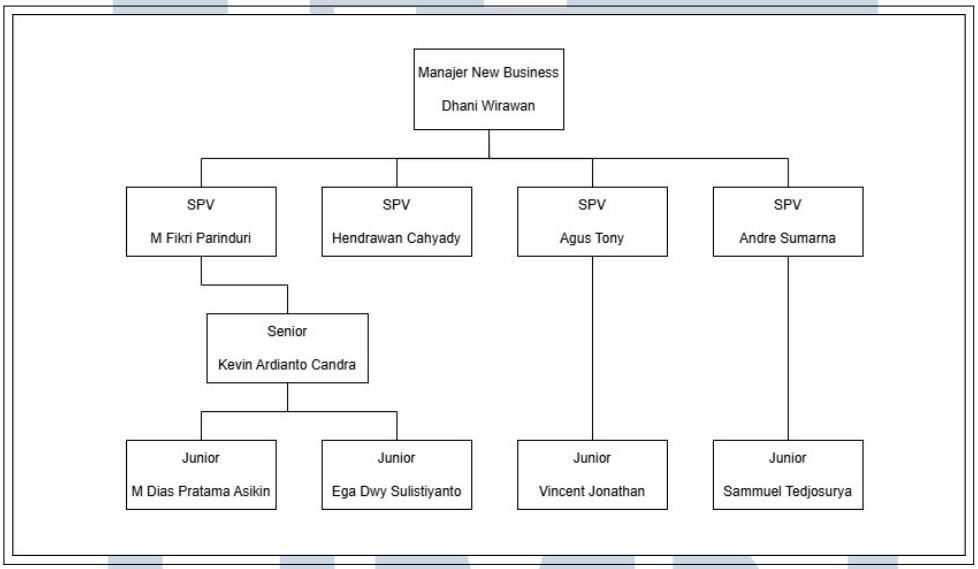
BAB 3
PELAKSANAAN KERJA MAGANG

3.1 Kedudukan dan Koordinasi

Pelaksanaan program magang dilaksanakan mulai tanggal 14 Januari 2025 hingga 14 Juni 2025 di PT Prima Solusi Computindo, dengan penempatan pada Tim *New Business* di bawah arahan langsung Pak Andre Sumarna selaku *Supervisor*. Struktur organisasi tim *New Business* dapat dilihat pada Gambar 3.1.



Gambar 3.1. Struktur organisasi perusahaan PT Prima Solusi Computindo

Pada pelaksanaan magang ini, saya ditempatkan sebagai *Web Developer Intern* dengan fokus utama pada *Design UI/UX* dan *Fullstack Developer*. Dalam melaksanakan tugas, saya bekerja sama secara intensif dengan *Supervisor* dan *Junior*, mengikuti prosedur kerja perusahaan yang menggunakan alat komunikasi Discord sebagai alat untuk berkoordinasi.

3.2 Tugas yang Dilakukan

Selama menjalani program magang di PT Prima Solusi Computindo, tanggung jawab yang diberikan adalah mengerjakan *project* LPDP, DAPEN BI dan DIFA. Produk tersebut adalah sebuah *platform* berbasis *web* yang menyediakan data pasar dan digunakan oleh tim *back office* perusahaan dalam mendukung kegiatan investasi. Tugas pertama adalah bertugas sebagai *fullstack developer* yaitu

melakukan pengembangan fitur produk LPDP dan DAPEN BI. Selain LPDP dan DIFA, tugas kedua adalah merancang dan mendesain tampilan DIFA dimulai dari mencari referensi, *design prototype*, sampai implementasi *website*. Tugas yang dilakukan setiap minggu dapat dilihat pada Tabel 3.1.

Tabel 3.1. Pekerjaan yang dilakukan tiap minggu selama pelaksanaan kerja magang

Minggu Ke -	Pekerjaan yang dilakukan
1	Pengenalan *Jobdesk*, sistem kerja, dan *project* yang akan dikerjakan.
2	Pemahaman tahap awal terhadap sistem *project* seperti bahasa yang digunakan, *flow web*, dan fungsi web.
3	Mencari *reference* dan menganalisis web lain dan web yang ingin dirombak, mulai dari *flow*, bentuk dan warna.
4	Pembuatan halaman daftar reksadana dengan figma.
5	Pembuatan isi dari halaman daftar reksadana dengan figma.
6	Pembuatan *flow* dan *prototype* web dengan figma.
7	Implementasi desain figma ke *codingan* dengan html, css, dan bootstrap.
8	Pembuatan *sidebar website*.
9	Pembuatan tabel dengan kendoGrid.
10	Pembuatan *chart* dengan ApexChart.
11	Menghubungkan web dengan API yang sudah ada.
12-17	Pengerjaan task LPDP dan DAPENBI.

