengan pengaturan yang diminimalkan (Kevin Burke, 2020).

Pengimplementasian Flask-REStful dimulai dengan penginstalan dengan cara

```
1 pip install flask-restful
```

dilanjutkan dengan menggunakan import Resource dan Api dari Flask-REStful, buat variabel yang akan menyimpan sebuah class Api dengan argumen Flask, kemudian buat class child dari sebuah class Resource, pada setiap class sudah dapat menerapkan fungsi dari parent Resource seperti get, post, dll, kemudian adalah tambahkan mapping class tersebut kepada string endpoint URL.

```
1 from flask import Flask
2 from flask_restful import Resource, Api
3
4 app = Flask(__name__)
5 api = Api(app)
6
7 class HelloWorld(Resource):
8     def get(self):
9         return {'hello': 'world'}
10
11 api.add_resource(HelloWorld, '/')
12
```

```

13 if __name__ == '__main__':
14     app.run(debug=True)

```

Listing II.1: Python example

b. Flask-Mongoengine

Flask-Mongoengine merupakan extension Flask yang menyediakan integrasi Flask ke MongoEngine. MongoEngine sendiri merupakan library python yang berperan sebagai Object Document Mapper dengan MongoDB (Streetlife, 2020).

Pengimplementasian Flask-Mongoengine dimulai dengan penginstalan dengan cara

```

1 pip install flask-mongoengine

```

dilanjutkan dengan import Mongoengine dari Flask-Mongoengine, lakukan inisiasi untuk app Flask, gunakan fungsi ".config" untuk melakukan konfigurasi database yang akan di pakai, lalu inisiasi MongoEngine dan gunakan fungsi "init_app" dengan memasukkan parameter app Flask.

```

1 import flask
2 from flask_mongoengine import MongoEngine
3
4
5 db = MongoEngine()
6 app = flask.Flask("example_app")
7 app.config["MONGODB_SETTINGS"] = [
8     {
9         "db": "project1",
10        "host": "localhost",
11        "port": 27017,
12        "alias": "default",
13    }
14]
15 db.init_app(app)

```

Penggunaan MongoEngine diawali dengan mendefinisikan dokumen yang akan disimpan dalam bentuk class yang memiliki parent Document. Mendefinisikan dokumen juga mengharuskan kita mendefinisikan field yang ada didalamnya dengan menetapkannya sebagai property class dan menetapkan jenis dari setiap property class tersebut (Ross Lawley, 2020).

```

1 from mongoengine import *
2 import datetime
3
4 class Page(Document):
5     title = StringField(max_length=200, required=True)
6     date_modified = DateTimeField(default=datetime.datetime.utcnow)
```

Setelah Pendefinisan dokumen, memanipulasi data dapat dilakukan dengan memanggil fungsi pada class tersebut. Berikut merupakan contoh dari manipulasi data pada Mongoengine.

```

1 from mongoengine import *
2
3 #Saving Documents
4 page = Page(title="Test Page")
5 page.save()
6 id = page.id
7
8 #Update Documents with id
9 page2 = Page.objects.get(id=id)
10 page2.update({title: "Test 123"})
11
12 #Delete Documents with id
13 page3 = Page.objects.get(id=id)
14 page3.delete()
```

E. Apache

Apache adalah perangkat lunak server yang dapat digunakan untuk membuat koneksi antara browser web dan server menggunakan protokol HyperText Transfer,

atau HTTP. Apache memiliki fitur yang cukup untuk melakukan konfigurasi dasar seperti pesan kesalahan dan otentikasi berbasis database. Saat digunakan, itu akan membuat kumpulan proses, atau daemon, untuk menangani permintaan. Arsitektur Apache didasarkan pada model client-server berbasis thread yang menggunakan modul; modul ini termasuk keamanan, penulisan ulang URL dan otentikasi kata sandi antara lain (Foundation, 2022).

1. Konfigurasi httpd.conf

Konfigurasi Apache pada server adalah dengan mendaftarkan Aplikasi pada file httpd.conf. Berikut merupakan contoh pendaftaran pada httpd.conf

```
1 WSGIDaemonProcess /fishapi python-path=/opt/rh/rh-python38/root/lib/pyt$  
2 WSGIProcessGroup /fishapi  
3 WSGIApplicationGroup %{GLOBAL}  
4 WSGIScriptAlias /fishapi /var/www/html/fishapi/index.wsgi  
5 WSGIScriptReloading on  
6 <Directory "/var/www/html/fishapi/fishapi">  
7   AllowOverride All  
8   Options +ExecCGI  
9   AddHandler cgi-script .cgi .pl .py  
10  Order allow,deny  
11  allow from all  
12 </Directory>  
13  
14 Alias /fishapi/static /var/www/html/fishapi/fishapi/static  
15 <Directory /var/www/html/fishapi/fishapi/static>  
16   AllowOverride All  
17   Order allow,deny  
18   allow from all  
19 </Directory>
```

F. Unit Testing

Unit testing merupakan salah satu tipe pengujian perangkat lunak dimana setiap fungsi atau komponen dari perangkat lunak diuji. Tujuan dari unit testing adalah untuk memastikan fungsi pada perangkat lunak sudah berjalan sesuai dengan ekspektasi (Hamilton, 2022). Menurut (Rosa, 2016), “Unit testing berfokus pada pengujian unit terkecil (komponen perangkat lunak atau modul) dari desain perangkat lunak. Semua fungsi pada perangkat lunak diuji untuk memastikan bahwa input dan output unit sesuai dengan yang diinginkan”.

Teknik yang dilakukan pada unit testing adalah “berfokus pada setiap unit perangkat lunak (misalnya komponen, kelas, atau objek konten dari aplikasi atau web) seperti yang diterapkan dalam kode program” (Pressman, 2012). Kode program dikaji apakah terdapat kesalahan. Kesalahan pada kode program dapat diidentifikasi dengan menggunakan White-Box Testing. Sedangkan menurut (Rosa, 2016), “White-Box Testing (pengujian kotak putih) merupakan pengujian perangkat lunak atau aplikasi dari segi desain dan kode program untuk mengetahui apakah perangkat lunak mampu menghasilkan fungsi-fungsi, masukan, dan keluaran yang sesuai dengan spesifikasi kebutuhan”.

Proses unit testing memastikan fungsi-fungsi pada aplikasi yang telah dikembangkan peneliti memenuhi persyaratan, dapat berjalan dengan baik, dan memiliki input serta output sesuai yang diinginkan. Berdasarkan definisi di atas dapat disimpulkan bahwa unit testing merupakan salah satu tipe pengujian fungsi atau unit pada perangkat lunak untuk memastikan apakah perangkat lunak mampu untuk menghasilkan input dan output sesuai dengan spesifikasi yang telah ditentukan.

G. User Acceptance Test (UAT)

Menurut (Perry, 2006) User Acceptance Test (UAT) yaitu pengujian untuk verifikasi apakah fungsi pada sistem telah berjalan dengan kebutuhan, pengujian dilakukan oleh user dimana user tersebut adalah staff/karyawan perusahaan yang langsung berinteraksi dengan sistem. Menurut Black, acceptance testing pada umumnya menunjukkan bahwa sistem telah memenuhi persyaratan-persyaratan tertentu. Pada pengembangan software dan hardware komersial, acceptance test biasanya disebut juga "alpha tests" (yang dilakukan oleh pengguna in-house) dan "beta tests" (yang dilakukan oleh pengguna yang sedang menggunakan atau akan menggunakan sistem tersebut).

Menurut (Hady dkk., 2020) menyebutkan bahwa pelaksanaan UAT terdapat pada akhir proses pengujian saat sistem siap digunakan. Tujuan utama UAT adalah untuk memvalidasi apakah sistem diterima atau ditolak, memenuhi spesifikasi sistem, dan mengembangkan perangkat lunak yang mampu memenuhi kebutuhan user.

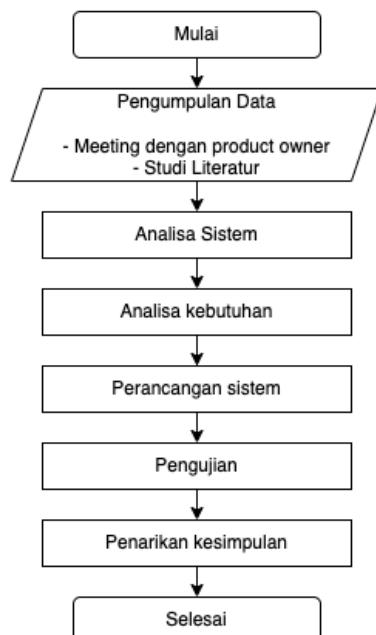
Proses UAT memastikan bahwa web service berjalan dapat berjalan dengan baik, mengbalikkan respon sesuai kebutuhan bagi kepentingan pengembangan frontend. Dari definisi yang telah dipaparkan dapat disimpulkan bahwa UAT adalah pengujian pada akhir proses yang dilakukan oleh pengguna pada sebuah sistem untuk memastikan fungsi-fungsi yang terdapat pada sistem tersebut telah berjalan dengan baik dan sesuai dengan kebutuhan pengguna.

BAB III

Metodologi Penelitian

Penelitian yang dilakukan oleh penulis akan menghasilkan produk tertentu dan akan dilakukan pengujian keefektifannya (Purnama, 2013). Menurut Suhadi Ibnu (Purnama, 2013), penelitian pengembangan bertujuan untuk menghasilkan suatu produk baik itu software ataupun hardware melalui prosedur yangumumnya dimulai dengan menganalisis kebutuhan, kemudian lanjut ke proses pengembangan, dan diakhiri dengan evaluasi.

Berdasarkan pengertian tersebut, penelitian yang akan dilaksanakan oleh penulis masuk ke dalam jenis Penelitian dan Pengembangan/Research and Development. Tahapan penelitian yang akan dilaksanakan penulis dalam perancangan aplikasi dapat dilihat pada **Gambar 3.1**.



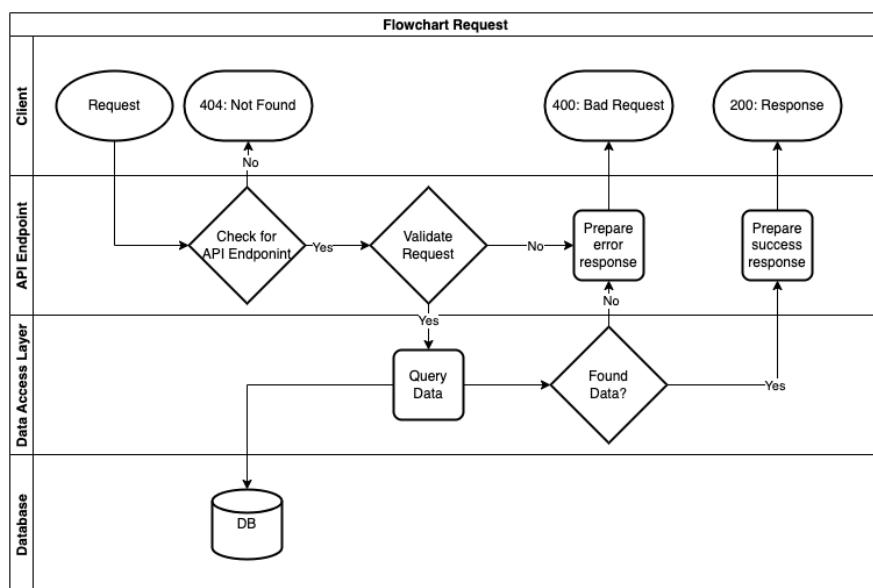
Gambar 3.1: Tahapan Peneltian

A. Pengumpulan Data

Data diambil dari hasil wawancara dengan owner J Farm Technology (JFT) Muhammad Eka Suryana, M.Kom., yang sekaligus merupakan klien dari penelitian ini. Selain itu, dilakukan studi literatur dengan membaca jurnal-jurnal yang terkait dengan topik penelitian. Untuk transkrip wawancara dapat dibaca pada Lampiran 1

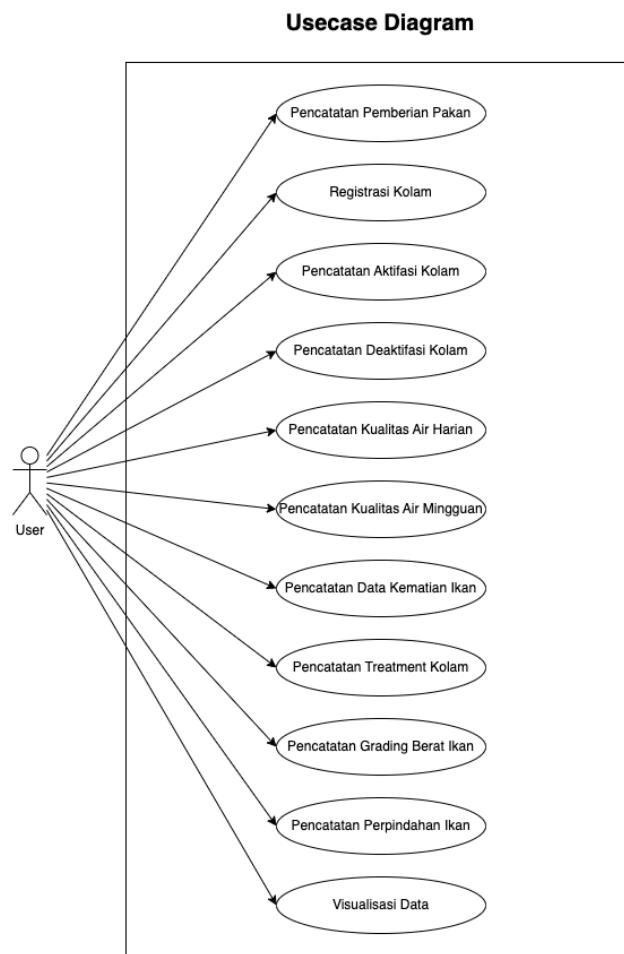
B. Analisa Sistem

Sistem yang akan dihasilkan oleh penelitian ini berupa sebuah web service, lebih tepatnya adalah Private API. Sistem akan menggunakan arsitektur REST yang mana sering juga disebut RESTful. Sistem akan memberikan respons dari setiap request yang dilakukan oleh client. Request yang dilakukan oleh client memakai protokol http dan response yang diberikan oleh sistem berupa JSON Object. Yang nantinya sistem akan dijalankan di online server agar bisa di akses oleh client menggunakan akses internet.



Gambar 3.2: Flowchart request

Dimulai dari request yang dilakukan oleh client, ada beberapa proses yang akan dilakukan oleh sistem. Proses yang dilakukan oleh sistem diantaranya adalah pengecekan API EndPoint, validasi request, pengambilan dan penulisan data di database, dan persiapan response. Pengecekan API EndPoint dilakukan oleh sistem untuk mengetahui request apa yang diminta oleh client dari URL yang diakses. validasi request dilakukan untuk memastikan data yang dikirim oleh user telah sesuai dengan protokol yang telah ditentukan. Setelahnya proses yang dilakukan adalah manipulasi atau membaca data di database. dan diakhiri dengan persiapan protokol response sebelum dikirim ke client.



Gambar 3.3: Usecase diagram

Gambar 3.3 merupakan usecase diagram user yang dibuat berdasarkan story yang telah didiskusikan bersama *scrum master*. usecase tersebut menggambarkan kegiatan apa saja yang bisa dilakukan oleh user dalam menggunakan aplikasi. Pada penerapan pada *backend usecase* nantinya akan dipecah lebih mendetail saat mendeskripsikan *task* dari *story*.

C. Analisa Kebutuhan

Berdasarkan uraian pada lampiran 1., prioritas fitur pada webservice perikanan modern terfokus pada pencatatan aktifitas pembudidaya ikan seperti pemberian pakan, pencatatan kematian ikan, dll.

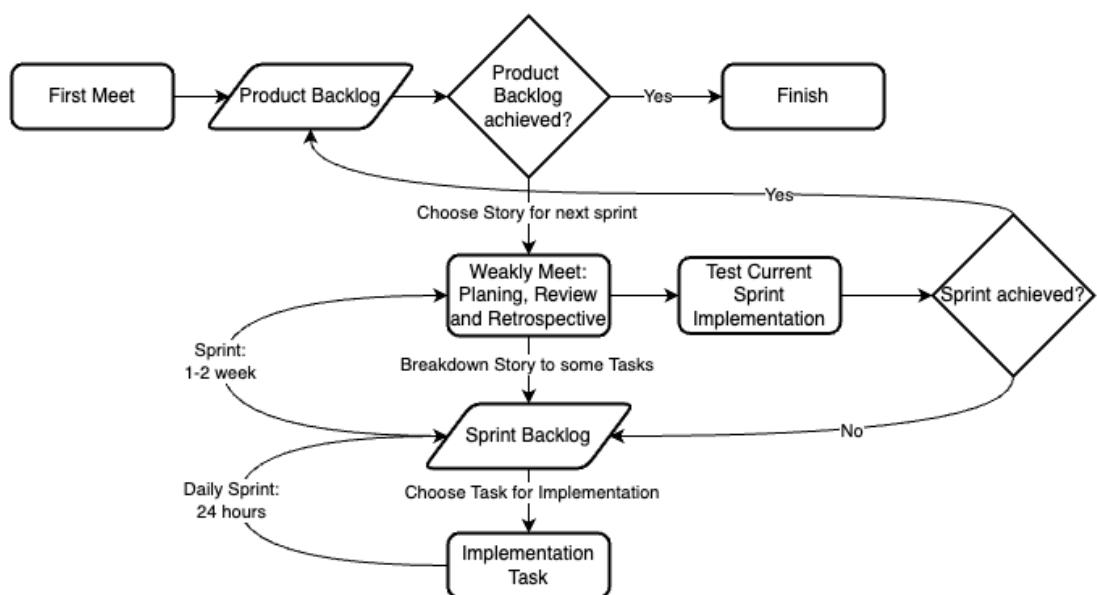
Perangkat keras dan perangkat lunak yang penulis butuhkan adalah sebagai berikut:

1. Perangkat keras, terdiri dari:
 - a. Laptop dengan spesifikasi Processor Apple M1 & Ram 8 GB
2. Perangkat lunak, terdiri atas:
 - a. MacOs ventura 13.0
 - b. Visual Studio Code sebagai IDE dan code editor
 - c. Python sebagai bahasa pemrograman penyusun aplikasi web service
 - d. Flask sebagai framework python
 - e. MongoDB sebagai basis data

D. Perancangan Sistem

Untuk keberlangsungan penelitian, penulis menggunakan metodologi pengembangan sistem scrum. Adapun tahapan-tahapan bagian dari scrum yang

dilakukan dalam proses penelitian skripsi yang berjudul “Perancangan Arsitektur Backend Server Teknologi Perikanan Modren Berikut Spesifikasi Web Service yang Mampu Berhubungan dengan Multi Platform dengan Metode Scrum”. Flowchart metodologi Scrum yang dapat dilihat pada **Gambar 3.4**.



Gambar 3.4: Flowchart Scrum

Tahapan pertama penelitian adalah melakukan meeting dengan product owner dan scrum master. Meeting tersebut bertujuan untuk mendefinisikan product backlog yang akan menjadi acuan sprint kedepannya. Berikutnya merupakan weekly meeting pertama, agenda pada meeting ini adalah sprint planning. Sprint planning dilakukan dengan cara memilih story dari product backlog yang akan dilakukan breakdown sehingga menghasilkan sprint backlog. Sprint backlog merupakan kumpulan dari task-task kecil yang akan diimplementasi oleh developer setiap harinya. Weekly meeting berikutnya memiliki agenda tambahan dibanding dengan weekly meeting pertama yaitu, adanya sprint review dan sprint retrospective sebelum dilakukannya sprint planning untuk minggu berikutnya. Sprint review dan

retrospective merupakan sebuah kajian dari task yang sudah dikerjakan oleh developer dan sprint yang dilakukan bersama oleh scrum master. Apabila tujuan dari sprint itu tercapai, maka scrum master akan menentukan story dari product backlog untuk sprint selanjutnya. Setelah semua product backlog tercapai maka sistem bisa dikatakan telah selesai.

Metode scrum terdiri dari beberapa komponen diantaranya adalah product backlog, sprint backlog, sprint, daily scrum, dan pengujian sistem. Berikut adalah penjelasan dari komponen-komponen tersebut:

1. Produk Backlog

Product backlog dibuat berdasarkan hasil diskusi dengan product owner sekaligus scrum master. Transkrip percakapan diskusi terdapat pada **Lampiran**. Setiap story yang ada di product backlog akan diselesaikan bertahap tiap minggunya. Berikut adalah tabel product backlog.

Tabel 3.1: *Product Backlog*

No	User Story	Sprint	Priority
1	Modul CRUD dan tampilan Pencatatan Pemberian pakan	1,2	High
2	Modul CRUD dan tampilan untuk Registrasi kolam	3	High
3	Modul CRUD dan tampilan untuk Masa Budidaya Kolam	4	High
4	Modul CRUD dan tampilan untuk Pencatatan data kematian ikan	5,6	Medium
5	Modul CRUD dan tampilan untuk Pencatatan Sortir ikan	7	Medium
6	Modul CRUD dan tampilan untuk Pencatatan Grading berat ikan	8	Medium
7	Modul CRUD dan tampilan untuk Pencatatan kualitas air harian	9	Low
8	Modul CRUD dan tampilan untuk Pencatatan kualitas air mingguan	10	Low
9	Modul CRUD dan tampilan untuk Pencatatan treatment kolam	11	Low

Dari **Tabel 3.1** di atas, Produk Backlog terdiri dari empat komponen yaitu, User Story, Sprint, dan Priority. Story merupakan sebuah pekerjaan yang menggambarkan suatu fitur atau suatu module, yang mana nantinya akan diuraikan kedalam sprint dalam bentuk task. Komponen sprint merupakan perkiraan pada sprint keberapakah story akan dikerjakan. Priority merupakan sebuah keterangan seberapa prioritasnya user story.

2. Sprint Backlog

Sprint backlog merupakan sebuah task atau pekerjaan yang cenderung lebih kecil atau ringan. Sprint backlog merupakan uraian dari satu story pada product backlog yang menghasilkan beberapa task ringan. Sprint backlog diuraikan oleh scrum master. kemudian, developer bisa memilih task mana yang akan dikerjakan dahulu. Berikut merupakan tabel sprint backlog pertama.

1. Sprint-1

Sprint-1 dilaksanakan selama delapan minggu yaitu pada tanggal 5 April sampai 31 Mei 2021. delapan minggu tersebut dikarenakan terdapatnya libur lebaran idul fitri selama 2 minggu yaitu pada tanggal 25 April - 9 May. Lalu enam minggu sisanya merupakan waktu yang dibutuhkan penulis untuk menulis penelitian, pembelajaran Scrum, pembelajaran framework flask, dan pembelajaran deployment pada server. Sprint ini merupakan uraian dari story "Pencatatan pemberian pakan" pada product backlog.

Tabel 3.2: Sprint-I backlog

No	Story	Task	Status
1	Create, Read, Update, dan Delete untuk Pencatatan Pemberian pakan	Merancang Database	Completed
2		Merancang struktur flask framework	Completed
3		Merancang class diagram	Completed
4		Implementasi model Pond, Feed_type, Feed_history	Tested
5		Implementasi controller EndpointApi	Tested
6		Deployment Sistem	Tested
7		Konfigurasi sistem di server	Tested
8		Merancang Dokumentasi API	Tested

3. Daily Scrum

Daily scrum tidak dilakukan secara meeting dikarenakan terbatasnya waktu scrum master dan developer. Maka dari itu, daily scrum digantikan dengan mencatat hambatan dan hal penting yang terjadi pada saat menjalankan sprint setiap harinya pada note github projects.

4. Sprint Review dan Sprint Retrospective

Sprint review dan retrospective dilakukan pada awal pekan tepatnya pada hari selasa. Sprint review dan retrospective dilakukan dengan cara voice call melalui platform telegram dan discord. Pembahasan dari meeting ini dimulai dari evaluasi perkembangan sistem, hambatan yang terjadi, dan penetapan sprint kedepannya.

E. Pengujian

Pada tahap ini peneliti akan melakukan uji backend server perikanan modern menggunakan dua jenis pengujian yaitu unit testing dan User Acceptance Test (UAT). Pengujian unit testing dilaksanakan oleh developer untuk memastikan fungsi-fungsi pada aplikasi yang telah dikembangkan dapat berjalan dengan baik. Sedangkan UAT dilaksanakan oleh scrum master untuk mengetahui apakah backend server sudah sesuai dengan kebutuhan dan layak untuk dipakai oleh developer frontend.

1. Unit Testing Skenario pada unit testing dibuat berdasarkan product backlog.

Adapun skenario dari unit testing yang akan diuji oleh developer terdapat pada

Tabel 3.3.

Tabel 3.3: Skenario unit testing.

Awal Tabel	
Uji unit	Skenario Pengujian
Pemberian Pakan	API pencatatan pemberian pakan
	API merubah data pemberian pakan
	API mendapatkan list data pemberian pakan pada suatu kolam
	API mendapatkan detail data pemberian pakan
	API hapus data pemeberian pakan
Registrasi Kolam	API pencatatan registasi kolam
	API merubah data kolam
	API mendapatkan list data kolam
	API mendapatkan detail data kolam
	API hapus data kolam

Lanjutan Tabel 3.3	
Uji unit	Skenario Pengujian
Musim Budidaya Kolam	API memulai musim budidaya pada kolam
	API mengakhiri musim budidaya atau panen pada kolam
	API mendapatkan list musim budidaya pada suatu kolam
Pencatatan Kematian Ikan	API pencatatan kematian ikan
	API merubah data kematian ikan
	API mendapatkan list data kematian ikan pada suatu musim budidaya
	API mendapatkan detail kematian ikan
	API hapus kematian ikan
Perpindahan Ikan Antar Kolam	API pencatatan perpindahan ikan antar kolam
	API merubah data perpindahan ikan antar kolam
	API mendapatkan list data perpindahan ikan pada suatu musim budidaya
	API mendapatkan detail perpindahan ikan
	API hapus data perpindahan ikan
Grading berat ikan	API pencatatan grading berat ikan
	API merubah data grading berat ikan
	API mendapatkan list data grading berat ikan pada suatu musim budidaya
	API mendapatkan detail grading berat ikan

Lanjutan Tabel 3.3	
Uji unit	Skenario Pengujian
	API hapus data grading berat ikan
Pencatatan kualitas air harian	API pencatatan kualitas air harian kolam
	API merubah data kualitas air harian kolam
	API mendapatkan list data kualitas air harian kolam pada suatu musim budidaya
	API mendapatkan detail kualitas air harian
	API hapus data kualitas air harian kolam
Pencatatan kualitas air mingguan	API pencatatan kualitas air mingguan kolam
	API merubah data kualitas air mingguan kolam
	API mendapatkan list data kualitas air mingguan kolam pada suatu musim budidaya
	API mendapatkan detail kualitas air mingguan kolam
	API hapus data kualitas air mingguan kolam
Pencatatan treatment kolam	API pencatatan treatment kolam
	API merubah data treatment kolam
	API mendapatkan list data treatment kolam pada suatu musim budidaya
	API mendapatkan detail treatment kolam
	API hapus data treatment kolam
Akhir Tabel 3.3	

2. User Acceptance Test Skenario pada User Acceptance Test dibuat berdasarkan fitur-fitur yang dapat diakses oleh owner. Adapun skenario dari UAT yang akan

dilaksanakan terdapat pada tabel

Tabel 3.4: Tabel *User Acceptance Test*

Uji Fitur	Skenario Pengujian
Pemberian pakan	Menampilkan view tabel data pemberian pakan per kolam
Registrasi kolam	Menampilkan view tabel data kolam yang terdaftar
Musim budidaya kolam	Menampilkan view tabel data musim budidaya per kolam
Kematian ikan	Menampilkan view tabel data kematian ikan per musim budidaya
Perpindahan ikan	Menampilkan view tabel data perpindahan ikan antar kolam
Grading berat ikan	Menampilkan view tabel data grading berat ikan per musim budidaya
Kualitas air harian	Menampilkan view tabel data kualitas air harian per musim budidaya
Kualitas air mingguan	Menampilkan view tabel data kualitas air mingguan per musim budidaya

BAB IV

Hasil Dan Pembahasan

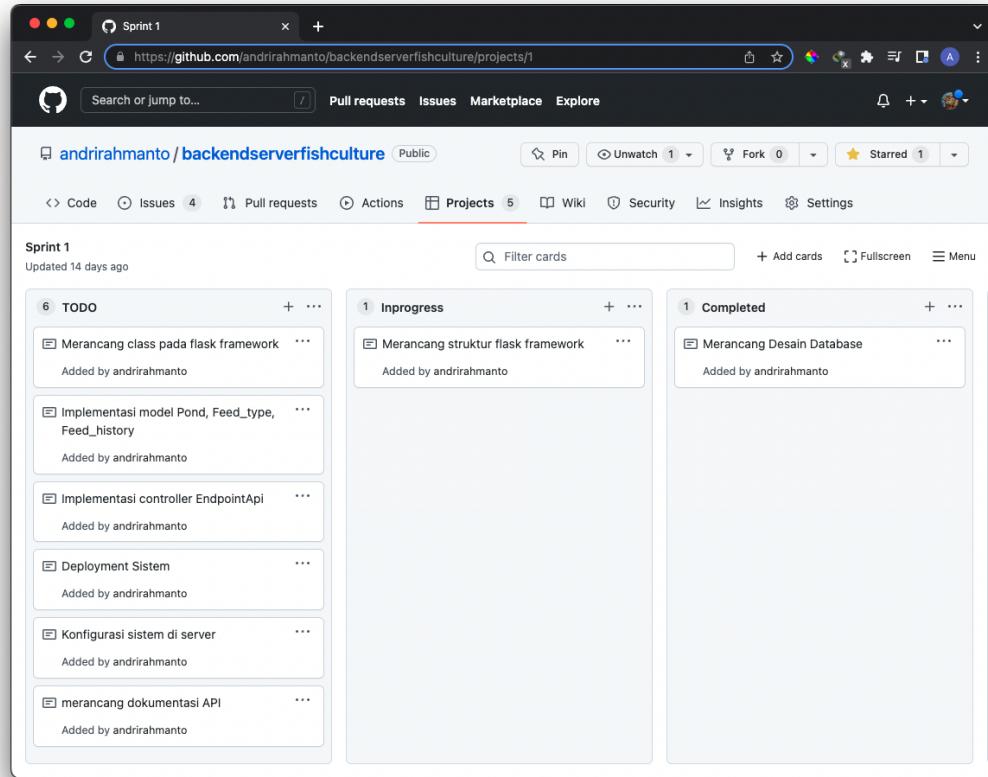
A. Perancangan Sistem

Perancangan Web service sebagai backend aplikasi budidaya ikan modern dengan metode Scrum. Dalam metode Scrum ini pengembangan sistem dilakukan dengan iterasi dalam setiap Sprint yang urutan pengerjaan Sprint-nya terdapat pada Product Backlog. Berikut laporan lanjutan dalam pengembangan sistem menggunakan metode scrum:

1. Sprint 1 Report

Pada sprint pertama ini story yang di pilih untuk di uraikan pada sprint kali ini adalah pencatatan pemberian pakan. Rentang waktu sprint ini sedikit lebih lama dari sprint lainnya sekitar 2-3 minggu, yang mana sprint lainnya hanya 1-2 minggu. Perbedaan tersebut terjadi karena penulis melakukan training terhadap bahasa dan framework yang akan digunakan dalam pengembangan sistem, sekaligus memberi waktu penulis untuk melakukan pendaftaran dan penulisan pada penelitian.

Tujuan dari sprint-1 adalah membuat CRUD berbentuk API untuk sebuah fitur pencatatan pemberian pakan. Dimulai dari mendesain database, desain struktur sistem, desain class pada sistem, membuat program dari desain yang sudah ada, dan yang terakhir adalah melakukan deployment sistem ke server. Penggunaan platform Github Projects untuk mempermudah manajemen task dan progress seperti pada **Gambar 4.1.**

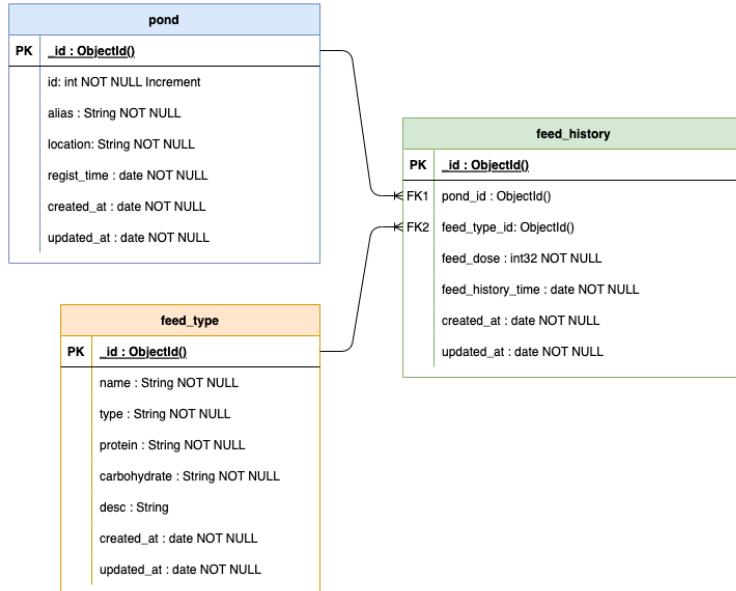


Gambar 4.1: Github Projects Sprint-1

Dari **Gambar 4.1** diatas, terdapat 4 kolom yang menggambarkan fase dari task tersebut. 4 fase itu adalah to do, in progress, completed, dan tested. To do menandakan fase dimana task baru hanya didaftarkan pada list. In progress menandakan fase dimana task sedang dalam proses penggerjaan oleh developer. Completed merupakan fase dimana task sudah selesai dikerjakan. Dan tested merupakan fase dimana task tertentu telah di test oleh developer dan scrum master.

a). Merancang Database

Desain Database atau biasa disebut Entity Relationship Diagram (ERD) pada sprint-1 menggambarkan struktur data dan relasi antar data yang akan disimpan pada database.



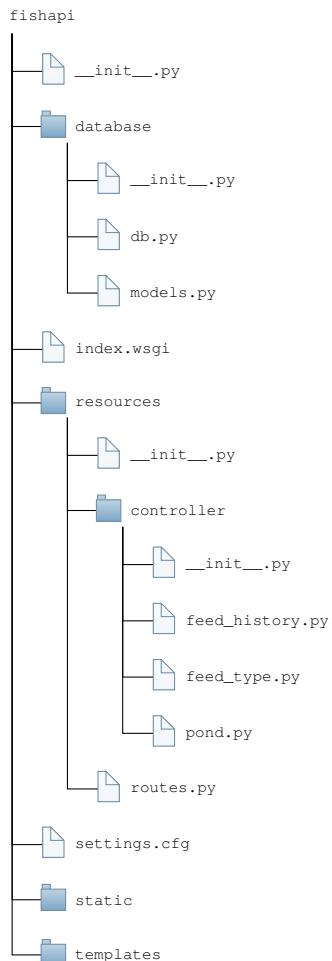
Gambar 4.2: ERD Database Sprint-1

Terdapat 3 tabel yang dibutuhkan untuk menunjang fitur pemberian pakan. Tabel *pond* tabel *pond* berfungsi untuk menyimpan data terkait kolam, dimana ada beberapa field yaitu nama dan lokasi kolam. Tabel *feed_type* berfungsi menyimpan data terkait tipe pakan, dimana field yang disimpan adalah nama pakan, tipe pakan, kandungan protein pakan dalam bentuk persentase, kandungan karbohidrat pakan dalam bentuk persentase, dan keterangan tambahan. Tabel *feed_history* berfungsi untuk menyimpan data terkait pemberian pakan, dimana field yang disimpan adalah id kolam, id tipe pakan, dosis pakan, dan waktu pemberian pakan.

b). Merancang Struktur Flask Framework

Rancangan struktur flask framework menggambarkan direktori-direktori pada sistem yang akan di buat. terdapat 4 direktori penting yaitu database, resource, static, template. Direktori database berisi file yang berkaitan dengan database mongodb yaitu, koneksi database dan model data pada database. Direktori

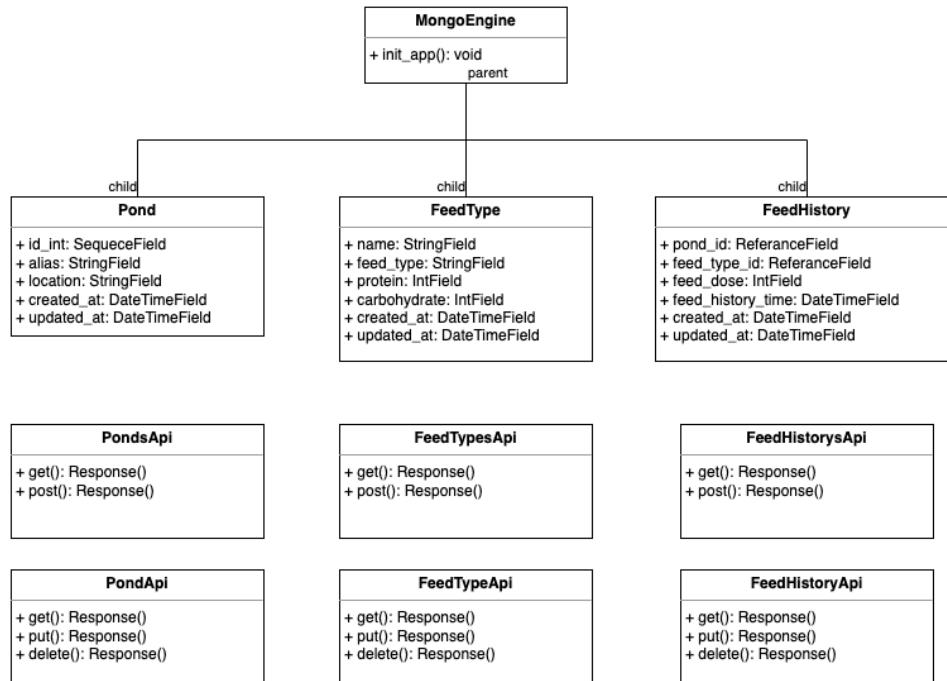
resource deskripsi routes dan controller untuk setiap logic API endpoint. Direktori static berisi hal-hal yang dibutuhkan untuk styling sebuah halaman html, yang biasanya berisi css file dan javascript file. Direktori template merupakan tempat penulis menyimpan html file. Pembagian direktori tersebut bertujuan untuk mempermudah pengolahan dalam pengembangan sistem kedepannya.



c). Merancang Class Diagram

Class Diagram menggambarkan kelas-kelas yang akan dipakai oleh sistem. Umumnya terdapat 3 kelas pada setiap module yaitu class model, controller, dan view. Untuk sprint-1 setiap modul hanya terdapat 2 class yaitu model, dan

controller.



Gambar 4.3: Class Diagram Sprint-1

Pada **Gambar 4.3** setiap module terdapat 3 class. Semisal module Pond terdapat 1 class untuk model dan 2 class controller API. class model merupakan class yang mewarisi sifat `MongoEngine.Document` yang merupakan sebuah package pada python untuk mempermudah manipulasi database. Sedangkan class controller API merupakan class yang dibuat untuk mendeskripsikan logic yang akan dijalankan setiap client mengakses API Endpoint.

d). Implementasi model Pond, Feed_type, Feed_history

Implementasi model dilakukan menggunakan bantuan package `Mongoengine` yang mana mengharuskan menjadikan `Mongoengine.Document` sebagai parent dari class tersebut. Pada setiap atribut class harus didefinisikan terhadap jenis field yang telah disediakan oleh `Mongoengine`.

1). Model Pond

```

1 class Pond(db.Document):
2     id_int = db.SequenceField(required=True)
3     alias = db.StringField(required=True)
4     location = db.StringField(required=True)
5     created_at = db.DateTimeField(default=datetime.datetime.now)
6     updated_at = db.DateTimeField(default=datetime.datetime.now)
7

```

2). Model Feed_type

```

1 class FeedType(db.Document):
2     name = db.StringField(required=True)
3     feed_type = db.StringField(required=True)
4     protein = db.IntField(required=True)
5     carbohydrate = db.IntField(required=True)
6     desc = db.StringField()
7     created_at = db.DateTimeField(default=datetime.datetime.now)
8     updated_at = db.DateTimeField(default=datetime.datetime.now)
9

```

3). Model Feed_history

```

1 class FeedHistory(db.Document):
2     pond_id = db.ReferenceField(Pond, required=True)
3     feed_type_id = db.ReferenceField(FeedType, required=True)
4     feed_dose = db.IntField(required=True)
5     feed_history_time = db.DateTimeField(default=datetime.datetime.now)
6     created_at = db.DateTimeField(default=datetime.datetime.now)
7     updated_at = db.DateTimeField(default=datetime.datetime.now)
8

```

e). Implementasi Controller EndpointAPI

1). Add Routes

```

1 def initialize_routes(api):
2     # pond
3     api.add_resource(PondsApi, '/api/ponds')

```

```

4     api.add_resource(PondApi, '/api/ponds/<id>')
5

```

2). PondsApi Class PondsApi

```

1 class PondsApi(Resource):
2     def get(self):
3         objects = Pond.objects()
4         response = []
5         for pond in objects:
6             pond = pond.to_mongo()
7             response.append(pond)
8         response_dump = json.dumps(response, default=str)
9         return Response(response_dump, mimetype="application/json", status
=200)
10
11    def post(self):
12        body = {
13            "alias": request.form.get("alias", None),
14            "location": request.form.get("location", None),
15        }
16        pond = Pond(**body).save()
17        id = pond.id
18        return {'id': str(id)}, 200
19

```

3). PondApi

```

1 class PondApi(Resource):
2     def put(self, id):
3         body = {
4             "alias": request.form.get("alias", None),
5             "location": request.form.get("location", None),
6             "updated_at": datetime.datetime.utcnow()
7         }
8         Pond.objects.get(id=id).update(**body)
9         return '', 200
10
11    def delete(self, id):
12        pond = Pond.objects.get(id=id).delete()

```

```

13         return '', 200
14
15     def get(self, id):
16         objects = Pond.objects.get(id=id)
17         pond = objects.to_mongo()
18         response_dump = json.dumps(pond, default=str)
19         return Response(response_dump, mimetype="application/json", status
20 =200)

```

f). Deployment Sistem

Sistem pada akhirnya dijalankan pada server agar dapat diakses client melalui akses internet. Pertama yang dilakukan adalah mengunggah repositori sistem ke repositori github. Repositori github terdapat pada https://github.com/andrirahmanto/backendserverfishculture/tree/sprint_01. Lalu setelahnya adalah menghubungkan personal computer penulis ke server dengan metode SSH (Secure Shell Connection). Menghubungkan dengan metode ssh dilakukan melalui terminal pada dengan cara

```
1 ssh <user>@<ip_address_server>
```

dan masukan password ketika diminta.

Selanjutnya hubungkan server dengan repositori github yang sudah dibuat menggunakan metode git remote.

```

1 mkdir /var/www/html/<name_your_repo>
2 git init
3 git remote add origin <link_github>
4 git pull origin main

```

dengan begitu server telah memiliki repo yang sama dengan yang ada di personal komputer. Langkah selanjutnya diikuti dengan penginstalan package yang dipakai pada sistem.

g). Konfigurasi Sistem pada Server

Konfigurasi bertujuan untuk menjalankan sistem yang sudah di upload ke server.

Tahapan pertama konfigurasi adalah memastikan bahwa sistem berbentuk package agar dapat di import. kemudian buat file index.wsgi seperti dibawah ini

```

1 import logging
2 import sys
3
4 logging.basicConfig(stream=sys.stderr)
5 sys.path.insert(0, '/var/www/html/fishapi')
6
7 from fishapi import create_app
8 application = create_app()

```

index.wsgi nantinya akan didaftarkan ke file httpd.conf pada server agar sistem dapat dieksekusi oleh server.

```

1 WSGIDaemonProcess /fishapi python-path=/opt/rh/rh-python38/root/lib/python3.8/site-packages
2 WSGIProcessGroup /fishapi
3 WSGIApplicationGroup %{GLOBAL}
4 WSGIScriptAlias /fishapi /var/www/html/fishapi/index.wsgi
5 WSGIScriptReloading on
6 <Directory "/var/www/html/fishapi/fishapi">
7   AllowOverride All
8   Options +ExecCGI
9   AddHandler cgi-script .cgi .pl .py
10  Order allow,deny
11  allow from all
12 </Directory>
13
14 Alias /fishapi/static /var/www/html/fishapi/fishapi/static
15 <Directory /var/www/html/fishapi/fishapi/static>
16   AllowOverride All
17   Order allow,deny
18   allow from all
19 </Directory>

```

Setelahnya lakukan restart pada apache pada server dengan menjalankan:

```
1 systemctl restart http
```

Setelah konfigurasi selesai, sistem akan di test pada browser dengan melakukan request.

```
1 [
2   {
3     "_id": "625d7727a9a73e090c65cda6",
4     "feed_dose": 2500,
5     "feed_history_time": "2022-04-18 21:35:19.414000",
6     "created_at": "2022-04-18 21:35:19.414000",
7     "updated_at": "2022-04-18 21:35:19.414000",
8     "pond": {
9       "_id": "625d7026a9a73e090c65cda1",
10      "id_int": 2,
11      "alias": "kolom 2",
12      "location": "lantai 2",
13      "created_at": "2022-04-18 21:05:26.183000",
14      "updated_at": "2022-04-18 21:05:26.183000"
15    },
16    "feed_type": {
17      "_id": "625d74d2a9a73e090c65cda4",
18      "name": "Queen pelet",
19      "feed_type": "pelet",
20      "protein": 50,
21      "carbohydrate": 50,
22      "desc": "Queen pelet hitam"
23    }
24  }
25]
```

Gambar 4.4: Request Get pemberian pakan Postman

h). Dokumentasi API

Dokumentasi API dibuat menggunakan postman, yang mana dapat diakses melalui link <https://documenter.getpostman.com/view/11714934/UVysybzY>. berikut beberapa contoh request dan response yang diberikan server: request cURL:

```
1 curl --location --request GET 'http://jft.web.id/fishapi/api/ponds'
```

response json:

```
1 [
2   {
3     "_id": "625d7026a9a73e090c65cdal",
4     "id_int": 2,
```

```

5   "alias": "alpha",
6   "location": "blok 1",
7   "shape": "bundar",
8   "material": "beton",
9   "length": 0,
10  "width": 0,
11  "diameter": 1.4,
12  "height": 1,
13  "build_at": "2022-04-18 21:05:26.183000",
14  "image_name": "kolam_1655141767.jpeg",
15  "area": 1.5399999999999998,
16  "image_link": "http://127.0.0.1:5000/api/ponds/image/625
d7026a9a73e090c65cda1",
17  "volume": 1.5399999999999998
18 },
19 { "_id": "625d7033a9a73e090c65cda2",.....},
20 { "_id": "62a62163e445ffb9c5f746f3",.....},
21 { "_id": "62a955888911334402ddb3b3",.....},
22 { "_id": "62a9a466299f257e382a8295",.....}
23 ]

```

i). Unit Testing

Unit Testing dilakukan pada akhir sprint. Adapun hasil dari unit testing yang telah dilaksanakan dapat dilihat pada tabel di bawah ini:

Tabel 4.1: Unit Testing Sprint 1

No	Skenario Pengujian	Kesesuaian		Kesimpulan
		Sesuai	Tidak	
1	Pencatatan pemberian pakan	✓		Diterima
2	Merubah data pemberian pakan	✓		Diterima
3	Mendapatkan list data pemberian pakan	✓		Diterima
4	Mendapatkan detail data pemberian pakan	✓		Diterima
5	Menghapus data pemberian pakan	✓		Diterima

2. Sprint 2 Report

Berikut merupakan report dari sprint ke-2 yang dilakukan pada tanggal 1 juni -7 juni 2022.

Tabel 4.2: Sprint-2 backlog

No	Story	Task	Status
1	Create, Read, Updte, dan Delete untuk Pencatatan	Membuat view pakan bulanan	Completed
2		Membuat view pakan harian	Completed

Pemberian pakan

1. View pakan bulanan

No	Kolam	Lokasi	Total Pemberian	Total Dosis	Rata-rata Dosis	Detail Masa Budidaya
1	alpha	blok 3	93 X	181 gr	1.946236559139785 gr / day	<button>Detail</button>
2	beta	Blok timur	30 X	67 gr	2.2333333333333334 gr / day	<button>Detail</button>
3	charlie	blok C	0 X	0 gr	None gr / day	<button>Detail</button>
4	zeta	blok barat	0 X	0 gr	None gr / day	<button>Detail</button>

Gambar 4.5: View pakan bulanan

2. View pakan harian

Pilih Tanggal
2022-10-02

Search

Hasil Untuk Tanggal 2022-10-02

No	Kolam	Lokasi	Total Dosis Hari Ini	Pemberian Pakan Hari Ini
1	alpha	blok 3	4 gr	Waktu Dosis Pakan
				06:00:00 WIB 2 gr Tapioka
				12:00:00 WIB 1 gr Tapioka
				18:00:00 WIB 1 gr Kangkung

Show 10 entries Search: Showing 1 to 3 of 3 entries Previous Next

Gambar 4.6: View pakan harian

3. Sprint 3 Report

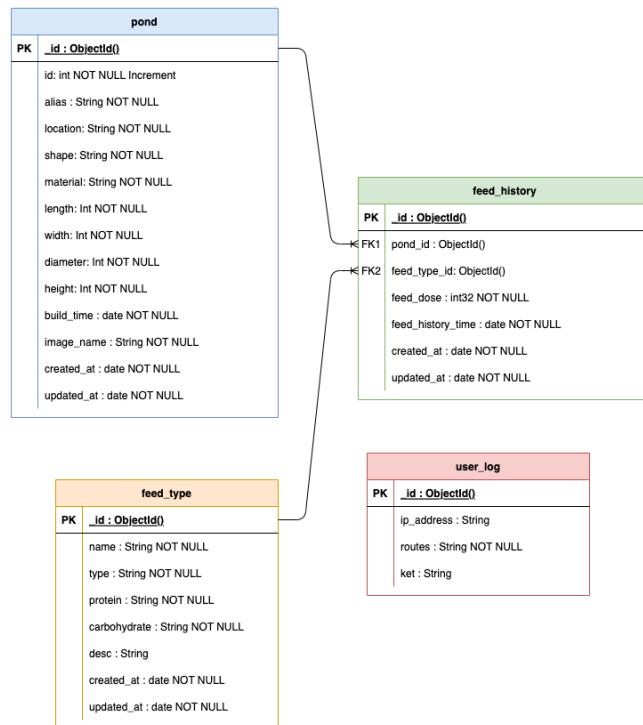
Berikut merupakan report dari sprint ke-3 yang dilakukan pada tanggal 8 juni

- 14 juni 2022.

Tabel 4.3: Sprint-3 backlog

No	Story	Task	Status
1	Create, Read, Update, dan Delete untuk Registrasi kolam	Membarui desain database	Completed
2		Membarui API entry kolam	Completed
3		Membarui API edit kolam	Completed
4		Membarui API fetch list kolam	Completed
5		Membuat API edit foto kolam	Completed
6		Membuat API fetch foto kolam	Completed
7		Membuat View detail kolam	Completed

1. Desain database

**Gambar 4.7: ERD Database Sprint-3**

Dengan berubahnya desain database diperlukan juga perubahan model pada source code, berikut perubahan pada source code model kolam.

```

1 # fishapi/database/model.py
2
3 class Pond(db.Document):
4     id_int = db.SequenceField(required=True)
5     alias = db.StringField(required=True)
6     location = db.StringField(required=True)
7     shape = db.StringField(required=True)
8     material = db.StringField(required=True)
9     length = db.FloatField(required=True, default=0)
10    width = db.FloatField(required=True, default=0)
11    diameter = db.FloatField(required=True, default=0)
12    height = db.FloatField(required=True, default=0)
13    image_name = db.StringField(required=True, default='default.jpg')
14    build_at = db.DateTimeField(default=datetime.datetime.now)
15    created_at = db.DateTimeField(default=datetime.datetime.now)
16    updated_at = db.DateTimeField(default=datetime.datetime.now)
```

2. Membarui API entry kolam

Perubahan terjadi pada controller API entry kolam, berikut merupakan perubahan source code controller API entry kolam.

```

1 # fishapi/resources/controller/pond.py
2
3 def post(self):
4     try:
5         body = {
6             "alias": request.form.get("alias", None),
7             "location": request.form.get("location", None),
8             "shape": request.form.get("shape", None),
9             "material": request.form.get("material", None),
10            "length": request.form.get("length", None),
11            "width": request.form.get("width", None),
12            "diameter": request.form.get("diameter", None),
13            "height": request.form.get("height", None),
14            "build_at": request.form.get("build_at", None),
15        }
```

```

16     pond = Pond(**body).save()
17
18     id = pond.id
19
20     response = {"message": "success add pond", "id": id}
21     response = json.dumps(response, default=str)
22
23     return Response(response, mimetype="application/json", status=200)
24
25 except Exception as e:
26
27     response = {"message": str(e)}
28
29     response = json.dumps(response, default=str)
30
31     return Response(response, mimetype="application/json", status=400)

```

Fungsi ini merupakan sebuah endpoint HTTP POST yang digunakan untuk menambahkan data kolam ke dalam database.

Pada bagian awal fungsi, `request.form.get` digunakan untuk membaca nilai dari setiap parameter dari `request` yang dikirim melalui `form-data`. Nilai-nilai ini kemudian digabungkan menjadi satu objek `body`.

Kemudian, objek `Pond` yang baru dibuat dengan parameter yang diambil dari objek `body` tersebut. Setelah itu, fungsi `"save()"` digunakan untuk menyimpan objek `Pond` ke dalam database.

Jika operasi penyimpanan berhasil dilakukan, fungsi akan memberikan respons dengan status kode 200 dan pesan "success add pond" beserta ID dari kolam yang baru saja ditambahkan. Jika terjadi kesalahan dalam proses ini, fungsi akan memberikan respons dengan status kode 400 dan pesan kesalahan dalam format JSON.

Fungsi `json.dumps()` digunakan untuk mengkonversi objek `response` ke dalam format JSON dan menentukan parameter `default=str` agar objek dapat di-serialize dalam format JSON. Terakhir, fungsi mengembalikan respons dengan header "application/json".

Berikut merupakan form untuk entry kolam.

Form	Jenis	Deskripsi
alias	REQUIRED STRING	alias/nama untuk kolam
location	REQUIRED STRING	lokasi kolam
shape	REQUIRED STRING VALUE : ['persegi', 'bundar']	bentuk kolam
material	REQUIRED STRING VALUE : ['tanah', 'terpal', 'beton']	bahan bangun kolam
length	REQUIRED DOUBLE FOR SHAPE ('persegi')	panjang kolam
width	REQUIRED DOUBLE FOR SHAPE ('persegi')	lebar kolam
diameter	REQUIRED DOUBLE FOR SHAPE ('bundar')	diameter kolam
height	REQUIRED DOUBLE	tinggi kolam
build_at	OPTIONAL STRING DATETIME { YYYY-mm-dd THH:MM:ss }	tanggal kolam dibuat

Tabel tersebut menjelaskan parameter-parameter yang dibutuhkan untuk membuat suatu kolam dalam suatu program atau sistem. Setiap baris di tabel tersebut menjelaskan satu parameter yang diperlukan untuk membuat kolam, yang terdiri dari tiga kolom:

- a. Form: kolom ini menjelaskan nama parameter yang dibutuhkan untuk membuat kolam
- b. Jenis: kolom ini menjelaskan jenis data yang diperlukan untuk nilai parameter tersebut, termasuk apakah data tersebut wajib atau opsional, dan jika wajib, tipe data apa yang harus digunakan.
- c. Deskripsi: kolom ini memberikan deskripsi singkat tentang arti parameter tersebut.

Dalam tabel tersebut, terdapat sembilan parameter yang dibutuhkan untuk membuat kolam, yaitu "alias", "location", "shape", "material", "length", "width", "diameter", "height", dan "build_at". Beberapa parameter adalah wajib, seperti "alias", "location", "shape", "material", dan "height", sedangkan beberapa parameter lainnya adalah opsional, seperti "build_at". Terdapat juga beberapa

parameter yang memerlukan jenis data khusus, seperti "shape" yang harus bernilai "persegi" atau "bundar", dan "build_at" yang harus merupakan datetime dalam format tertentu.

Berikut merupakan hasil test request dari API entry kolam.

cURL:

```

1 curl --location 'http://jft.web.id/fishapi/api/ponds' \
2 --form 'alias="epsilon"' \
3 --form 'location="blok 3"' \
4 --form 'shape="persegi"' \
5 --form 'material="tanah"' \
6 --form 'length="5"' \
7 --form 'width="3"' \
8 --form 'height="0.7"'

```

response json:

```

1 {
2     "message": "success add pond",
3     "id": "62aa118fa95cfc494c5a4a6"
4 }

```

3. Membarui API edit kolam

Perubahan terjadi pada controller API edit kolam, berikut merupakan perubahan source code controller API edit kolam.

```

1 # fishapi/resources/controller/pond.py
2
3 def put(self, id):
4     try:
5         body = request.form.to_dict(flat=True)
6         Pond.objects.get(id=id).update(**body)
7         response = {"message": "success change data pond", "id": id}
8         response = json.dumps(response, default=str)
9         return Response(response, mimetype="application/json", status=200)
10    except Exception as e:
11        response = {"message": str(e)}

```

```

12     response = json.dumps(response, default=str)
13
14     return Response(response, mimetype="application/json", status=400)
15

```

Fungsi ini merupakan sebuah endpoint HTTP PUT yang digunakan untuk mengubah data kolam yang sudah ada di dalam database.

Saat endpoint ini dipanggil, fungsi akan membaca input yang diberikan melalui form-data dan membuat objek body dengan nilai-nilai parameter tersebut.

Kemudian, fungsi "update()" pada objek Pond yang memiliki ID yang sesuai akan dipanggil dengan parameter nilai-nilai dari objek body tersebut. Fungsi ini akan mengubah nilai dari setiap parameter yang terkait dengan ID kolam tersebut di dalam database.

Jika operasi pengubahan berhasil dilakukan, fungsi akan memberikan respons dengan status kode 200 dan pesan "success change data pond" beserta ID dari kolam yang diubah. Jika terjadi kesalahan dalam proses ini, fungsi akan memberikan respons dengan status kode 400 dan pesan kesalahan dalam format JSON.

Fungsi json.dumps() digunakan untuk mengkonversi objek response ke dalam format JSON dan menentukan parameter default=str agar objek dapat di-serialize dalam format JSON. Terakhir, fungsi mengembalikan respons dengan header "application/json".

Berikut merupakan hasil test request yang dari API edit kolam.

cURL:

```

1 curl --location -g --request PUT 'http://jft.web.id/fishapi/api/ponds/{pond_id}'
2   \
3   --form 'material="terpal"'

```

response json:

```

1 {
2     "message": "success change data pond",
3     "id": "625d7033a9a73e090c65cda2"
4 }
```

4. Membarui API fetch list kolam

Perubahan terjadi pada controller API fetch list kolam, berikut merupakan perubahan source code controller API fetch list kolam.

```

1 # fishapi/resources/controller/pond.py
2
3 def get(self):
4     try:
5         url = url_for('pondimageapidummy', _external=True)
6         pipeline = [
7             {"$addFields": {
8                 "area": {"$cond": {
9                     "if": {"$eq": ["$shape", "persegi"]},
10                    "then": {"$multiply": ["$length", "$width"]},
11                    "else": {"$divide": [
12                        {"$multiply": [22, "$diameter", "$diameter"]},
13                        28
14                    ]}},
15                } },
16                "image_link": {"$concat": [url, "/", {"$toString": "$_id"}]}
17            } },
18            {"$addFields": {
19                "volume": {"$multiply": ["$area", "$height"]}}
20            } },
21            {"$project": {
22                "pond_id": 0,
23                "feed_type_id": 0,
24                "created_at": 0,
25                "updated_at": 0,
26            } }
27        ]
28        ponds = Pond.objects.aggregate(pipeline)
29        list_ponds = list(ponds)
```

```

30     response = json.dumps(list_ponds, default=str)
31
32     return Response(response, mimetype="application/json", status=200)
33 except Exception as e:
34     response = {"message": str(e)}
35
36     response = json.dumps(response, default=str)
37
38     return Response(response, mimetype="application/json", status=400)

```

Fungsi ini merupakan sebuah endpoint HTTP GET yang digunakan untuk mengambil data kolam yang sudah ada di dalam database.

Saat endpoint ini dipanggil, fungsi akan melakukan query terhadap database untuk mengambil data kolam yang sudah ada. Query ini dilakukan dengan menggunakan objek pipeline yang akan menambahkan field "area" dan "image_link" pada hasil query.

Field "area" akan diisi dengan nilai luas kolam yang dihitung berdasarkan shape, length, width, dan diameter kolam. Field "image_link" akan diisi dengan URL gambar kolam yang diambil dari endpoint "pondimageapidummy".

Setelah itu, field "volume" akan ditambahkan ke hasil query dengan menghitung nilai volume kolam berdasarkan field "area" dan "height".

Terakhir, hasil query akan diproyeksikan ke dalam objek yang hanya mengandung field-field tertentu saja seperti "alias", "location", "shape", "material", "length", "width", "diameter", "height", "area", "volume", dan "image_link".

Jika query berhasil dilakukan, fungsi akan memberikan respons dengan status kode 200 dan daftar kolam dalam format JSON. Jika terjadi kesalahan dalam proses ini, fungsi akan memberikan respons dengan status kode 400 dan pesan kesalahan dalam format JSON.

Berikut merupakan hasil test request yang dari API fetch list kolam.

cURL:

```
1 curl --location 'http://jft.web.id/fishapi/api/ponds'
```

response json:

```
1 [
2   {
3     "_id": "625d7026a9a73e090c65cda1",
4     "id_int": 2,
5     "alias": "alpha",
6     "location": "blok 1",
7     "shape": "bundar",
8     "material": "beton",
9     "length": 0,
10    "width": 0,
11    "diameter": 1.4,
12    "height": 1,
13    "build_at": "2022-04-18 21:05:26.183000",
14    "image_name": "kolam_1655141767.jpeg",
15    "area": 1.5399999999999998,
16    "image_link": "http://127.0.0.1:5000/api/ponds/image/625d7026a9a73e090c65cda1",
17    "volume": 1.5399999999999998
18  },
19  {
20    "_id": "625d7033a9a73e090c65cda2",
21    "id_int": 3,
22    "alias": "beta",
23    "location": "blok 1",
24    "shape": "persegi",
25    "material": "terpal",
26    "length": 8,
27    "width": 4,
28    "diameter": 0,
29    "height": 1,
30    "build_at": "2022-04-18 21:05:39.608000",
31    "image_name": "default.jpg",
32    "area": 32,
33    "image_link": "http://127.0.0.1:5000/api/ponds/image/625d7033a9a73e090c65cda2",
34    "volume": 32
```

```

35     },
36     { . . . .
37 ]

```

5. Menambahkan API edit foto kolam

Penambahan API untuk fungsi edit foto kolam, berikut merupakan penambahan source code controller API edit foto kolam.

```

1 # fishapi/resources/controller/pond.py
2
3 def put(self, id):
4     try:
5         file = request.files['image']
6         if not file:
7             response = {"message": "no file selected"}
8             response = json.dumps(response, default=str)
9             return Response(response, mimetype="application/json", status
=400)
10        if not allowed_file(file.filename):
11            response = {"message": "file type not allowed"}
12            response = json.dumps(response, default=str)
13            return Response(response, mimetype="application/json", status
=400)
14        filename = secure_filename(file.filename)
15        filename = pad_timestamp(filename)
16        path = os.path.join(current_app.instance_path,
17                            current_app.config['UPLOAD_DIR'])
18        try:
19            os.makedirs(path)
20        except OSError:
21            pass
22        filepath = os.path.join(path, filename)
23        file.save(filepath)
24        # database
25        objects = Pond.objects.get(id=id)
26        pond = objects.to_mongo()
27        old_image_name = pond["image_name"]
28        new_image_name = filename

```

```

29         if old_image_name != "default.jpg":
30             os.remove(os.path.join(path, old_image_name))
31
32         data = {
33             "image_name": new_image_name
34         }
35
36         objects.update(**data)
37
38         id = objects.id
39
40         response = {"message": "success change image", "id": id}
41
42         response = json.dumps(response, default=str)
43
44         return Response(response, mimetype="application/json", status=200)
45
46     except Exception as e:
47
48         response = {"message": str(e)}
49
50         response = json.dumps(response, default=str)
51
52         return Response(response, mimetype="application/json", status=400)

```

Kode tersebut merupakan implementasi dari method PUT pada API untuk mengganti gambar kolam ikan berdasarkan ID kolam. Pertama, API akan menerima request PUT yang berisi file gambar yang ingin diganti dan ID kolam yang ingin diubah gambarnya. Jika tidak ada file gambar yang dipilih, maka akan mengembalikan response "no file selected" dengan status code 400. Jika file yang dipilih tidak diperbolehkan, maka akan mengembalikan response "file type not allowed" dengan status code 400.

Selanjutnya, filename dari file gambar tersebut akan diproses dengan fungsi secure_filename dan pad_timestamp untuk memastikan nama file yang aman dan unik. Path untuk menyimpan file gambar akan dibuat, dan file akan disimpan di path tersebut.

Kemudian, API akan mengambil data kolam ikan berdasarkan ID yang diberikan dari database. Gambar lama dari kolam ikan tersebut akan dihapus jika bukan "default.jpg" dan digantikan dengan gambar baru yang telah dipilih.

Terakhir, API akan mengembalikan response "success change image" beserta ID kolam ikan yang diubah gambarnya dengan status code 200. Jika terjadi error pada

proses ini, maka akan mengembalikan response dengan pesan error dan status code 400.

Berikut merupakan hasil test request yang dari API edit foto kolam. Simulasi dibuat dengan mengisikan form 'image' dengan path image yang ada di direktori local user.

cURL:

```
1 curl --location -g --request PUT 'http://jft.web.id/fishapi/api/ponds/image/{
    pond_id}' \
2 --form 'image=@"/Users/andrirahmanto/Downloads/kolam.jpeg'"
```

response json:

```
1 {
2     "message": "success change image",
3     "id": "62a62163e445ffb9c5f746f3"
4 }
```

6. Membuat View detail kolam

No	Kolam	Bentuk	Dimensi	Tinggi	Luas	Volume	Material	Dibangun Pada
1	alpha	persegi	5.0 m X 3.0 m (PxL)	0.7 m	15.0 m ²	10.5 m ³	tanah	21-11-2022
2	beta	persegi	4.0 m X 2.0 m (PxL)	2.0 m	8.0 m ²	16.0 m ³	tanah	22-11-2022

Gambar 4.8: View detail kolam

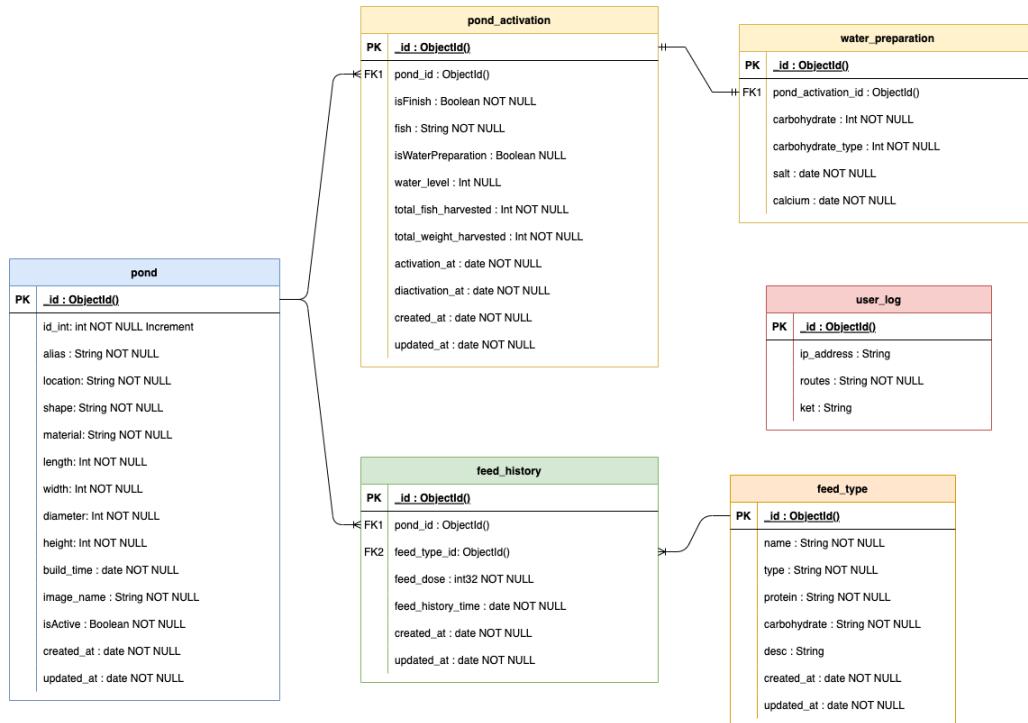
4. Sprint 4 Report

Berikut merupakan report dari sprint ke-4 yang dilakukan pada tanggal 15 juni - 21 juni 2022.

Tabel 4.4: *Sprint-4 backlog*

No	Story	Task	Status
1	Create, Read, Updte, dan Delete untuk Masa Budidaya Kolam	Membarui desain database	Completed
2		Implementasi API aktifasi kolam	Completed
3		Implementasi API deaktifasi kolam	Completed
4		Implementasi API fetch list status kolam	Completed
5		Implementasi API fetch musim budidaya per kolam	Completed
6		Membuat view status budidaya semua kolam	Completed
7		Membuat view list budidaya per kolam	Completed
8		Membuat view detail budidaya	Completed

1. Membarui desain database



Gambar 4.9: ERD Database Sprint-4

Dengan berubahnya desain database diperlukan juga penambahan model pada source code, berikut perubahan pada source code model.

```

1 # fishapi/database/model.py
2
3 class PondActivation(db.Document):
4     id_int = db.IntField(required=True)
5     pond_id = db.ReferenceField(Pond, required=True)
6     isFinish = db.BooleanField(required=True, default=False)
7     isWaterPreparation = db.BooleanField(required=True, default=False)
8     water_level = db.FloatField(required=True, default=0)
9     total_fish_harvested = db.IntField(required=True, default=0)
10    total_weight_harvested = db.IntField(required=True, default=0)
11    activated_at = db.DateTimeField(default=datetime.datetime.now)
12    deactivated_at = db.DateTimeField(default=None)
13    deactivated_description = db.StringField(default=None)
14    constanta_oversize = db.FloatField(required=True, default=1.3)
15    constanta_undersize = db.FloatField(required=True, default=0.7)

```

```

16     created_at = db.DateTimeField(default=datetime.datetime.now)
17     updated_at = db.DateTimeField(default=datetime.datetime.now)

```

2. Implementasi API aktifasi kolam

Perubahan terjadi pada controller API entry musim budidaya, berikut merupakan perubahan source code controller API entry musim budidaya.

```

1 # fishapi/database/pondactivation.py
2
3 class PondActivationApi(Resource):
4
5     def post(self, pond_id):
6
7         pond = Pond.objects.get(id=pond_id)
8
9         pipeline_year = {'$match': {'$expr': {'$and': [
10
11             {'$eq': ['$pond_id', '$toObjectId': pond_id]},
12             {'$eq': [{'$dateToString': {
13                 'format': "%Y", 'date': "$created_at"}}, getYearToday()
14             ()]}],
15
16             ]}}
17
18         list_pond_year = PondActivation.objects.aggregate(pipeline_year)
19
20         list_pond_year = list(list_pond_year)
21
22         id_int = len(list_pond_year) + 1
23
24         if pond.isActive == True:
25
26             response = {"message": "status pond is already active"}
27
28             response = json.dumps(response, default=str)
29
30             return Response(response, mimetype="application/json", status=400)
31
32         fishes = request.form.get("fish", "[]")
33
34         fishes = json.loads(fishes)
35
36         if len(fishes) < 1:
37
38             response = {"message": "There is no fish"}
39
40             response = json.dumps(response, default=str)
41
42             return Response(response, mimetype="application/json", status=400)
43
44         isWaterPreparation = request.form.get("isWaterPreparation", False)
45
46         if isWaterPreparation == "true":
47
48             isWaterPreparation = True
49
50         else:
51
52             isWaterPreparation = False
53
54         water_level = request.form.get("water_level", None)

```

```

31     activated_at = request.form.get(
32         "activated_at", datetime.datetime.now())
33     pond_activation_data = {
34         "id_int": id_int,
35         "pond_id": pond_id,
36         "isFinish": False,
37         "isWaterPreparation": isWaterPreparation,
38         "water_level": water_level,
39         "activated_at": activated_at
40     }
41     pondActivation = PondActivation(**pond_activation_data).save()
42     pondActivation_id = pondActivation.id
43     if isWaterPreparation == True:
44         carbohydrate = request.form.get("carbohydrate", None)
45         carbohydrate_type = request.form.get("carbohydrate_type", None)
46         salt = request.form.get("salt", None)
47         calcium = request.form.get("calcium", None)
48         water_preparation_data = {
49             "pond_activation_id": pondActivation_id,
50             "carbohydrate": carbohydrate,
51             "carbohydrate_type": carbohydrate_type,
52             "salt": salt,
53             "calcium": calcium,
54         }
55         water_preparation = WaterPreparation(
56             **water_preparation_data).save()
57         pond.update(**{"isActive": True})
58         for fish in fishes:
59             # save fish log
60             data = {
61                 "pond_id": pond_id,
62                 "pond_activation_id": pondActivation_id,
63                 "type_log": "activation",
64                 "fish_type": fish['type'],
65                 "fish_amount": fish['amount'],
66                 "fish_total_weight": fish['weight']
67             }
68             fishlog = FishLog(**data).save()
69             response = {"message": "success to activation pond"}
70             response = json.dumps(response, default=str)

```

```
71 |     return Response(response, mimetype="application/json", status=200)
```

Kode tersebut adalah sebuah fungsi dalam sebuah REST API yang bertujuan untuk mengaktifkan kolam ikan. Pada awal fungsi, dilakukan pencarian kolam ikan berdasarkan id kolam yang diberikan.

Selanjutnya, dilakukan operasi agregasi MongoDB untuk mencari data aktivasi pada tahun ini dengan menggunakan metode \$match dan memanfaatkan fungsi agregasi lainnya seperti \$and, \$eq, \$dateToString, dan getYearToday(). Hasil agregasi kemudian disimpan dalam variabel list_pond_year.

Selanjutnya, variabel id_int dihitung berdasarkan panjang dari list_pond_year ditambah 1. Kemudian, dilakukan beberapa validasi seperti memeriksa apakah kolam sudah aktif atau belum, dan apakah terdapat ikan yang akan dimasukkan ke dalam kolam.

Jika semua validasi telah dilakukan, data aktivasi kolam akan disimpan dalam database, lalu dilakukan pembaruan pada status kolam menjadi aktif. Data ikan yang dimasukkan ke dalam kolam juga akan disimpan dalam bentuk log pada tabel FishLog.

Terakhir, fungsi mengembalikan respons dalam format JSON yang berisi pesan berhasil atau gagalnya aktivasi kolam.

Berikut merupakan form untuk entry musim budidaya.

Tabel 4.5: Form entry musim budidaya kolam.

Form	Jenis Data	Deskripsi
fish	REQUIRED STRING FORMAT list(dict()) jenis ikan : ["nila merah", "nila hitam", "lele", "patin", "mas"] ex: "[{"lele": 100}, {"patin": 200}]"	jenis ikan dan jumlah ikan
isWaterPreparation	REQUIRED STRING BOOLEAN	variable bila aktifasi kolam menggunakan preparasi air
water_level	REQUIRED DOUBLE	ketinggian air dalam meter (m)
activated_at	OPTIONAL STRING DATE FORMAT : isodate()	timestamp aktifasi kolam
carbohydrate	REQUIRED INTEGER IF isWaterPreparation == True	banyak karbohidrat dalam (Gram/ml)
carbohydrate_type	REQUIRED STRING IF isWaterPreparation == True VALUE : ["gula", "molase", "trigu", "tapioka",]	jenis karbohidrat yang digunakan
salt	REQUIRED INTEGER IF isWaterPreparation == True	berat garam yang digunakan (Kg)
calcium	REQUIRED INTEGER IF isWaterPreparation == True	berat kapur yang digunakan (gram)

Tabel di atas berisi informasi tentang parameter yang harus diisi atau opsional

dalam aktivasi atau memulai musim budidaya kolam ikan. Berikut adalah penjelasan masing-masing kolom pada tabel tersebut:

- (a) Form: Merupakan nama parameter yang harus diisi atau opsional dalam aktivasi kolam ikan.
- (b) Jenis Data: Menjelaskan jenis data yang harus diisi pada parameter tersebut. Contohnya, string, boolean, integer, atau double.
- (c) Deskripsi: Berisi deskripsi singkat tentang informasi yang harus diisi pada parameter tersebut.