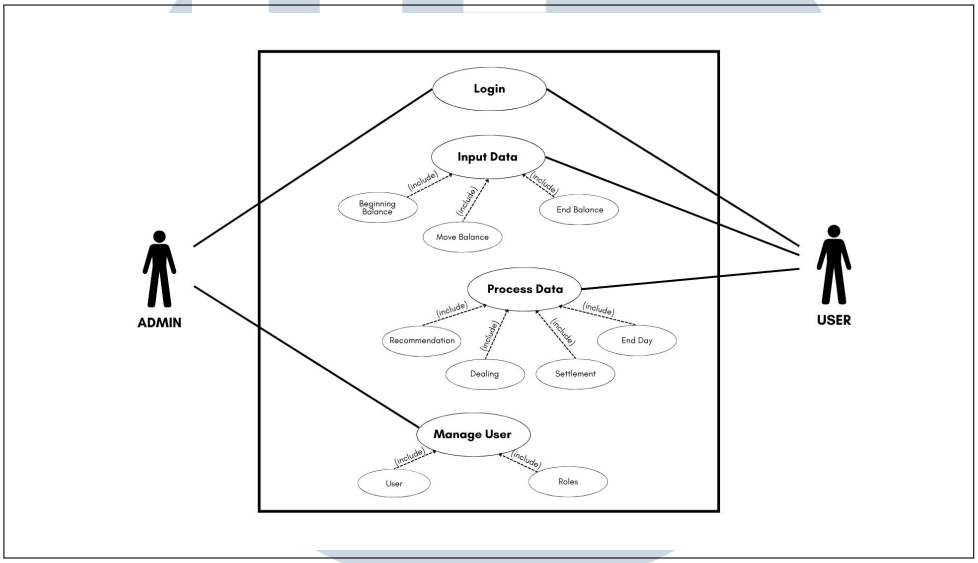
3.3 Uraian Pelaksanaan Magang

3.3.1 LPDP

LPDP merupakan salah satu klien yang menggunakan jasa dari RADSoft. Web yang dibuat untuk LPDP adalah web pendukung investasi yang digunakan untuk mencatat dan mengawasi kegiatan yang berhubungan dengan investasi. Pada *project* LPDP ini, tugas yang diberikan berupa pemeliharaan dan pengembangan fitur-fitur web baik yang sudah ada maupun yang akan ditambahkan. Dalam task ini, *developer* menerima arahan dari *implementor* untuk menambahkan atau memperbaiki beberapa kasus, fitur, maupun *bug* yang terdapat pada web LPDP. Tugas ini memerlukan keterampilan di bidang *Fullstack Development*, khususnya

dalam penggunaan bahasa pemrograman *JavaScript*, *CSharp*, pengelolaan *query* basis data dan OOP.

A Usecase Diagram LPDP

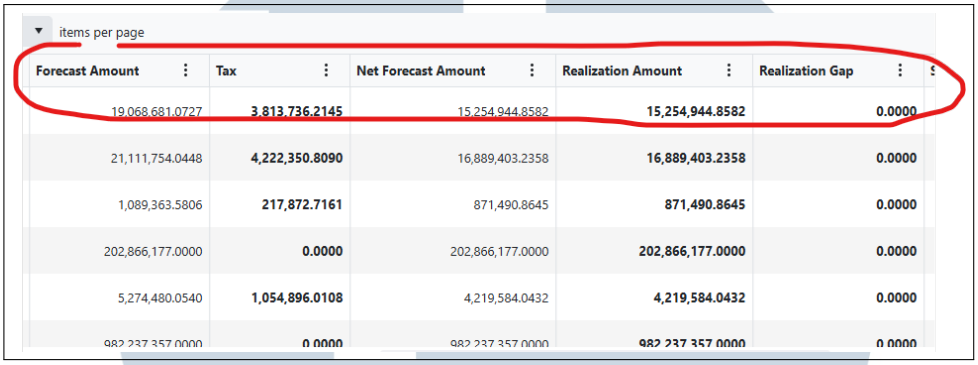


Gambar 3.2. Usecase Diagram LPDP

Usecase Diagram LPDP pada Gambar 3.2 menggambarkan cara kerja web LPDP. Pada web LPDP terdapat dua aktor yaitu *Admin* dan *User*. *Admin* dapat melakukan *login* dengan *user admin*. *Admin* juga dapat melakukan pengaturan pada *Management User*. *Management User* dapat mengatur *User* dan *Roles* untuk *user*. *User* dapat melakukan *Input* data yang dimana *input* data ini dibagi menjadi 3 yaitu *beginning balance*, *move balance* dan *end balance*. Lalu *user* dapat mengakses *process* data dengan *output Recommendation*, *Dealing*, *Settlement* dan *End day*.