Beberapa contoh parameter yang harus diisi pada aktivasi kolam ikan di antaranya adalah jenis ikan dan jumlahnya, ketinggian air dalam meter, dan variable bila aktivasi kolam menggunakan preparasi air. Ada juga parameter yang opsional, seperti timestamp aktifasi kolam. Selain itu, ada parameter yang harus diisi jika variable bila aktivasi kolam menggunakan preparasi air diaktifkan, seperti banyak karbohidrat dalam gram/ml dan jenis karbohidrat yang digunakan. Ada juga parameter yang harus diisi jika menggunakan kapur atau garam dalam preparasi air, yaitu berat kapur dan berat garam yang digunakan. Semua parameter ini harus diisi dengan format yang sudah ditentukan pada kolom Jenis Data.

Berikut merupakan hasil test request dari API aktivasi kolam.

cURL:

```

1 curl --location -g 'http://jft.web.id/fishapi/api/ponds/{pond_id}/activation' \
2 --form 'fish=[{"lele": 100}, {"patin": 200}]"' \
3 --form 'isWaterPreparation="false"' \
4 --form 'water_level="100"'

```

response json:

```

1 {
2     "message": "success to activation pond"
3 }
```

3. Implementasi API deaktivasi kolam

Penambahan terjadi pada controller API entry musim budidaya, berikut merupakan perubahan source code controller API entry panen musim budidaya.

```

1 # fishapi/database/pondactivation.py
2
3 class PondDeactivationApi(Resource):
4
5     def post(self, pond_id):
6
7         pond = Pond.objects.get(id=pond_id)
8
9         if pond.isActive == False:
10
11             response = {"message": "status pond is already not active"}
12
13             response = json.dumps(response, default=str)
14
15             return Response(response, mimetype="application/json", status=400)
16
17         # get last pond_activation
18
19         pond_activation = PondActivation.objects(
20             pond_id=pond_id, isFinish=False).order_by('-activated_at').first()
21
22         fishes = request.form.get("fish", "[]")
23
24         fishes = json.loads(fishes)
25
26         total_fish_harvested = 0
27
28         total_weight_harvested = 0
29
30         for fish in fishes:
31
32             # save fish log
33
34             data = {
35
36                 "pond_id": pond_id,
37
38                 "pond_activation_id": pond_activation.id,
39
40                 "type_log": "deactivation",
41
42                 "fish_type": fish['type'],
43
44                 "fish_amount": fish['amount'],
45
46                 "fish_total_weight": fish['weight']
47
48             }
49
50             total_fish_harvested += fish['amount']
51
52             total_weight_harvested += fish['weight']
53
54             fishlog = FishLog(**data).save()
55
56             print(data)
```

```

31     print(total_fish_harvested)
32
33     print(total_weight_harvested)
34
35     # get args form data
36
37     # update pond_activation
38
39     pond_deactivation_data = {
40
41         "isFinish": True,
42
43         "total_fish_harvested": total_fish_harvested,
44
45         "total_weight_harvested": total_weight_harvested,
46
47         "deactivated_at": request.form.get("deactivated_at", datetime.
        datetime.now()),
48
49         "deactivated_description": "Normal"
50     }
51
52     pond_activation.update(**pond_deactivation_data)
53
54     # update pond isActive
55
56     pond.update(**{"isActive": False})
57
58     response = {"message": "success to deactivation pond"}
59
60     response = json.dumps(response, default=str)
61
62     return Response(response, mimetype="application/json", status=200)

```

Kode ini merupakan sebuah kelas Python dengan nama PondDeactivationApi yang diturunkan dari kelas Resource. Kelas ini memiliki satu method yaitu post() yang mengambil satu argumen yaitu pond_id.

Dalam method post(), dilakukan pengambilan data Pond dengan id tertentu yang diberikan sebagai argumen. Kemudian, jika atribut isActive pada objek Pond bernilai False, maka akan dikembalikan respons dengan pesan "status pond is already not active" dan status kode 400.

Selanjutnya, dilakukan pengambilan PondActivation terakhir yang memiliki pond_id tertentu dan isFinish bernilai False. Kemudian, dilakukan pengolahan data ikan dengan mengambil nilai dari form data "fish", yang kemudian di-parse dari string JSON menjadi list Python. Dalam loop for, dilakukan penyimpanan data FishLog untuk setiap ikan yang terdapat dalam list fishes.

Setelah itu, dilakukan pembaruan data pada objek PondActivation dengan

mengubah nilai atribut isFinish menjadi True dan menambahkan data mengenai total jumlah ikan yang dipanen (total_fish_harvested) dan total berat ikan yang dipanen (total_weight_harvested), serta waktu dan deskripsi pada saat deaktivasi. Selanjutnya, dilakukan pembaruan data pada objek Pond dengan mengubah nilai atribut isActive menjadi False.

Akhirnya, dilakukan pembuatan respons dengan pesan "success to deactivation pond" dan status kode 200. Pesan respons dan data lainnya kemudian di-serialize menjadi string JSON dengan menggunakan modul json dan dikembalikan dalam bentuk response dengan tipe konten "application/json".

Berikut merupakan form untuk entry panen musim budidaya.

Tabel 4.6: Form entry panen musim budidaya kolam.

Form	Jenis Data	Deskripsi
total_fish_harvested	REQUIRED INTEGER	banyak ikan yang di angkat
total_weight_harvested	REQUIRED INTEGER	berat seluruh ikan yang di angkat
diactived_at	OPTIONAL STRING DATE FORMAT : isodate() DEFAULT : now()	timestamp diaktifasi kolam

Tabel ini terdiri dari 3 baris, dimana setiap baris merepresentasikan sebuah parameter yang dapat digunakan dalam suatu API. Setiap baris terdiri dari 3 kolom yaitu:

- (a) Form : Merupakan nama parameter yang digunakan dalam API.
- (b) Jenis Data : Menjelaskan jenis data yang diterima oleh parameter tersebut.

Pada tabel ini, Jenis Data terdiri dari REQUIRED INTEGER dan

OPTIONAL STRING DATE FORMAT : isodate() DEFAULT : now().

- i. REQUIRED INTEGER artinya bahwa parameter tersebut wajib diisi dengan nilai berupa bilangan bulat (integer).
 - ii. OPTIONAL STRING DATE FORMAT : isodate() DEFAULT : now() artinya bahwa parameter tersebut bersifat opsional dan dapat diisi dengan nilai berupa string yang merepresentasikan tanggal dan waktu dalam format ISO (isodate), dengan nilai default saat tidak diisi adalah waktu saat ini.
- (c) Deskripsi : Menjelaskan secara singkat tentang parameter tersebut. Pada tabel ini, Deskripsi menjelaskan tentang banyak ikan yang diangkat, berat seluruh ikan yang diangkat, dan timestamp diaktifasi kolam.

Berikut merupakan hasil test request dari API deaktivasi/panen masa budidaya kolam.

cURL:

```
1 curl --location -g 'http://jft.web.id/fishapi/api/ponds/{pond_id}/diactivation' \
2 --form 'total_fish harvested="300" \
3 --form 'total_weight harvested="3000"
```

response json:

```
1 {
2   "message": "success to diactivation pond"
3 }
```

4. Implementasi API fetch list status kolam

Penambahan terjadi pada controller API fetch list status kolam, berikut merupakan perubahan source code controller API fetch list status kolam.

```
1 # fishapi/database/pondactivation.py
2
```

```

3 class PondsStatusApi(Resource):
4     def get(self):
5         pipeline = [
6             {'$lookup': {
7                 'from': 'pond_activation',
8                 'let': {"pondid": "$_id"},
9                 'pipeline': [
10                     {'$match': {'$expr': {'$and': [
11                         {'$eq': ['$pond_id', '$$pondid']},
12                     ]}}},
13                     {'$lookup': {
14                         'from': 'water_preparation',
15                         'let': {"pond_activation_id": "$_id"},
16                         'pipeline': [
17                             {'$match': {
18                                 '$expr': {'$eq': ['$pond_activation_id', '$$pond_activation_id']}},
19                             {"$project": {
20                                 "created_at": 0,
21                                 "updated_at": 0,
22                             }}}},
23                         ],
24                         'as': 'water_preparation'
25                     }},
26                     {"$addFields": {
27                         "water_preparation": {"$first": "$water_preparation"}
28                     }},
29                     {"$project": {
30                         "pond_id": 0,
31                         "feed_type_id": 0,
32                         "created_at": 0,
33                         "updated_at": 0,
34                     }}}},
35                 ],
36                 'as': 'pond_activation_list'
37             }},
38             {"$addFields": {
39                 "total_activation": {"$size": "$pond_activation_list"},
40             }},
41             {"$project": {

```

```

42         "location": 0,
43         "shape": 0,
44         "material": 0,
45         "length": 0,
46         "width": 0,
47         "diameter": 0,
48         "height": 0,
49         "image_name": 0,
50         "pond_activation_list": 0,
51         "updated_at": 0,
52         "created_at": 0,
53     } }
54 ]
55 ponds = Pond.objects().aggregate(pipeline)
56 response = list(ponds)
57 response = json.dumps(response, default=str)
58 return Response(response, mimetype="application/json", status=200)

```

Class ini memiliki sebuah method GET, yang berfungsi untuk mengambil data status kolam ikan dari database.

Method GET ini menggunakan sebuah pipeline pada MongoDB, yaitu pipeline yang terdiri dari beberapa tahap, yaitu:

- (a) Lookup: Melakukan join dengan tabel "pond_activation" berdasarkan _id kolam ikan pada tabel "pond". Join ini dilakukan menggunakan operasi \$lookup pada MongoDB.
- (b) Match: Melakukan filtering pada data yang telah di-join dengan tabel "pond_activation", dengan kondisi bahwa pond_id pada tabel "pond_activation" harus sama dengan _id pada tabel "pond".
- (c) Lookup: Melakukan join dengan tabel "water_preparation" berdasarkan pond_activation_id pada tabel "pond_activation". Join ini dilakukan menggunakan operasi \$lookup pada MongoDB.

- (d) Match: Melakukan filtering pada data yang telah di-join dengan tabel "water_preparation", dengan kondisi bahwa pond_activation_id pada tabel "water_preparation" harus sama dengan _id pada tabel "pond_activation".
- (e) Project: Mengembalikan data pada tabel "water_preparation" ke dalam variabel "water_preparation" pada tabel "pond_activation".
- (f) Project: Menghilangkan beberapa kolom pada tabel "pond_activation".
- (g) AddFields: Menambahkan kolom "total_activation" pada tabel "pond".
- (h) Project: Menghilangkan beberapa kolom pada tabel "pond".

Setelah pipeline tersebut dijalankan, hasilnya akan di-serialize menjadi format JSON, dan kemudian dikirimkan sebagai response dengan HTTP status code 200.

Berikut merupakan hasil test request dari API fetch list status kolam.

cURL:

```
1 curl --location 'http://jft.web.id/fishapi/api/ponds/status'
```

response json:

```

1 [
2   {
3     "_id": "625d7026a9a73e090c65cd1",
4     "id_int": 2,
5     "alias": "alpha",
6     "build_at": "2022-04-18 21:05:26.183000",
7     "isActive": true,
8     "total_activation": 2
9   },
10  {
11    "_id": "625d7033a9a73e090c65cd2",
12    "id_int": 3,
13    "alias": "beta",
14    "build_at": "2022-04-18 21:05:39.608000",
15    "isActive": false,
```

```

16     "total_activation": 0
17 },
18 {
19     "_id": "62a62163e445ffb9c5f746f3",
20     "id_int": 4,
21     "alias": "charlie",
22     "build_at": "2022-06-13 00:24:51.473000",
23     "isActive": false,
24     "total_activation": 0
25 },
26 {
27     "_id": "62a955888911334402ddb3b3",
28     "id_int": 5,
29     "alias": "delta",
30     "build_at": "2022-06-15 10:44:08.180000",
31     "isActive": false,
32     "total_activation": 0
33 },....
34 ]

```

5. Implementasi API fetch musim budidaya per kolam

Penambahan terjadi pada controller API fetch musim budidaya per kolam, berikut merupakan perubahan source code controller API musim budidaya per kolam.

```

1 # fishapi/database/pondactivation.py
2
3 class PondStatusApi(Resource):
4     def get(self, pond_id):
5         pond_objects = Pond.objects.get(id=pond_id)
6         pipeline = [
7             {'$match': {'$expr': {'$eq': ['$_id', {'$toObjectId': pond_id}]}}},
8             {'$lookup': {
9                 'from': 'pond_activation',
10                 'let': {"pondid": "$_id"},
11                 'pipeline': [
12                     {'$match': {'$expr': {'$and': [
13                         {'$eq': ['$pond_id', '$$pondid']},
14                     ]}}},

```

```

15      {"$sort": {"activated_at": -1}},
16      {'$lookup': {
17          'from': 'fish_log',
18          'let': {"pond_activation_id": "$_id"},
19          'pipeline': [
20              {'$match': {
21                  '$expr': {'$and': [
22                      {'$eq': ['$pond_activation_id', '$$pond_activation_id']},
23                      {'$eq': ['$type_log', 'activation']}
24                  ]}}
25              },
26              {"$project": {
27                  "created_at": 0,
28                  "updated_at": 0,
29              }},
30              {"$group": {
31                  "_id": "$fish_type",
32                  "fish_type": {"$first": "$fish_type"},
33                  "fish_amount": {"$sum": "$fish_amount"},
34                  "fish_total_weight": {"$sum": "$fish_total_weight"
35              }},
36              {"$sort": {"fish_type": -1}},
37              {"$project": {
38                  "_id": 0,
39              }},
40          ],
41          'as': 'fish_stock'
42      }},
43      {'$lookup': {
44          'from': 'fish_log',
45          'let': {"pond_activation_id": "$_id"},
46          'pipeline': [
47              {'$match': {
48                  '$expr': {'$and': [
49                      {'$eq': ['$pond_activation_id', '$$pond_activation_id']},
50                      {'$ne': ['$type_log', 'deactivation']}
51                  ]}}
52              },
53          ]
54      }
55  }
56
57  
```

```

54     } },
55     {"$project": {
56       "created_at": 0,
57       "updated_at": 0,
58     } },
59     {"$group": {
60       "_id": "$fish_type",
61       "fish_type": {"$first": "$fish_type"},
62       "fish_amount": {"$sum": "$fish_amount"}
63     } },
64     {"$sort": {"fish_type": -1}},
65     {"$project": {
66       "_id": 0,
67     } },
68   ],
69   'as': 'fish_live'
70 },
71 {'$lookup': {
72   'from': 'fish_log',
73   'let': {"pond_activation_id": "$_id"},
74   'pipeline': [
75     {'$match': {
76       '$expr': {'$and': [
77         {'$eq': ['$pond_activation_id',
78           '$$pond_activation_id']},
79         {'$eq': ['$type_log', 'death']}
80       ]}}
81     } },
82     {"$project": {
83       "created_at": 0,
84       "updated_at": 0,
85     } },
86     {"$group": {
87       "_id": "$fish_type",
88       "fish_type": {"$first": "$fish_type"},
89       "fish_amount": {"$sum": "$fish_amount"}
90     } },
91     {"$sort": {"fish_type": -1}},
92     {"$project": {
93       "_id": 0,

```

```

94         } },
95     ],
96     'as': 'fish_death'
97   } },
98   {'$lookup': {
99     'from': 'fish_log',
100    'let': {"pond_activation_id": "$_id"},
101    'pipeline': [
102      {'$match': {
103        '$expr': {'$and': [
104          {'$eq': ['$pond_activation_id',
105            '$$pond_activation_id']},
106          {'$eq': ['$type_log', 'deactivation']}
107        ] }
108      } },
109      {"$project": {
110        "created_at": 0,
111        "updated_at": 0,
112      } },
113      {"$group": {
114        "_id": "$fish_type",
115        "fish_type": {"$first": "$fish_type"},
116        "fish_amount": {"$sum": "$fish_amount"},
117        "fish_total_weight": {"$sum": "$fish_total_weight"
118      }},
119      {"$sort": {"fish_type": -1}},
120      {"$project": {
121        "_id": 0,
122      } },
123    ],
124    'as': 'fish harvested'
125  } },
126  {'$lookup': {
127    'from': 'feed_history',
128    'let': {"pond_activation_id": "$_id"},
129    'pipeline': [
130      {'$match': {
131        '$expr': {'$and': [
132          {'$eq': ['$pond_activation_id',

```

```

133                               '$$pond_activation_id']}},
134                         ] }
135                   } },
136                 ],
137                   'as': 'feed_history'
138             } },
139             {'$lookup': {
140               'from': 'water_preparation',
141               'let': {"pond_activation_id": "$_id"},
142               'pipeline': [
143                 {'$match': {
144                   '$expr': {'$eq': ['$pond_activation_id', '$$pond_activation_id']}},
145                   '$project': {
146                     "created_at": 0,
147                     "updated_at": 0,
148                   } }
149                 ],
150                   'as': 'water_preparation'
151             } },
152             {"$addFields": {
153               "water_preparation": {"$first": "$water_preparation"},

154               "total_fish": {"$sum": "$fish_live.fish_amount"},

155               "survival_rate": {"$cond": [
156                 {"$eq": [{"$sum": "$fish_stock.fish_amount"}, 0]},

157                 0,
158                 {"$multiply": [{"$divide": [{"$sum": "$fish_live.

fish_amount"}, {
159                   "$sum": "$fish_stock.fish_amount"}]}, 100]} }

160             ] },
161               "weight_growth": {"$subtract": [{"$sum": "$fish harvested

.fish_total_weight"}, {"$sum": "$fish_stock.fish_total_weight"}]},

162               "total_dose": {"$sum": "$feed_history.feed_dose"},

163               # "fcr": {"$sum": {"$divide": [{"$sum": "$fish_live.

fish_amount"}, {"$sum": "$fish_stock.fish_amount"}]}},

164             } },
165             {"$addFields": {
166               "fcr": {"$cond": [
167                 {"$eq": [{"$sum": "$total_dose"}, 0]},

168                 0,

```

```

169             {"$sum": {"$divide": [
170                 "$weight_growth", "$total_dose"]}}
171         ] },
172     } },
173     {"$project": {
174         "pond_id": 0,
175         "feed_history": 0,
176         "feed_type_id": 0,
177         "created_at": 0,
178         "updated_at": 0,
179     } }
180 ],
181     'as': 'pond_activation_list'
182 } },
183 {"$addFields": {
184     "total_activation": {"$size": "$pond_activation_list"},
185     "pond_activation_list": '$pond_activation_list',
186
187 } },
188 {"$project": {
189         "location": 0,
190         "shape": 0,
191         "material": 0,
192         "length": 0,
193         "width": 0,
194         "diameter": 0,
195         "height": 0,
196         "image_name": 0,
197         "updated_at": 0,
198         "created_at": 0,
199     } }
200 ]
201 ponds = Pond.objects().aggregate(pipeline)
202 ponds = list(ponds)
203 ponds = dict(ponds[0])
204 response = json.dumps(ponds, default=str)
205 return Response(response, mimetype="application/json", status=200)

```

Kode diatas adalah sebuah API untuk mengembalikan data status kolam ikan.

Ketika API ini diakses dengan GET request, akan mengembalikan data status dari kolam ikan dengan id yang diberikan sebagai parameter. Data tersebut meliputi informasi tentang stok ikan hidup, ikan yang mati, ikan yang dipanen, dan sebagainya.

Seluruh data status kolam ikan yang dibutuhkan diperoleh dengan satu query melalui pipeline. Pipeline tersebut terdiri dari beberapa operasi seperti \$match, \$lookup, \$group, dan \$addFields yang digunakan untuk menggabungkan data dari beberapa tabel berbeda dan melakukan pengolahan data.

Setelah data berhasil diperoleh melalui pipeline, data tersebut akan dikembalikan dalam format JSON sebagai response dari GET request pada API ini.

Berikut merupakan hasil test request dari API fetch musim budidaya per kolam.

cURL:

```
1 curl --location -g 'http://jft.web.id/fishapi/api/ponds/status/{pond_id}'
```

response json:

```
1 {
2     "_id": "625d7026a9a73e090c65cda1",
3     "id_int": 2,
4     "alias": "alpha",
5     "build_at": "2022-04-18 21:05:26.183000",
6     "isActive": true,
7     "pond_activation_list": [
8         {
9             "_id": "62af3d637cf22faa567235ad",
10            "isFinish": true,
11            "fish": [
12                {
13                    "lele": 100
14                },
15                {
16                    "patin": 200
17                }
18            ]
19        }
20    ]
21 }
```

```

18     ],
19     "isWaterPreparation": true,
20     "water_level": 100,
21     "total_fish_harvested": 300,
22     "total_weight_harvested": 3000,
23     "activated_at": "2022-06-19 22:14:43.952000",
24     "diactivated_at": "2022-06-19 23:10:03.469000",
25     "water_preparation": {
26       "_id": "62af3d637cf22faa567235ae",
27       "pond_activation_id": "62af3d637cf22faa567235ad",
28       "carbohydrate": 100,
29       "carbohydrate_type": "gula",
30       "salt": 100,
31       "calcium": 100
32     }
33   },
34   {
35     "_id": "62af4c24f6a3ffba25a6be6a",
36     "isFinish": false,
37     "fish": [
38       {
39         "lele": 100
40       },
41       {
42         "patin": 200
43       }
44     ],
45     "isWaterPreparation": false,
46     "water_level": 100,
47     "total_fish_harvested": 0,
48     "total_weight_harvested": 0,
49     "activated_at": "2022-06-19 23:17:40.501000"
50   }
51 ],
52 "total_activation": 2
53 }
```

6. Membuat view status budidaya semua kolam

No	Kolam	Total Masa Budidaya	Status	Action
1	Raiden	7 X	Active	<button>Detail</button>
2	Prometheus	7 X	Closed	<button>Detail</button>
3	Wendy	1 X	Closed	<button>Detail</button>
4	Mazda	2 X	Active	<button>Detail</button>
5	epsilon	0 X	Not Active	<button>Detail</button>
6	K1	0 X	Not Active	<button>Detail</button>
7	K1	0 X	Not Active	<button>Detail</button>
8	Alpha	0 X	Not Active	<button>Detail</button>

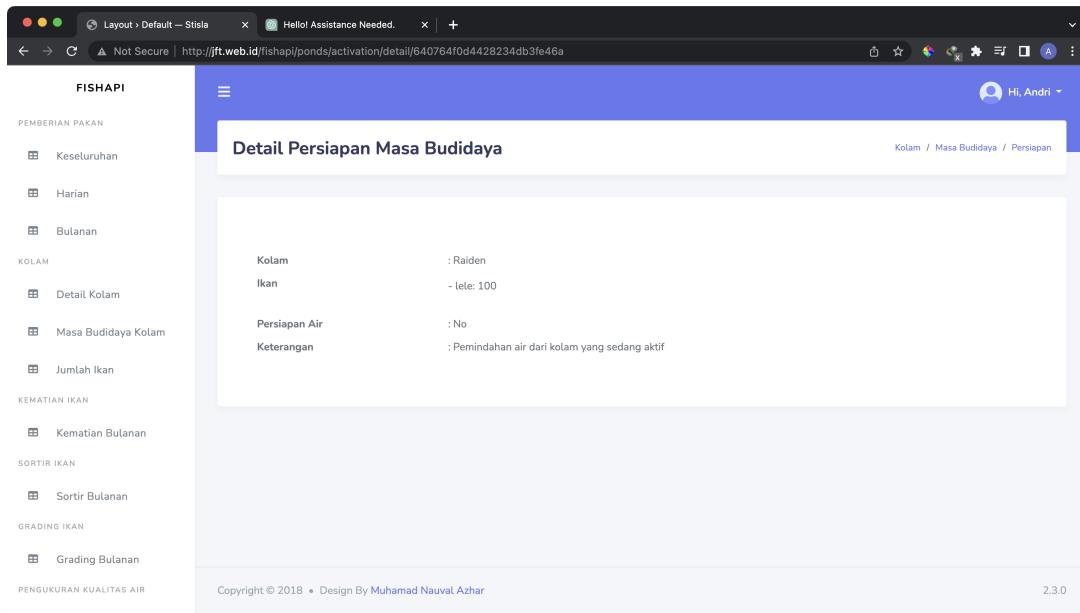
Gambar 4.10: View status kolam

7. Membuat view list budidaya per kolam

No	Status	Mulai Tanggal	Selesai Tanggal	Ikan	Jumlah Ikan Panen	Berat Ikan Panen	Detail Persiapan
1	Unfinish	07-03-2023	None	lele: 100	0	0 Kg	<button>Detail</button>
2	Finish	07-03-2023	07-03-2023	nila hitam: 100	0	20 Kg	<button>Detail</button>
3	Finish	28-02-2023	07-03-2023	nila hitam: 100	0	29 Kg	<button>Detail</button>

Gambar 4.11: View list musim budidaya per kolam

8. Membuat view detail budidaya



Gambar 4.12: View detail musim budidaya

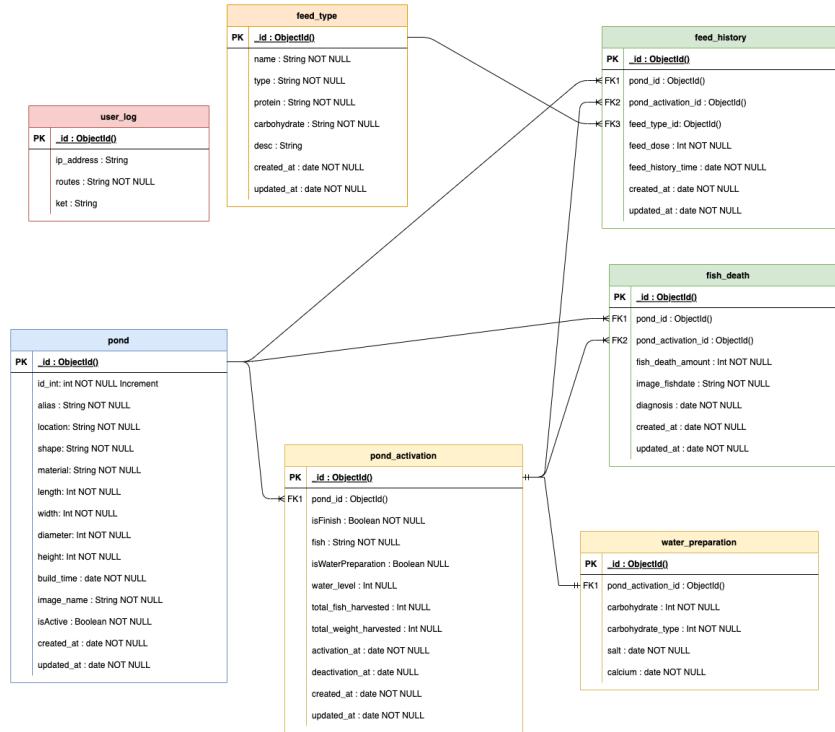
5. Sprint 5 Report

Berikut merupakan report dari sprint ke-5 yang dilakukan pada tanggal 22 juni - 28 juni 2022.

Tabel 4.7: Sprint-5 backlog

No	Story	Task	Status
1		Membarui desain database	Completed
2	Delete untuk Pencatatan data kematian ikan	Implementasi controller entry kematian ikan	Completed
3		Implementasi controller edit kematian ikan	Completed
4		Implementasi controller fetch list kematian ikan berdasarkan kolam	Next Spirnt
5		Membuat view rekap kematian ikan perbulan	Next Sprint

1. Membarui desain database



Gambar 4.13: ERD Database Sprint-5

Dengan berubahnya desain database diperlukan juga penambahan model pada source code, berikut perubahan pada source code model.

```

1 # fishapi/database/model.py
2
3 class FishDeath(db.Document):
4     pond_id = db.ReferenceField(Pond, required=True)
5     pond_activation_id = db.ReferenceField(PondActivation, required=True)
6     image_name = db.StringField(required=True)
7     diagnosis = db.StringField(default=datetime.datetime.now)
8     death_at = db.DateTimeField(default=datetime.datetime.now)
9     created_at = db.DateTimeField(default=datetime.datetime.now)
10    updated_at = db.DateTimeField(default=datetime.datetime.now)

```

2. Implementasi API entry kematian ikan

Implementasi controller API entry kematian ikan, berikut merupakan perubahan source code controller API entry kematian ikan.

```

1 # fishapi/database/fishdeath.py
2
3 class PondActivationApi(Resource):
4     def post(self):
5         try:
6             pond_id = request.form.get("pond_id", None)
7             pond = Pond.objects.get(id=pond_id)
8             if pond.isActive == False:
9                 response = {"message": "status pond is not active"}
10            response = json.dumps(response, default=str)
11            return Response(response, mimetype="application/json", status
12                           =400)
13            pond_activation = PondActivation.objects(
14                pond_id=pond_id, isFinish=False).order_by('-activated_at').first
15            ()
16            try:
17                file = request.files['image', None]
18                if not allowed_file(file.filename):
19                    response = {"message": "file type not allowed"}
20                    response = json.dumps(response, default=str)
21                    return Response(response, mimetype="application/json", status
22                           =400)
23                    filename = secure_filename(file.filename)
24                    filename = pad_timestamp(filename)
25                    path = os.path.join(current_app.instance_path,
26                                         current_app.config['UPLOAD_DIR'])
27                    try:
28                        os.makedirs(path)
29                    except OSError:
30                        pass
31                    filepath = os.path.join(path, filename)
32                    file.save(filepath)
33                    except:
34                        filename = "default.jpg"
35                        fish_death_amount = request.form.get("fish_death_amount", "[]")
36                        fish_death_amount = json.loads(fish_death_amount)
37                        if len(fish_death_amount) < 1:
38                            response = {"message": "There is no fish"}
39                            response = json.dumps(response, default=str)

```

```

37         return Response(response, mimetype="application/json", status
38 =400)
39
40     body = {
41         "pond_id": pond.id,
42         "pond_activation_id": pond_activation.id,
43         "image_name": filename,
44         "diagnosis": request.form.get("diagnosis", None),
45     }
46
47     fishdeath = FishDeath(**body).save()
48
49     id = fishdeath.id
50
51     for fish in fish_death_amount:
52
53         # save fish log
54
55         data = {
56             "pond_id": pond_id,
57             "pond_activation_id": pond_activation.id,
58             "fish_death_id": id,
59             "type_log": "death",
60             "fish_type": fish['type'],
61             "fish_amount": int(fish['amount']) * -1
62         }
63
64         fishlog = FishLog(**data).save()
65
66         response = {"message": "success add fishdeath"}
67
68         response = json.dumps(response, default=str)
69
70         return Response(response, mimetype="application/json", status=200)
71
72     except Exception as e:
73
74         response = {"message": str(e)}
75
76         response = json.dumps(response, default=str)
77
78         return Response(response, mimetype="application/json", status=400)

```

Kode ini adalah sebuah fungsi yang menangani permintaan POST yang dikirimkan ke server.

Pertama, fungsi ini mencoba untuk mengambil nilai "pond_id" dari permintaan yang diterima. Jika nilai ini tidak ada, maka nilai "pond_id" akan diatur menjadi None. Kemudian fungsi mencoba untuk mendapatkan objek Pond dari database dengan menggunakan nilai "pond_id" yang diperoleh sebelumnya.

Jika nilai isActive dari objek Pond adalah False, maka fungsi akan mengembalikan

pesan kesalahan dengan kode status 400. Jika nilai isActive adalah True, maka fungsi akan mencoba mendapatkan objek PondActivation dengan menggunakan nilai "pond_id" yang diperoleh sebelumnya. Objek PondActivation yang diperoleh akan memiliki isFinish=False dan diurutkan berdasarkan tanggal aktivasi terbaru.

Selanjutnya, fungsi akan mencoba untuk mendapatkan file gambar dari permintaan yang diterima. Jika jenis file tidak diizinkan, maka fungsi akan mengembalikan pesan kesalahan dengan kode status 400. Jika jenis file diizinkan, maka fungsi akan menyimpan file gambar di dalam direktori yang ditentukan.

Setelah itu, fungsi akan mencoba untuk mengambil nilai "fish_death_amount" dari permintaan yang diterima dan mengonversinya menjadi objek Python. Jika nilai "fish_death_amount" kosong, maka fungsi akan mengembalikan pesan kesalahan dengan kode status 400.

Kemudian, fungsi akan membuat objek FishDeath dengan menggunakan nilai-nilai yang diperoleh sebelumnya, dan menyimpannya ke dalam database. Fungsi juga akan membuat objek FishLog untuk setiap jenis ikan yang tercatat mati, dan menyimpannya ke dalam database.

Terakhir, fungsi akan mengembalikan pesan sukses dengan kode status 200, atau pesan kesalahan dengan kode status 400 jika terjadi kesalahan selama proses. Semua pesan yang dikirimkan dalam format JSON.

Terakhir, fungsi mengembalikan respons dalam format JSON yang berisi pesan berhasil atau gagalnya aktivasi kolam.

Berikut merupakan form untuk entry musim budidaya.

Tabel 4.8: Form entry kematian ikan.

Form	Jenis Data	Deskripsi
pond_id	REQUIRED STRING	id kolam yang mengalami kematian ikan
fish_death_amount	REQUIRED JSON FISH TYPE: ["nila hitam", "nila merah", "lele", "patin", "mas",] Ex: [{"type": "lele", "amount": 5}, {"type": "patin", "amount": 9}]	tipe dan banyak ikan
image	OPTIONAL FILE	foto kematian ikan
diagnosis	REQUIRED STRING	diagnosa kematian ikan

Tabel tersebut merupakan deskripsi dari jenis data yang diperlukan pada form untuk menambahkan data kematian ikan pada suatu kolam. Form ini meminta input berupa pond_id yang merupakan string yang mengacu pada id kolam yang mengalami kematian ikan. Selain itu, input fish_death_amount juga diperlukan dalam bentuk JSON dengan jenis data yang terdiri dari "nila hitam", "nila merah", "lele", "patin", dan "mas". Fish_death_amount menyimpan informasi tentang jenis dan banyaknya ikan yang mati. Input file image bersifat opsional dan meminta foto kematian ikan, sedangkan diagnosis yang diperlukan dalam bentuk string merujuk pada diagnosa kematian ikan.

Berikut merupakan hasil test request dari API entry kematian ikan.

cURL:

```

1 curl --location 'http://jft.web.id/fishapi/api/fishdeath' \
2 --form 'pond_id="625d7026a9a73e090c65cda1"' \

```

```

3 --form 'fish_death_amount="[{\"lele\": 10}, {"patin":20}]"' \
4 --form 'image=@"/Users/andrirahmanto/Downloads/Mass-fish-death-Menindee.jpeg"' \
5 --form 'diagnosis="mati karena sakit"'

```

response json:

```

1 {
2     "message": "success add fishdeath",
3     "id": "62b9adfa793e0f39dbaa1739"
4 }

```

3. Implementasi API edit kematian ikan

Implementasi controller API edit kematian ikan, berikut merupakan perubahan source code controller API edit kematian ikan.

```

1 # fishapi/database/fishdeath.py
2
3 def put(self, id):
4     try:
5         body = request.form.to_dict(flat=True)
6         FishDeath.objects.get(id=id).update(**body)
7         response = {"message": "success change data fish death", "id": id}
8         response = json.dumps(response, default=str)
9         return Response(response, mimetype="application/json", status=200)
10    except Exception as e:
11        response = {"message": str(e)}
12        response = json.dumps(response, default=str)
13        return Response(response, mimetype="application/json", status=400)
14    return

```

Kode ini merupakan fungsi untuk mengubah data kematian ikan pada database. Fungsi ini menggunakan HTTP method PUT dan menerima parameter id, yang merupakan id dari data kematian ikan yang ingin diubah.

Pada blok try, terdapat dua baris kode. Baris pertama mengubah data request.form menjadi dictionary yang disimpan dalam variabel body. Baris kedua memanggil

fungsi update pada model FishDeath dengan mengirimkan id dan dictionary body sebagai parameter.

Jika proses berhasil, akan mengembalikan response dengan status 200 dan message "success change data fish death" beserta id dari data yang diubah. Jika terjadi error, maka akan mengembalikan response dengan status 400 dan message error yang muncul pada variabel e yang diubah menjadi string.

Blok terakhir menggunakan return untuk mengakhiri fungsi.

Berikut merupakan form untuk edit kematian ikan.

Tabel 4.9: Form edit kematian ikan.

Form	Jenis Data	Deskripsi
pond_id	REQUIRED STRING	id kolam yang mengalami kematian ikan
fish_death_amount	OPTIONAL JSON FISH TYPE: ["nila hitam", "nila merah", "lele", "patin", "mas",] Ex: [{"type": "lele", "amount": 5}, {"type": "patin", "amount": 9}]	tipe dan banyak ikan
diagnosis	OPTIONAL STRING	diagnosa kematian ikan

Tabel ini adalah deskripsi dari form edit kematian ikan dan memiliki tiga kolom yaitu Form, Jenis Data, dan Deskripsi.

Form menjelaskan nama dari field input pada form, Jenis Data menjelaskan tipe data dari field input, sedangkan Deskripsi memberikan penjelasan singkat mengenai deskripsi dari field input tersebut.

Pada tabel ini, terdapat 3 field input yaitu pond_id, fish_death_amount, dan diagnosis. Kolom Jenis Data menunjukkan apakah field input tersebut merupakan tipe data REQUIRED STRING atau OPTIONAL JSON FISH TYPE dan Deskripsi memberikan penjelasan mengenai field input tersebut.

Field pond_id wajib diisi dan berisi id kolam yang mengalami kematian ikan, sedangkan field fish_death_amount dan diagnosis bersifat opsional. Field fish_death_amount berisi tipe dan banyak ikan yang mati dan ditulis dalam format JSON, sedangkan field diagnosis berisi diagnosa kematian ikan. Berikut merupakan hasil test request dari API entry kematian ikan.

cURL:

```

1 curl --location -g --request PUT 'http://jft.web.id/fishapi/api/fishdeath/{'
2   fishdeath_id}' \
3 --form 'fish_death_amount="[\\"lele\\": 10}, {\"patin\":20}]"' \
4 --form 'diagnosis="mati karena sakit"'
```

response json:

```

1 {
2   "message": "success change data fish death",
3   "id": "62b9adfa793e0f39dbaa1739"
4 }
```

6. Sprint 6 Report

Berikut merupakan report dari sprint ke-5 yang dilakukan pada tanggal 29 juni - 5 juli 2022.

Tabel 4.10: Sprint-6 backlog

No	Story	Task	Status
1		Implementasi controller delete kematian ikan	Complete
2		Membuat view rekap kematian ikan perbulan	Complete

1. Implementasi API fetch list kematian ikan berdasarkan kolam

Implementasi controller API fetch list kematian ikan berdasarkan kolam, berikut merupakan source code controller API fetch list kematian ikan berdasarkan kolam.

```

1 # fishapi/database/fishdeath.py
2
3 def get(self):
4     try:
5         url = url_for('fishdeathimageapidummy', _external=True)
6         pipeline = [
7             {'$lookup': {
8                 'from': 'pond_activation',
9                 'let': {"pondid": "$_id"},
10                'pipeline': [
11                    {'$match': {'$expr': {'$and': [
12                        {'$eq': ['$pond_id', '$$pondid']},
13                        {'$eq': ['$isFinish', False]}],
14                    ]}}},
15                    {"$lookup": {
16                        "from": "fish_death",
17                        "let": {"activationid": "$_id"},
18                        "pipeline": [
19                            {'$match': {'$expr': {'$and': [
20                                {'$eq': ['$pond_activation_id',
21                                    '$$activationid']},
22                            ]}}},
23                            {"$lookup": {
24                                'from': 'fish_log',

```

```

25      'let': {"fish_death_id": "$_id"},
26      'pipeline': [
27          {'$match': {
28              '$expr': {'$and': [
29                  {'$eq': ['$fish_death_id', '$$fish_death_id']},
30                  {'$eq': ['$type_log', 'death']}
31              ]}}
32          }},
33          {"$project": {
34              "created_at": 0,
35              "updated_at": 0,
36          }}}
37      ],
38      'as': 'fish'
39  },
40  {"$addFields": {
41      "image_link": {"$concat": [url, "/", {"$toString": "$_id"}]}
42  },
43      {"$project": {
44          "pond_id": 0,
45          "pond_activation_id": 0,
46          "created_at": 0,
47          "updated_at": 0,
48      }}}
49  ],
50      'as': "fish_death_list",
51  },
52  {"$project": {
53      "pond_id": 0,
54      "feed_type_id": 0,
55      "created_at": 0,
56      "updated_at": 0,
57  }}}
58  ],
59      'as': 'pond_activation_list'
60  },
61  {"$addFields": {
62

```

```

64         "pond_activation": {"$first": "$pond_activation_list"},  

65     }},  

66     {"$project": {  

67         "location": 0,  

68         "shape": 0,  

69         "material": 0,  

70         "length": 0,  

71         "width": 0,  

72         "diameter": 0,  

73         "height": 0,  

74         "image_name": 0,  

75         "pond_activation_list": 0,  

76         "updated_at": 0,  

77         "created_at": 0,  

78     }}  

79 ]  

80 ponds = Pond.objects.aggregate(pipeline)  

81 list_ponds = list(ponds)  

82 response = json.dumps(list_ponds, default=str)  

83 return Response(response, mimetype="application/json", status=200)  

84 except Exception as e:  

85     response = {"message": str(e)}  

86     response = json.dumps(response, default=str)  

87 return Response(response, mimetype="application/json", status=400)

```

Kode yang diberikan adalah sebuah method get yang ada di dalam sebuah class. Method ini menerima request GET untuk mengambil data dari MongoDB database.

Berikut adalah langkah-langkah yang dilakukan oleh method get:

- a. Membuat URL dari fishdeathimageapidummy dengan menggunakan url_for dan menyimpannya ke dalam variabel url.
- b. Membuat sebuah pipeline untuk melakukan aggregate query pada collection Pond.

- c. Pada pipeline tersebut, melakukan lookup terhadap collection pond_activation dengan menggunakan field _id dari collection Pond sebagai input.
- d. Pada hasil lookup pertama, mencocokkan nilai dari field isFinish yang bernilai False.
- e. Pada hasil lookup kedua, mencocokkan nilai dari field pond_activation_id yang sama dengan _id dari hasil lookup pertama.
- f. Pada hasil lookup ketiga, mencocokkan nilai dari field fish_death_id yang sama dengan _id dari hasil lookup kedua dan type_log yang bernilai death.
- g. Melakukan projeksi untuk mengambil field-field tertentu yang dibutuhkan dan mengecualikan beberapa field yang tidak dibutuhkan.
- h. Menyimpan hasil query dalam variabel ponds.
- i. Mengubah hasil query menjadi sebuah list dan mengubahnya menjadi JSON dengan menggunakan json.dumps.
- j. Membungkus hasil JSON kedalam sebuah response dengan menggunakan Response.
- k. Jika terdapat exception, maka akan membuat sebuah response dengan pesan error dan mengembalikan response tersebut.
- l. Inti dari kode ini adalah melakukan sebuah aggregate query pada collection Pond dengan menggunakan beberapa lookup dan projeksi untuk mengambil data yang dibutuhkan dan menyajikan hasil query dalam bentuk JSON.

Berikut merupakan hasil test request dari API get list kematian ikan.

cURL:

```
curl --location 'http://jft.web.id/fishapi/api/fishdeath'
```

response json:

```
1 [
2 {
3     "_id": "62a62163e445ffb9c5f746f3",
4     "id_int": 4,
5     "alias": "charlie",
6     "build_at": "2022-06-13 00:24:51.473000",
7     "isActive": false
8 },
9 {
10     "_id": "625d7033a9a73e090c65cda2",
11     "id_int": 3,
12     "alias": "beta",
13     "build_at": "2022-04-18 21:05:39.608000",
14     "isActive": true,
15     "pond_activation": {
16         "_id": "62b6b2f1a8a50041ee6350a4",
17         "isFinish": false,
18         "fish": [
19             {
20                 "lele": 100
21             },
22             {
23                 "patin": 200
24             }
25         ],
26         "isWaterPreparation": false,
27         "water_level": 100,
28         "total_fish_harvested": 0,
29         "total_weight_harvested": 0,
30         "activated_at": "2022-06-25 14:02:09.881000",
31         "fish_death_list": []
32     }
33 },
34 {
35     "_id": "625d7026a9a73e090c65cdaf",
36     "id_int": 2,
37     "alias": "alpha",
38     "build_at": "2022-04-18 21:05:26.183000",
39     "isActive": true,
```

```

40     "pond_activation": {
41         "_id": "62af4c24f6a3ffba25a6be6a",
42         "isFinish": false,
43         "fish": [
44             {
45                 "lele": 100
46             },
47             {
48                 "patin": 200
49             }
50         ],
51         "isWaterPreparation": false,
52         "water_level": 1.34,
53         "total_fish_harvested": 0,
54         "total_weight_harvested": 0,
55         "activated_at": "2022-06-19 23:17:40.501000",
56         "fish_death_list": [
57             {
58                 "_id": "62b9adfa793e0f39dbaa1739",
59                 "fish_death_amount": [
60                     {
61                         "lele": 10
62                     },
63                     {
64                         "patin": 20
65                     }
66                 ],
67                 "image_name": "Mass-fish-death-Menindee_1656335866.jpeg",
68                 "diagnosis": "mati karena sakit",
69                 "image_link": "http://127.0.0.1:5000/api/fishdeath/image/62
b9adfa793e0f39dbaa1739"
70             }
71         ]
72     }
73 },
74 {
75     "_id": "62a955888911334402ddb3b3",
76     "id_int": 5,
77     "alias": "delta",
78     "build_at": "2022-06-15 10:44:08.180000",

```

```

79      "isActive": false
80  },
81  {
82    "_id": "62ada3ff1dc3a711668e7ca3",
83    "id_int": 7,
84    "alias": "epsilon",
85    "isActive": false,
86    "build_at": "2022-06-18 17:07:59.396000"
87  }
88 ]

```

2. Membuat view rekap kematian ikan perbulan

The screenshot shows a web browser window for the FISHAPI application. The URL is <http://jft.web.id/fishapi/fishdeaths/2022-11>. The page title is "Kematian Ikan". On the left, there is a sidebar with navigation links for PEMBERIAN PAKAN, KOLAM, KEMATIAN IKAN, SORTIR IKAN, GRADING IKAN, and PENGUKURAN KUALITAS AIR. The main content area displays a search form with a dropdown for "Pilih Bulan" set to "2022-11" and a green "Search" button. Below the search form, the results for "Hasil untuk Bulan November 2022" are shown in a table. The table has columns: No, Tanggal, Waktu, Kolam, Jumlah Kematian, Diagnosis, and Foto. There is one entry: No 1, Tanggal 10-11-2022, Waktu 01:02 WIB, Kolam alpha, Jumlah Kematian lele: -1, Diagnosis mati karena sakit, and a thumbnail for the Foto. At the bottom of the table, it says "Showing 1 to 1 of 1 entries".

Gambar 4.14: View list kematian perbulan

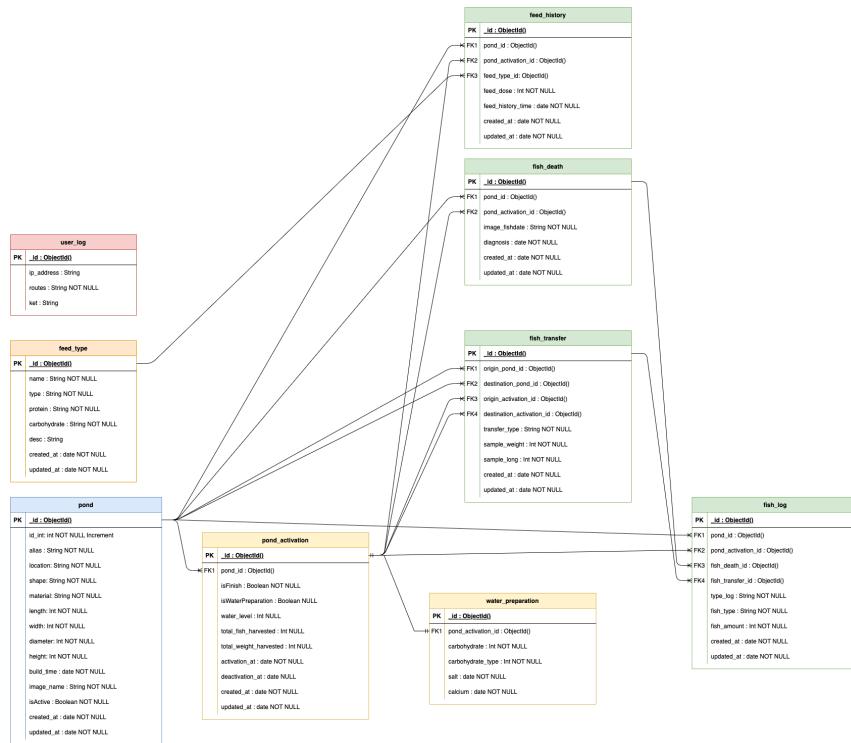
7. Sprint 7 Report

Berikut merupakan report dari sprint ke-7 yang dilakukan pada tanggal 6 juni - 12 juni 2022.

Tabel 4.11: *Sprint-7 backlog*

No	Story	Task	Status
1	Create, Read, Update, dan Delete untuk Pencatatan perpindahan ikan	Membarui desain database	Completed
2		Menambahkan routes API	Completed
3		Implementasi controller entry perpindahan ikan	Completed
4		Implementasi controller fetch list perpindahan ikan	Completed
5		Implementasi controller edit perpindahan ikan	Completed
6		Implementasi controller delete perpindahan ikan	Completed
7		Implementasi controller fetch perpindahan ikan dengan id	Completed
8		Membuat view rekap perpindahan ikan	Completed

1. Membarui desain database



Gambar 4.15: ERD Database Sprint-7

Dengan berubahnya desain database diperlukan juga penambahan model pada source code, berikut perubahan pada source code model.

```

1 # fishapi/database/model.py
2
3 class FishTransfer(db.Document):
4     origin_pond_id = db.ReferenceField(Pond, required=True)
5     destination_pond_id = db.ReferenceField(Pond, required=True)
6     origin_activation_id = db.ReferenceField(PondActivation, required=True)
7     destination_activation_id = db.ReferenceField(
8         PondActivation, required=True)
9     transfer_type = db.StringField(required=True)
10    sample_weight = db.IntField(required=True)
11    sample_long = db.IntField(required=True)
12    created_at = db.DateTimeField(default=datetime.datetime.now)
13    updated_at = db.DateTimeField(default=datetime.datetime.now)

```

2. Menambahkan routes API

```

1 # fishapi/resource/routes.py
2
3 # fish transfer
4     api.add_resource(FishTransfersApi, '/api/fishtransfer')
5     api.add_resource(FishTransferApi, '/api/fishtransfer/<id>')

```

3. Implementasi controller entry perpindahan ikan

Implementasi controller API entry perpindahan ikan, berikut merupakan perubahan source code controller API entry perpindahan ikan.

```

1 # fishapi/database/fishtransfer.py
2
3 class FishTransfersApi(Resource):
4
5     def post(self):
6
7         try:
8
9             origin_pond_id = request.form.get("origin_pond_id", None)
10            origin_pond = Pond.objects.get(id=origin_pond_id)
11
12            if origin_pond.isActive == False:
13
14                response = {"message": "status pond is not active"}
15
16                response = json.dumps(response, default=str)
17
18                return Response(response, mimetype="application/json", status
19
20                =400)
21
22            origin_activation = PondActivation.objects(
23
24                pond_id=origin_pond_id, isFinish=False).order_by('-activated_at')
25
26                .first()
27
28            destination_pond_id = request.form.get("destination_pond_id", None)
29            destination_pond = Pond.objects.get(id=destination_pond_id)
30
31            if destination_pond.isActive == False:
32
33                response = {"message": "status pond is not active"}
34
35                response = json.dumps(response, default=str)
36
37                return Response(response, mimetype="application/json", status
38
39                =400)
40
41            destination_activation = PondActivation.objects(
42
43                pond_id=destination_pond_id, isFinish=False).order_by('-
44                activated_at').first()
45
46            fish_grading_id = request.form.get("fish_grading_id", None)
47            transfer_type = request.form.get("transfer_type", None)
48            transfer_method = request.form.get("transfer_method", None)

```

```

25     sample_weight = request.form.get("sample_weight", None)
26     sample_long = request.form.get("sample_long", None)
27     fishes = request.form.get("fish", "[]")
28     fishes = json.loads(fishes)
29     if len(fishes) < 1:
30         response = {"message": "There is no fish"}
31         response = json.dumps(response, default=str)
32         return Response(response, mimetype="application/json", status
=400)
33
34     # if transfer method is "kering" deactived pond
35     if transfer_method == "kering":
36         # update activation
37         pond_deactivation_data = {
38             "isFinish": True,
39             "total_fish_harvested": request.form.get("total_fish_harvested", None),
40             "total_weight_harvested": request.form.get("total_weight_harvested", None),
41             "deactivated_at": request.form.get("deactivated_at", datetime
.datetime.now()),
42             "deactivated_description": "sortir kering"
43         }
44         origin_activation.update(**pond_deactivation_data)
45         pond.update(**{"isActive": False})
46
47         # update fish grading [optional]
48         if fish_grading_id and transfer_method != "kering":
49             # pengecekan fish_grading
50             fish_grading = FishGrading.objects.get(id=fish_grading_id)
51             # update fishgrading
52             if transfer_type == "oversized_transfer":
53                 fish_grading.update(**{"isOversizeTransferred": True})
54             else:
55                 fish_grading.update(**{"isUndersizeTransferred": True})
56
57         # save data
58         data = {
59             "origin_pond_id": origin_pond_id,
60             "destination_pond_id": destination_pond_id,
61             "origin_activation_id": origin_activation.id,
62             "destination_activation_id": destination_activation.id,
63             "fish_grading_id": None if transfer_method == "kering" else

```

```

    fish_grading_id,
61        "transfer_type": None if transfer_method == "kering" else
62        transfer_type,
63        "transfer_method": transfer_method,
64        "transfer_type": transfer_type,
65        "sample_weight": sample_weight,
66        "sample_long": sample_long
67    }
68
69    fish_transfer = FishTransfer(**data).save()
70
71    # transfer out
72
73    for fish in fishes:
74
75        # save fish log
76
77        data = {
78            "pond_id": origin_pond_id,
79            "pond_activation_id": origin_activation.id,
80            "fish_transfer_id": fish_transfer.id,
81            "type_log": "transfer_out",
82            "fish_type": fish['type'],
83            "fish_amount": fish['amount'] * -1
84        }
85
86        fishlog = FishLog(**data).save()
87
88    # transfer in
89
90    for fish in fishes:
91
92        # save fish log
93
94        data = {
95            "pond_id": destination_pond_id,
96            "pond_activation_id": destination_activation.id,
97            "fish_transfer_id": fish_transfer.id,
98            "type_log": "transfer_in",
99            "fish_type": fish['type'],
100           "fish_amount": fish['amount']
101       }
102
103       fishlog = FishLog(**data).save()
104
105       response = {"message": "success add fishdeath"}
106
107       response = json.dumps(response, default=str)
108
109       return Response(response, mimetype="application/json", status=200)
110
111   except Exception as e:
112
113       response = {"message": str(e)}
114
115       response = json.dumps(response, default=str)
116
117       return Response(response, mimetype="application/json", status=400)

```

Pada awalnya, kode tersebut melakukan beberapa hal berikut:

- (1) Menerima data dari permintaan POST yang dikirim oleh klien.
- (2) Memeriksa status kolam asal (origin_pond) menggunakan origin_pond_id. Jika kolam tidak aktif, maka akan dikembalikan respons dengan pesan "status kolam tidak aktif".
- (3) Mengambil aktivasi kolam asal yang belum selesai (isFinish=False) berdasarkan origin_pond_id dan diurutkan berdasarkan waktu aktivasi terakhir (activated_at).
- (4) Memeriksa status kolam tujuan (destination_pond) menggunakan destination_pond_id. Jika kolam tidak aktif, maka akan dikembalikan respons dengan pesan "status kolam tidak aktif".
- (5) Mengambil aktivasi kolam tujuan yang belum selesai (isFinish=False) berdasarkan destination_pond_id dan diurutkan berdasarkan waktu aktivasi terakhir (activated_at).

Selanjutnya, kode tersebut melakukan beberapa tugas lainnya:

- (1) Mengambil data seperti fish_grading_id, transfer_type, transfer_method, sample_weight, sample_long, dan fishes dari permintaan POST.
- (2) Memeriksa apakah terdapat ikan dalam fishes. Jika tidak ada ikan, maka akan dikembalikan respons dengan pesan "Tidak ada ikan".
- (3) Jika metode transfer adalah "kering", maka akan dilakukan deaktivasi kolam asal. Data aktivasi kolam akan diperbarui dengan mengatur isFinish menjadi True dan mengisi informasi terkait seperti jumlah ikan yang dipanen, bobot total yang dipanen, waktu deaktivasi, dan deskripsi deaktivasi.

- (4) Jika terdapat fish_grading_id dan metode transfer bukan "kering", maka akan dilakukan pembaruan pada penilaian ikan. Jika transfer_type adalah "oversized_transfer", maka isOversizeTransferred akan diatur sebagai True. Jika tidak, isUndersizeTransferred akan diatur sebagai True.
- (5) Data yang diperoleh akan disimpan dalam variabel data untuk digunakan nanti.
- (6) Dilakukan penyimpanan data transfer ikan menggunakan model FishTransfer.
- (7) Dilakukan transfer ikan keluar (transfer out) dan transfer ikan masuk (transfer in) untuk setiap ikan dalam fishes. Untuk setiap ikan, data log ikan akan disimpan menggunakan model FishLog.
- (8) Setelah semua operasi selesai, respons dengan pesan "success add fishdeath" akan dikirim kembali kepada klien.

Jika terjadi kesalahan selama proses tersebut, akan ditangkap oleh blok except dan respons dengan pesan kesalahan yang sesuai akan dikirim kembali kepada klien.

Berikut merupakan form untuk entry perpindahan ikan antar kolam.

Tabel 4.12: Form entry kematian ikan.

Form	Jenis Data	Deskripsi
origin_pond_id	REQUIRED STRING	id kolam asal ikan yang akan di transfer
destination_pond_id	REQUIRED STRING	id kolam tujuan ikan yang akan di transfer

Lanjutan Tabel 4.12		
Form	Jenis Data	Deskripsi
fish	REQUIRED JSON FISH TYPE: ["nila hitam", "nila merah", "lele", "patin", "mas",] Ex: [{"type": "lele", "amount": 5}, {"type": "patin", "amount": 9}]	tipe dan banyak ikan
transfer_method	REQUIRED STRING TYPE: ["kering", "basah"]	metode transfer ikan, jika kering kolam akan dianggap panen sekaligus
sample_weight	REQUIRED INT	sample berat ikan yang dipindahkan
sample_long	REQUIRED INT	sample panjang ikan yang dipindahkan
transfer_type	REQUIRED STRING TYPE: ["oversized_transfer", "undersized_transfer"]	tipe transfer, "oversized_transfer" adalah perpindahan yang dilakukan karena ikan terlalu besar dari pada ikan yang ada di kolam asal, sedangkan "undersized_transfer" adalah kebalikannya"

Lanjutan Tabel 4.12			
Form	Jenis Data	Deskripsi	
total_fish harvested	REQUIRED "transfer_method" "kering" INT	IF IS	total ikan yang dipanen bila "transfer_method" adalah "kering"
total_weight harvested	REQUIRED "transfer_method" "kering" INT	IF IS	total berat ikan yang dipanen bila "transfer_method" adalah "kering"

Berikut adalah penjelasan rinci dari setiap kolom dalam tabel tersebut:

- (a) "origin_pond_id": Variabel yang diperlukan, berupa string yang mengidentifikasi ID kolam asal ikan yang akan ditransfer.
- (b) "destination_pond_id": Variabel yang diperlukan, berupa string yang mengidentifikasi ID kolam tujuan ikan yang akan ditransfer.
- (c) "fish": Variabel yang diperlukan, berupa JSON yang mengandung tipe dan jumlah ikan yang akan ditransfer. Contoh format JSON: ["type": "lele", "amount": 5, "type": "patin", "amount": 9]. Menggambarkan jenis dan jumlah ikan yang akan ditransfer.
- (d) "transfer_method": Variabel yang diperlukan, berupa string yang mengindikasikan metode transfer ikan. Jika nilainya "kering", itu berarti kolam akan dianggap sedang dipanen secara keseluruhan. Jika nilainya "basah", maka transfer akan dilakukan dengan metode lain.
- (e) "sample_weight": Variabel yang diperlukan, berupa bilangan bulat (integer) yang mewakili berat sampel ikan yang akan ditransfer.

- (f) "sample_long": Variabel yang diperlukan, berupa bilangan bulat (integer) yang mewakili panjang sampel ikan yang akan ditransfer.
- (g) "transfer_type": Variabel yang diperlukan, berupa string yang menentukan tipe transfer. Nilainya bisa "oversized_transfer" jika ikan yang ditransfer terlalu besar dibandingkan dengan ikan di kolam asal, atau "undersized_transfer" jika ikan yang ditransfer terlalu kecil dibandingkan dengan ikan di kolam asal.
- (h) "total_fish_harvested": Variabel yang diperlukan jika "transfer_method" adalah "kering", berupa bilangan bulat (integer) yang mewakili total jumlah ikan yang dipanen.
- (i) "total_weight_harvested": Variabel yang diperlukan jika "transfer_method" adalah "kering", berupa bilangan bulat (integer) yang mewakili total berat ikan yang dipanen.

Berikut merupakan hasil test request dari API entry perpindahan ikan antar kolam.

cURL:

```

1 curl --location 'http://jft.web.id/fishapi/api/fishtransfer' \
2 --form 'origin_pond_id="62a62163e445ffb9c5f746f3"' \
3 --form 'destination_pond_id="625d7033a9a73e090c65cda2"' \
4 --form 'fish="[{\"type\": \"lele\", \"amount\": 30}, {\"type\": \"patin\", \"amount\":
5 \"\": 10}]"' \
6 --form 'sample_weight="20"' \
--form 'sample_long="50"'

```

response json:

```

1 {
2   "message": "success add fishtransfer"
3 }

```

4. Implementasi API fetch list perpindahan ikan

Implementasi controller API fetch list perpindahan ikan, berikut merupakan source code controller API fetch list perpindahan ikan.

```

1 # fishapi/resource/fishtransfer.py
2
3 def get(self):
4     try:
5         pipeline = [
6             {'$lookup': {
7                 'from': 'pond',
8                 'let': {"pondid": "$origin_pond_id"},
9                 'pipeline': [
10                     {'$match': {
11                         '$expr': {'$eq': ['$id', '$$pondid']}},
12                     {"$project": {
13                         "created_at": 0,
14                         "updated_at": 0,
15                     }})
16                 ],
17                 'as': 'origin_pond'
18             }},
19             {'$lookup': {
20                 'from': 'pond',
21                 'let': {"pondid": "$destination_pond_id"},
22                 'pipeline': [
23                     {'$match': {
24                         '$expr': {'$eq': ['$id', '$$pondid']}},
25                     {"$project": {
26                         "created_at": 0,
27                         "updated_at": 0,
28                     }})
29                 ],
30                 'as': 'destination_pond'
31             }},
32             {"$addFields": {
33                 "origin_pond": {"$first": "$origin_pond"},
34                 "destination_pond": {"$first": "$destination_pond"},
35             }},
36             {'$lookup': {
37                 'from': 'fish_log',

```

```

38     'let': {"fish_transfer_id": "$_id"},  

39     'pipeline': [  

40         {'$match': {  

41             '$expr': {'$and': [  

42                 {'$eq': ['$fish_transfer_id',  

43                     '$$fish_transfer_id']},  

44                 {'$eq': ['$type_log',  

45                     'transfer_out']}],  

46             ]}  

47         },  

48         {"$project": {  

49             "created_at": 0,  

50             "updated_at": 0,  

51         }}  

52     ],  

53     'as': 'fish'  

54 },  

55 {"$project": {  

56     "updated_at": 0,  

57     "created_at": 0,  

58 }}  

59  

60 ]
61 fishtransfers = FishTransfer.objects.aggregate(pipeline)
62 fishtransfers = list(fishtransfers)
63 response = json.dumps(fishtransfers, default=str)
64 return Response(response, mimetype="application/json", status=200)
65 except Exception as e:
66     response = {"message": str(e)}
67     response = json.dumps(response, default=str)
68     return Response(response, mimetype="application/json", status=400)

```

Kode diatas adalah sebuah fungsi yang mengambil data transfer ikan dari database. Fungsi ini menggunakan operasi agregasi MongoDB untuk melakukan penggabungan (lookup) antara koleksi "fish_transfer" dengan koleksi "pond" dan "fish_log" untuk mengambil data terkait.

Pertama, fungsi ini mendefinisikan sebuah pipeline yang berisi serangkaian

operasi agregasi. Operasi pertama adalah \$lookup yang menghubungkan koleksi "fish_transfer" dengan koleksi "pond" berdasarkan origin_pond_id. Hasil penggabungan ini disimpan dalam field origin_pond. Operasi yang sama juga dilakukan untuk menggabungkan koleksi "fish_transfer" dengan koleksi "pond" berdasarkan destination_pond_id, dan hasilnya disimpan dalam field destination_pond.

Selanjutnya, dilakukan operasi \$addFields untuk mengambil nilai pertama dari field origin_pond dan destination_pond, dan menyimpannya dalam field yang sama. Setelah itu, dilakukan \$lookup kembali untuk menggabungkan koleksi "fish_transfer" dengan koleksi "fish_log" berdasarkan _id dari transfer ikan. Hasil penggabungan ini disimpan dalam field fish.

Akhirnya, dilakukan operasi \$project untuk menghilangkan field updated_at dan created_at dari hasil akhir. Seluruh pipeline diterapkan pada koleksi "fish_transfer" menggunakan fungsi aggregate, dan hasilnya dikonversi menjadi daftar Python. Data tersebut kemudian diubah menjadi format JSON menggunakan json.dumps, dan dikembalikan sebagai respons dengan tipe konten "application/json" dan kode status 200.

Jika terjadi kesalahan selama proses, pengecualian akan ditangkap dan sebuah respons JSON yang berisi pesan kesalahan akan dikirim dengan tipe konten "application/json" dan kode status 400.

Berikut merupakan hasil test request dari API fetch list perpindahan ikan antar kolam.

cURL:

```
1 curl --location 'http://jft.web.id/fishapi/api/fishtransfer'
```

response json:

```
1 [
2 {
3     "_id": "62d564220d9bde3218b4951b",
4     "origin_pond_id": "62a62163e445ffb9c5f746f3",
5     "destination_pond_id": "625d7033a9a73e090c65cda2",
6     "origin_activation_id": "62d3f2180d7265ab60f9cb83",
7     "destination_activation_id": "62d562e676d3a668927c511f",
8     "sample_weight": 20,
9     "sample_long": 50,
10    "origin_pond": {
11        "_id": "62a62163e445ffb9c5f746f3",
12        "id_int": 4,
13        "alias": "charlie",
14        "location": "blok 2",
15        "shape": "persegi",
16        "material": "tanah",
17        "length": 5,
18        "width": 3,
19        "diameter": 0,
20        "height": 1,
21        "build_at": "2022-06-13 00:24:51.473000",
22        "image_name": "kolam_1656312365.jpeg",
23        "isActive": true
24    },
25    "destination_pond": {
26        "_id": "625d7033a9a73e090c65cda2",
27        "id_int": 3,
28        "alias": "beta",
29        "location": "blok 10",
30        "shape": "persegi",
31        "material": "terpal",
32        "length": 8,
33        "width": 4,
34        "diameter": 0,
35        "height": 1,
36        "build_at": "2022-04-18 21:05:39.608000",
37        "image_name": "default.jpg",
38        "isActive": true
39    }
}
```

```
40     "fish": [
41         {
42             "_id": "62d564220d9bde3218b4951c",
43             "pond_id": "62a62163e445ffb9c5f746f3",
44             "pond_activation_id": "62d3f2180d7265ab60f9cb83",
45             "fish_transfer_id": "62d564220d9bde3218b4951b",
46             "type_log": "transfer_out",
47             "fish_type": "lele",
48             "fish_amount": -20
49         },
50         {
51             "_id": "62d564220d9bde3218b4951d",
52             "pond_id": "62a62163e445ffb9c5f746f3",
53             "pond_activation_id": "62d3f2180d7265ab60f9cb83",
54             "fish_transfer_id": "62d564220d9bde3218b4951b",
55             "type_log": "transfer_out",
56             "fish_type": "patin",
57             "fish_amount": -20
58         }
59     ],
60 },
61 {
62     "_id": "62d57d8b08d03adbb8cff042",
63     "origin_pond_id": "62a62163e445ffb9c5f746f3",
64     "destination_pond_id": "625d7033a9a73e090c65cda2",
65     "origin_activation_id": "62d3f2180d7265ab60f9cb83",
66     "destination_activation_id": "62d562e676d3a668927c511f",
67     "sample_weight": 20,
68     "sample_long": 50,
69     "origin_pond": {
70         "_id": "62a62163e445ffb9c5f746f3",
71         "id_int": 4,
72         "alias": "charlie",
73         "location": "blok 2",
74         "shape": "persegi",
75         "material": "tanah",
76         "length": 5,
77         "width": 3,
78         "diameter": 0,
79         "height": 1,
```

```
80      "build_at": "2022-06-13 00:24:51.473000",
81      "image_name": "kolam_1656312365.jpeg",
82      "isActive": true
83    },
84    "destination_pond": {
85      "_id": "625d7033a9a73e090c65cda2",
86      "id_int": 3,
87      "alias": "beta",
88      "location": "blok 10",
89      "shape": "persegi",
90      "material": "terpal",
91      "length": 8,
92      "width": 4,
93      "diameter": 0,
94      "height": 1,
95      "build_at": "2022-04-18 21:05:39.608000",
96      "image_name": "default.jpg",
97      "isActive": true
98    },
99    "fish": [
100      {
101        "_id": "62d57d8b08d03adbb8cff043",
102        "pond_id": "62a62163e445ffb9c5f746f3",
103        "pond_activation_id": "62d3f2180d7265ab60f9cb83",
104        "fish_transfer_id": "62d57d8b08d03adbb8cff042",
105        "type_log": "transfer_out",
106        "fish_type": "lele",
107        "fish_amount": -30
108      },
109      {
110        "_id": "62d57d8b08d03adbb8cff044",
111        "pond_id": "62a62163e445ffb9c5f746f3",
112        "pond_activation_id": "62d3f2180d7265ab60f9cb83",
113        "fish_transfer_id": "62d57d8b08d03adbb8cff042",
114        "type_log": "transfer_out",
115        "fish_type": "patin",
116        "fish_amount": -10
117      }
118    ]
119  }
```

120]

5. Implementasi API edit perpindahan ikan

Implementasi controller API edit perpindahan ikan, berikut merupakan perubahan source code controller API edit perpindahan ikan.

```

1 # fishapi/database/fishtransfer.py
2
3 def put(self, id):
4     try:
5         body = request.form.to_dict(flat=True)
6         FishDeath.objects.get(id=id).update(**body)
7         response = {"message": "success change data fish death", "id": id}
8         response = json.dumps(response, default=str)
9         return Response(response, mimetype="application/json", status=200)
10    except Exception as e:
11        response = {"message": str(e)}
12        response = json.dumps(response, default=str)
13    return Response(response, mimetype="application/json", status=400)

```

Kode di atas adalah sebuah fungsi yang digunakan untuk mengubah data kematian ikan dalam database. Fungsi ini menerima parameter id yang merupakan ID unik dari data kematian ikan yang ingin diubah.

Pada awalnya, fungsi ini mengambil data dari permintaan (request) dalam bentuk formulir dan mengonversinya menjadi kamus (dictionary) menggunakan metode `to_dict(flat=True)`. Kamus ini kemudian disimpan dalam variabel `body`.

Selanjutnya, fungsi ini menggunakan metode `get()` dari model `FishDeath` untuk mendapatkan objek kematian ikan berdasarkan ID yang diberikan. Setelah itu, metode `update()` dipanggil pada objek tersebut dengan meneruskan `body` sebagai argumen. Ini akan mengubah nilai-nilai properti objek sesuai dengan nilai-nilai dalam kamus `body`.

Setelah berhasil mengubah data kematian ikan, fungsi ini mengembalikan respons sukses dalam format JSON. Pesan respons berisi informasi bahwa data kematian ikan telah berhasil diubah, beserta ID dari data yang telah diubah. Pesan respons kemudian diubah menjadi bentuk JSON menggunakan json.dumps, dan dikembalikan sebagai respons dengan tipe konten "application/json" dan kode status 200.

Namun, jika terjadi kesalahan selama proses, pengecualian akan ditangkap dan sebuah respons JSON yang berisi pesan kesalahan akan dikirim. Pesan kesalahan akan menampilkan detail dari pengecualian yang terjadi. Respons JSON tersebut juga dikembalikan dengan tipe konten "application/json" dan kode status 400.

Berikut merupakan hasil test request dari API edit perpindahan ikan antar kolam.

cURL:

```
1 curl --location -g --request PUT 'http://jft.web.id/fishapi/api/fishtransfer/{
2   fishtransfer_id}' \
2 --form 'sample_weight="20"' \
```

response json:

```
1 {
2   "message": "success change data fish transfer",
3   "id": "62b9adfa793e0f39dbaa1739"
4 }
```

6. Implementasi API delete perpindahan ikan

Implementasi controller API delete perpindahan ikan, berikut merupakan perubahan source code controller API delete perpindahan ikan.

```
1 # fishapi/database/fishtransfer.py
2
3 def delete(self, id):
4     try:
5         fishtransfer = FishTransfer.objects.get(id=id)
```

```

6         # delete fish
7
8         fishes = FishLog.objects.aggregate([
9             {'$match': {
10                 '$expr': {'$eq': ['$fish_transfer_id', {'$toObjectId': id
11             }]}},
12             ])
13
14             for fish in fishes:
15
16                 fishlog = FishLog.objects.get(id=fish['_id'])
17
18                 fishlog.delete()
19
20             # delete data
21
22             fishtransfer.delete()
23
24             response = {"message": "success delete fishtransfer"}
25
26             response = json.dumps(response, default=str)
27
28             return Response(response, mimetype="application/json", status=200)
29
30         except Exception as e:
31
32             response = {"message": str(e)}
33
34             response = json.dumps(response, default=str)
35
36             return Response(response, mimetype="application/json", status=400)

```

Kode di atas merupakan sebuah fungsi yang digunakan untuk menghapus data transfer ikan dari database. Fungsi ini menerima parameter id yang merupakan ID unik dari data transfer ikan yang akan dihapus.

Pada awalnya, fungsi ini menggunakan metode get() dari model FishTransfer untuk mendapatkan objek transfer ikan berdasarkan ID yang diberikan. Objek tersebut disimpan dalam variabel fishtransfer.

Selanjutnya, fungsi ini menggunakan agregasi MongoDB untuk mencari dan mendapatkan daftar ikan yang terkait dengan transfer ikan yang akan dihapus. Pencarian dilakukan berdasarkan kesesuaian nilai fish_transfer_id dengan ID transfer ikan yang diberikan. Hasil pencarian tersebut disimpan dalam variabel fishes.

Setelah mendapatkan daftar ikan terkait, fungsi ini melakukan iterasi melalui daftar tersebut. Pada setiap iterasi, fungsi menghapus objek FishLog yang sesuai

dengan ID ikan yang ditemukan.

Setelah menghapus semua data ikan terkait, fungsi ini menghapus objek transfer ikan itu sendiri dengan menggunakan metode delete() pada objek fishtransfer.

Setelah berhasil menghapus data transfer ikan, fungsi ini mengembalikan respons sukses dalam format JSON. Pesan respons menyatakan bahwa data transfer ikan telah berhasil dihapus. Pesan respons kemudian diubah menjadi bentuk JSON menggunakan json.dumps, dan dikembalikan sebagai respons dengan tipe konten "application/json" dan kode status 200.

Namun, jika terjadi kesalahan selama proses, pengecualian akan ditangkap dan sebuah respons JSON yang berisi pesan kesalahan akan dikirim. Pesan kesalahan akan menampilkan detail dari pengecualian yang terjadi. Respons JSON tersebut juga dikembalikan dengan tipe konten "application/json" dan kode status 400.

Berikut merupakan hasil test request dari API delete perpindahan ikan antar kolam.

cURL:

```
1 curl --location --request DELETE 'http://jft.web.id/fishapi/api/fishtransfer/62
d564220d9bde3218b4951b'
```

response json:

```
1 {
2     "message": "success delete fishtransfer"
3 }
```

7. Implementasi API fetch detail perpindahan ikan

Implementasi controller API fetch detail perpindahan ikan, berikut merupakan source code controller API fetch detail perpindahan ikan.

```
1 # fishapi/resource/fishtransfer.py
2
3 def get(self, id):
```



```

44                               '$$fish_transfer_id']),
45                               {'$eq': ['$type_log',
46                                         'transfer_out']},
47                         ]})
48           },
49           {"$project": {
50             "created_at": 0,
51             "updated_at": 0,
52           }})
53         ],
54         'as': 'fish'
55       },
56       {"$project": {
57         "updated_at": 0,
58         "created_at": 0,
59       }})
60     ]
61   }
62 fishtransfers = FishTransfer.objects.aggregate(pipeline)
63 fishtransfers = list(fishtransfers)
64 fishtransfer = fishtransfers[0]
65 response = json.dumps(fishtransfer, default=str)
66 return Response(response, mimetype="application/json", status=200)
67 except Exception as e:
68   response = {"message": str(e)}
69 response = json.dumps(response, default=str)
70 return Response(response, mimetype="application/json", status=400)

```

Kode di atas merupakan sebuah fungsi yang digunakan untuk mendapatkan data transfer ikan berdasarkan ID dari database. Fungsi ini menerima parameter id yang merupakan ID unik dari transfer ikan yang ingin diperoleh.