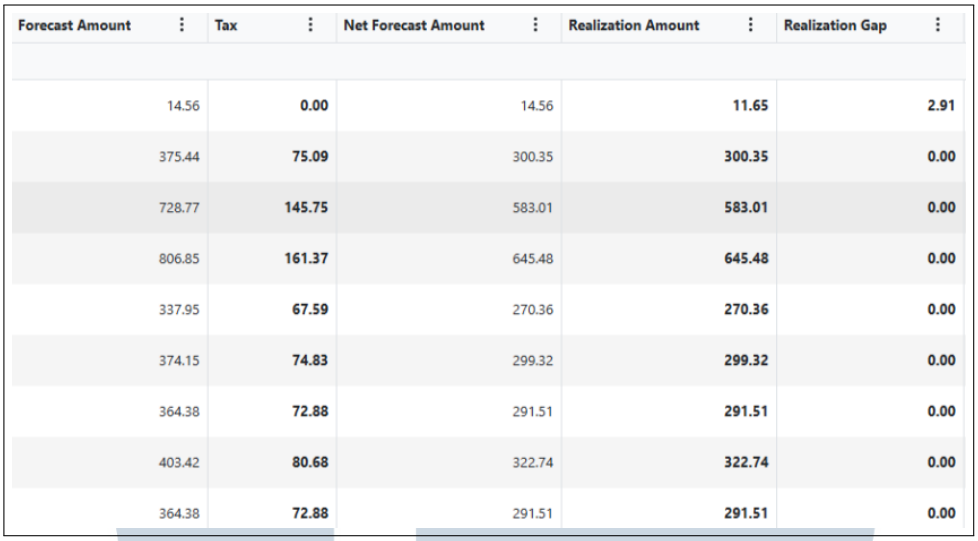
B Task yang dikerjakan

B.1 Merubah Decimal Digit Pada Menu Cash Recon



Gambar 3.3. Tampilan Grid sebelum perubahan

Gambar 3.3 memperlihatkan tampilan *grid* pada kolom *Forecast Amount*, *Tax*, *Net Forecast Amount*, *Realization Amount*, dan *Realization Gap* masih menampilkan 4 digit dibelakang koma. Pada kasus ini, *user* meminta untuk tampilan yang pada *grid* di beberapa kolom tersebut untuk dirubah menjadi dua digit *decimal*. Langkah pertama yang dilakukan adalah mencari menu yang ingin dirubah yaitu *Cash Recon*, *file* yang perlu dirubah terkait tampilan pada *grid* yaitu *Javascript*. Lalu pada file tersebut terdapat sebuah fungsi *grid* yang dibuat dari KendoGrid yang di dalam nya terdapat sebuah variabel *grid options*. Di dalam *Grid Options* terdapat sebuah fungsi *number format* yang dapat memberikan jumlah decimal digit yang ingin ditampilkan, *number format* tersebut lalu dirubah yang awal nya '0:n4' menjadi '0:n2'. Setelah selesai decimal digit pada *field* tersebut akan menjadi 2 saja seperti Gambar 3.4



Forecast Amount	Tax	Net Forecast Amount	Realization Amount	Realization Gap
14.56	0.00	14.56	11.65	2.91
375.44	75.09	300.35	300.35	0.00
728.77	145.75	583.01	583.01	0.00
806.85	161.37	645.48	645.48	0.00
337.95	67.59	270.36	270.36	0.00
374.15	74.83	299.32	299.32	0.00
364.38	72.88	291.51	291.51	0.00
403.42	80.68	322.74	322.74	0.00
364.38	72.88	291.51	291.51	0.00

Gambar 3.4. Tampilan Grid sesudah perubahan

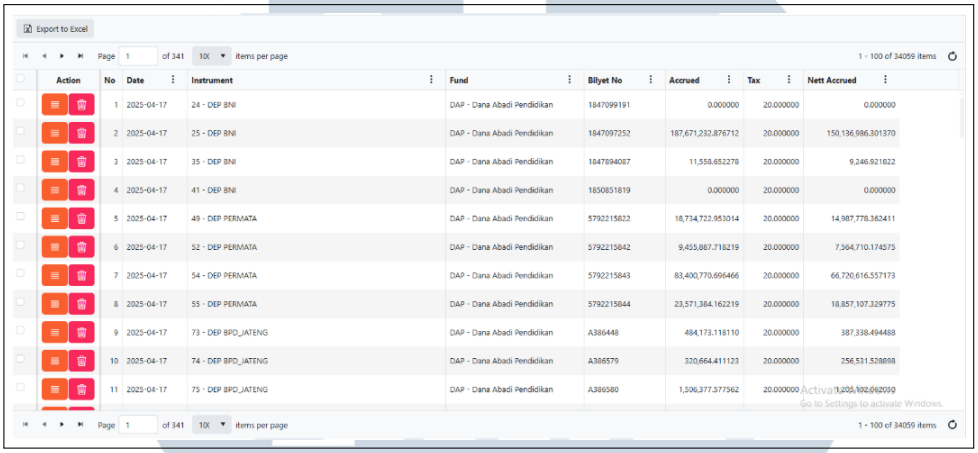
B.2 Perbaikan Bug - Pop Up Approve Failed Dealing Bond

Pada Menu *Bond Proposal* terdapat sebuah fitur yang dapat meng-*approve Dealing Bond* namun pada saat *approve* gagal dilakukan notifikasi *pop up* pada fitur ini tidak muncul. Langkah awal untuk menyelesaikan kasus ini adalah dengan mencari letak bug tersebut, karena bug ini terjadi pada bagian *frontend* dan fungsinya yang bermasalah sehingga tidak muncul notifikasi, maka pada file *Javascriptnya* adalah kemungkinan terbesarnya. Lalu untuk mengetahui apakah benar salah satu fungsi pada *Javascript* tersebut terdapat kekeliruan adalah dengan menyematkan *debugger*. *Debugger* ditaruh pada sebuah fungsi yang dapat memunculkan *warning* pada fitur tersebut apabila *approve* dinyatakan gagal. Ketika berhasil ditemukan sumber masalah tersebut, ternyata pada bagian *try catch error*, terdapat *syntax* yang kurang benar seperti gambar dibawah ini.

B.3 Penambahan Fitur Export to Excel pada Menu Accrued Deposito

Pada kasus ini, *user* meminta untuk menu Accrued Deposito ditambahkan fitur *Export to Excel* seperti pada Gambar 3.5 dan data yang di *export* adalah semua data yang ada pada halaman tersebut. Karena pada kasus ini data yang ingin di *export* adalah semua data yang ada maka untuk fitur ini dapat menggunakan *library UI* dari kendo yang dapat membuat fitur ini. Pada file *javascript*, terdapat sebuah fungsi dari KendoGrid yang berfungsi untuk menampilkan data dalam bentuk tabel atau *grid*. Pada fungsi tersebut, untuk menambahkan tombol *export*

to excel perlu menambahkan '*toolbars*: ' dengan isi '[*excel*]' dan di dalam '*excel*' perlu menambahkan '*fileName*:' untuk nama file *export*, '*filterable*:' untuk menambahkan fitur *filter* pada *header* tabel dan '*allPages*:' untuk menampilkan seluruh data.



Gambar 3.5. Tampilan Grid dengan fitur export to excel

B.4 Industry List Summary Export to Excel

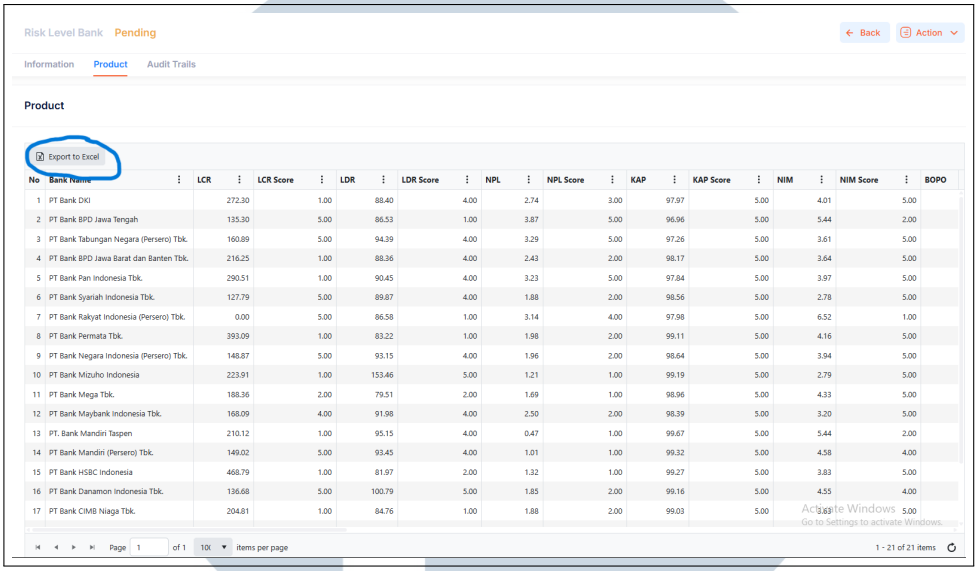
Pada Menu ini *user* meminta agar Menu *Industry List Summary* ditambahkan fitur *Export to Excel* dikarenakan Menu ini tidak dapat mengakses DIFA. *User* meminta agar seluruh data pada *grid* ditampilkan pada *Excel* maka untuk menyelesaikan task ini dapat menggunakan *library* dari Kendo yang telah menyediakan fitur untuk *Export* seluruh data KendoGrid ke *Excel*.

```
function initGrid() {
  const gridId = 'grid_industry_summary_df';
  const gridOptions = {
    dataSource: { get: getIndustrySummaryDf },
    heightAdj: 380,
    defaults: {
      numberFormat: '{0:n2}',
      width: '450px',
    },
    autoFitColumn: true,
    topPager: true,
    rowNumber: 1,
    toolbars: ['excel'],
    excel: {
      fileName: 'Industry List Summary.xlsx',
      filterable: true,
      allPages: true,
    },
```

Gambar 3.6. Pecahan codingan export to excel Kendo

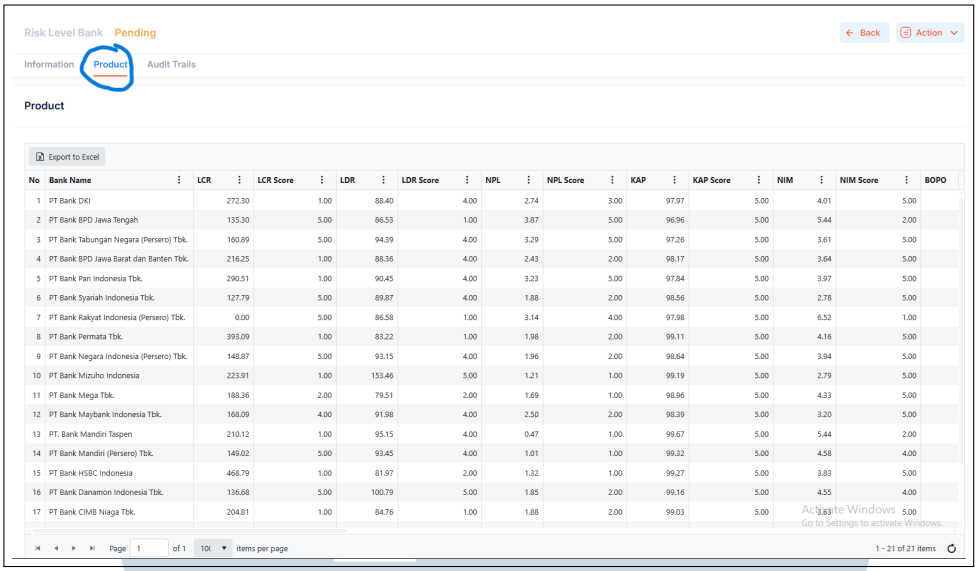
Gambar 3.6 menunjukkan sebuah pecahan *codingan* dari *function Initgrid* yang dimana *function* ini berfungsi untuk inisialisasi *grid*. Pada *codingan* tersebut terdapat properti *Toolbars Excel* yang berfungsi untuk menambahkan *toolbar* dengan tombol *ekspor* ke *Excel*. Lalu dibawahnya terdapat *Filename* untuk nama file *output*, *Filterable* untuk menambahkan fitur *filter* pada *header* tabel, dan *allPages* untuk mengekspor semua data (bukan hanya halaman yang terlihat)..