Pada awalnya, fungsi ini menggunakan agregasi MongoDB dengan pipeline untuk mencocokkan transfer ikan berdasarkan kesesuaian ID dengan ID yang diberikan. Pipeline ini terdiri dari beberapa tahap:

- (a) Tahap pertama, \$match, digunakan untuk mencocokkan dokumen

berdasarkan nilai `_id` yang sama dengan ID yang diberikan.

- (b) Selanjutnya, terdapat dua tahap `$lookup` yang digunakan untuk melakukan operasi join dengan koleksi "pond". Kedua tahap ini berguna untuk mencari informasi kolam asal dan kolam tujuan dari transfer ikan. Kedua tahap ini menggunakan variabel `$origin_pond_id` dan `$destination_pond_id` untuk mencocokkan nilai `_id` dengan nilai dari kolom tersebut di koleksi "pond". Tahap ini juga menggunakan tahap `$project` untuk menghilangkan field "created_at" dan "updated_at" dari hasil join.
- (c) Setelah itu, menggunakan tahap `$addFields`, ditambahkan dua field baru, yaitu "origin_pond" dan "destination_pond", yang nilainya diambil dari hasil join sebelumnya. Field-field ini hanya mengambil nilai pertama dari hasil join menggunakan operator `$first`.
- (d) Kemudian, terdapat tahap `$lookup` lainnya untuk melakukan operasi join dengan koleksi "fish_log". Tahap ini mencari log ikan yang terkait dengan transfer ikan berdasarkan nilai `_id` transfer ikan tersebut. Pencocokan dilakukan berdasarkan nilai `$fish_transfer_id` dengan nilai dari field `_id` di koleksi "fish_log". Seperti sebelumnya, tahap ini juga menggunakan tahap `$project` untuk menghilangkan field "created_at" dan "updated_at" dari hasil join.
- (e) Terakhir, menggunakan tahap `$project`, dihilangkan juga field "updated_at" dan "created_at" dari hasil akhir.

Setelah pipeline selesai, fungsi ini menggunakan metode `aggregate()` dari model `FishTransfer` dengan menggunakan pipeline yang telah dibuat untuk mendapatkan hasilnya. Hasilnya kemudian dikonversi menjadi list dan disimpan dalam variabel `fishtransfers`. Karena hanya ingin mendapatkan satu transfer ikan berdasarkan ID,

maka transfer ikan pertama dari list tersebut diambil dan disimpan dalam variabel fishtransfer.

Selanjutnya, hasil transfer ikan tersebut diubah menjadi bentuk JSON menggunakan json.dumps(). JSON tersebut kemudian dikembalikan sebagai respons dengan tipe konten "application/json" dan kode status 200.

Namun, jika terjadi kesalahan selama proses, pengecualian akan ditangkap dan sebuah respons JSON yang berisi pesan kesalahan akan dikirim. Pesan kesalahan akan menampilkan detail dari pengecualian yang terjadi. Respons JSON tersebut juga dikembalikan dengan tipe konten "application/json" dan kode status 400.

Berikut merupakan hasil test request dari API fetch list perpindahan ikan antar kolam.

cURL:

```
1 curl --location 'http://jft.web.id/fishapi/api/fishtransfer/62
d564220d9bde3218b4951b'
```

response json:

```
1 {
2   "_id": "62d564220d9bde3218b4951b",
3   "origin_pond_id": "62a62163e445ffb9c5f746f3",
4   "destination_pond_id": "625d7033a9a73e090c65cda2",
5   "origin_activation_id": "62d3f2180d7265ab60f9cb83",
6   "destination_activation_id": "62d562e676d3a668927c511f",
7   "sample_weight": 20,
8   "sample_long": 50,
9   "origin_pond": {
10     "_id": "62a62163e445ffb9c5f746f3",
11     "id_int": 4,
12     "alias": "charlie",
13     "location": "blok 2",
14     "shape": "persegi",
15     "material": "tanah",
16     "length": 5,
```

```
17     "width": 3,
18     "diameter": 0,
19     "height": 1,
20     "build_at": "2022-06-13 00:24:51.473000",
21     "image_name": "kolam_1656312365.jpeg",
22     "isActive": true
23   },
24   "destination_pond": {
25     "_id": "625d7033a9a73e090c65cda2",
26     "id_int": 3,
27     "alias": "beta",
28     "location": "blok 10",
29     "shape": "persegi",
30     "material": "terpal",
31     "length": 8,
32     "width": 4,
33     "diameter": 0,
34     "height": 1,
35     "build_at": "2022-04-18 21:05:39.608000",
36     "image_name": "default.jpg",
37     "isActive": true
38   },
39   "fish": [
40     {
41       "_id": "62d564220d9bde3218b4951c",
42       "pond_id": "62a62163e445ffb9c5f746f3",
43       "pond_activation_id": "62d3f2180d7265ab60f9cb83",
44       "fish_transfer_id": "62d564220d9bde3218b4951b",
45       "type_log": "transfer_out",
46       "fish_type": "lele",
47       "fish_amount": -20
48     },
49     {
50       "_id": "62d564220d9bde3218b4951d",
51       "pond_id": "62a62163e445ffb9c5f746f3",
52       "pond_activation_id": "62d3f2180d7265ab60f9cb83",
53       "fish_transfer_id": "62d564220d9bde3218b4951b",
54       "type_log": "transfer_out",
55       "fish_type": "patin",
56       "fish_amount": -20
```

```

57     }
58 ]
59 }
```

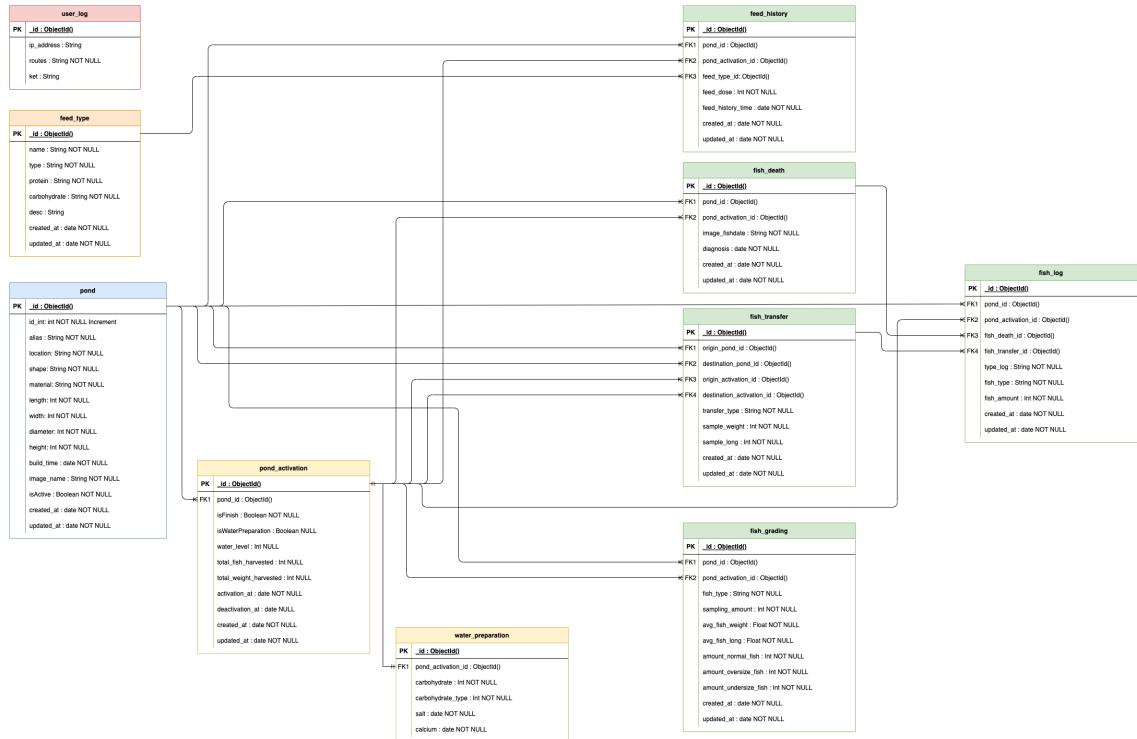
8. Sprint 8 Report

Berikut merupakan report dari sprint ke-8 yang dilakukan pada tanggal 13 juli - 19 juli 2022.

Tabel 4.13: *Sprint-8 backlog*

No	Story	Task	Status
1	Create, Read, Updte, dan Delete untuk Pencatatan grading berat ikan	Membarui desain database	Completed
2		Menambahkan routes API	Completed
3		Implementasi controller entry grading berat ikan	Completed
4		Implementasi controller fetch list grading berat ikan	Completed
5		Implementasi controller edit grading berat ikan	Completed
6		Implementasi controller delete grading berat ikan	Completed
7		Implementasi controller fetch grading berat ikan dengan id	Completed
8		Membuat view rekap grading berat ikan	Completed

1. Membarui desain database



Gambar 4.16: ERD Database Sprint-8

Dengan berubahnya desain database diperlukan juga penambahan model pada source code, berikut perubahan pada source code model.

```

1 # fishapi/database/model.py
2
3 class FishGrading(db.Document):
4     pond_id = db.ReferenceField(Pond, required=True)
5     pond_activation_id = db.ReferenceField(PondActivation, required=True)
6     isOversizeTransferred = db.BooleanField(required=True, default=False)
7     isUndersizeTransferred = db.BooleanField(required=True, default=False)
8     fish_type = db.StringField(required=True)
9     sampling_amount = db.IntField(required=True)
10    avg_fish_weight = db.FloatField(required=True)
11    avg_fish_long = db.FloatField(required=True)
12    amount_normal_fish = db.IntField(required=True)
13    amount_oversize_fish = db.IntField(required=True)
14    amount_undersize_fish = db.IntField(required=True)
15    grading_at = db.DateTimeField(default=datetime.datetime.now)

```

```

16     created_at = db.DateTimeField(default=datetime.datetime.now)
17     updated_at = db.DateTimeField(default=datetime.datetime.now)

```

2. Menambahkan routes API

```

1 # fishapi/resource/routes.py
2
3 # fish grading
4     api.add_resource(FishGradingsApi, '/api/fishgradings')
5     api.add_resource(FishGradingApi, '/api/fishgradings/<id>')
6     api.add_resource(FishGradingApiByActivation,
7                     '/api/fishgradings/activation/<activation_id>')
8     # graph
9     api.add_resource(FishGradingGraphApi,
10                      '/api/fishgradings/graph', endpoint='api.graph')

```

3. Implementasi controller entry grading berat ikan

Implementasi controller API entry grading berat ikan, berikut merupakan perubahan source code controller API entry grading berat ikan.

```

1 # fishapi/resource/fishgrading.py
2
3 class FishGradingsApi(Resource):
4     def post(self):
5         try:
6             pond_id = request.form.get("pond_id", None)
7             pond = Pond.objects.get(id=pond_id)
8             if pond['isActive'] == False:
9                 response = {"message": "pond is not active"}
10                response = json.dumps(response, default=str)
11                return Response(response, mimetype="application/json", status
12                               =400)
13                pond_activation = PondActivation.objects(
14                    pond_id=pond_id, isFinish=False).order_by('-activated_at').first
15                ()
16                body = {
17                    "pond_id": pond.id,
18                    "pond_activation_id": pond_activation.id,

```

```

17     "fish_type": request.form.get("fish_type", None),
18     "sampling_amount": request.form.get("sampling_amount", None),
19     "avg_fish_weight": request.form.get("avg_fish_weight", None),
20     "avg_fish_long": request.form.get("avg_fish_long", None),
21     "amount_normal_fish": request.form.get("amount_normal_fish", None
22     ),
23     "amount_oversize_fish": request.form.get("amount_oversize_fish",
24     None),
25     "amount_undersize_fish": request.form.get("amount_undersize_fish"
26     , None),
27     "created_at": created_at,
28     "grading_at": created_at,
29   }
30   fishgrading = FishGrading(**body).save()
31   id = fishgrading.id
32   return {'id': str(id)}, 200
33 except Exception as e:
34   response = {"message": str(e)}
35   response = json.dumps(response, default=str)
36   return Response(response, mimetype="application/json", status=400)

```

Kode di atas adalah sebuah API untuk mengelola data grading ikan. API ini menggunakan framework Flask-RESTful dan memiliki dua metode, yaitu get dan post.

Metode get digunakan untuk mendapatkan daftar grading ikan. Pada metode ini, digunakan agregasi MongoDB dengan pipeline untuk melakukan beberapa tahap pengolahan data. Pertama, terdapat dua tahap \$lookup yang digunakan untuk melakukan operasi join dengan koleksi "pond" dan "pond_activation". Tahap-tahap ini mencocokkan nilai _id dari transfer ikan dengan nilai dari field "pond_id" dan "pond_activation_id" di koleksi yang di-lookup. Setelah itu, dengan menggunakan tahap \$addFields, ditambahkan dua field baru yaitu "pond" dan "pond_activation" yang nilainya diambil dari hasil join sebelumnya menggunakan operator \$first. Tahap \$project digunakan untuk menghilangkan

field "updated_at" dan "created_at" dari hasil akhir.

Setelah pipeline selesai, fungsi ini menggunakan metode aggregate() dari model FishGrading dengan menggunakan pipeline yang telah dibuat untuk mendapatkan hasilnya. Hasilnya kemudian dikonversi menjadi list dan disimpan dalam variabel list_fishgradings.

Selanjutnya, hasil grading ikan tersebut diubah menjadi bentuk JSON menggunakan json.dumps(). JSON tersebut kemudian dikembalikan sebagai respons dengan tipe konten "application/json" dan kode status 200.

Jika terjadi kesalahan selama proses, pengecualian akan ditangkap dan sebuah respons JSON yang berisi pesan kesalahan akan dikirim. Pesan kesalahan akan menampilkan detail dari pengecualian yang terjadi. Respons JSON tersebut juga dikembalikan dengan tipe konten "application/json" dan kode status 400.

Metode post digunakan untuk membuat data grading ikan baru. Pada metode ini, terlebih dahulu dilakukan pengambilan nilai pond_id dari form data yang dikirim. Kemudian, dilakukan pencarian kolam ("pond") berdasarkan pond_id tersebut. Jika kolam tidak aktif (nilai "isActive" adalah False), maka dikembalikan respons JSON dengan pesan "pond is not active" dan kode status 400.

Selanjutnya, dilakukan pencarian aktivasi kolam ("pond_activation") terbaru yang belum selesai (nilai "isFinish" adalah False) berdasarkan pond_id. Nilai ini digunakan untuk mengisi field "pond_activation_id" pada data grading ikan yang akan dibuat.

Data grading ikan yang akan dibuat terdiri dari beberapa field yang diambil dari form data yang dikirim. Field "created_at" dan "grading_at" diisi dengan waktu saat ini. Data grading ikan tersebut kemudian disimpan ke dalam database menggunakan metode save().

Setelah data berhasil disimpan, ID dari data grading ikan yang baru dibuat diambil dan dikembalikan sebagai respons JSON dengan kode status 200.

Jika terjadi kesalahan selama proses, pengecualian akan ditangkap dan sebuah respons JSON yang berisi pesan kesalahan akan dikirim. Pesan kesalahan akan menampilkan detail dari pengecualian yang terjadi. Respons JSON tersebut juga dikembalikan dengan tipe konten "application/json" dan kode status 400.

Berikut merupakan form untuk entry grading berat ikan antar kolam.

Tabel 4.14: Form entry kematian ikan.

Form	Jenis Data	Deskripsi
pond_id	REQUIRED STRING	id kolam saat melakukan grading berat
constanta_oversize	REQUIRED DOUBLE	konstanta yang menentukan berapa berat ikan yang dikategorikan oversize
constanta_undersize	REQUIRED DOUBLE	konstanta yang menentukan berapa berat ikan yang dikategorikan undersize
fish_type	REQUIRED STRING TYPE; ["lele", "patin", "nila merah", "nila hitam", "mas"]	tipe ikan yang di grading beratnya
sampling_amount	REQUIRED INT	total jumlah ikan yang dilakukan sampling berat
avg_fish_weight	REQUIRED DOUBLE	rata-rata berat ikan yang di sampling

Lanjutan Tabel 4.14		
Form	Jenis Data	Deskripsi
avg_fish_long	REQUIRED DOUBLE	rata-rata panjang ikan yang di sampling
amount_normal_fish	REQUIRED INT	jumlah ikan yang dikategorikan normal
amount_oversize_fish	REQUIRED INT	jumlah ikan yang oversize
amount_undersize_fish	REQUIRED INT	jumlah ikan yang undersize

Tabel di atas merupakan deskripsi dari kolom-kolom yang digunakan dalam suatu entitas data grading ikan. Tabel ini menjelaskan setiap kolom beserta tipe datanya dan penjelasan singkat tentang fungsinya.

Kolom pertama adalah "pond_id" yang merupakan kolom wajib dengan tipe data STRING. Kolom ini digunakan untuk menyimpan ID kolam tempat dilakukannya grading berat pada ikan.

Kolom kedua dan ketiga adalah "constanta_oversize" dan "constanta_undersize" yang merupakan kolom wajib dengan tipe data DOUBLE. Kedua kolom ini digunakan untuk menyimpan konstanta yang menentukan batas berat ikan yang dikategorikan sebagai oversize atau undersize.

Kolom berikutnya adalah "fish_type" yang merupakan kolom wajib dengan tipe data STRING. Kolom ini digunakan untuk menyimpan tipe ikan yang sedang dilakukan grading berat.

Selanjutnya, terdapat kolom "sampling_amount" yang merupakan kolom wajib dengan tipe data INT. Kolom ini digunakan untuk menyimpan total jumlah ikan yang dilakukan sampling berat.

Kolom "avg_fish_weight" dan "avg_fish_long" adalah kolom wajib dengan tipe data DOUBLE. Kolom-kolom ini digunakan untuk menyimpan rata-rata berat dan panjang ikan yang diambil sebagai sampel.

Kolom "amount_normal_fish", "amount_oversize_fish", dan "amount_undersize_fish" adalah kolom wajib dengan tipe data INT. Kolom-kolom ini digunakan untuk menyimpan jumlah ikan yang dikategorikan sebagai normal, oversize, dan undersize, sesuai dengan hasil grading berat yang dilakukan.

Berikut merupakan hasil test request dari API entry perpindahan ikan antar kolam.

cURL:

```

1 curl --location 'http://jft.web.id/fishapi/api/fishgradings' \
2 --form 'pond_id="{pond_id}"' \
3 --form 'fish_type="lele"' \
4 --form 'sampling_amount="20"' \
5 --form 'avg_fish_weight="50"' \
6 --form 'avg_fish_long="10"' \
7 --form 'amount_normal_fish="18"' \
8 --form 'amount_oversize_fish="1"' \
9 --form 'amount_undersize_fish="1"'

```

response json:

```

1 {
2   "message": "success add fish grading",
3   "id": "62e105707ac8f837667faa70"
4 }

```

4. Implementasi API fetch list grading berat ikan

Implementasi controller API fetch list grading berat ikan, berikut merupakan source code controller API fetch list grading berat ikan.

```

1 # fishapi/resource/fishgrading.py
2

```

```

3 def get(self):
4     try:
5         pipeline = [
6             {'$lookup': {
7                 'from': 'pond',
8                 'let': {"pondid": "$pond_id"}, 
9                 'pipeline': [
10                     {'$match': {'$expr': {'$eq': ['$id', '$$pondid']}}, 
11                     {"$project": {
12                         "_id": 1,
13                         "alias": 1,
14                         "location": 1,
15                         "build_at": 1,
16                         "isActive": 1,
17                     }}, 
18                 ],
19                 'as': 'pond'
20             }},
21             {'$lookup': {
22                 'from': 'pond_activation',
23                 'let': {"activationid": "$pond_activation_id"}, 
24                 'pipeline': [
25                     {'$match': {
26                         '$expr': {'$eq': ['$id', '$$activationid']}}, 
27                     {"$project": {
28                         "_id": 1,
29                         "isFinish": 1,
30                         "isWaterPreparation": 1,
31                         "water_level": 1,
32                         "activated_at": 1
33                     }}, 
34                 ],
35                 'as': 'pond_activation'
36             }},
37             {"$addFields": {
38                 "pond": {"$first": "$pond"}, 
39                 "pond_activation": {"$first": "$pond_activation"}, 
40             }},
41             {"$project": {
42                 "updated_at": 0,

```

```

43         "created_at": 0,
44     } }
45 ]
46 fishgrading = FishGrading.objects.aggregate(pipeline)
47 list_fishgradings = list(fishgrading)
48 response = json.dumps(list_fishgradings, default=str)
49 return Response(response, mimetype="application/json", status=200)
50 except Exception as e:
51     response = {"message": str(e)}
52 response = json.dumps(response, default=str)
53 return Response(response, mimetype="application/json", status=400)

```

Kode di atas merupakan implementasi dari metode get dalam suatu API. Kode ini bertujuan untuk mengambil data grading ikan dari sumber data yang terhubung dan mengembalikan respons dalam format JSON.

Pada awalnya, kode ini mendefinisikan sebuah pipeline yang akan digunakan untuk melakukan operasi agregasi pada objek FishGrading. Pipeline ini terdiri dari beberapa tahap yang dijalankan secara berurutan. Tahap pertama menggunakan operasi \$lookup untuk melakukan join dengan koleksi "pond" berdasarkan kolom "pond_id". Hasil join ini disimpan dalam field "pond".

Selanjutnya, terdapat tahap kedua yang juga menggunakan operasi \$lookup. Tahap ini melakukan join dengan koleksi "pond_activation" berdasarkan kolom "pond_activation_id". Hasil join ini disimpan dalam field "pond_activation".

Setelah itu, menggunakan tahap "\$addFields", nilai field "pond" dan "pond_activation" diubah menjadi hanya menyimpan dokumen pertama dari hasil join, dengan menggunakan operasi \$first.

Tahap terakhir menggunakan operasi "\$project" untuk menghilangkan field "updated_at" dan "created_at" dari hasil akhir.

Setelah pipeline selesai, kode melakukan agregasi pada objek FishGrading

dengan menggunakan pipeline yang telah dibuat. Hasil agregasi ini kemudian diubah menjadi list dan dikonversi menjadi JSON menggunakan json.dumps.

Jika tidak terjadi exception selama proses ini, respons berisi data grading ikan dalam format JSON dikembalikan dengan status 200. Namun, jika terjadi exception, sebuah pesan error dikirimkan dalam format JSON dengan status 400.

Berikut merupakan hasil test request dari API fetch list perpindahan ikan antar kolam.

cURL:

```
1 curl --location 'http://jft.web.id/fishapi/api/fishgradings'
```

response json:

```
1 [
2   {
3     "_id": "62e105307ac8f837667faa6f",
4     "pond_id": "62a62163e445ffb9c5f746f3",
5     "pond_activation_id": "62d3f2180d7265ab60f9cb83",
6     "fish_type": "lele",
7     "sampling_amount": 10,
8     "avg_fish_weight": 50,
9     "avg_fish_long": 10,
10    "amount_normal_fish": 8,
11    "amount_oversize_fish": 1,
12    "amount_undersize_fish": 1,
13    "pond": {
14      "_id": "62a62163e445ffb9c5f746f3",
15      "alias": "charlie",
16      "location": "blok 2",
17      "build_at": "2022-06-13 00:24:51.473000",
18      "isActive": true
19    },
20    "pond_activation": {
21      "_id": "62d3f2180d7265ab60f9cb83",
22      "isFinish": false,
23      "isWaterPreparation": true,
```

```

24     "water_level": 100,
25     "activated_at": "2022-07-17 18:27:20.511000"
26   }
27 },
28 {
29   "_id": "62e105707ac8f837667faa70",
30   "pond_id": "62a62163e445ffb9c5f746f3",
31   "pond_activation_id": "62d3f2180d7265ab60f9cb83",
32   "fish_type": "lele",
33   "sampling_amount": 20,
34   "avg_fish_weight": 50,
35   "avg_fish_long": 10,
36   "amount_normal_fish": 18,
37   "amount_oversize_fish": 1,
38   "amount_undersize_fish": 1,
39   "pond": {
40     "_id": "62a62163e445ffb9c5f746f3",
41     "alias": "charlie",
42     "location": "blok 2",
43     "build_at": "2022-06-13 00:24:51.473000",
44     "isActive": true
45   },
46   "pond_activation": {
47     "_id": "62d3f2180d7265ab60f9cb83",
48     "isFinish": false,
49     "isWaterPreparation": true,
50     "water_level": 100,
51     "activated_at": "2022-07-17 18:27:20.511000"
52   }
53 }
54 ]

```

5. Implementasi API edit grading berat ikan

Implementasi controller API edit grading berat ikan, berikut merupakan perubahan source code controller API edit grading berat ikan.

```

1 # fishapi/database/fishgrading.py
2

```

```

3 def put(self, id):
4     try:
5         body = request.form.to_dict(flat=True)
6         FishGrading.objects.get(id=id).update(**body)
7         response = {
8             "message": "success change data fish grading", "id": id}
9         response = json.dumps(response, default=str)
10        return Response(response, mimetype="application/json", status=200)
11    except Exception as e:
12        response = {"message": str(e)}
13        response = json.dumps(response, default=str)
14        return Response(response, mimetype="application/json", status=400)

```

Kode yang diberikan merupakan implementasi metode put dalam suatu API. Tujuan dari kode ini adalah untuk memperbarui data grading ikan berdasarkan id yang diberikan.

Pertama, kode ini mengambil data dari permintaan (request) dalam format form dan mengubahnya menjadi dictionary menggunakan metode `to_dict(flat=True)`. Data tersebut akan disimpan dalam variabel `body`.

Selanjutnya, menggunakan metode `get()` dari model `FishGrading`, dilakukan pencarian data grading ikan berdasarkan id yang diberikan. Setelah data ditemukan, dilakukan pembaruan (`update()`) pada atribut-atribut data menggunakan nilai yang ada dalam `body`.

Apabila pembaruan berhasil dilakukan, kode akan mengembalikan respons yang berisi pesan sukses beserta id data yang telah diubah. Pesan dan respons tersebut akan diubah menjadi format JSON menggunakan `json.dumps()`.

Namun, apabila terjadi kesalahan atau exception dalam proses pembaruan, kode akan menangkap exception tersebut dan mengembalikan respons yang berisi pesan error. Respons tersebut juga akan diubah menjadi format JSON sebelum dikirimkan kembali kepada pengguna.

Berikut merupakan hasil test request dari API edit grading berat ikan.

cURL:

```

1 curl --location --request PUT 'http://jft.web.id/fishapi/api/fishgradings/62
2   e105707ac8f837667faa70' \
3   --form 'fish_type="lele"' \
4   --form 'sampling_amount="21"' \
5   --form 'avg_fish_weight="50"' \
6   --form 'avg_fish_long="10"' \
7   --form 'amount_normal_fish="18"' \
8   --form 'amount_oversize_fish="2"' \
9   --form 'amount_undersize_fish="1"'
```

response json:

```

1 {
2   "message": "success change data fish grading",
3   "id": "62e105707ac8f837667faa70"
4 }
```

6. Implementasi API delete grading berat ikan

Implementasi controller API delete grading berat ikan, berikut merupakan perubahan source code controller API delete grading berat ikan.

```

1 # fishapi/database/fishgrading.py
2
3 def delete(self, id):
4     try:
5         fishgrading = FishGrading.objects.get(id=id).delete()
6         response = {"message": "success delete fish grading"}
7         response = json.dumps(response, default=str)
8         return Response(response, mimetype="application/json", status=200)
9     except Exception as e:
10        response = {"message": str(e)}
11        response = json.dumps(response, default=str)
12        return Response(response, mimetype="application/json", status=400)
```

Tujuan dari kode ini adalah untuk menghapus data grading ikan berdasarkan id yang diberikan.

Pertama, menggunakan metode get() dari model FishGrading, dilakukan pencarian data grading ikan berdasarkan id yang diberikan. Setelah data ditemukan, metode delete() akan dipanggil untuk menghapus data tersebut.

Apabila penghapusan berhasil dilakukan, kode akan mengembalikan respons yang berisi pesan sukses untuk memberitahu bahwa data grading ikan telah berhasil dihapus.

Namun, jika terjadi kesalahan atau exception dalam proses penghapusan, kode akan menangkap exception tersebut dan mengembalikan respons yang berisi pesan error yang menjelaskan kesalahan yang terjadi.

Selanjutnya, respons tersebut akan diubah menjadi format JSON menggunakan json.dumps() sebelum dikirimkan kembali kepada pengguna melalui objek Response dengan jenis konten application/json dan status kode 200 untuk respons sukses, atau status kode 400 untuk respons yang mengindikasikan terjadinya kesalahan.

Berikut merupakan hasil test request dari API delete grading berat ikan.

cURL:

```
1 curl --location --request DELETE 'http://jft.web.id/fishapi/api/fishgradings/62
e10cda1370aa28b4905284'
```

response json:

```
1 {
2   "message": "success delete fish grading"
3 }
```

7. Implementasi API fetch detail grading berat ikan

Implementasi controller API fetch detail grading berat ikan, berikut merupakan source code controller API fetch detail grading berat ikan.

```

1 # fishapi/resource/fishgrading.py
2
3 def get(self, id):
4     try:
5         pipeline = [
6             {'$match': {'$expr': {'$eq': ['$id', '$toObjectId': id]}},
7             {'$lookup': {
8                 'from': 'pond',
9                 'let': {"pondid": "$pond_id"},
10                'pipeline': [
11                    {'$match': {'$expr': {'$eq': ['$id', '$$pondid']}},
12                    {"$project": {
13                        "_id": 1,
14                        "alias": 1,
15                        "location": 1,
16                        "build_at": 1,
17                        "isActive": 1,
18                    } }
19                ],
20                'as': 'pond'
21            },
22            {'$lookup': {
23                'from': 'pond_activation',
24                'let': {"activationid": "$pond_activation_id"},
25                'pipeline': [
26                    {'$match': {
27                        '$expr': {'$eq': ['$id', '$$activationid']}},
28                    {"$project": {
29                        "_id": 1,
30                        "isFinish": 1,
31                        "isWaterPreparation": 1,
32                        "water_level": 1,
33                        "activated_at": 1
34                    } }
35                ],
36                'as': 'pond_activation'
37            },

```

```

38         {"$addFields": {
39             "pond": {"$first": "$pond"},
40             "pond_activation": {"$first": "$pond_activation"},
41         }},
42         {"$project": {
43             "updated_at": 0,
44             "created_at": 0,
45         }}
46     ]
47     fishgrading = FishGrading.objects.aggregate(pipeline)
48     list_fishgradings = list(fishgrading)
49     response = json.dumps(list_fishgradings[0], default=str)
50     return Response(response, mimetype="application/json", status=200)
51 except Exception as e:
52     response = {"message": str(e)}
53     response = json.dumps(response, default=str)
54     return Response(response, mimetype="application/json", status=400)

```

Kode di atas merupakan implementasi metode get dalam suatu API yang digunakan untuk mendapatkan data grading ikan berdasarkan id yang diberikan.

Pada awalnya, kode tersebut membentuk sebuah pipeline yang berisi serangkaian operasi agregasi untuk mengambil data. Pertama, dilakukan pencocokan (\$match) dengan membandingkan _id dengan id yang diterima setelah diubah menjadi objek ObjectId menggunakan \$toObjectId.

Selanjutnya, dilakukan operasi \$lookup dua kali untuk menggabungkan data dengan koleksi "pond" dan "pond_activation" berdasarkan nilai yang sesuai. Dalam operasi \$lookup, variabel (let) digunakan untuk menyimpan nilai yang akan digunakan dalam pipeline yang sesuai. Setelah itu, dengan menggunakan \$project, hanya kolom-kolom tertentu yang dipilih untuk ditampilkan.

Kemudian, dengan menggunakan FishGrading.objects.aggregate(pipeline), pipeline akan dieksekusi dan menghasilkan objek agregat. Objek agregat tersebut kemudian dikonversi menjadi list menggunakan list(fishgrading). Dalam kasus

ini, karena hanya ingin mengambil data pertama dari hasil tersebut, digunakan indeks [0] untuk mengakses elemen pertama dari list.

Hasil akhir dari elemen pertama tersebut kemudian diubah menjadi JSON menggunakan json.dumps, dan dikembalikan sebagai respons dengan tipe konten "application/json" dan status kode 200 jika berhasil. Jika terjadi kesalahan, tangkapan Exception akan menghasilkan pesan kesalahan yang dikirim sebagai respons dengan status kode 400.

Berikut merupakan hasil test request dari API fetch list grading berat ikan.

cURL:

```
1 curl --location 'http://jft.web.id/fishapi/api/fishgradings/62
e105707ac8f837667faa70'
```

response json:

```
1 {
2     "_id": "62e105707ac8f837667faa70",
3     "pond_id": "62a62163e445ffb9c5f746f3",
4     "pond_activation_id": "62d3f2180d7265ab60f9cb83",
5     "fish_type": "lele",
6     "sampling_amount": 20,
7     "avg_fish_weight": 50,
8     "avg_fish_long": 10,
9     "amount_normal_fish": 18,
10    "amount_oversize_fish": 1,
11    "amount_undersize_fish": 1,
12    "pond": {
13        "_id": "62a62163e445ffb9c5f746f3",
14        "alias": "charlie",
15        "location": "blok 2",
16        "build_at": "2022-06-13 00:24:51.473000",
17        "isActive": true
18    },
19    "pond_activation": {
20        "_id": "62d3f2180d7265ab60f9cb83",
21        "isFinish": false,
```

```

22     "isWaterPreparation": true,
23     "water_level": 100,
24     "activated_at": "2022-07-17 18:27:20.511000"
25   }
26 }
```

9. Sprint 9 Report

Berikut merupakan report dari sprint ke-9 yang dilakukan pada tanggal 13 juli - 19 juli 2022.

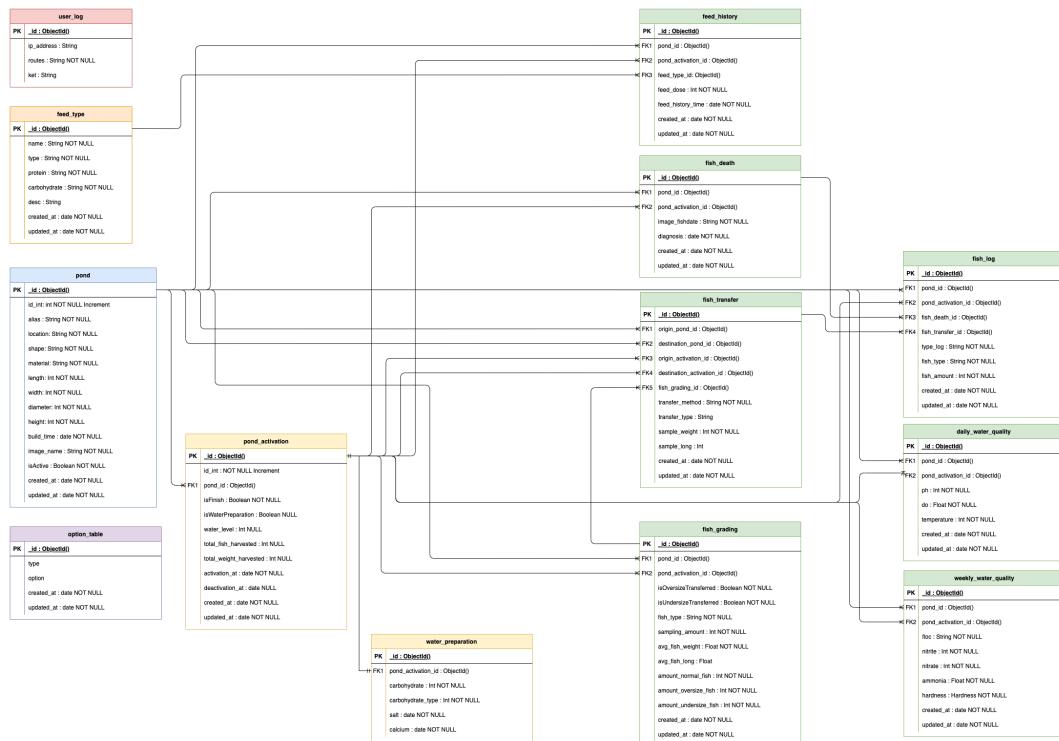
Tabel 4.15: Sprint-09 backlog.