B.5 Risk Level Bank - Tambahkan fitur Ekspor ke Excel dan Memindahkan Grid



Gambar 3.7. Tampilan grid dengan fitur export to excel

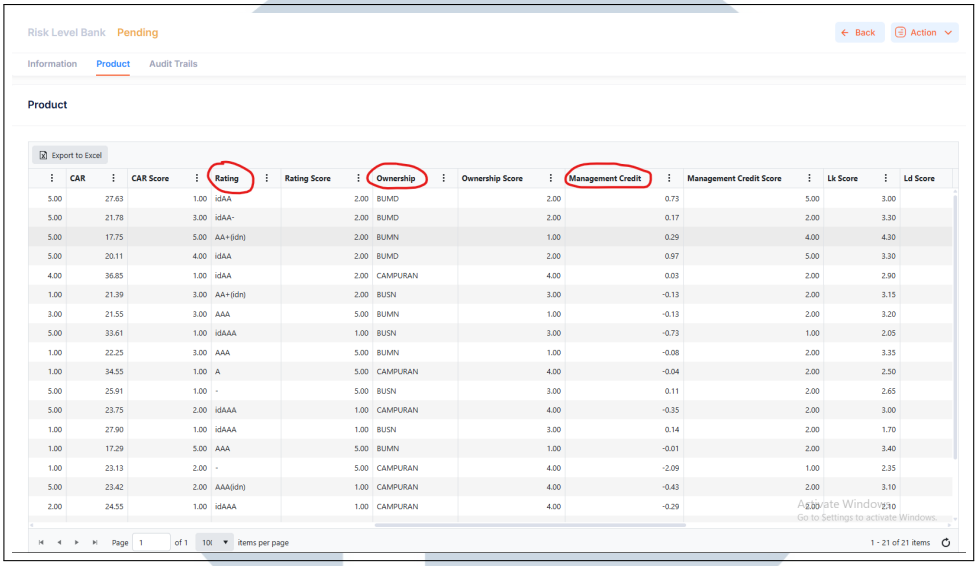
Pada task ini *user* meminta agar Menu *Risk Level Bank* ditambahkan fitur *Export to Excel* seperti pada Gambar 3.7 dan *Grid Bank* pada *page detail* dipindahkan ke *tab product*. Untuk penambahan fitur *Export to Excel* dapat menggunakan *library* dari Kendo dengan penambahan pada KendoGrid. Setelah menambahkan *Toolbars Excel*, tombol *Export to Excel* akan muncul pada bagian atas kanan *grid*. Fungsi tombol tersebut akan berfungsi setelah menambahkan *fileName*, *filterable* dan *allPages* pada properti *excel*.



Gambar 3.8. Hasil setelah grid dipindahkan ke tab product

Untuk perpindahan *grid bank* pada *page detail* seperti pada Gambar 3.8, tahap pertama yang dilakukan adalah dengan membuat *tab product* terlebih dahulu. *Tab product* dibuat dengan *Bootstrap card* dan *tab* yang ditambahkan di samping *tab Information*. Tombol *tab* dibuat dengan *font color* berwarna biru dan ketika tombol di klik, tombol akan membuka *card product*. Lalu pada *body card* ditambahkan *Grid Bank* yang diambil dari *tab Information* dengan menambah *id grid* pada *card body*. Untuk memindahkannya dibutuhkan beberapa penyesuaian dengan membuat *class product* pada file *js*. *Class product* dibuat agar *product* dibuat lebih *ekslusif* sehingga pada saat ingin memanggil fungsi pada *product* tidak bentrok. Lalu data ditarik ketika *user* membuka *tab product*, *grid* di *fetch* pada saat data ditarik dari API.

B.6 Risk Level Bank - Tampilkan Value Rating, Kepemilikan, dan Manajemen Kredit



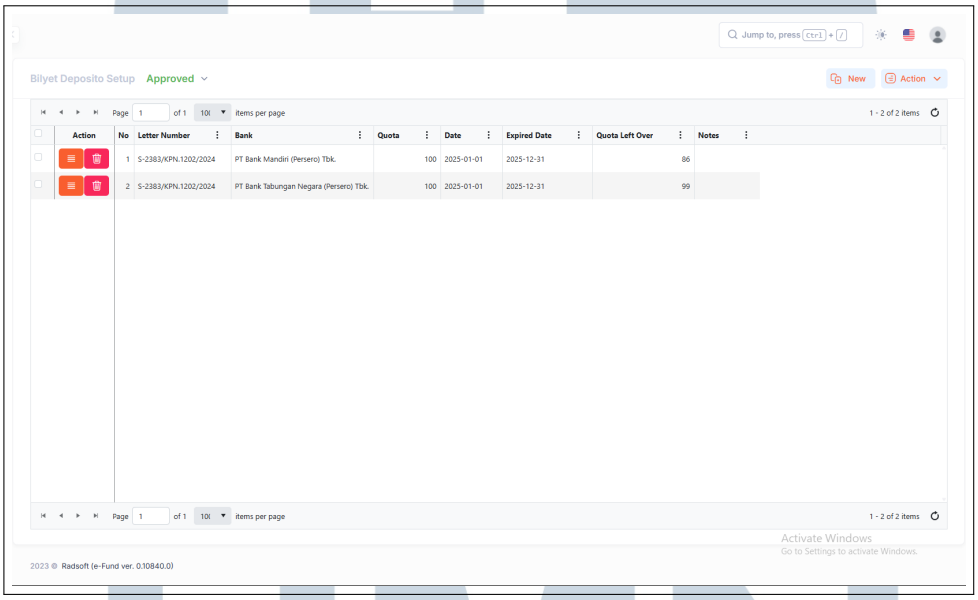
Gambar 3.9. Hasil setelah grid ditambahkan field