No	Story	Task	Status
1	Create, Read, Updte, dan	Membarui desain database	Completed
2	Delete untuk Pencatatan	Menambahkan routes API	Completed
3	kualitas kolam harian dan mingguan	Implementasi controller entry Pencatatan kualitas kolam harian	Completed
4		Implementasi controller fetch list Pencatatan kualitas kolam harian	Completed
5		Implementasi controller edit Pencatatan kualitas kolam harian	Completed
6		Implementasi controller delete Pencatatan kualitas kolam harian	Completed

Lanjutan Tabel 4.15

No	Story	Task	Status
7		Implementasi controller fetch detail Pencatatan kualitas kolam harian dengan id	Completed
8		Implementasi controller entry Pencatatan kualitas kolam mingguan	Completed
9		Implementasi controller fetch list Pencatatan kualitas kolam mingguan	Completed
10		Implementasi controller edit Pencatatan kualitas kolam mingguan	Completed
11		Implementasi controller delete Pencatatan kualitas kolam mingguan	Completed
12		Implementasi controller fetch detail Pencatatan kualitas kolam mingguan dengan id	Completed
13		Membuat view rekap Pencatatan kualitas kolam harian	Completed
14		Membuat view rekap Pencatatan kualitas kolam mingguan	Completed

1. Membarui desain database



Gambar 4.17: ERD Database Sprint-9

Dengan berubahnya desain database diperlukan juga penambahan model pada source code, berikut perubahan pada source code model.

```

1 # fishapi/database/model.py
2
3 class DailyWaterQuality(db.Document):
4     pond_id = db.ReferenceField(Pond, required=True)
5     pond_activation_id = db.ReferenceField(PondActivation, required=True)
6     ph = db.IntField(required=True)
7     do = db.FloatField(required=True)
8     temperature = db.IntField(required=True)
9     created_at = db.DateTimeField(default=datetime.datetime.now)
10    updated_at = db.DateTimeField(default=datetime.datetime.now)
11
12
13 class WeeklyWaterQuality(db.Document):
14     floc_option = ('0-10', '11-30', '31-50', '51-100', '101-300', '>300')

```

```

15     nitrite_option = (0, 1, 5, 10, 20, 40, 80)
16     nitrate_option = (0, 10, 25, 50, 100, 250, 500)
17     ammonia_option = (0, 0.25, 1.5, 3, 5)
18     hardness_option = (0, 25, 50, 125, 250, 425)
19
20     pond_id = db.ReferenceField(Pond, required=True)
21     pond_activation_id = db.ReferenceField(PondActivation, required=True)
22     floc = db.StringField(required=True, choices=floc_option)
23     nitrite = db.IntField(required=True, choices=nitrite_option)
24     nitrate = db.IntField(required=True, choices=nitrate_option)
25     ammonia = db.FloatField(required=True, choices=ammonia_option)
26     hardness = db.IntField(required=True, choices=hardness_option)
27     created_at = db.DateTimeField(default=datetime.datetime.now)
28     updated_at = db.DateTimeField(default=datetime.datetime.now)

```

2. Menambahkan routes API

```

1 # fishapi/resource/routes.py
2
3 # daily water
4     api.add_resource(DailyWaterQualitiesApi, '/api/dailywaterquality')
5     api.add_resource(DailyWaterQualityApi, '/api/dailywaterquality/<id>')
6
7     # weekly water
8     api.add_resource(WeeklyWaterQualitiesApi, '/api/weeklywaterquality')
9     api.add_resource(WeeklyWaterQualityApi, '/api/weeklywaterquality/<id>')

```

3. Implementasi controller entry pencatatan kualitas kolam harian

Implementasi controller API entry pencatatan kualitas kolam harian, berikut merupakan perubahan source code controller API entry pencatatan kualitas kolam harian.

```

1 # fishapi/resource/dailywaterquality.py
2
3 class DailyWaterQualitiesApi(Resource):
4     def post(self):
5         try:
6             pond_id = request.form.get("pond_id", None)

```

```

7     pond = Pond.objects.get(id=pond_id)
8
9     if pond['isActive'] == False:
10
11         response = {"message": "pond is not active"}
12
13         response = json.dumps(response, default=str)
14
15         return Response(response, mimetype="application/json", status
16
17             =400)
18
19         pond_activation = PondActivation.objects(
20
21             pond_id=pond_id, isFinish=False).order_by('-activated_at').first
22
23         ()
24
25         body = {
26
27             "pond_id": pond.id,
28
29             "pond_activation_id": pond_activation.id,
30
31             "ph": request.form.get("ph", None),
32
33             "do": request.form.get("do", None),
34
35             "temperature": request.form.get("temperature", None),
36
37         }
38
39         print(body)
40
41         dailywaterquality = DailyWaterQuality(**body).save()
42
43         id = dailywaterquality.id
44
45         response = {
46
47             "message": "success add data daily water quality", "id": id}
48
49         response = json.dumps(response, default=str)
50
51         return Response(response, mimetype="application/json", status=200)
52
53     except Exception as e:
54
55         response = {"message": str(e)}
56
57         response = json.dumps(response, default=str)
58
59         return Response(response, mimetype="application/json", status=400)

```

Kode di atas merupakan implementasi sebuah API endpoint dengan kelas DailyWaterQualitiesApi. Kode tersebut berfungsi untuk menangani permintaan HTTP POST yang diterima pada endpoint tersebut. Berikut adalah penjelasan langkah-langkah yang dilakukan oleh kode tersebut:

- (a) Metode post() digunakan untuk menangani permintaan POST pada endpoint DailyWaterQualitiesApi.
- (b) Di dalam blok try, pertama-tama dilakukan pengambilan nilai pond_id dari

data form yang dikirim dalam permintaan.

- (c) Kemudian, dilakukan pencarian objek Pond berdasarkan pond_id yang diperoleh.
- (d) Jika nilai isActive pada objek Pond adalah False, artinya kolam tidak aktif, maka akan mengembalikan respons dengan pesan error yang sesuai.
- (e) Selanjutnya, dilakukan pencarian objek PondActivation terbaru (dengan isFinish bernilai False) berdasarkan pond_id yang diperoleh, dengan urutan berdasarkan activated_at secara menurun.
- (f) Data yang diperoleh dari form dan informasi kolam dan aktivasi yang ditemukan digabungkan ke dalam variabel body.
- (g) Objek DailyWaterQuality baru dibuat dengan menggunakan nilai-nilai dari body dan kemudian disimpan ke database dengan menggunakan metode .save(). Hasilnya akan disimpan dalam variabel dailywaterquality.
- (h) Dari objek dailywaterquality, diambil nilai id dan disimpan dalam variabel id.
- (i) Hasil akhir yang akan dikembalikan sebagai respons adalah pesan sukses beserta id dari data yang berhasil ditambahkan dalam format JSON.
- (j) Jika terjadi exception atau kesalahan, akan di-handle dengan mengembalikan respons dengan pesan error yang sesuai dalam format JSON dan status kode 400.

Dengan demikian, kode tersebut mengimplementasikan logika untuk menambahkan data kualitas air harian (DailyWaterQuality) ke dalam database berdasarkan permintaan POST yang diterima.

Berikut merupakan form untuk entry pencatatan kualitas kolam harian.

Tabel 4.16: Form entry pencatatan kolam harian.

Form	Jenis Data	Deskripsi
pond_id	REQUIRED STRING	id kolam yang akan dilakukan pencatatan kualitas air harian
ph	REQUIRED INT	tingkat ph air satuan (pH)
do	REQUIRED INT	oksigen yang terlarut dalam air (ppm)
temprature	REQUIRED INT	suhu air kolam (°C)

Tabel di atas menggambarkan struktur data yang diperlukan untuk melakukan pencatatan kualitas air harian dalam kolam. Berikut adalah penjelasan untuk setiap kolom dalam tabel:

- (a) pond_id: Kolom ini merupakan string yang wajib diisi. Nilai yang dimasukkan adalah ID kolam yang akan dicatat kualitas air harianya. Kolom ini digunakan untuk mengaitkan catatan kualitas air dengan kolam yang sesuai.
- (b) ph: Kolom ini merupakan integer yang wajib diisi. Nilai yang dimasukkan adalah tingkat pH air dalam satuan pH (potensi hidrogen). pH mengindikasikan tingkat keasaman atau kebasaan air. Pengukuran ini penting karena kualitas air yang baik untuk ikan seringkali terkait dengan tingkat pH yang sesuai.
- (c) do: Kolom ini merupakan integer yang wajib diisi. Nilai yang dimasukkan adalah tingkat oksigen terlarut dalam air dalam satuan ppm (parts per million). Oksigen terlarut penting untuk kelangsungan hidup ikan dan

organisme akuatik lainnya. Pemantauan kadar oksigen terlarut membantu memastikan bahwa lingkungan air di dalam kolam memadai bagi ikan.

- (d) temperature: Kolom ini merupakan integer yang wajib diisi. Nilai yang dimasukkan adalah suhu air kolam dalam satuan derajat Celsius. Suhu air mempengaruhi kesehatan dan pertumbuhan ikan. Pengukuran suhu air membantu memantau perubahan suhu yang dapat memengaruhi kondisi ikan dan keseimbangan ekosistem di dalam kolam.

Dengan menggunakan struktur data yang terdefinisi dalam tabel ini, pengguna dapat mencatat kualitas air harian dengan mengisi nilai-nilai yang sesuai untuk setiap kolom. Data ini penting untuk pemantauan dan pengelolaan yang efektif dalam budidaya ikan, serta untuk menjaga kesehatan dan kualitas lingkungan air di dalam kolam.

Berikut merupakan hasil test request dari API entry pencatatan kolam harian.

cURL:

```

1 curl --location 'http://jft.web.id/fishapi/api/dailywaterquality' \
2 --form 'pond_id="{pond_id}"' \
3 --form 'ph="1"' \
4 --form 'do="6"' \
5 --form 'temperature="20"'

```

response json:

```

1 {
2   "message": "success add data daily water quality",
3   "id": "62e8b800ef4edacc5bb18b05"
4 }

```

4. Implementasi API fetch list pencatatan kolam harian

Implementasi controller API fetch list pencatatan kolam harian, berikut merupakan source code controller API fetch list pencatatan kolam harian.

```

1 # fishapi/resource/dailywaterquality.py
2
3 class DailyWaterQualitysApi(Resource):
4     def get(self):
5         try:
6             pipeline = [
7                 {"$sort": {"created_at": -1}},
8                 {"'$lookup': {
9                     'from': 'pond',
10                    'let': {"pondid": "$pond_id"},
11                    'pipeline': [
12                        {'$match': {'$expr': {'$eq': ['$id', '$$pondid']} }},
13                        {"$project": {
14                            "_id": 1,
15                            "alias": 1,
16                            "location": 1,
17                            "build_at": 1,
18                            "isActive": 1,
19                        } }
20                    ],
21                    'as': 'pond'
22                }},
23                {"'$lookup': {
24                    'from': 'pond_activation',
25                    'let': {"activationid": "$pond_activation_id"},
26                    'pipeline': [
27                        {'$match': {
28                            '$expr': {'$eq': ['$id', '$$activationid']} },
29                        {"$project": {
30                            "_id": 1,
31                            "isFinish": 1,
32                            "isWaterPreparation": 1,
33                            "water_level": 1,
34                            "activated_at": 1
35                        } }
36                    ],
37                    'as': 'pond_activation'
38                }},
39                {"$addFields": {

```

```

40     "pond": {"$first": "$pond"},  

41     "pond_activation": {"$first": "$pond_activation"},  

42     "ph_desc": {  

43         "$switch":  

44             {  

45                 "branches": [  

46                     {  

47                         "case": {"$lt": ["$ph", 6]},  

48                         "then": "berbahaya"  

49                     },  

50                     {  

51                         "case": {"$gt": ["$ph", 8]},  

52                         "then": "berbahaya"  

53                     }  

54                 ],  

55                 "default": "normal"  

56             }  

57         },  

58     "do_desc": {  

59         "$switch":  

60             {  

61                 "branches": [  

62                     {  

63                         "case": {"$or": [  

64                             {"$lt": ["$do", 3]},  

65                             {"$gt": ["$do", 7.5]}  

66                         ]},  

67                         "then": "berbahaya"  

68                     },  

69                     {  

70                         "case": {"$or": [  

71                             {"$and": [{"$gte": ["$do", 3]}, {  

72                             "$lte": ["$do", 4]}]},  

73                             {"$and": [{"$gt": ["$do", 6]}, {  

74                             "$lte": ["$do", 7.5]}]}  

75                         ]},  

76                         "then": "semi berbahaya"  

77                     }  

78                 ],  

79                 "default": "normal"  


```

```

80                     }
81             }
82         } },
83         {"$project": {
84             "updated_at": 0,
85             "created_at": 0,
86         } }
87     ]
88     dailywaterquality = DailyWaterQuality.objects.aggregate(pipeline)
89     list_dailywaterqualitys = list(dailywaterquality)
90     response = json.dumps(list_dailywaterqualitys, default=str)
91     return Response(response, mimetype="application/json", status=200)
92 except Exception as e:
93     response = {"message": str(e)}
94     response = json.dumps(response, default=str)
95     return Response(response, mimetype="application/json", status=400)

```

Kode di atas merupakan implementasi dari sebuah API dengan nama DailyWaterQualitysApi yang bertujuan untuk mengambil data kualitas air harian (DailyWaterQuality). Berikut penjelasan tentang kode tersebut:

API ini menyediakan metode get() yang digunakan untuk mengambil data kualitas air harian. Dalam metode tersebut, dilakukan serangkaian operasi pengolahan data menggunakan Aggregation Framework pada MongoDB.

Pertama, sebuah pipeline dibuat sebagai langkah-langkah pengolahan data. Pada langkah awal, data diurutkan berdasarkan waktu pembuatan (created_at) secara menurun (descending) menggunakan operasi \$sort.

Selanjutnya, dilakukan operasi \$lookup untuk melakukan join dengan dua koleksi lain, yaitu "pond" dan "pond_activation". Dalam langkah ini, dilakukan pencocokan (\$match) berdasarkan nilai _id pada koleksi "pond" dan "pond_activation" dengan nilai yang berasal dari koleksi "DailyWaterQuality". Hasil join disimpan dalam array pond dan pond_activation.

Setelah itu, dilakukan penambahan kolom baru (\$addFields) pada dokumen dengan menggunakan hasil join sebelumnya. Kolom baru yang ditambahkan antara lain pond (berisi data kolam yang terkait), pond_activation (berisi data aktivasi kolam yang terkait), ph_desc (deskripsi tingkat pH), dan do_desc (deskripsi tingkat oksigen terlarut). Deskripsi tersebut ditentukan berdasarkan kondisi yang didefinisikan dengan operator \$switch.

Langkah terakhir dalam pipeline adalah menggunakan operasi \$project untuk menghilangkan kolom updated_at dan created_at dari hasil akhir.

Setelah pipeline selesai, dilakukan agregasi pada koleksi "DailyWaterQuality" menggunakan pipeline yang telah dibuat. Hasil agregasi kemudian diubah menjadi list dan dikonversi menjadi format JSON. Respon JSON ini kemudian dikembalikan sebagai respons dari API dengan kode status 200 (OK).

Namun, jika terjadi kesalahan selama proses, dilakukan penanganan eksepsi (exception) yang menghasilkan pesan error. Pesan error tersebut kemudian dikonversi menjadi format JSON dan dikembalikan sebagai respons dari API dengan kode status 400 (Bad Request).

Berikut merupakan hasil test request dari API fetch list pencatatan kolam harian.

cURL:

```
1 curl --location 'http://jft.web.id/fishapi/api/dailywaterquality'
```

response json:

```
1 [
2   {
3     "_id": "647edbe028aa8efac0b6c2f2",
4     "pond_id": "647d8beb28aa8efac0b6c2ed",
5     "pond_activation_id": "647d8c0b28aa8efac0b6c2ee",
6     "ph": 1,
7     "do": 6.0,
```

```
8      "temperature": 20,
9      "pond": {
10        "_id": "647d8beb28aa8efac0b6c2ed",
11        "alias": "charlie",
12        "location": "blok 3",
13        "isActive": true,
14        "build_at": "2023-06-05 14:16:59.384000"
15      },
16      "pond_activation": {
17        "_id": "647d8c0b28aa8efac0b6c2ee",
18        "isFinish": false,
19        "isWaterPreparation": false,
20        "water_level": 100.0,
21        "activated_at": "2023-06-05 14:17:31.973000"
22      },
23      "ph_desc": "berbahaya",
24      "do_desc": "normal"
25    },
26    {
27      "_id": "647edbe028aa8efac0b6c2f2",
28      "pond_id": "647d8beb28aa8efac0b6c2ed",
29      "pond_activation_id": "647d8c0b28aa8efac0b6c2ee",
30      "ph": 1,
31      "do": 6.0,
32      "temperature": 20,
33      "pond": {
34        "_id": "647d8beb28aa8efac0b6c2ed",
35        "alias": "charlie",
36        "location": "blok 3",
37        "isActive": true,
38        "build_at": "2023-06-05 14:16:59.384000"
39      },
40      "pond_activation": {
41        "_id": "647d8c0b28aa8efac0b6c2ee",
42        "isFinish": false,
43        "isWaterPreparation": false,
44        "water_level": 100.0,
45        "activated_at": "2023-06-05 14:17:31.973000"
46      },
47      "ph_desc": "berbahaya",
```

```

48         "do_desc": "normal"
49     }
50 ]

```

5. Implementasi API edit pencatatan kolam harian

Implementasi controller API edit pencatatan kolam harian, berikut merupakan perubahan source code controller API edit pencatatan kolam harian.

```

1 # fishapi/database/dailywaterquality.py
2
3 class DailyWaterQualityApi(Resource):
4     def put(self, id):
5         try:
6             body = request.form.to_dict(flat=True)
7             DailyWaterQuality.objects.get(id=id).update(**body)
8             response = {
9                 "message": "success change data daily water quality", "id": id}
10            response = json.dumps(response, default=str)
11            return Response(response, mimetype="application/json", status=200)
12        except Exception as e:
13            response = {"message": str(e)}
14            response = json.dumps(response, default=str)
15            return Response(response, mimetype="application/json", status=400)

```

Kode di atas merupakan implementasi dari sebuah API dengan nama DailyWaterQualityApi yang bertujuan untuk mengubah (update) data kualitas air harian (DailyWaterQuality) berdasarkan ID tertentu. Berikut adalah penjelasan tentang kode tersebut:

API ini menyediakan metode put() yang digunakan untuk mengubah data kualitas air harian. Metode ini menerima parameter id yang merupakan ID dari data yang akan diubah.

Dalam metode put(), terdapat beberapa langkah yang dilakukan. Pertama, nilai dari form request diubah menjadi bentuk dictionary menggunakan fungsi

`request.form.to_dict(flat=True)`. Dictionary ini akan berisi data yang dikirimkan dalam form request.

Selanjutnya, dilakukan operasi `update()` pada objek `DailyWaterQuality` dengan menggunakan ID yang diberikan. Operasi `update()` ini akan memperbarui data dengan nilai-nilai yang terdapat dalam dictionary body. Data yang diubah akan sesuai dengan kolom-kolom yang ada dalam objek `DailyWaterQuality`.

Setelah proses pembaruan selesai, dibuat respons yang akan dikirimkan kembali kepada pengguna. Respons ini berupa pesan sukses yang mengindikasikan bahwa data kualitas air harian telah berhasil diubah. Respons tersebut juga mencakup ID dari data yang telah diubah. Pesan dan ID tersebut kemudian diubah menjadi format JSON menggunakan fungsi `json.dumps()`.

Apabila terjadi kesalahan selama proses pembaruan, penanganan eksepsi (exception handling) akan terjadi. Pesan error yang dihasilkan akan dikonversi menjadi format JSON dan dikirimkan sebagai respons dengan kode status 400 (Bad Request).

Dengan demikian, melalui API ini pengguna dapat mengirimkan permintaan untuk mengubah data kualitas air harian berdasarkan ID tertentu, dan akan menerima respons yang mengindikasikan keberhasilan atau kegagalan pembaruan data.

Berikut merupakan hasil test request dari API edit pencatatan kolam harian.

cURL:

```

1 curl --location --request PUT 'http://jft.web.id/fishapi/api/dailywaterquality/62
   e89ce7846724c2d2984687' \
2 --form 'ph="2"' \
3 --form 'do="7"' \
4 --form 'temperature="20"'

```

response json:

```

1 {
2     "message": "success change data daily water quality",
3     "id": "62e89ce7846724c2d2984687"
4 }
```

6. Implementasi API delete pencatatan kolam harian

Implementasi controller API delete pencatatan kolam harian, berikut merupakan perubahan source code controller API delete pencatatan kolam harian.

```

1 # fishapi/database/dailywaterquality.py
2
3 class DailyWaterQualityApi(Resource):
4     def delete(self, id):
5         try:
6             dailywaterquality = DailyWaterQuality.objects.get(id=id).delete()
7             response = {"message": "success delete daily water quality"}
8             response = json.dumps(response, default=str)
9             return Response(response, mimetype="application/json", status=200)
10        except Exception as e:
11            response = {"message": str(e)}
12            response = json.dumps(response, default=str)
13            return Response(response, mimetype="application/json", status=400)
```

API ini menyediakan metode delete() yang digunakan untuk menghapus data kualitas air harian. Metode ini menerima parameter id yang merupakan ID dari data yang akan dihapus.

Dalam metode delete(), terdapat beberapa langkah yang dilakukan. Pertama, dilakukan pengambilan objek DailyWaterQuality berdasarkan ID yang diberikan menggunakan metode get(). Kemudian, dilakukan operasi delete() pada objek tersebut untuk menghapus data dari koleksi.

Setelah proses penghapusan selesai, dibuat respons yang akan dikirimkan kembali kepada pengguna. Respons ini berupa pesan sukses yang mengindikasikan bahwa

data kualitas air harian telah berhasil dihapus. Pesan tersebut kemudian diubah menjadi format JSON menggunakan fungsi `json.dumps()`.

Apabila terjadi kesalahan selama proses penghapusan, penanganan eksepsi (exception handling) akan terjadi. Pesan error yang dihasilkan akan dikonversi menjadi format JSON dan dikirimkan sebagai respons dengan kode status 400 (Bad Request).

Dengan demikian, melalui API ini pengguna dapat mengirimkan permintaan untuk menghapus data kualitas air harian berdasarkan ID tertentu, dan akan menerima respons yang mengindikasikan keberhasilan atau kegagalan penghapusan data.

Berikut merupakan hasil test request dari API delete pencatatan kolam harian.

cURL:

```
1 curl --location --request DELETE 'http://jft.web.id/fishapi/api/dailywaterquality
/62e8b800ef4edacc5bb18b05'
```

response json:

```
1 {
2     "message": "success delete daily water quality"
3 }
```

7. Implementasi API fetch detail pencatatan kolam harian dengan id

Implementasi controller API fetch detail pencatatan kolam harian, berikut merupakan source code controller API fetch detail pencatatan kolam harian.

```
1 # fishapi/resource/dailywaterquality.py
2
3 class DailyWaterQualityApi(Resource):
4     def get(self, id):
5         try:
6             pipeline = [
7                 {'$match': {'$expr': {'$eq': ['$id', '$toObjectId': id]}}, },
8                 {'$lookup': {
```

```
9      'from': 'pond',
10     'let': {"pondid": "$pond_id"},
11     'pipeline': [
12       {'$match': {'$expr': {'$eq': ['$id', '$$pondid']} }},
13       {"$project": {
14         "_id": 1,
15         "alias": 1,
16         "location": 1,
17         "build_at": 1,
18         "isActive": 1,
19       } }
20     ],
21     'as': 'pond'
22   },
23   {'$lookup': {
24     'from': 'pond_activation',
25     'let': {"activationid": "$pond_activation_id"},
26     'pipeline': [
27       {'$match': {
28         '$expr': {'$eq': ['$id', '$$activationid']} }},
29       {"$project": {
30         "_id": 1,
31         "isFinish": 1,
32         "isWaterPreparation": 1,
33         "water_level": 1,
34         "activated_at": 1
35       } }
36     ],
37     'as': 'pond_activation'
38   },
39   {"$addFields": {
40     "pond": {"$first": "$pond"},  

41     "pond_activation": {"$first": "$pond_activation"},  

42     "ph_desc": {
43       "$switch": {
44         "branches": [
45           {
46             "case": {"$lt": ["$ph", 6]},  

47             "then": "berbahaya"
48           }
49         ]
50       }
51     }
52   }}
```

```

49         },
50         {
51             "case": {"$gt": ["$ph", 8]},
52             "then": "berbahaya"
53         }
54     ],
55     "default": "normal"
56 }
57 },
58 "do_desc": {
59     "$switch":
60     {
61         "branches": [
62             {
63                 "case": {"$or": [
64                     {"$lt": ["$do", 3]},
65                     {"$gt": ["$do", 7.5]}]
66                 },
67                 "then": "berbahaya"
68             },
69             {
70                 "case": {"$or": [
71                     {"$and": [{"$gte": ["$do", 3]}, {"$lte": ["$do", 4]}]}, {
72                     {"$and": [{"$gt": ["$do", 6]}, {"$lte": ["$do", 7.5]}]}]
73                 },
74                 "then": "semi berbahaya"
75             },
76             "default": "normal"
77         }
78     ],
79     "default": "normal"
80 }
81 }
82 },
83 "$project": {
84     "updated_at": 0,
85     "created_at": 0,
86 }
87 ]
88 dailywaterquality = DailyWaterQuality.objects.aggregate(pipeline)

```

```

89     list_dailywaterqualitys = list(dailywaterquality)
90
91     response = json.dumps(list_dailywaterqualitys[0], default=str)
92
93     return Response(response, mimetype="application/json", status=200)
94
95     except Exception as e:
96
97         response = {"message": str(e)}
98
99         response = json.dumps(response, default=str)
100
101    return Response(response, mimetype="application/json", status=400)

```

Kode di atas merupakan implementasi dari sebuah API dengan nama DailyWaterQualityApi yang digunakan untuk mendapatkan (get) data kualitas air harian (DailyWaterQuality) berdasarkan ID tertentu. Berikut adalah penjelasan tentang kode tersebut:

API ini menyediakan metode get() yang digunakan untuk mengambil data kualitas air harian berdasarkan ID. Metode ini menerima parameter id yang merupakan ID dari data yang akan diambil.

Dalam metode get(), terdapat beberapa langkah yang dilakukan. Pertama, terdapat penggunaan pipeline yang berisi serangkaian tahapan untuk melakukan agregasi dan pencarian data menggunakan MongoDB. Tahapan-tahapan tersebut meliputi pencocokan (\$match) berdasarkan ID yang diberikan, pencarian relasi dengan koleksi pond dan pond_activation menggunakan \$lookup, dan penambahan (\$addFields) beberapa field baru ke hasil agregasi.

Beberapa field baru yang ditambahkan adalah ph_desc dan do_desc, yang bergantung pada nilai ph dan do pada data kualitas air harian. Nilai-nilai tersebut diuji menggunakan operator \$switch untuk menentukan deskripsi kualitas air berdasarkan batasan-batasan tertentu.

Setelah tahapan agregasi selesai, hasil aggregasi tersebut dikonversi menjadi daftar (list) dan diambil elemen pertama (list_dailywaterqualitys[0]). Hasil tersebut kemudian diubah menjadi format JSON menggunakan fungsi

json.dumps().

Apabila terjadi kesalahan selama proses pengambilan data, penanganan eksepsi (exception handling) akan terjadi. Pesan error yang dihasilkan akan dikonversi menjadi format JSON dan dikirimkan sebagai respons dengan kode status 400 (Bad Request).

Berikut merupakan hasil test request dari API fetch pencatatan kolam harian berdasarkan id.

cURL:

```
1 curl --location 'http://jft.web.id/fishapi/api/dailywaterquality/62
e89ce7846724c2d2984687'
```

response json:

```
1 {
2   "_id": "62e89ce7846724c2d2984687",
3   "pond_id": "62a62163e445ffb9c5f746f3",
4   "pond_activation_id": "62d3f2180d7265ab60f9cb83",
5   "ph": 7,
6   "do": 4,
7   "temperature": 25,
8   "pond": {
9     "_id": "62a62163e445ffb9c5f746f3",
10    "alias": "charlie",
11    "location": "blok 2",
12    "build_at": "2022-06-13 00:24:51.473000",
13    "isActive": true
14  },
15  "pond_activation": {
16    "_id": "62d3f2180d7265ab60f9cb83",
17    "isFinish": false,
18    "isWaterPreparation": true,
19    "water_level": 100,
20    "activated_at": "2022-07-17 18:27:20.511000"
21  },
22  "ph_desc": "normal",
```

```

23     "do_desc": "semi berbahaya"
24 }
```

8. Implementasi controller entry pencatatan kualitas kolam mingguan

Implementasi controller API entry pencatatan kualitas kolam mingguan, berikut merupakan perubahan source code controller API entry pencatatan kualitas kolam mingguan.

```

1 # fishapi/resource/weeklywaterquality.py
2
3 class WeeklyWaterQualitysApi(Resource):
4     def post(self):
5         try:
6             pond_id = request.form.get("pond_id", None)
7             pond = Pond.objects.get(id=pond_id)
8             if pond['isActive'] == False:
9                 response = {"message": "pond is not active"}
10            response = json.dumps(response, default=str)
11            return Response(response, mimetype="application/json", status
12                           =400)
13            pond_activation = PondActivation.objects(
14                pond_id=pond_id, isFinish=False).order_by('-activated_at').first
15            ()
16            body = {
17                "pond_id": pond.id,
18                "pond_activation_id": pond_activation.id,
19                "floc": request.form.get("floc", None),
20                "nitrite": request.form.get("nitrite", None),
21                "nitrate": request.form.get("nitrate", None),
22                "ammonia": request.form.get("ammonia", None),
23                "hardness": request.form.get("hardness", None),
24            }
25            weeklywaterquality = WeeklyWaterQuality(**body).save()
26            id = weeklywaterquality.id
27            response = {
28                "message": "success add data weekly water quality", "id": id}
29            response = json.dumps(response, default=str)
30            return Response(response, mimetype="application/json", status=200)
```

```

29     except Exception as e:
30         response = {"message": str(e)}
31         response = json.dumps(response, default=str)
32     return Response(response, mimetype="application/json", status=400)

```

Kode di atas merupakan implementasi dari sebuah API dengan nama WeeklyWaterQualityApi. API ini digunakan untuk menambahkan (post) data kualitas air mingguan (WeeklyWaterQuality) ke dalam sistem. Berikut adalah penjelasan mengenai kode tersebut:

API ini menyediakan metode post() yang digunakan untuk menangani permintaan untuk menambahkan data kualitas air mingguan ke sistem. Metode ini menerima data yang dikirimkan dalam format form-data melalui permintaan HTTP POST.

Pertama, kode ini mengambil nilai pond_id dari permintaan menggunakan request.form.get(). Nilai ini digunakan untuk mendapatkan objek kolam (pond) dari basis data menggunakan metode objects.get() dari model Pond. Jika kolam dengan ID tersebut tidak aktif (isActive = False), maka API akan memberikan respons dengan pesan error yang menyatakan "pond is not active".

Selanjutnya, kode ini mencari aktivasi kolam (pond_activation) terbaru yang belum selesai (isFinish = False) berdasarkan pond_id. Aktivasi kolam ini diurutkan berdasarkan waktu aktivasi (activated_at) secara menurun (descending) menggunakan metode order_by() dan diambil aktivasi pertama menggunakan first(). Informasi ini akan digunakan dalam pembuatan objek WeeklyWaterQuality.

Data untuk objek WeeklyWaterQuality, seperti floc, nitrite, nitrate, ammonia, dan hardness, diambil dari permintaan menggunakan request.form.get(). Kemudian, objek WeeklyWaterQuality baru dibuat dengan menggunakan data tersebut dan disimpan ke dalam basis data menggunakan metode save().

Setelah penyimpanan berhasil, respons sukses dengan pesan "success add data weekly water quality" dan ID objek yang baru dibuat (id) dikirimkan sebagai respons dalam format JSON.

Namun, jika terjadi kesalahan selama proses tersebut, blok except akan menangkap eksepsi yang terjadi dan memberikan respons dengan pesan error yang dihasilkan.

Dengan demikian, melalui API ini pengguna dapat mengirimkan permintaan untuk menambahkan data kualitas air mingguan ke sistem. Jika permintaan berhasil, pengguna akan menerima respons sukses dengan ID data yang baru ditambahkan. Jika terjadi kesalahan, respons error akan diberikan dengan pesan yang menjelaskan kesalahan tersebut.

Berikut merupakan form untuk entry pencatatan kualitas kolam mingguan.

Tabel 4.17: Form entry pencatatan kolam mingguan.

Form	Jenis Data	Deskripsi
pond_id	REQUIRED STRING	id kolam yang akan dilakukan pencatatan kualitas air mingguan
floc	OPTION STRING VALUE: ['0-10', '11-30', '31-50', '51-100', '101-300', '>300']	kadar floc dalam air
nitrite	OPTION INT VALUE: [0, 1, 5, 10, 20, 40, 80]	kadar nitrite pada air
nitrate	OPTION INT VALUE: [0, 10, 25, 50, 100, 250, 500]	kadar nitrate pada air

Lanjutan Tabel 4.17		
Form	Jenis Data	Deskripsi
ammonia	OPTION DOUBLE VALUE: [0, 0.25, 1.5, 3, 5]	kadar ammonia pada air
hardness	OPTION INT VALUE: [0, 25, 50, 125, 250, 425]	tingkat hardness air

Berikut adalah penjelasan mengenai setiap kolom pada tabel:

- (a) pond_id (REQUIRED STRING): Kolom ini menyimpan ID kolam yang akan digunakan untuk mencatat kualitas air mingguan. ID kolam harus diberikan dalam format string dan merupakan data yang wajib diisi.
- (b) floc (OPTION STRING): Kolom ini mencatat kadar floc dalam air. Floc adalah partikel-partikel padat yang mengendap dalam air kolam. Nilai yang dapat diisikan adalah dalam bentuk string dengan pilihan nilai yang telah ditentukan, yaitu "0-10", "11-30", "31-50", "51-100", "101-300", atau ">300". Kolom ini bersifat opsional, sehingga tidak wajib diisi.
- (c) nitrite (OPTION INT): Kolom ini mencatat kadar nitrite pada air. Nitrite adalah senyawa nitrogen yang dapat menjadi racun bagi ikan dan organisme akuatik lainnya jika terlalu tinggi. Nilai yang dapat diisikan adalah dalam bentuk integer (bilangan bulat) dengan pilihan nilai yang telah ditentukan, yaitu 0, 1, 5, 10, 20, 40, atau 80. Kolom ini bersifat opsional, sehingga tidak wajib diisi.
- (d) nitrate (OPTION INT): Kolom ini mencatat kadar nitrate pada air. Nitrate adalah senyawa nitrogen yang juga dapat menjadi racun bagi ikan dan organisme akuatik jika terlalu tinggi. Nilai yang dapat diisikan adalah

dalam bentuk integer dengan pilihan nilai yang telah ditentukan, yaitu 0, 10, 25, 50, 100, 250, atau 500. Kolom ini bersifat opsional, sehingga tidak wajib diisi.

- (e) ammonia (OPTION DOUBLE): Kolom ini mencatat kadar ammonia pada air. Ammonia adalah senyawa nitrogen beracun yang dihasilkan oleh limbah organik dalam kolam. Nilai yang dapat diisikan adalah dalam bentuk double (bilangan desimal) dengan pilihan nilai yang telah ditentukan, yaitu 0, 0.25, 1.5, 3, atau 5. Kolom ini bersifat opsional, sehingga tidak wajib diisi.
- (f) hardness (OPTION INT): Kolom ini mencatat tingkat kekerasan air (hardness). Kekerasan air dapat dipengaruhi oleh kandungan mineral seperti kalsium dan magnesium. Nilai yang dapat diisikan adalah dalam bentuk integer dengan pilihan nilai yang telah ditentukan, yaitu 0, 25, 50, 125, 250, atau 425. Kolom ini bersifat opsional, sehingga tidak wajib diisi.

Tabel tersebut memberikan informasi tentang jenis data yang harus diisi dalam setiap kolom dan deskripsi singkat tentang masing-masing kolom. Data-data ini digunakan untuk mencatat kualitas air mingguan dalam kolam dan dapat digunakan untuk analisis dan pemantauan kondisi lingkungan akuatik.

Berikut merupakan hasil test request dari API entry pencatatan kolam mingguan.

cURL:

```

1 curl --location 'http://jft.web.id/fishapi/api/weeklywaterquality' \
2 --form 'pond_id="{pond_id}"' \
3 --form 'floc="11-30"' \
4 --form 'nitrite="20"' \
5 --form 'nitrate="100"' \
6 --form 'ammonia="1.5"' \
7 --form 'hardness="125"'

```

response json:

```

1  {
2      "message": "success add data weekly water quality",
3      "id": "62e92ea6602825ffd3a1d2c1"
4  }

```

9. Implementasi API fetch list pencatatan kolam mingguan

Implementasi controller API fetch list pencatatan kolam mingguan, berikut merupakan source code controller API fetch list pencatatan kolam mingguan.

```

1 # fishapi/resource/weeklywaterquality.py
2
3 class WeeklyWaterQualitiesApi(Resource):
4     def get(self):
5         try:
6             pipeline = [
7                 {"$sort": {"created_at": -1}},
8                 {'$lookup': {
9                     'from': 'pond',
10                    'let': {"pondid": "$pond_id"},
11                    'pipeline': [
12                        {'$match': {'$expr': {'$eq': ['$id', '$$pondid']} }},
13                        {"$project": {
14                            "_id": 1,
15                            "alias": 1,
16                            "location": 1,
17                            "build_at": 1,
18                            "isActive": 1,
19                        } }
20                    ],
21                    'as': 'pond'
22                }},
23                {'$lookup': {
24                    'from': 'pond_activation',
25                    'let': {"activationid": "$pond_activation_id"},
26                    'pipeline': [
27                        {'$match': {
28                            '$expr': {'$eq': ['$id', '$$activationid']} }},
29                        {"$project": {

```

```

30         "_id": 1,
31         "isFinish": 1,
32         "isWaterPreparation": 1,
33         "water_level": 1,
34         "activated_at": 1
35     } }
36 ],
37 'as': 'pond_activation'
38 },
39 {"$addFields": {
40     "pond": {"$first": "$pond"},
41     "pond_activation": {"$first": "$pond_activation"},
42     "floc_desc": {
43         "$$switch": {
44             {
45                 "branches": [
46                     {"case": {
47                         "$eq": ["$floc", "0-10"]}, "then": "
48                     pembentukan},
49                     {"case": {
50                         "$eq": ["$floc", "11-30"]}, "then": "
51                     normal},
52                     {"case": {
53                         "$eq": ["$floc", "31-50"]}, "then": "
54                     tebal},
55                     {"case": {
56                         "$eq": ["$floc", "51-100"]}, "then": "
57                     abnormal},
58                     {"case": {
59                         "$eq": ["$floc", "101-300"]}, "then": "
60                     berbahaya},
61                     {"case": {
62                         "$eq": ["$floc", ">300"]}, "then": "
63                     deadzone},
64                 ],
65                 "default": "normal"
66             }
67         },
68         "nitrite_desc": {
69             "$$switch": {
70                 {
71                     "branches": [
72                         {"case": {
73                             "$eq": ["$nitrite", "0-10"]}, "then": "
74                             rendah},
75                         {"case": {
76                             "$eq": ["$nitrite", "11-30"]}, "then": "
77                             sedang},
78                         {"case": {
79                             "$eq": ["$nitrite", "31-50"]}, "then": "
80                             tinggi},
81                         {"case": {
82                             "$eq": ["$nitrite", "51-100"]}, "then": "
83                             sangattinggi}
84                     ]
85                 }
86             }
87         }
88     }
89 }
90 }
```

```

64
65         {
66             "branches": [
67                 {"case": {
68                     "$eq": ["$ammonia", 0]}, "then": "tidak
69                     ada"}, ,
70                     {"case": {
71                         "$eq": ["$ammonia", 1]}, "then": "sedikit
72                     "}, ,
73                     {"case": {
74                         "$eq": ["$ammonia", 5]}, "then": "aman"}, ,
75                     {"case": {
76                         "$eq": ["$ammonia", 10]}, "then": "pekat"
77                     }, ,
78                     {"case": {
79                         "$eq": ["$ammonia", 20]}, "then": "banyak
80                     "}, ,
81                     {"case": {
82                         "$eq": ["$ammonia", 40]}, "then": "
83                     berbahaya"}, ,
84                     {"case": {
85                         "$eq": ["$ammonia", 80]}, "then": "
86                     deadzone"}, ,
87                     ],
88                     "default": "normal"
89                 }
90             },
91             "nitrate_desc": {
92                 "$switch":
93                 {
94                     "branches": [
95                         {"case": {
96                             "$eq": ["$ammonia", 0]}, "then": "tidak
97                             ada"}, ,
98                         {"case": {
99                             "$eq": ["$ammonia", 10]}, "then": "sedikit
100                         "}, ,
101                         {"case": {
102                             "$eq": ["$ammonia", 25]}, "then": "aman"
103                         }, ,
104                         {"case": {
105                             "$eq": ["$ammonia", 30]}, "then": "pekat"
106                         }
107                     }
108                 }
109             }
110         }
111     }
112 }
```

```

95           "$eq": ["$ammonia", 50]}, "then": "pekat"
96     },
97   {"case": {
98     "$eq": ["$ammonia", 100]}, "then": "
99       banyak",
100      {"case": {
101        "$eq": ["$ammonia", 250]}, "then": "
102          berbahaya"),
103        {"case": {
104          "$eq": ["$ammonia", 500]}, "then": "
105            deadzone},
106          ],
107          "default": "normal"
108        }
109      },
110      "ammonia_desc": {
111        "$switch":
112        {
113          "branches": [
114            {"case": {
115              "$eq": ["$ammonia", 0]}, "then": "sedikit
116            "},
117            {"case": {
118              "$eq": ["$ammonia", 0.25]}, "then": "aman
119            "},
120            {"case": {
121              "$eq": ["$ammonia", 1.5]}, "then": "pekat
122            "},
123            {"case": {
124              "$eq": ["$ammonia", 3]}, "then": "banyak"
125            },
126            {"case": {
127              "$eq": ["$ammonia", 5]}, "then": "
128          berbahaya"),
129          ],
130          "default": "normal"
131        }
132      },
133      "hardness_desc": {
134        "$switch":
```

```

126
127         {
128             "branches": [
129                 {"case": {
130                     "$eq": ["$ammonia", 0]}, "then": "sedikit"
131                 },
132                 {"case": {
133                     "$eq": ["$ammonia", 25]}, "then": "aman"
134                 },
135                 {"case": {
136                     "$eq": ["$ammonia", 50]}, "then": "pekat"
137                 },
138                 {"case": {
139                     "$eq": ["$ammonia", 125]}, "then": "banyak"
140                 },
141                 {"case": {
142                     "$eq": ["$ammonia", 250]}, "then": "berbahaya"
143                 },
144                 {"case": {
145                     "$project": {
146                         "updated_at": 0,
147                         "created_at": 0,
148                     }})
149                 ],
150             weeklywaterquality = WeeklyWaterQuality.objects.aggregate(pipeline)
151             list_weeklywaterqualitys = list(weeklywaterquality)
152             response = json.dumps(list_weeklywaterqualitys, default=str)
153             return Response(response, mimetype="application/json", status=200)
154         except Exception as e:
155             response = {"message": str(e)}
156             response = json.dumps(response, default=str)
157             return Response(response, mimetype="application/json", status=400)

```

Kode di atas adalah implementasi dari API WeeklyWaterQualitysApi yang digunakan untuk mengambil data kualitas air mingguan dari database. Berikut adalah penjelasan mengenai kode tersebut:

- (a) Pada blok try, pertama-tama kita mendefinisikan sebuah pipeline untuk melakukan agregasi data menggunakan MongoDB Aggregation Framework. Pipeline ini terdiri dari beberapa tahap yang akan diterapkan pada data WeeklyWaterQuality.
- (b) Tahap pertama dalam pipeline adalah \$sort, yang digunakan untuk mengurutkan data berdasarkan kolom created_at secara descending (urutan terbalik), sehingga data terbaru akan muncul di bagian atas.
- (c) Tahap selanjutnya adalah \$lookup. Di sini dilakukan operasi join dengan koleksi pond dan pond_activation. Pada \$lookup pertama, kita melakukan join dengan koleksi pond menggunakan kolom pond_id dari WeeklyWaterQuality dan _id dari pond. Hasil join ini akan disimpan dalam field pond pada dokumen WeeklyWaterQuality. \$lookup kedua dilakukan dengan koleksi pond_activation menggunakan kolom pond_activation_id dari WeeklyWaterQuality dan _id dari pond_activation. Hasil join ini akan disimpan dalam field pond_activation pada dokumen WeeklyWaterQuality.
- (d) Setelah dilakukan join, tahap \$addFields digunakan untuk menambahkan beberapa field baru ke dokumen WeeklyWaterQuality. Field yang ditambahkan antara lain floc_desc, nitrite_desc, nitrate_desc, ammonia_desc, dan hardness_desc. Field-field ini akan memiliki nilai berdasarkan kondisi-kondisi yang ditentukan menggunakan operator \$switch. Misalnya, pada field floc_desc, dilakukan pengecekan nilai field floc dan kemudian menentukan nilai berdasarkan kondisi yang sesuai. Jika

nilai floc adalah "0-10", maka nilai floc_desc akan menjadi "pembentukan".

- (e) Tahap terakhir adalah \$project, yang digunakan untuk mengatur proyeksi output. Di sini dilakukan penghapusan field updated_at dan created_at dari output agar tidak ditampilkan dalam respons API.
- (f) Setelah selesai mengatur pipeline, kita menjalankannya menggunakan aggregate() pada objek WeeklyWaterQuality dengan parameter pipeline yang telah dibuat. Hasil aggregasi ini kemudian dikonversi menjadi list dan disimpan dalam variabel list_weeklywaterqualitys.
- (g) Setelah itu, variabel list_weeklywaterqualitys diubah menjadi format JSON menggunakan json.dumps(). Respons JSON ini kemudian dikirimkan sebagai respons API dengan status 200 jika sukses.
- (h) Pada blok except, jika terjadi exception, respons akan berisi pesan error yang diambil dari exception yang terjadi.

Kode tersebut mengimplementasikan logika untuk mengambil data kualitas air mingguan dari database, melakukan operasi join dengan koleksi lain, dan melakukan transformasi pada nilai-nilai tertentu sebelum mengirimkan respons dalam format JSON.

Berikut merupakan hasil test request dari API fetch list pencatatan kolam harian.

cURL:

```
1 curl --location 'http://jft.web.id/fishapi/api/weeklywaterquality'
```

response json:

```
1 [
2   {
3     "_id": "62e92ea6602825ffd3a1d2c1",
4     "pond_id": "62a62163e445ffb9c5f746f3",
5     "pond_activation_id": "62d3f2180d7265ab60f9cb83",
```

```
6      "floc": "11-30",
7      "nitrite": 20,
8      "nitrate": 100,
9      "ammonia": 1.5,
10     "hardness": 125,
11     "pond": {
12       "_id": "62a62163e445ffb9c5f746f3",
13       "alias": "charlie",
14       "location": "blok 2",
15       "build_at": "2022-06-13 00:24:51.473000",
16       "isActive": true
17     },
18     "pond_activation": {
19       "_id": "62d3f2180d7265ab60f9cb83",
20       "isFinish": false,
21       "isWaterPreparation": true,
22       "water_level": 100,
23       "activated_at": "2022-07-17 18:27:20.511000"
24     },
25     "floc_desc": "normal",
26     "nitrite_desc": "normal",
27     "nitrate_desc": "normal",
28     "ammonia_desc": "pekat",
29     "hardness_desc": "normal"
30   },
31   {
32     "_id": "62e92e4d57ca7c50fd16795b",
33     "pond_id": "62a62163e445ffb9c5f746f3",
34     "pond_activation_id": "62d3f2180d7265ab60f9cb83",
35     "floc": "0-10",
36     "nitrite": 5,
37     "nitrate": 10,
38     "ammonia": 3,
39     "hardness": 250,
40     "pond": {
41       "_id": "62a62163e445ffb9c5f746f3",
42       "alias": "charlie",
43       "location": "blok 2",
44       "build_at": "2022-06-13 00:24:51.473000",
45       "isActive": true
```

```
46     },
47     "pond_activation": {
48       "_id": "62d3f2180d7265ab60f9cb83",
49       "isFinish": false,
50       "isWaterPreparation": true,
51       "water_level": 100,
52       "activated_at": "2022-07-17 18:27:20.511000"
53     },
54     "floc_desc": "pembentukan",
55     "nitrite_desc": "normal",
56     "nitrate_desc": "normal",
57     "ammonia_desc": "banyak",
58     "hardness_desc": "normal"
59   },
60   {
61     "_id": "62e8d6263019b1590982a5e8",
62     "pond_id": "62a62163e445ffb9c5f746f3",
63     "pond_activation_id": "62d3f2180d7265ab60f9cb83",
64     "floc": "0-10",
65     "nitrite": 6,
66     "nitrate": 20,
67     "ammonia": 345,
68     "hardness": 534,
69     "pond": {
70       "_id": "62a62163e445ffb9c5f746f3",
71       "alias": "charlie",
72       "location": "blok 2",
73       "build_at": "2022-06-13 00:24:51.473000",
74       "isActive": true
75     },
76     "pond_activation": {
77       "_id": "62d3f2180d7265ab60f9cb83",
78       "isFinish": false,
79       "isWaterPreparation": true,
80       "water_level": 100,
81       "activated_at": "2022-07-17 18:27:20.511000"
82     },
83     "floc_desc": "pembentukan",
84     "nitrite_desc": "normal",
85     "nitrate_desc": "normal",
```

```

86     "ammonia_desc": "normal",
87     "hardness_desc": "normal"
88   }
89 ]

```

10. Implementasi API edit pencatatan kolam mingguan

Implementasi controller API edit pencatatan kolam mingguan, berikut merupakan perubahan source code controller API edit pencatatan kolam mingguan.

```

1 # fishapi/database/weeklywaterquality.py
2
3 class WeeklyWaterQualityApi(Resource):
4     def put(self, id):
5         try:
6             body = request.form.to_dict(flat=True)
7             WeeklyWaterQuality.objects.get(id=id).update(**body)
8             response = {
9                 "message": "success change data weekly water quality", "id": id}
10            response = json.dumps(response, default=str)
11            return Response(response, mimetype="application/json", status=200)
12        except Exception as e:
13            response = {"message": str(e)}
14            response = json.dumps(response, default=str)
15            return Response(response, mimetype="application/json", status=400)

```

Kode di atas adalah implementasi dari API WeeklyWaterQualityApi yang digunakan untuk memperbarui data kualitas air mingguan berdasarkan ID tertentu. Berikut adalah penjelasan mengenai kode tersebut:

Pada blok try, API ini menerima ID sebagai parameter saat melakukan permintaan PUT. Permintaan tersebut digunakan untuk mengidentifikasi entitas WeeklyWaterQuality yang akan diperbarui.

Selanjutnya, kita mendapatkan body permintaan dari

request.form.to_dict(flat=True). request.form mengandung data yang dikirim dalam permintaan PUT, dan dengan menggunakan metode to_dict(flat=True), kita mengonversi data tersebut menjadi dictionary tunggal yang berisi pasangan kunci-nilai.

Dalam langkah selanjutnya, kita menggunakan metode get() pada model WeeklyWaterQuality dengan ID yang diberikan untuk mengambil entitas yang sesuai dari database. Kemudian, kita memanggil metode update() pada entitas yang ditemukan dan memberikan argumen **body untuk memperbarui nilai-nilai kolom dengan nilai yang baru dari body permintaan.

Jika tidak terjadi kesalahan selama proses tersebut, kita mengirimkan respons berhasil dengan status 200. Respons ini berisi pesan sukses dan ID dari entitas WeeklyWaterQuality yang telah diperbarui.

Pada blok except, jika terjadi exception selama proses tersebut, kita mengirimkan respons dengan pesan error yang diambil dari exception yang terjadi, bersama dengan status 400.

Dengan demikian, kode ini mengimplementasikan logika untuk menerima permintaan PUT dengan ID sebagai parameter, mengambil data dari permintaan, dan memperbarui entitas WeeklyWaterQuality yang sesuai dengan nilai-nilai baru.

Berikut merupakan hasil test request dari API edit pencatatan kolam mingguan.

cURL:

```

1 curl --location --request PUT 'http://jft.web.id/fishapi/api/weeklywaterquality
2 /62e92ea6602825ffd3ald2c1' \
3 --form 'floc="11-30"' \
4 --form 'nitrite="20"' \
5 --form 'nitrate="100"' \
6 --form 'ammonia="1.5"' \

```

```
6 --form 'hardness="250'"
```

response json:

```
1 {
2     "message": "success change data weekly water quality",
3     "id": "62e92ea6602825ffd3a1d2c1"
4 }
```

11. Implementasi API delete pencatatan kolam mingguan

Implementasi controller API delete pencatatan kolam mingguan, berikut merupakan perubahan source code controller API delete pencatatan kolam mingguan.

```
1 # fishapi/database/weeklywaterquality.py
2
3 class WeeklyWaterQualityApi(Resource):
4     def delete(self, id):
5         try:
6             weeklywaterquality = WeeklyWaterQuality.objects.get(id=id).delete()
7             response = {"message": "success delete weekly water quality"}
8             response = json.dumps(response, default=str)
9             return Response(response, mimetype="application/json", status=200)
10        except Exception as e:
11            response = {"message": str(e)}
12            response = json.dumps(response, default=str)
13            return Response(response, mimetype="application/json", status=400)
```

Kode di atas adalah implementasi dari API WeeklyWaterQualityApi yang digunakan untuk menghapus data kualitas air mingguan berdasarkan ID tertentu. Berikut adalah penjelasan mengenai kode tersebut:

Pada blok try, API ini menerima ID sebagai parameter saat melakukan permintaan DELETE. Permintaan tersebut digunakan untuk mengidentifikasi entitas WeeklyWaterQuality yang akan dihapus.

Selanjutnya, kita menggunakan metode `get()` pada model `WeeklyWaterQuality` dengan ID yang diberikan untuk mengambil entitas yang sesuai dari database. Kemudian, kita memanggil metode `delete()` pada entitas tersebut untuk menghapusnya dari database.

Jika penghapusan berhasil dan tidak terjadi kesalahan selama proses tersebut, kita mengirimkan respons berhasil dengan status 200. Respons ini berisi pesan sukses yang mengindikasikan bahwa data kualitas air mingguan telah berhasil dihapus.

Pada blok `except`, jika terjadi exception selama proses tersebut, kita mengirimkan respons dengan pesan error yang diambil dari exception yang terjadi, bersama dengan status 400.

Dengan demikian, kode ini mengimplementasikan logika untuk menerima permintaan `DELETE` dengan ID sebagai parameter, menghapus entitas `WeeklyWaterQuality` yang sesuai dari database, dan mengirimkan respons berhasil jika penghapusan berhasil dilakukan.

Berikut merupakan hasil test request dari API delete pencatatan kolam harian.

cURL:

```
1 curl --location --request DELETE 'http://jft.web.id/fishapi/api/
  weeklywaterquality/62e92ea6602825ffd3a1d2c1'
```

response json:

```
1 {
2   "message": "success delete weekly water quality"
3 }
```

12. Implementasi API fetch detail pencatatan kolam mingguan dengan id

Implementasi controller API fetch detail pencatatan kolam mingguan, berikut merupakan source code controller API fetch detail pencatatan kolam mingguan.

```
1 # fishapi/resource/weeklywaterquality.py
2
3 class WeeklyWaterQualityApi(Resource):
4     def get(self, id):
5         try:
6             pipeline = [
7                 {"$match": {"$expr": {"$eq": ["$_id", {"$toObjectId": "id"}]}},
8                 {"$sort": {"created_at": -1}},
9                 {"$lookup": {
10                     "from": "pond",
11                     "let": {"pondid": "$pond_id"},
12                     "pipeline": [
13                         {"$match": {"$expr": {"$eq": ["$_id", '$$pondid']}},
14                         {"$project": {
15                             "_id": 1,
16                             "alias": 1,
17                             "location": 1,
18                             "build_at": 1,
19                             "isActive": 1,
20                         } }
21                     ],
22                     "as": 'pond'
23                 }},
24                 {"$lookup": {
25                     "from": 'pond_activation',
26                     "let": {"activationid": "$pond_activation_id"},
27                     "pipeline": [
28                         {"$match": {
29                             '$expr': {"$eq": ["$_id", '$$activationid']}},
30                         {"$project": {
31                             "_id": 1,
32                             "isFinish": 1,
33                             "isWaterPreparation": 1,
34                             "water_level": 1,
35                             "activated_at": 1
36                         } }
37                     ],
38                     "as": 'pond_activation'
39                 }},
40             ]
41             result = self._db.pond.aggregate(pipeline)
42             return result
43         except Exception as e:
44             print(e)
45             return {"error": str(e)}
```

```

40      {"$addFields": {
41        "pond": {"$first": "$pond"},
42        "pond_activation": {"$first": "$pond_activation"},
43        "floc_desc": {
44          "$switch": {
45            "branches": [
46              {"case": {
47                "$eq": ["$floc", "0-10"]}, "then": "pembentukan",
48                {"case": {
49                  "$eq": ["$floc", "11-30"]}, "then": "normal",
50                  {"case": {
51                    "$eq": ["$floc", "31-50"]}, "then": "tebal",
52                    {"case": {
53                      "$eq": ["$floc", "51-100"]}, "then": "abnormal",
54                      {"case": {
55                        "$eq": ["$floc", "101-300"]}, "then": "berbahaya",
56                        {"case": {
57                          "$eq": ["$floc", ">300"]}, "then": "deadzone",
58                          [,
59                            "default": "normal"
60                          }
61                        },
62                      "nitrite_desc": {
63                        "$switch": {
64                          "branches": [
65                            {"case": {
66                              "$eq": ["$ammonia", 0]}, "then": "tidak
67 ada"}, ,
68                            {"case": {
69                              "$eq": ["$ammonia", 1]}, "then": "sedikit
70   "},
71                            {"case": {

```

```

72           "$eq": ["$ammonia", 5]}, "then": "aman"}, {
73             "case": {
74               "$eq": ["$ammonia", 10]}, "then": "pekat"
75             },
76             {"case": {
77               "$eq": ["$ammonia", 20]}, "then": "banyak"
78             },
79             {"case": {
80               "$eq": ["$ammonia", 40]}, "then": "berbahaya",
81             },
82             {"case": {
83               "$eq": ["$ammonia", 80]}, "then": "deadzone",
84             ],
85             "default": "normal"
86           }
87         },
88         "nitrate_desc": {
89           "$switch":
90             {
91               "branches": [
92                 {"case": {
93                   "$eq": ["$ammonia", 0]}, "then": "tidak
94                   ada"}, {
95                   {"case": {
96                     "$eq": ["$ammonia", 10]}, "then": "sedikit"}, {
97                     {"case": {
98                       "$eq": ["$ammonia", 25]}, "then": "aman"
99                     },
100                     {"case": {
101                       "$eq": ["$ammonia", 50]}, "then": "pekat"
102                     },
103                     {"case": {
104                       "$eq": ["$ammonia", 100]}, "then": "banyak",
105                     },
106                     {"case": {
107                       "$eq": ["$ammonia", 250]}, "then": "berbahaya"}, {
108                       {"case": {
109                         "$eq": ["$ammonia", 250]}}, "then": {
110                           {"case": {
111                             "$eq": ["$ammonia", 250]}}, "then": "berbahaya"
112                           }
113                         }
114                       ]
115                     }
116                   }
117                 }
118               }
119             }
120           ]
121         }
122       }
123     }
124   }
125 }
```

```

102
103     "deadzone"}, ,
104         ],
105         "default": "normal"
106     }
107 },
108     "ammonia_desc": {
109         "$switch":
110     {
111         "branches": [
112             {"case": {
113                 "$eq": ["$ammonia", 0]}, "then": "sedikit"
114             },
115             {"case": {
116                 "$eq": ["$ammonia", 0.25]}, "then": "aman"
117             },
118             {"case": {
119                 "$eq": ["$ammonia", 1.5]}, "then": "pekat"
120             },
121             {"case": {
122                 "$eq": ["$ammonia", 3]}, "then": "banyak"
123             },
124             {"case": {
125                 "$eq": ["$ammonia", 5]}, "then": "berbahaya"}, ,
126                 ],
127                 "default": "normal"
128             }
129         },
130         "hardness_desc": {
131             "$switch":
132         {
133             "branches": [
134                 {"case": {
135                     "$eq": ["$ammonia", 0]}, "then": "sedikit"
136                 },
137                 {"case": {
138                     "$eq": ["$ammonia", 25]}, "then": "aman"
139                 },
140                 {"case": {
141                     "$eq": ["$ammonia", 50]}, "then": "sangat_aman"
142                 }
143             }
144         }
145     }
146 }
```

```

134                               "$eq": ["$ammonia", 50]}, "then": "pekat"
135                           },
136                           {"case": {
137                               "$eq": ["$ammonia", 125]}, "then": "
138                               banyak},
139                               {"case": {
140                                   "$eq": ["$ammonia", 250]}, "then": "
141                               berbahaya},
142                               {"case": {
143                                   "$eq": ["$ammonia", 425]}, "then": "
144                               deadzone},
145                               ],
146                               "default": "normal"
147                           }
148                       },
149                       {"$project": {
150                           "updated_at": 0,
151                           "created_at": 0,
152                           }
153                       ]
154               weeklywaterquality = WeeklyWaterQuality.objects.aggregate(pipeline)
155               list_weeklywaterqualitys = list(weeklywaterquality)
156               response = json.dumps(list_weeklywaterqualitys[0], default=str)
157               return Response(response, mimetype="application/json", status=200)
158           except Exception as e:
159               response = {"message": str(e)}
160               response = json.dumps(response, default=str)
161               return Response(response, mimetype="application/json", status=400)

```

Kode di atas adalah implementasi dari API WeeklyWaterQualityApi yang digunakan untuk mengambil data kualitas air mingguan berdasarkan ID tertentu. Berikut adalah penjelasan mengenai kode tersebut:

- (a) Pada blok try, API ini menerima ID sebagai parameter saat melakukan permintaan GET. Permintaan tersebut digunakan untuk mengidentifikasi entitas WeeklyWaterQuality yang akan diambil.

- (b) Kemudian, kita menggunakan aggregation framework pada MongoDB untuk menghasilkan pipa (pipeline) untuk operasi pencocokan, pengurutan, dan lookup yang kompleks. Pipa ini berisi beberapa tahapan:
- i. Pada tahap pertama, menggunakan operasi \$match untuk mencocokkan _id dengan ID yang diberikan.
 - ii. Pada tahap berikutnya, menggunakan operasi \$sort untuk mengurutkan data berdasarkan created_at secara menurun.
 - iii. Setelah itu, menggunakan operasi \$lookup untuk menggabungkan data dari koleksi "pond" berdasarkan pond_id. Lookup ini mengambil beberapa field spesifik dari koleksi "pond" dan memasukkannya ke dalam field pond.
 - iv. Selanjutnya, menggunakan operasi \$lookup lagi untuk menggabungkan data dari koleksi "pond_activation" berdasarkan pond_activation_id. Lookup ini mengambil beberapa field spesifik dari koleksi "pond_activation" dan memasukkannya ke dalam field pond_activation.
 - v. Kemudian, menggunakan operasi \$addFields untuk menambahkan field-field baru ke dokumen, seperti floc_desc, nitrite_desc, nitrate_desc, ammonia_desc, dan hardness_desc. Nilai-nilai field ini dihasilkan berdasarkan operasi \$switch yang memetakan nilai-nilai field awal ke nilai deskriptif yang lebih terbaca.
 - vi. Terakhir, menggunakan operasi \$project untuk menghilangkan field updated_at dan created_at dari hasil akhir.
- (c) Setelah menjalankan agregasi, kita mengambil hasilnya sebagai daftar objek weeklywaterquality dan mengonversinya menjadi JSON menggunakan json.dumps(). Kita kemudian mengirimkan respons dengan hasil JSON tersebut dan status 200.

- (d) Pada blok except, jika terjadi exception selama proses tersebut, kita mengirimkan respons dengan pesan error yang diambil dari exception yang terjadi, bersama dengan status 400.

Dengan demikian, kode ini mengimplementasikan logika untuk menerima permintaan GET dengan ID sebagai parameter, mengambil data kualitas air mingguan yang sesuai dari database menggunakan aggregation framework, dan mengirimkan respons dengan data yang berhasil diambil jika operasi tersebut berhasil dilakukan.

Berikut merupakan hasil test request dari API fetch pencatatan kolam mingguan berdasarkan id.

cURL:

```
1 curl --location 'http://jft.web.id/fishapi/api/weeklywaterquality/62
e92ea6602825ffd3a1d2c1'
```

response json:

```
1 {
2   "_id": "62e92ea6602825ffd3a1d2c1",
3   "pond_id": "62a62163e445ffb9c5f746f3",
4   "pond_activation_id": "62d3f2180d7265ab60f9cb83",
5   "floc": "11-30",
6   "nitrite": 20,
7   "nitrate": 100,
8   "ammonia": 1.5,
9   "hardness": 125,
10  "pond": {
11    "_id": "62a62163e445ffb9c5f746f3",
12    "alias": "charlie",
13    "location": "blok 2",
14    "build_at": "2022-06-13 00:24:51.473000",
15    "isActive": true
16  },
17  "pond_activation": {
```

```

18     "_id": "62d3f2180d7265ab60f9cb83",
19     "isFinish": false,
20     "isWaterPreparation": true,
21     "water_level": 100,
22     "activated_at": "2022-07-17 18:27:20.511000"
23   },
24   "floc_desc": "normal",
25   "nitrite_desc": "normal",
26   "nitrate_desc": "normal",
27   "ammonia_desc": "pekat",
28   "hardness_desc": "normal"
29 }

```

13. Membuat view rekap pencatatan kolam harian

The screenshot shows a web-based application for managing pond treatments. On the left, there's a sidebar with various navigation links such as 'Harian', 'Bulanan', 'KOLAM', 'Detail Kolam', 'Masa Budidaya Kolam', 'Jumlah Ikan', 'KEMATIAN IKAN', 'Kematian Bulanan', 'SORTIR IKAN', 'Sortir Bulanan', 'GRADING IKAN', 'Grading Bulanan', 'PENGUKURAN KUALITAS AIR', 'Harian', 'Mingguan', and 'TREATMENT KOLAM'. The main content area has a title 'Treatment Kolam' and a sub-section 'Treatment Kolam / Bulanan'. It features a search bar with 'Search' and a date input 'Pilih Bulan' set to '2023-06'. Below this is a table titled 'Hasil untuk Bulan June 2023' with columns: No, Tanggal Treatment, Kolam, Masa Budidaya, Tipe Treatment, Pergantian Air, Dosis Garam, Kultur Probiotik, and Karbon. Two entries are listed: entry 1 for 'beta' on '05-06-2023' with 'karantina' treatment and entry 2 for 'charlie' on '05-06-2023' with 'ringan' treatment. The table includes a 'Show 10 entries' dropdown and a 'Search' input.

No	Tanggal Treatment	Kolam	Masa Budidaya	Tipe Treatment	Pergantian Air	Dosis Garam	Kultur Probiotik	Karbon
1	05-06-2023	beta	1	karantina	100%	-	-	-
2	05-06-2023	charlie	1	ringan	0%	-	10 G/ml	10 (gula)

Gambar 4.18: View list pencatatan kolam harian

14. Membuat view rekap pencatatan kolam mingguan

The screenshot shows a web-based application interface titled "Treatment Kolam". On the left, there is a sidebar with various navigation links under categories like KOLAM, KEMATIAN IKAN, SORTIR IKAN, GRAADING IKAN, PENGUKURAN KUALITAS AIR, and TREATMENT KOLAM. The main content area is titled "Hasil untuk Bulan June 2023" and displays a table of treatment records. The table has columns for No, Tanggal Treatment, Kolam, Masa Budidaya, Tipe Treatment, Pergantian Air, Dosis Garam, Kultur Probiotik, and Karbon. Two entries are listed:

No	Tanggal Treatment	Kolam	Masa Budidaya	Tipe Treatment	Pergantian Air	Dosis Garam	Kultur Probiotik	Karbon
1	05-06-2023	beta	1	karantina	100%	-	-	-
2	05-06-2023	charlie	1	ringan	0%	-	10 G/ml	10 (gula)

At the bottom of the table, it says "Showing 1 to 2 of 2 entries". There are also "Previous" and "Next" buttons.

Gambar 4.19: View list pencatatan kolam mingguan

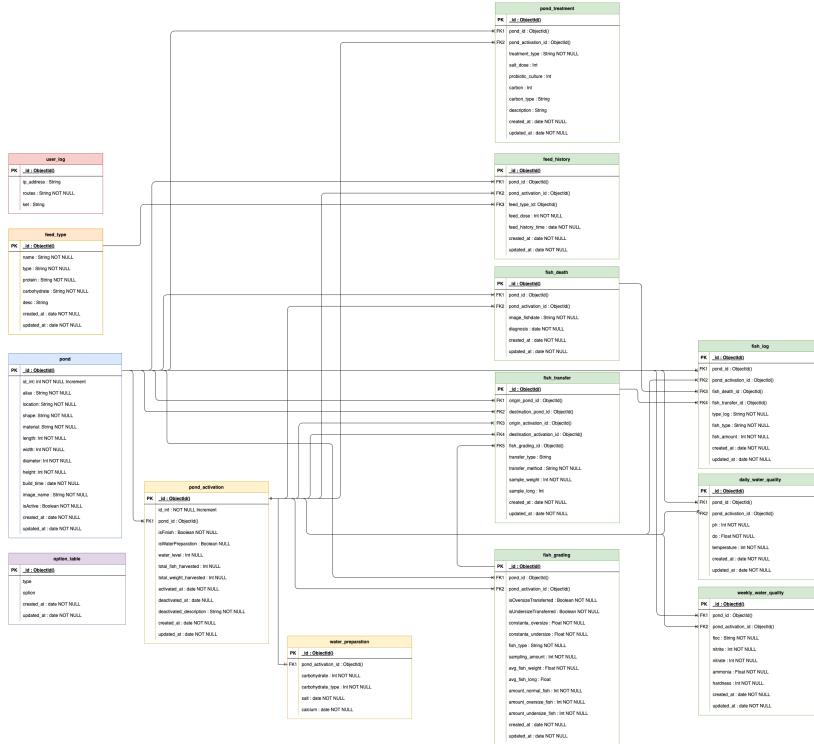
10. Sprint 10 Report

Berikut merupakan report dari sprint ke-10 yang dilakukan pada tanggal 20 juli - 26 juli 2022.

Tabel 4.18: Sprint-10 backlog

No	Story	Task	Status
1	Create, Read, Update, dan Delete untuk Treatment kolam	Membarui desain database	Completed
2		Menambahkan routes API	Completed
3		Implementasi controller entry treatment kolam	Completed
4		Implementasi controller fetch list treatment kolam	Completed
5		Implementasi controller edit treatment kolam	Completed
6		Implementasi controller delete treatment kolam	Completed
7		Implementasi controller fetch detail treatment kolam dengan id	Completed
8		Membuat view rekap treatment kolam	Completed

1. Membarui desain database



Gambar 4.20: ERD Database Sprint-10

Dengan berubahnya desain database diperlukan juga penambahan model pada source code, berikut perubahan pada source code model.

```

1 # fishapi/database/model.py
2
3 class PondTreatment(db.Document):
4     treatment_type_option = ("ringan", "karantina", "pergantian air")
5     carbohydrate_type_option = ("", "gula", "molase", "terigu", "tapioka")
6
7     pond_id = db.ReferenceField(Pond, required=True)
8     pond_activation_id = db.ReferenceField(PondActivation, required=True)
9     treatment_type = db.StringField(required=True, choices=treatment_type_option)
10    water_change = db.IntField()
11    salt = db.IntField()
12    probiotic_culture = db.IntField()
13    carbohydrate = db.IntField()
14    carbohydrate_type = db.StringField(required=True, choices=
15        carbohydrate_type_option, default="")

```

```
15     description = db.StringField(default="")
16     created_at = db.DateTimeField(default=datetime.datetime.now)
17     updated_at = db.DateTimeField(default=datetime.datetime.now)
```

2. Menambahkan routes API

```
1 # fishapi/resource/routes.py
2
3 # pond treatment
4     api.add_resource(PondTreatmentsApi, '/api/pondtreatment')
5     api.add_resource(PondTreatmentApi, '/api/pondtreatment/<id>')
```

3. Implementasi controller entry treatment kolam

Implementasi controller API entry treatment kolam, berikut merupakan perubahan source code controller API entry treatment kolam.

```
1 # fishapi/resource/pondtreatment.py
2
3 class PondTreatmentsApi(Resource):
4     def post(self):
5         try:
6             pond_id = request.form.get("pond_id", None)
7             pond = Pond.objects.get(id=pond_id)
8             if pond['isActive'] == False:
9                 response = {"message": "pond is not active"}
10            response = json.dumps(response, default=str)
11            return Response(response, mimetype="application/json", status
12                           =400)
13
14            pond_activation = PondActivation.objects(
15                pond_id=pond_id, isFinish=False).order_by('-activated_at').first
16            ()
17
18            treatment_type = request.form.get("treatment_type", None)
19
20            if treatment_type == "karantina":
21
22                body = {
23
24                    "pond_id": pond_id,
25
26                    "pond_activation_id": pond_activation.id,
27
28                    "treatment_type": treatment_type,
29
30                    "water_change": 100,
```

```
    "description": request.form.get("description", None),
    }

pondtreatment = PondTreatment(**body).save()
id = pondtreatment.id

# update activation and pond
pond_deactivation_data = {
    "isFinish": True,
    "total_fish_harvested": request.form.get(
        "total_fish_harvested", None),
    "total_weight_harvested": request.form.get(
        "total_weight_harvested", None),
    "deactivated_at": request.form.get("deactivated_at", datetime.datetime.now()),
    "deactivated_description": "karantina total"
}

pond_activation = PondActivation.objects(
    pond_id=pond_id, isFinish=False).order_by('-activated_at').first()

pond_activation.update(**pond_deactivation_data)
pond.update(**{"isActive": False})

elif treatment_type == "ringan":
    body = {
        "pond_id": pond_id,
        "pond_activation_id": pond_activation.id,
        "treatment_type": treatment_type,
        "water_change": 0,
        "salt": request.form.get("salt_dose", None),
        "probiotic_culture": request.form.get("probiotic_culture",
        None),
        "carbohydrate": request.form.get("carbohydrate", None),
        "carbohydrate_type": request.form.get("carbohydrate_type",
        None),
    }
    pondtreatment = PondTreatment(**body).save()
    id = pondtreatment.id

elif treatment_type == "pergantian air":
    body = {
        "pond_id": pond_id,
        "pond_activation_id": pond_activation.id,
        "treatment_type": treatment_type,
    }
```

```

55         "water_change": request.form.get("water_change", 0)
56     }
57     pondtreatment = PondTreatment(**body).save()
58     id = pondtreatment.id
59 else:
60     response = {
61         "message": "treatment type just allow ['ringan','berat']"
62     response = json.dumps(response, default=str)
63     return Response(response, mimetype="application/json", status
=400)
64     response = {
65         "message": "success add data pond treatment", "id": id}
66     response = json.dumps(response, default=str)
67     return Response(response, mimetype="application/json", status=200)
68 except Exception as e:
69     response = {"message": str(e)}
70     response = json.dumps(response, default=str)
71     return Response(response, mimetype="application/json", status=400)

```

Kode di atas adalah implementasi sebuah API untuk menangani permintaan POST pada endpoint PondTreatmentsApi.

Pertama, kode tersebut mengambil nilai pond_id dari formulir permintaan menggunakan request.form.get("pond_id", None). Kemudian, dilakukan pencarian objek Pond berdasarkan pond_id yang diperoleh.

Selanjutnya, dilakukan pengecekan apakah kolom isActive pada objek Pond memiliki nilai False. Jika iya, maka dikembalikan respons dengan pesan "pond is not active" dan status kode 400.

Selanjutnya, dilakukan pencarian objek PondActivation berdasarkan pond_id dan isFinish=False dengan pengurutan berdasarkan activated_at secara menurun menggunakan .order_by('-activated_at').first(). Nilai treatment_type juga diambil dari formulir permintaan.

Kemudian, dilakukan pengecekan terhadap treatment_type. Jika nilainya adalah

"karantina", maka dibentuk sebuah body yang berisi data-data untuk membuat objek PondTreatment dengan beberapa nilai yang diambil dari formulir permintaan. Objek PondTreatment tersebut kemudian disimpan ke database menggunakan .save(). Nilai id dari objek tersebut diambil untuk digunakan sebagai respons.

Selanjutnya, dilakukan pembaruan (update) pada objek PondActivation dan Pond. Data-data yang diperbarui disimpan dalam variabel pond_deactivation_data dan pond_activation yang merupakan objek PondActivation terakhir yang ditemukan. Objek-objek tersebut diupdate menggunakan .update().

Jika treatment_type adalah "ringan", maka langkah yang serupa dilakukan untuk membentuk objek PondTreatment dengan data yang sesuai.

Jika treatment_type adalah "pergantian air", maka juga dilakukan langkah yang serupa untuk membentuk objek PondTreatment dengan data yang sesuai.

Jika treatment_type tidak sesuai dengan ketiga opsi yang diberikan ("karantina", "ringan", "pergantian air"), maka dikembalikan respons dengan pesan "treatment type just allow ['ringan','berat']" dan status kode 400.

Terakhir, jika tidak terjadi kesalahan, respons berhasil dibentuk dengan pesan "success add data pond treatment" dan nilai id dari objek PondTreatment yang telah dibuat. Respons tersebut dikonversi menjadi JSON dan dikembalikan dengan tipe konten "application/json" dan status kode 200. Jika terjadi kesalahan, tangkapan Exception akan menghasilkan pesan kesalahan yang dikirim sebagai respons dengan status kode 400.

Berikut merupakan form untuk entry grading berat ikan antar kolam.

Tabel 4.19: Form entry treatment kolam.

Form	Jenis Data	Deskripsi
pond_id	REQUIRED STRING	id kolam yang akan dilakukan treatment
treatment_type	REQUIRED STRING VALUE : ["ringan", "karantina", "pergantian air"]	tipe treatment yang akan dilakukan
salt	OPTIONAL INT	takaran penambahan garam ke kolam dalam satuan gram
probiotic_culture	OPTIONAL INT	takaran penambahan probiotik ke kolam dalam satuan gram
carbohydrate	OPTIONAL INT	takaran penambahan zat karbon ke kolam dalam satuan gram
carbohydrate_type	REQUIRED IF "carbohydrate" >0 STRING VALUE : ["", "gula", "molase", "terigu", "tapioka"]	tipe zat karbon
description	OPTIONAL STRING	keterangan treatment kolam
total_fish_harvested	REQUIRED IF "treatment_type" == "karantina" INT	jumlah ikan yang di panen pada saat kolam di karantina

Lanjutan Tabel 4.19			
Form	Jenis Data		Deskripsi
total_weight_harvested	REQUIRED "treatment_type" "karantina" INT	IF ==	jumlah berat ikan yang di panen pada saat kolam di karantina dalam satuan Kg
water_change	REQUIRED "treatment_type" "pergantian air" INT	IF ==	persentase pergantian air pada kolam

Tabel di atas merupakan daftar parameter yang dapat digunakan dalam formulir permintaan (request form) pada API PondTreatmentsApi. Setiap kolom dalam tabel memiliki informasi yang relevan terkait jenis data yang diharapkan dan deskripsi dari parameter tersebut. Berikut adalah penjelasan dari setiap kolom:

- (a) pond_id: Parameter ini harus diisi dengan tipe data STRING yang wajib ada.
Parameter ini digunakan untuk mengidentifikasi kolam yang akan diberikan treatment.
- (b) treatment_type: Parameter ini harus diisi dengan tipe data STRING yang wajib ada. Nilai yang dapat diterima adalah "ringan", "karantina", atau "pergantian air". Parameter ini menentukan jenis treatment yang akan dilakukan pada kolam.
- (c) salt: Parameter ini bersifat opsional dan harus diisi dengan tipe data INT.
Parameter ini digunakan untuk menentukan takaran penambahan garam ke dalam kolam dalam satuan gram.
- (d) probiotic_culture: Parameter ini bersifat opsional dan harus diisi dengan tipe data INT. Parameter ini digunakan untuk menentukan takaran penambahan

probiotik ke dalam kolam dalam satuan gram.