Task ini adalah lanjutan dari task sebelumnya. Pada task ini terdapat kebutuhan *user* untuk menambahkan dan menampilkan *field Value Rating*, Kepemilikan atau '*Ownership*' dan Manajemen Kredit pada *index Grid* seperti pada Gambar 3.9. Ketiga *field* tersebut diambil dari beberapa tabel yang berbeda, *field Value Rating* diambil dari kolom *Rating* di tabel *Bank Rating*, *field* Kepemilikan diambil kolom *Bank Category* dari tabel *Bank*, dan *field* Manajemen Kredit diambil dari hasil pengurangan NPL *Gross* tahun ini dengan tahun lalu, setelah itu dibuat menjadi kolom *score*. Tahap pertama yang harus dilakukan adalah dengan menambahkan '*Rating*', '*Ownership*' dan MCredit pada file *Views*. File *Views* adalah file yang digunakan untuk membawa data dari *backend* ke *frontend* yang telah di proses. Setelah itu pada *query* pengambilan data *grid* di *index*, diperlukan menambahkan *filter* pengambilan data hanya yang bertipe DEPO. Lalu untuk *query* pengambilan data *grid product* ditambahkan *field* '*Rating*', '*Ownership*' dan '*MCredit*' dengan *json*. Di dalam *json* sudah terdapat ketiga data *field* tersebut sehingga diperlukan memanggil *value* tersebut saja yang terletak di dalam *json*. Peletakan ketiga *field* tersebut juga diletakkan disebelah *field score* mereka. Tidak lupa untuk menambahkan *field* tersebut ke *Frontend* persisnya di KendoGrid *product* agar KendoGrid dapat menarik data dari *Backend* sesuai dengan posisi dan data yang diinginkan.

B.7 Pembuatan Menu BilyetDepositoSetup

Menu BilyetDepositoSetup merupakan menu yang berfungsi untuk mencatat data bilyet deposito setup yang dibuat user. Menu ini terdiri dari *page dashboard* dan *detail* yang memiliki beberapa fitur seperti *filter, create new, update data, approve, reject*, dan *void*. Berikut adalah rincian pembuatan menu BilyetDepositoSetup.

1. BilyetDepositoSetupList



Gambar 3.10. Tampilan Grid Menu BilyetDepositoSetup

Gambar 3.10 merupakan tampilan *grid* pada *dashboard* menu BilyetDepositoSetup. *Grid* dibuat dengan tujuan agar *user* dapat melihat data hasil *inputan*, serta dapat melakukan beberapa aksi untuk mengubah data tersebut. Tampilan *grid* dibuat dengan *library* dari KendoGrid untuk fungsinya dan html, *Bootstrap* untuk tampilannya.

Tabel 3.2. API Grid Data

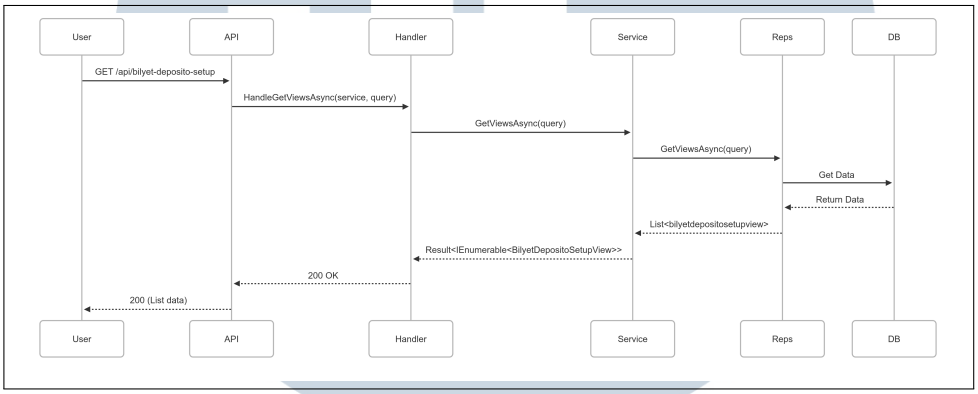
Nama API	BilyetDepositoSetupGetList
Method	GET
Endpoint Path	/api/bilyet-deposito-setup/

Request Body/Params	SimpleQuery : IdQuery
	{
	DateFrom : DateOnly
	DateTo: DateOnly
	Status: int
	}

Response	IEnumerable(BilyetDepositoSetupView) - (Daftar setup bilyet deposito)
Permission Required	bilyet-deposito-setup:read
Status Codes	200 OK

Gambar 3.11. API Call BilyetDepositoSetupGetList

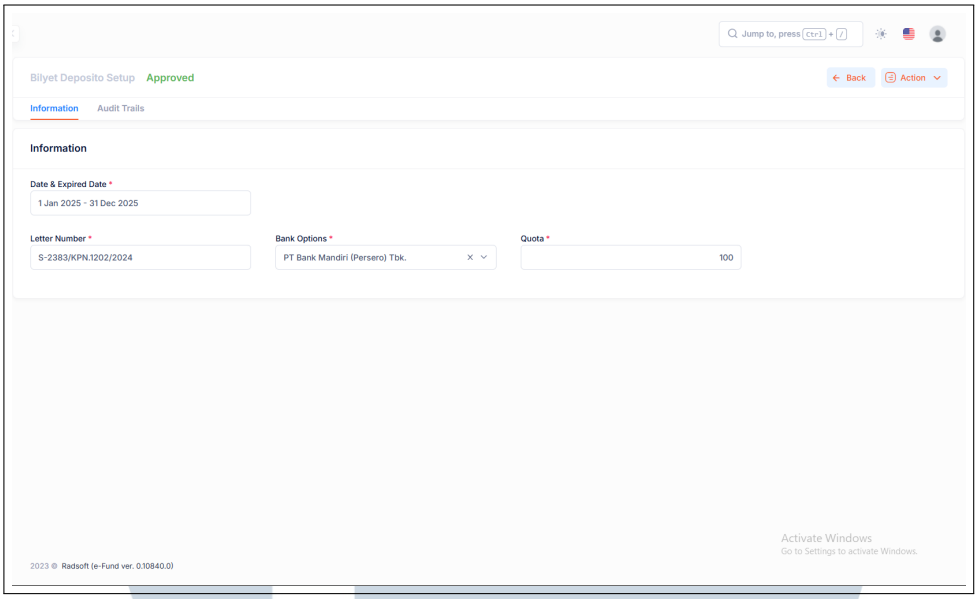
Pada Tabel 3.2 menunjukkan API yang digunakan untuk *Grid* yaitu BilyetDepositoSetupGetList. API ini digunakan untuk mendapatkan daftar data dari *database* menggunakan *method GET* dengan *endpoint* /api/bilyet-deposito-setup/. Parameter yang diperlukan adalah Status, DateFrom dan DateTo. *Response* yang dikembalikan berupa *enumerable* dari *BilyetDepositoSetupView* yang berisi daftar data seperti pada Gambar 3.11. Untuk mengakses API ini, pengguna memerlukan *permission bilyet-deposito-setup:read*. *Status code* yang mungkin dikembalikan adalah 200 OK.



Gambar 3.12. Sequence Diagram Grid

Sequence diagram pada Gambar 3.12 menggambarkan proses pengambilan data Bilyet Deposito Setup dalam sebuah sistem berbasis API. Alur dimulai ketika *User* mengirimkan permintaan *HTTP GET* ke *endpoint* /api/bilyet-deposito-setup. Permintaan ini diterima oleh API, yang kemudian meneruskannya ke *Handler* untuk diproses lebih lanjut. *Handler* memanggil *method HandleGetViewsAsync(service, query)* yang bertugas mengelola logika permintaan. Selanjutnya, *Handler* memanggil *method GetViewsAsync(query)* dari *Service*, yang berfungsi untuk menghubungkan *Handler* dengan *Reps*. *Service* kemudian berinteraksi dengan *Repository* (*Reps*) untuk mengambil data dari *Database*. Setelah data berhasil diambil dalam bentuk List¡BilyetDepositoSetupView¿, hasil tersebut dikembalikan secara berurutan melalui *Service, Handler,* dan akhirnya API. Sistem kemudian mengirimkan respons HTTP 200 (OK) kepada *User*, disertai dengan daftar data yang diminta.

2. BilyetDepositoSetupGet



Gambar 3.13. Tampilan Page Detail Menu BilyetDepositoSetup

Gambar 3.13 merupakan tampilan *page detail* pada menu BilyetDepositoSetup. *Page detail* dibuat dengan tujuan agar *user* dapat membuat data baru, melihat detail data dan mengubah data. Tampilan *page* dibuat dengan html dan *Bootstrap* untuk tampilannya.

Tabel 3.3. API Grid Detail Data

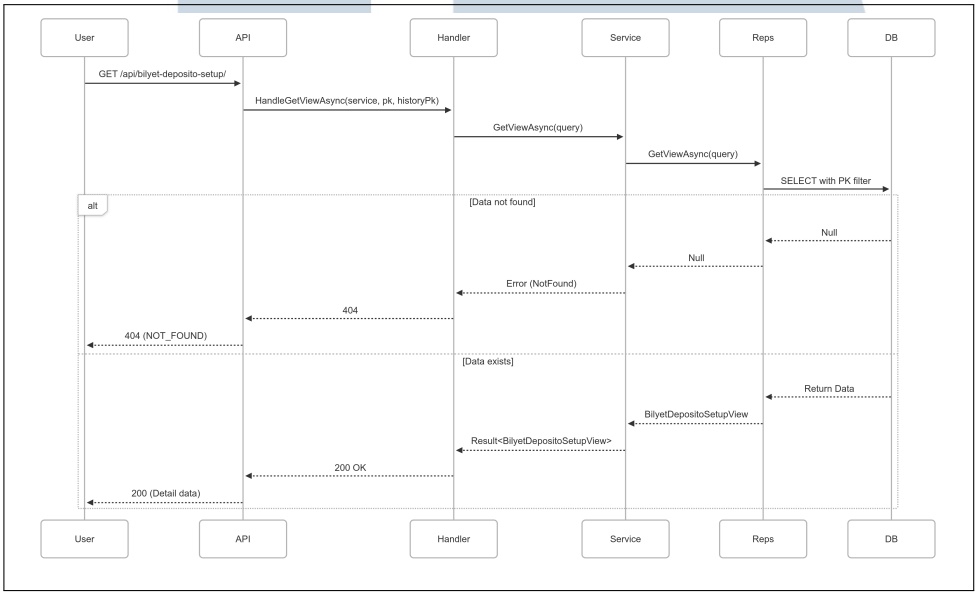
Nama API	BilyetDepositoSetupGet
Method	GET
Endpoint Path	/api/bilyet-deposito-setup/pk/historyPk

Request Body/Params	{
	Pk: long
	HistoryPk: long
	}

Response	BilyetDepositoSetupView (Detail setup bilyet deposito)
Permission Required	bilyet-deposito-setup:read
Status Codes	200 OK