- (e) carbohydrate: Parameter ini bersifat opsional dan harus diisi dengan tipe data INT. Parameter ini digunakan untuk menentukan takaran penambahan zat karbon ke dalam kolam dalam satuan gram.
- (f) carbohydrate_type: Parameter ini harus diisi jika nilai dari "carbohydrate" lebih besar dari 0. Parameter ini harus diisi dengan tipe data STRING. Nilai yang dapat diterima adalah "", "gula", "molase", "terigu", atau "tapioka". Parameter ini menentukan jenis zat karbon yang ditambahkan.
- (g) description: Parameter ini bersifat opsional dan harus diisi dengan tipe data STRING. Parameter ini digunakan untuk memberikan keterangan tambahan mengenai treatment kolam.
- (h) total_fish_harvested: Parameter ini harus diisi jika "treatment_type" memiliki nilai "karantina". Parameter ini harus diisi dengan tipe data INT. Parameter ini menentukan jumlah ikan yang dipanen saat kolam dalam kondisi karantina.
- (i) total_weight_harvested: Parameter ini harus diisi jika "treatment_type" memiliki nilai "karantina". Parameter ini harus diisi dengan tipe data INT. Parameter ini menentukan jumlah berat ikan yang dipanen saat kolam dalam kondisi karantina, dalam satuan kilogram.
- (j) water_change: Parameter ini harus diisi jika "treatment_type" memiliki nilai "pergantian air". Parameter ini harus diisi dengan tipe data INT. Parameter ini menentukan persentase pergantian air pada kolam.

Tabel ini memberikan panduan yang berguna dalam menggunakan API PondTreatmentsApi dengan benar, menunjukkan parameter apa yang harus disertakan dalam permintaan dan jenis data yang diharapkan untuk setiap

parameter tersebut.

Berikut merupakan hasil test request dari API entry treatment kolam.

cURL:

```

1 curl --location 'http://jft.web.id/fishapi/api/pondtreatment' \
2 --form 'pond_id="{pond_id}"' \
3 --form 'treatment_type="karantina"' \
4 --form 'description="penyakit ikan sekolam"'

```

response json:

```

1 {
2     "message": "success add data pond treatment",
3     "id": "62f5248cae2842f914ee7797"
4 }

```

4. Implementasi API fetch list treatment kolam

Implementasi controller API fetch list treatment kolam, berikut merupakan source code controller API fetch list treatment kolam.

```

1 # fishapi/resource/pondtreatment.py
2
3 def get(self):
4     try:
5         pipeline = [
6             {"$sort": {"created_at": 1}},
7             {'$lookup': {
8                 'from': 'pond',
9                 'let': {"pondid": "$pond_id"},
10                'pipeline': [
11                    {'$match': {'$expr': {'$eq': ['$id', '$$pondid']} }},
12                    {"$project": {
13                        "_id": 1,
14                        "alias": 1,
15                        "location": 1,
16                        "build_at": 1,
17                        "isActive": 1,
18                    } }
19                ] }
20            ]
21            result = self.db.pondtreatment.aggregate(pipeline)
22            return result
23        except Exception as e:
24            print(e)
25            return {"error": str(e)}
26    
```

```

19         ],
20         'as': 'pond'
21     } },
22     {'$lookup': {
23         'from': 'pond_activation',
24         'let': {"activationid": "$pond_activation_id"},
25         'pipeline': [
26             {'$match': {
27                 '$expr': {'$eq': ['$id', '$activationid']}},
28                 {"$project": {
29                     "_id": 1,
30                     "isFinish": 1,
31                     "isWaterPreparation": 1,
32                     "water_level": 1,
33                     "activated_at": 1
34                 } }
35             ],
36             'as': 'pond_activation'
37         } },
38         {"$addFields": {
39             "pond": {"$first": "$pond"},

40             "pond_activation": {"$first": "$pond_activation"},

41         } },
42         {"$project": {
43             "updated_at": 0,
44             "created_at": 0,
45         } }
46     ]
47     pondtreatment = PondTreatment.objects.aggregate(pipeline)
48     list_pondtreatments = list(pondtreatment)
49     response = json.dumps(list_pondtreatments, default=str)
50     return Response(response, mimetype="application/json", status=200)
51 except Exception as e:
52     response = {"message": str(e)}
53     response = json.dumps(response, default=str)
54     return Response(response, mimetype="application/json", status=400)

```

Kode di atas merupakan implementasi metode `get()` dalam sebuah class yang mengatur API. Metode ini digunakan untuk mengambil data `PondTreatment`

(pengobatan kolam) dari database menggunakan agregasi dengan pipeline MongoDB. Berikut adalah penjelasan langkah-langkah yang dilakukan oleh kode tersebut:

- (a) Sebuah pipeline MongoDB dibuat untuk mengatur tahapan-tahapan pemrosesan data.
- (b) Pada tahap pertama, data akan diurutkan berdasarkan waktu pembuatan (created_at) secara ascending (1).
- (c) Tahap berikutnya adalah melakukan operasi \$lookup untuk melakukan join (penggabungan) dengan koleksi pond. Tahap ini menggabungkan data
- (d) PondTreatment dengan data kolam yang sesuai berdasarkan pond_id. Hasil join ini akan mencakup proyeksi hanya beberapa field yang spesifik.
- (e) Selanjutnya, dilakukan operasi \$lookup kedua untuk melakukan join dengan koleksi pond_activation. Tahap ini menggabungkan data PondTreatment dengan data aktivasi kolam yang sesuai berdasarkan pond_activation_id. Seperti sebelumnya, hasil join ini juga mencakup proyeksi hanya beberapa field yang spesifik.
- (f) Setelah join, dilakukan operasi \$addFields untuk menambahkan dua field baru, yaitu pond dan pond_activation. Nilai dari kedua field ini diambil dari elemen pertama dalam array hasil join.
- (g) Tahap selanjutnya adalah operasi \$project yang digunakan untuk mengatur proyeksi output, dalam hal ini menghilangkan field updated_at dan created_at.
- (h) Setelah pipeline terbentuk, pipeline ini digunakan untuk melakukan agregasi pada koleksi PondTreatment menggunakan metode aggregate(). Hasil agregasi ini akan berupa kursor MongoDB.

- (i) Kursor hasil agregasi diubah menjadi list dengan menggunakan fungsi list().
- (j) Hasil list PondTreatment yang sudah diubah ke dalam format JSON menggunakan json.dumps().
- (k) Hasil akhir dikembalikan sebagai respons dengan tipe Response yang memiliki tipe konten "application/json" dan status kode 200.
- (l) Jika terjadi exception atau kesalahan, akan di-handle dengan mengembalikan respons dengan pesan error yang sesuai dalam format JSON dan status kode 400.

Dengan demikian, kode tersebut melakukan proses pengambilan data PondTreatment dari database menggunakan agregasi dengan pipeline, menggabungkan data dengan koleksi lain, dan mengatur format dan status respons yang dihasilkan.

Berikut merupakan hasil test request dari API fetch list treatment kolam.

cURL:

```
1 curl --location 'http://jft.web.id/fishapi/api/pondtreatment'
```

response json:

```
1 [
2   {
3     "_id": "62f5248cae2842f914ee7797",
4     "pond_id": "62a62163e445ffb9c5f746f3",
5     "pond_activation_id": "62d3f2180d7265ab60f9cb83",
6     "treatment_type": "ringan",
7     "probiotic_culture": 10,
8     "carbohydrate": 10,
9     "carbohydrate_type": "gula",
10    "pond": {
11      "_id": "62a62163e445ffb9c5f746f3",
12      "alias": "charlie",
13      "location": "blok 2",
```

```

14     "build_at": "2022-06-13 00:24:51.473000",
15     "isActive": false
16   },
17   "pond_activation": {
18     "_id": "62d3f2180d7265ab60f9cb83",
19     "isFinish": true,
20     "isWaterPreparation": true,
21     "water_level": 100,
22     "activated_at": "2022-07-17 18:27:20.511000"
23   }
24 },
25 [
26   {
27     "_id": "62f52c34307b0c2008380309",
28     "pond_id": "62a62163e445ffb9c5f746f3",
29     "pond_activation_id": "62d3f2180d7265ab60f9cb83",
30     "treatment_type": "karantina",
31     "carbohydrate_type": "",
32     "description": "penyakit ikan sekolam",
33     "pond": {
34       "_id": "62a62163e445ffb9c5f746f3",
35       "alias": "charlie",
36       "location": "blok 2",
37       "build_at": "2022-06-13 00:24:51.473000",
38       "isActive": false
39     },
40     "pond_activation": {
41       "_id": "62d3f2180d7265ab60f9cb83",
42       "isFinish": true,
43       "isWaterPreparation": true,
44       "water_level": 100,
45       "activated_at": "2022-07-17 18:27:20.511000"
46     }
47   ]

```

5. Implementasi API edit treatment kolam

Implementasi controller API edit treatment kolam, berikut merupakan perubahan source code controller API edit treatment kolam.

```

1 # fishapi/database/pondtreatment.py
2
3 def put(self, id):
4     try:
5         body = request.form.to_dict(flat=True)
6         PondTreatment.objects.get(id=id).update(**body)
7         response = {
8             "message": "success change data pond treatment", "id": id}
9         response = json.dumps(response, default=str)
10        return Response(response, mimetype="application/json", status=200)
11    except Exception as e:
12        response = {"message": str(e)}
13        response = json.dumps(response, default=str)
14        return Response(response, mimetype="application/json", status=400)

```

Kode di atas merupakan implementasi metode `put()` dalam sebuah modul `pondtreatment.py` yang berfungsi untuk memperbarui data `PondTreatment` (pengobatan kolam) dalam database. Berikut adalah penjelasan langkah-langkah yang dilakukan oleh kode tersebut:

- (a) Metode `put()` menerima parameter `id` yang merupakan identifikasi unik dari `PondTreatment` yang akan diperbarui.
- (b) Di dalam blok `try`, pertama-tama dilakukan pengambilan data dari form `request` menggunakan `request.form.to_dict(flat=True)`. Data tersebut dikonversi menjadi bentuk dictionary dengan argumen `flat=True`.
- (c) Selanjutnya, menggunakan metode `get()` dari `PondTreatment.objects` dengan memanfaatkan `id` yang diberikan, data `PondTreatment` yang sesuai diambil dari database.
- (d) Data `PondTreatment` tersebut kemudian diperbarui menggunakan metode `update()` dengan argumen `**body`, yang berarti data pada field-field yang sesuai dalam `PondTreatment` akan diperbarui dengan nilai baru dari `body`

(dictionary yang berisi data dari form request).

- (e) Setelah proses pembaruan selesai, respons berhasil dengan pesan success dan id PondTreatment yang telah diperbarui dibuat.
- (f) Respons tersebut diubah menjadi format JSON menggunakan json.dumps().
- (g) Hasil akhir dikembalikan sebagai respons dengan tipe Response yang memiliki tipe konten "application/json" dan status kode 200.
- (h) Jika terjadi exception atau kesalahan, akan di-handle dengan mengembalikan respons dengan pesan error yang sesuai dalam format JSON dan status kode 400.

Dengan demikian, kode tersebut melakukan proses pembaruan data PondTreatment dengan menggunakan data yang diterima dari form request. Data PondTreatment diambil dari database menggunakan get() dan kemudian diperbarui dengan nilai baru. Respons yang dihasilkan mengindikasikan keberhasilan pembaruan atau kesalahan yang terjadi.

Berikut merupakan hasil test request dari API edit treatment kolam.

cURL:

```

1 curl --location --request PUT 'http://jft.web.id/fishapi/api/pondtreatment/62
2   f5248cae2842f914ee7797' \
3   --form 'salt="20"' \
4   --form 'probiotic_culture="20"' \
5   --form 'carbohydrate="20"' \
5   --form 'carbohydrate_type="molase"'

```

response json:

```

1 {
2   "message": "success change data pond treatment",
3   "id": "62f5248cae2842f914ee7797"
4 }

```

6. Implementasi API delete treatment kolam

Implementasi controller API delete treatment kolam, berikut merupakan perubahan source code controller API delete treatment kolam.

```

1 # fishapi/database/pondtreatment.py
2
3 def delete(self, id):
4     try:
5         pondtreatment = PondTreatment.objects.get(id=id).delete()
6         response = {"message": "success delete pond treatment"}
7         response = json.dumps(response, default=str)
8         return Response(response, mimetype="application/json", status=200)
9     except Exception as e:
10        response = {"message": str(e)}
11        response = json.dumps(response, default=str)
12        return Response(response, mimetype="application/json", status=400)

```

Kode di atas merupakan implementasi metode delete() dalam modul pondtreatment.py yang bertanggung jawab untuk menghapus data PondTreatment (pengobatan kolam) dari database. Berikut adalah penjelasan langkah-langkah yang dilakukan oleh kode tersebut:

Metode delete() menerima parameter id yang merupakan identifikasi unik dari PondTreatment yang akan dihapus. Di dalam blok try, pertama-tama dilakukan pengambilan data PondTreatment dari database menggunakan metode get() dari PondTreatment.objects dengan menggunakan id yang diberikan. Setelah data PondTreatment berhasil diambil, dilakukan penghapusan data tersebut dari database menggunakan metode delete(). Setelah proses penghapusan selesai, respons berhasil dengan pesan success dihasilkan. Respons tersebut diubah menjadi format JSON menggunakan json.dumps(). Hasil akhir dikembalikan sebagai respons dengan tipe Response yang memiliki tipe konten "application/json" dan status kode 200. Jika terjadi exception atau kesalahan,

akan di-handle dengan mengembalikan respons dengan pesan error yang sesuai dalam format JSON dan status kode 400.

Dengan demikian, kode tersebut melakukan proses penghapusan data PondTreatment dari database berdasarkan id yang diberikan. Jika penghapusan berhasil, akan mengembalikan respons dengan pesan keberhasilan. Jika terjadi kesalahan, akan mengembalikan respons dengan pesan error yang sesuai.

Berikut merupakan hasil test request dari API delete treatment kolam.

cURL:

```
1 curl --location --request DELETE 'http://jft.web.id/fishapi/api/pondtreatment/62
e8b800ef4edacc5bb18b05'
```

response json:

```
1 {
2     "message": "success delete pond treatment"
3 }
```

7. Implementasi API fetch detail treatment kolam dengan id

Implementasi controller API fetch detail treatment kolam, berikut merupakan source code controller API fetch detail treatment kolam.

```
1 # fishapi/resource/pondtreatment.py
2
3 def get(self, id):
4     try:
5         pipeline = [
6             {'$match': {'$expr': {'$eq': ['$id', {'$toObjectId': id}]}}},
7             {'$lookup': {
8                 'from': 'pond',
9                 'let': {"pondid": "$pond_id"},
10                'pipeline': [
11                    {'$match': {'$expr': {'$eq': ['$id', '$$pondid']}},
12                    {"$project": {
13                        "_id": 1,
```

```

14         "alias": 1,
15         "location": 1,
16         "build_at": 1,
17         "isActive": 1,
18     } }
19 ],
20     'as': 'pond'
21 },
22 {'$lookup': {
23     'from': 'pond_activation',
24     'let': {"activationid": "$pond_activation_id"},
25     'pipeline': [
26         {'$match': {
27             '$expr': {'$eq': ['$id', '$activationid']}},
28             {"$project": {
29                 "_id": 1,
30                 "isFinish": 1,
31                 "isWaterPreparation": 1,
32                 "water_level": 1,
33                 "activated_at": 1
34             } }
35         ],
36         'as': 'pond_activation'
37     },
38     {"$addFields": {
39         "pond": {"$first": "$pond"}, 
40         "pond_activation": {"$first": "$pond_activation"}, 
41     }},
42     {"$project": {
43         "updated_at": 0,
44         "created_at": 0,
45     } }
46 ]
47 pondtreatment = PondTreatment.objects.aggregate(pipeline)
48 list_pondtreatments = list(pondtreatment)
49 response = json.dumps(list_pondtreatments[0], default=str)
50 return Response(response, mimetype="application/json", status=200)
51 except Exception as e:
52     response = {"message": str(e)}
53 response = json.dumps(response, default=str)

```

54

```
return Response(response, mimetype="application/json", status=400)
```

Kode di atas merupakan implementasi metode get() dalam modul pondtreatment.py yang bertanggung jawab untuk mendapatkan data PondTreatment (pengobatan kolam) berdasarkan id yang diberikan dari database. Berikut adalah penjelasan langkah-langkah yang dilakukan oleh kode tersebut:

- (a) Metode get() menerima parameter id yang merupakan identifikasi unik dari PondTreatment yang ingin didapatkan.
- (b) Di dalam blok try, sebuah pipeline MongoDB disusun untuk melakukan agregasi data PondTreatment berdasarkan id yang diberikan.
- (c) Pipeline ini terdiri dari beberapa tahapan:
 - i. Pertama, dilakukan pencocokan (\$match) berdasarkan ekspresi yang membandingkan _id dengan id yang dikonversi menggunakan \$toObjectId.
 - ii. Selanjutnya, dilakukan operasi \$lookup untuk menggabungkan (join) dengan koleksi "pond". Hasilnya akan disimpan dalam field "pond".
 - iii. Kemudian, dilakukan operasi \$lookup untuk menggabungkan (join) dengan koleksi "pond_activation". Hasilnya akan disimpan dalam field "pond_activation".
 - iv. Selanjutnya, menggunakan \$addFields, field "pond" dan "pond_activation" diatur sebagai nilai pertama (\$first) dari array yang dihasilkan oleh operasi \$lookup, sehingga hanya satu dokumen yang disimpan.
 - v. Terakhir, menggunakan \$project, field "updated_at" dan "created_at" dihilangkan dari hasil akhir.

- (d) Setelah pipeline selesai, dilakukan agregasi data PondTreatment menggunakan PondTreatment.objects.aggregate() dengan pipeline yang telah disusun.
- (e) Hasil agregasi kemudian diubah menjadi list menggunakan list().
- (f) Dalam hal ini, dianggap bahwa hasil agregasi akan menghasilkan satu dokumen PondTreatment. Oleh karena itu, hanya elemen pertama dari list yang diambil.
- (g) Hasil tersebut diubah menjadi format JSON menggunakan json.dumps().
- (h) Hasil akhir dikembalikan sebagai respons dengan tipe Response yang memiliki tipe konten "application/json" dan status kode 200.
- (i) Jika terjadi exception atau kesalahan, akan di-handle dengan mengembalikan respons dengan pesan error yang sesuai dalam format JSON dan status kode 400.

Dengan demikian, kode tersebut melakukan pengambilan data PondTreatment dari database berdasarkan id yang diberikan menggunakan operasi agregasi MongoDB. Jika pengambilan data berhasil, akan mengembalikan respons dengan data PondTreatment dalam format JSON. Jika terjadi kesalahan, akan mengembalikan respons dengan pesan error yang sesuai.

Berikut merupakan hasil test request dari API fetch treatment kolam berdasarkan id.

cURL:

```
curl --location 'http://jft.web.id/fishapi/api/pondtreatment/62  
f5248cae2842f914ee7797'
```

response json:

```
1  {
2      "_id": "62f5248cae2842f914ee7797",
3      "pond_id": "62a62163e445ffb9c5f746f3",
4      "pond_activation_id": "62d3f2180d7265ab60f9cb83",
5      "treatment_type": "ringan",
6      "probiotic_culture": 10,
7      "carbohydrate": 10,
8      "carbohydrate_type": "gula",
9      "pond": {
10         "_id": "62a62163e445ffb9c5f746f3",
11         "alias": "charlie",
12         "location": "blok 2",
13         "build_at": "2022-06-13 00:24:51.473000",
14         "isActive": false
15     },
16     "pond_activation": {
17         "_id": "62d3f2180d7265ab60f9cb83",
18         "isFinish": true,
19         "isWaterPreparation": true,
20         "water_level": 100,
21         "activated_at": "2022-07-17 18:27:20.511000"
22     }
23 }
```

8. Membuat view rekap treatment kolam

No	Tanggal Treatment	Kolam	Masa Budidaya	Tipe Treatment	Pergantian Air	Dosis Garam	Kultur Probiotik	Karbon
1	05-06-2023	beta	1	karantina	100%	-	-	-
2	05-06-2023	charlie	1	ringan	0%	-	10 G/ml	10 (gula)

Gambar 4.21: View list treatment kolam perbulan

B. Uji Sistem

Pengujian sistem dilakukan menggunakan dua tipe yaitu Unit Testing dan User Acceptance Test (UAT). Pengujian dilaksanakan pada saat seluruh User Story pada Product Backlog telah diimplementasikan. Unit testing aplikasi dilakukan terhadap satu frontend developer, sedangkan pengujian User Acceptance Test dilakukan terhadap satu scrum master dan owner.

1. Unit Testing

Unit testing yang dilakukan oleh penulis dilaksanakan setiap sebelum sprint berakhir. Adapun hasil dari unit testing yang telah dilaksanakan dapat dilihat pada tabel di bawah ini:

Tabel 4.20: Unit testing fitur pemberian pakan.

Awal Tabel			
Skenario Pengujian	Kesesuaian		Kesimpulan
	sesuai	tidak sesuai	
API pencatatan pemberian pakan	✓		Diterima
API merubah data pemberian pakan	✓		Diterima
API mendapatkan list data pemberian pakan pada suatu kolam	✓		Diterima
API mendapatkan detail data pemberian pakan	✓		Diterima
API hapus data pemeberian pakan	✓		Diterima

Akhir Tabel 4.20

Tabel 4.21: Unit testing fitur registrasi kolam.

Awal Tabel			
Skenario Pengujian	Kesesuaian		Kesimpulan
	sesuai	tidak sesuai	
API pencatatan registasi kolam	✓		Diterima
API merubah data kolam	✓		Diterima
API mendapatkan list data kolam	✓		Diterima
API mendapatkan detail data kolam	✓		Diterima
API hapus data kolam	✓		Diterima

Akhir Tabel 4.21

Tabel 4.22: Unit testing fitur musim budidaya kolam.

Awal Tabel			
Skenario Pengujian	Kesesuaian		Kesimpulan
	sesuai	tidak sesuai	
API memulai musim budidaya pada kolam	✓		Diterima
API mengakhiri musim budidaya atau panen pada kolam	✓		Diterima
API mendapatkan list status kolam	✓		Diterima
API mendapatkan list musim budidaya pada suatu kolam	✓		Diterima
Akhir Tabel 4.22			

Tabel 4.23: Unit testing fitur kematian ikan.

Awal Tabel			
Skenario Pengujian	Kesesuaian		Kesimpulan
	sesuai	tidak sesuai	
API pencatatan kematian ikan	✓		Diterima
API merubah data kematian ikan	✓		Diterima
API mendapatkan list data kematian ikan pada suatu musim budidaya	✓		Diterima
API mendapatkan detail kematian ikan	✓		Diterima
API hapus kematian ikan	✓		Diterima
Akhir Tabel 4.23			

Tabel 4.24: Unit testing fitur perpindahan ikan antar kolam.

Awal Tabel			
Skenario Pengujian	Kesesuaian		Kesimpulan
	sesuai	tidak sesuai	
API pencatatan perpindahan ikan antar kolam	✓		Diterima
API merubah data perpindahan ikan antar kolam	✓		Diterima
API mendapatkan list data perpindahan ikan pada suatu musim budidaya	✓		Diterima
API mendapatkan detail perpindahan ikan	✓		Diterima
API hapus data perpindahan ikan	✓		Diterima

Akhir Tabel 4.24

Tabel 4.25: Unit testing fitur grading berat ikan.

Awal Tabel			
Skenario Pengujian	Kesesuaian		Kesimpulan
	sesuai	tidak sesuai	
API pencatatan grading berat ikan	✓		Diterima
API merubah data grading berat ikan	✓		Diterima
API mendapatkan list data grading berat ikan pada suatu musim budidaya	✓		Diterima
API mendapatkan detail grading berat ikan	✓		Diterima
API hapus data grading berat ikan	✓		Diterima

Akhir Tabel 4.25

Tabel 4.26: Unit testing fitur pencatatan kualitas air harian.

Awal Tabel			
Skenario Pengujian	Kesesuaian		Kesimpulan
	sesuai	tidak sesuai	
API pencatatan kualitas air harian kolam	✓		Diterima
API merubah data kualitas air harian kolam	✓		Diterima
API mendapatkan list data kualitas air harian kolam pada suatu musim budidaya	✓		Diterima
API mendapatkan detail kualitas air harian	✓		Diterima
API hapus data kualitas air harian kolam	✓		Diterima

Akhir Tabel 4.26

Tabel 4.27: Unit testing fitur pencatatan kualitas air mingguan.

Awal Tabel			
Skenario Pengujian	Kesesuaian		Kesimpulan
	sesuai	tidak sesuai	
API pencatatan kualitas air mingguan kolam	✓		Diterima
API merubah data kualitas air mingguan kolam	✓		Diterima
API mendapatkan list data kualitas air mingguan kolam pada suatu musim budidaya	✓		Diterima
API mendapatkan detail kualitas air mingguan	✓		Diterima
API hapus data kualitas air mingguan kolam	✓		Diterima

Akhir Tabel 4.27

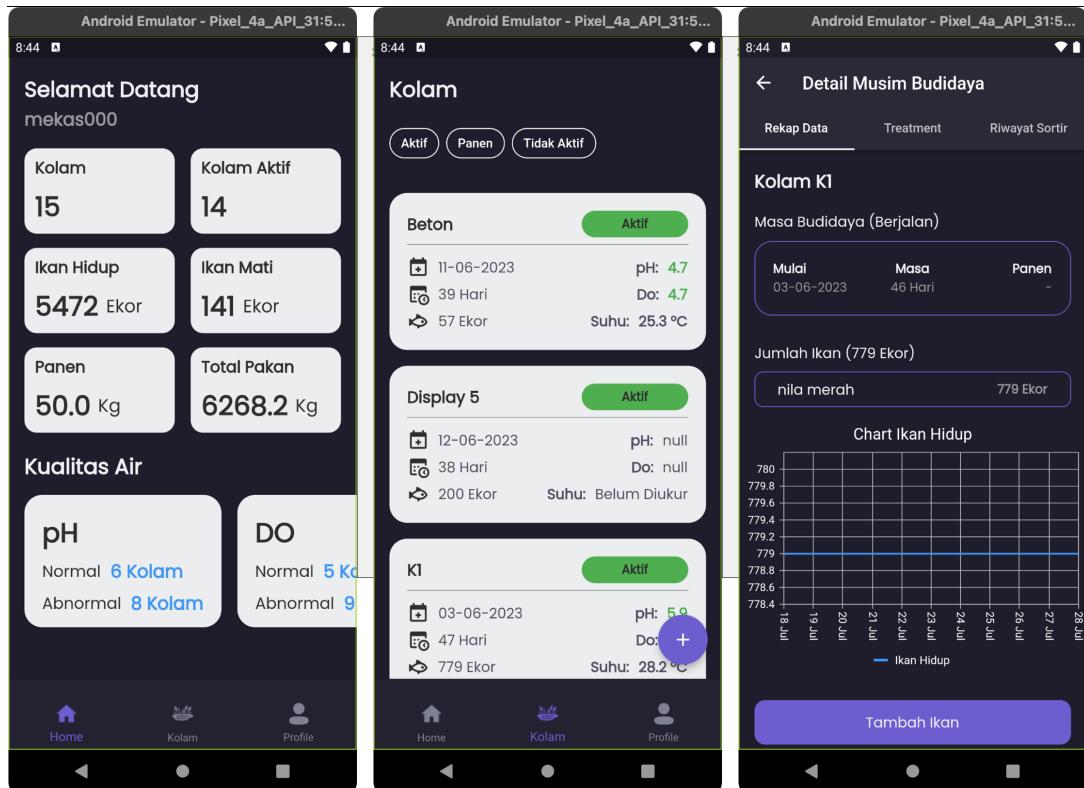
Tabel 4.28: Unit testing fitur pencatatan treatment.

Awal Tabel			
Skenario Pengujian	Kesesuaian		Kesimpulan
	sesuai	tidak sesuai	
API pencatatan treatment kolam	✓		Diterima
API merubah data treatment kolam	✓		Diterima
API mendapatkan list data treatment kolam pada suatu musim budidaya	✓		Diterima
API mendapatkan detail treatment kolam	✓		Diterima
API hapus data treatment kolam	✓		Diterima
Akhir Tabel 4.28			

2. Pengujian Web Service

Pengujian Web Service merupakan pengujian yang dilakukan oleh frontend developer yang akan menggunakan API pada aplikasinya. Dalam khusus ini API digunakan pada aplikasi Aqua Breeding yang telah sedang dikembangkan dipenelitian lain yang sedang berjalan bersama penelitian ini. Pengujian ini bersifat berkelanjutan sesuai progress penelitian aplikasi yang dilakukan.

Pengujian dilakukan secara luring dengan frontend developer pada tanggal 16 juni 2023 yang bertempat di suatu coffee shop jakarta. Pengujian berjalan lancar dengan beberapa sedikit penyesuaian pada API. Penyesuaian bersifat minor pada respon beberapa API agar data yang dikirim sesuai dengan UI & UX aplikasi yang sudah dibuat. Berikut merupakan beberapa tampilan halaman sistem Aqua Breeding berbasis android yang telah mengimplementasikan API yang telah dibuat.



Gambar 4.22: Tampilan Aqua Breeding berbasis android

3. Pengujian UAT (User Acceptance Test)

User Acceptance Test terhadap scrum master dilaksanakan pada tanggal 7 Desember 2022 secara luring bertempat di Gedung Dewi Sartika Universitas Negeri Jakarta. Adapun hasil dari UAT yang telah dilaksanakan dapat dilihat pada tabel di bawah ini:

Tabel 4.29: Daftar pengujian UAT.

Skenario Pengujian UAT	Awal Tabel		Kesimpulan
	Kesesuaian sesuai	Kesesuaian belum	
Tampilan tabel pencatatan pemberian pakan keseluruhan		✓	Butuh Penyesuaian
Tampilan tabel pencatatan pemberian pakan harian	✓		Diterima
Tampilan tabel pencatatan pemberian pakan bulanan	✓		Diterima
Tampilan tabel kolam yang sudah di registrasi		✓	Butuh Penyesuaian
Tampilan tabel masa budidaya kolam	✓		Diterima
Tampilan tabel jumlah Ikan setiap	✓		Diterima
Tampilan tabel data kematian ikan		✓	Butuh Penyesuaian
Tampilan tabel data sortir ikan	✓		Diterima
Tampilan tabel data grading berat ikan	✓		Diterima
Tampilan tabel data kualitas air harian		✓	Butuh Penyesuaian
Tampilan tabel data kualitas air mingguan		✓	Butuh Penyesuaian
Tampilan tabel data treatment kolam	✓		Diterima
Akhir Tabel 4.29			

Penyesuaian pada UAT bersifat minor seperti penambahan kolom, menambahkan keterangan satuan ukur, dan rata - rata data di dalam suatu kolam. Perbaikan ini dikerjakan selama satu minggu setelah UAT. Pertemuan dilakukan lagi setelah satu minggu, yaitu pada tanggal 14 Desember 2023 untuk melaporkan perbaikan kepada scrum master.

BAB V

KESIMPULAN DAN SARAN

A. Kesimpulan

Berdasarkan hasil implementasi dan pengujian fitur sistem web service yang telah dirancang, maka diperoleh kesimpulan sebagai berikut:

1. Terimplementasikannya prototipe backend web service aqua breeding versi pertama, yang berfokus kepada fitur pencatatan dan visualisasi data budidaya ikan. Adapun perancangannya dilakukan dengan metode Scrum dengan tahapan penyusunan Product Backlog, Sprint Backlog, dan dikerjakan dalam sepuluh Sprint.
2. Berdasarkan hasil pengujian web service yang dilakukan terhadap frontend developer, didapatkan bahwa API berhasil diimplementasikan kedalam prototipe aplikasi aqua breeding berbasis android.
3. Berdasarkan hasil User Acceptance Test terhadap satu pembudidaya, didapatkan bahwa prototipe API dan tampilan admin versi pertama sudah sesuai dan menghasilkan kesimpulan untuk fitur yang akan di tambahkan pada versi kedua.

B. Saran

Adapun saran untuk penelitian selanjutnya adalah:

1. Berdasarkan diskusi dengan owner, harus dimulainya pengembangan frontend sistem dengan mengintegrasikan API yang sudah di buat ke aplikasi aqua breeding berbasis android agar sistem dapat dipakai oleh para pembudidaya.

2. Berdasarkan diskusi dengan owner, pada versi selanjutnya menambahkan fitur multi farm, multi user, login, dan logout secara API maupun aplikasi. Agar dapat dirilis dan dapat dipakai secara umum oleh pembudidaya lain.

LAMPIRAN A

Transkrip Percakapan

Hari: Selasa

Tanggal: 19 April 2022

PL: Penulis

KL: Klein(Pemilik Farm)

PL: Sistem apa yang akan di buat?

KL: Kita akan membuat *Backend server* berbentuk *REST API* untuk perikanan modren

PL: Apa saja requirement dan fitur yang dibutuhkan oleh sistem ini?

KL: Fitur utama yang harus ada adalah pencatatan pemberian pakan, registrasi kolam, aktivasi-deaktivasi kolam, pencatatan kualitas air harian dan mingguan, pencatatan data kematian harian, pencatatan treatment kolam, *grading* berat ikan kolam, perpindahan ikan antar kolam

PL: Dimulai dari manakah penggerjaan fitur2 tersebut?

KL: Dimulai dari pemberian pakan

/Users/andrirahmanto/flask/fishapigian/fishapiv2/resources/controller/fishsort.py

DAFTAR PUSTAKA

- Adani, M. R. (2020). Pengenalan apa itu framework dan jenisnya untuk web development. <https://www.sekawanmedia.co.id/blog/pengertian-framework>.
- Aep Setiawan, Erlin Arlitasari, M. Z. A. H. (2022). Monitoring pemberian pakan ikan otomatis menggunakan iot di labolatorium perikanan sekolah vokasi ipb. *JINTEKS (Jurnal Informatika Teknologi dan Sains)*.
- Ahmad Rifa'i, M. Udin Harun Al Rasyid, A. I. G. (2021). Sistem pemantauan dan kontrol otomatis kualitas air berbasis iot menggunakan platform node-red untuk budidaya udang. *JTT (Jurnal Teknologi Terapan)*.
- Alim, H. (2022). Fish movement tracking dengan menggunakan metode gmm dan kalman filter. *Program Studi Ilmu Komputer Fakultas Matematika Dan Ilmu Pengetahuan Alam Universitar Negeri Jakarta 2022*.
- Dhita Widhiastika, Sobakhul Munir Siroj, A. W. U. B. A. P. N. F. R. (2021). Perancangan aplikasi jual beli produk perikanan berbasis mobile android (studi kasus : Fo-klik). *Jurnal Ilmu Perikanan dan Kelautan Indonesia*.
- Foundation, A. S. (2022). Apache http server. <https://httpd.apache.org/>.
- Hadi, F. P. (2021). Rancang bangun web service dan website sebagai storage engine dan monitoring data sensing untuk budidaya ikan air tawar. *Program Studi Ilmu Komputer Fakultas Matematika Dan Ilmu Pengetahuan Alam Universitar Negeri Jakarta 2021*.
- Hady, E. L., Haryono, K., dan Rahayu, N. W. (2020). User acceptance testing (uat) pada purwarupa sistem tabungan santri (studi kasus: Pondok pesantren al-mawaddah). *Jurnal Ilmiah Multimedia dan Komunikasi*, 5(1).

- Hamilton, T. (2022). Unit testing tutorial – what is, types and test example.
<https://www.guru99.com/unit-testing-guide.html>.
- Hema Krishnan, T. Santhanakrishnan, M. E. (2016). Mongodb – a comparison with nosql databases). *International Journal of Scientific and Engineering Research*.
- infishta (2019). Jenis-jenis budaya ikan air tawar. <https://infishta.com/blogs/jenis-jenis-budidaya-ikan-air-tawar>.
- Kevin Burke, Kyle Conroy, R. H. (2020). Flaskrestful. <https://flask-restful.readthedocs.io/en/latest>.
- Kunda, D. dan Phiri, H. (2017). A comparative study of nosql and relational database.
- Nugraha, B. (2022). Ekstrasi latar depan pada citra ikan dengan metode grabcut yang diautomasi menggunakan saliency map. *Program Studi Ilmu Komputer Fakultas Matematika Dan Ilmu Pengetahuan Alam Universitar Negeri Jakarta 2022*.
- Perry, W. (2006). *Effective Methods for Software Testing, Third Edition*. John Wiley, Sons, Inc., USA.
- Pramleonita, M., Yuliani, N., Arizal, R., dan Wardoyo, S. E. (2018). Parameter fisika dan kimia air kolam ikan nila hitam (*oreochromis niloticus*). *Jurnal Sains Natural*.
- Pressman, R. S. (2012). Rekayasa perangkat lunak pendekatan praktisi. *Yogyakarta: Andi, 2010.*
- Purnama, S. (2013). Metode penelitian dan pengembangan (pengenalan untuk mengembangkan produk pembelajaran bahasa arab). *LITERASI*.
- Rasouli, M. (2020). Flask route a complete overview. <https://medium.com/analytics-vidhya/flask-route-a-complete-overview-d66065bfa867>.

- Rosa, A. S. (2016). Rekayasa perangkat lunak terstruktur dan berorientasi objek.
- Ross Lawley, Marr, B. G. (2020). Mongoengine user documentation.
<https://docs.mongoengine.org>.
- Streetlife (2020). Flask-mongoengine documentation.
<http://docs.mongoengine.org/projects/flask-mongoengine/en/latest>.
- Supriyati, D. M. R. (2018). Model perancangan sistem informasi akuntansi budidaya perikanan berbasis sak emkm dan android. *Program Studi Komputerisasi Akuntansi Fakultas Teknik dan Ilmu Komputer-Universitas Komputer Indonesia*.
- Yudhis Thiro Kabul Yunior, K. (2019). Sistem monitoring kualitas air pada budidaya perikanan berbasis iot dan manajemen data. *Creative Information Technology Journal (CITEC JOURNAL)*.