Pada Tabel 3.3 menunjukkan API yang digunakan untuk mengambil

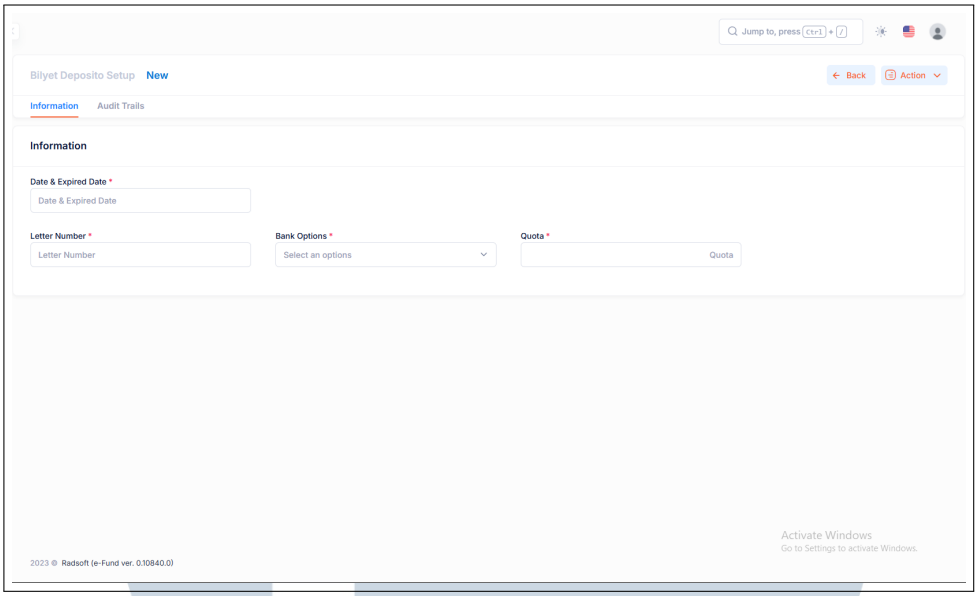
data detail adalah BilyetDepositoSetupGet. API ini berfungsi untuk mendapatkan detail data berdasarkan *primary key* (pk) dan *history key* (*historyPK*). Menggunakan *method GET* dengan *endpoint* /api/bilyet-deposito-setup/pk/historyPK. Parameter yang diperlukan adalah pk dan *historyPK*. *Response* berupa *BilyetDepositoSetupView* yang berisi detail data. *Permission* yang diperlukan adalah *bilyet-deposito-setup:read*, dan *status code* yang dikembalikan adalah 200 OK.



Gambar 3.14. Sequence Diagram Grid Detail

Sequence Diagram pada Gambar 3.14 mengilustrasikan cara sistem menampilkan detail data. Pertama, Pengguna mengirim permintaan *GET* dengan parameter Pk dan HistoryPk, kemudian sistem akan mengambil data dari *database* melalui *layered architecture* yang terdiri dari handler, service, dan repository. Jika data tidak ditemukan, sistem memberikan *respons 404*, sedangkan jika berhasil akan mengembalikan data lengkap dengan *status 200*.

3. BilyetDepositoSetupAdd



Gambar 3.15. Tampilan Create New Data BilyetDepositoSetup

Gambar 3.15 merupakan tampilan fitur *create new data* pada menu BilyetDepositoSetup. *Create new data* dibuat dengan tujuan agar *user* dapat menambahkan data berdasarkan *field* yang telah disediakan. Setelah mengisi *field* yang disediakan, *user* dapat melakukan *save* pada tombol *action* di kanan atas.

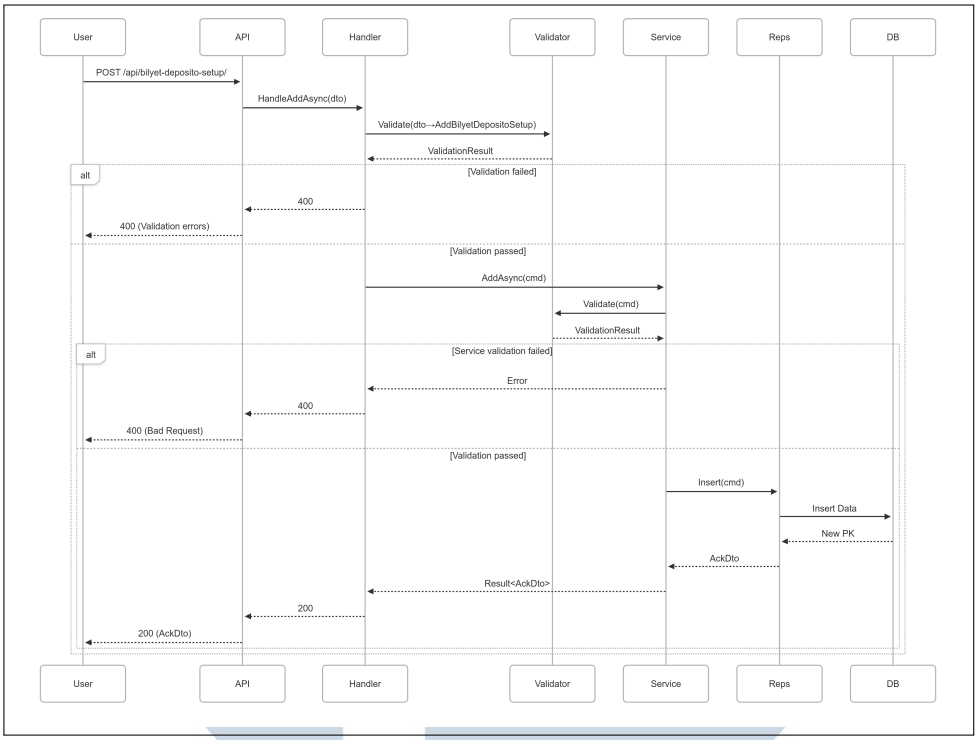
Tabel 3.4. API Add Data

Nama API	BilyetDepositoSetupAdd
Method	POST
Endpoint Path	/api/bilyet-deposito-setup/

Request Body/Params	AddBilyetDepositoSetupDto
	{
	Date: DateOnly
	LetterNo: string
	BankPk: long
	Quota: long
	ExpiredDate: DateOnly
	}

Response	AckDto (Konfirmasi sukses) atau ValidationProblemDetails (Jika validasi gagal)
Permission Required	bilyet-deposito-setup:add
Status Codes	200 OK, 400 Bad Request

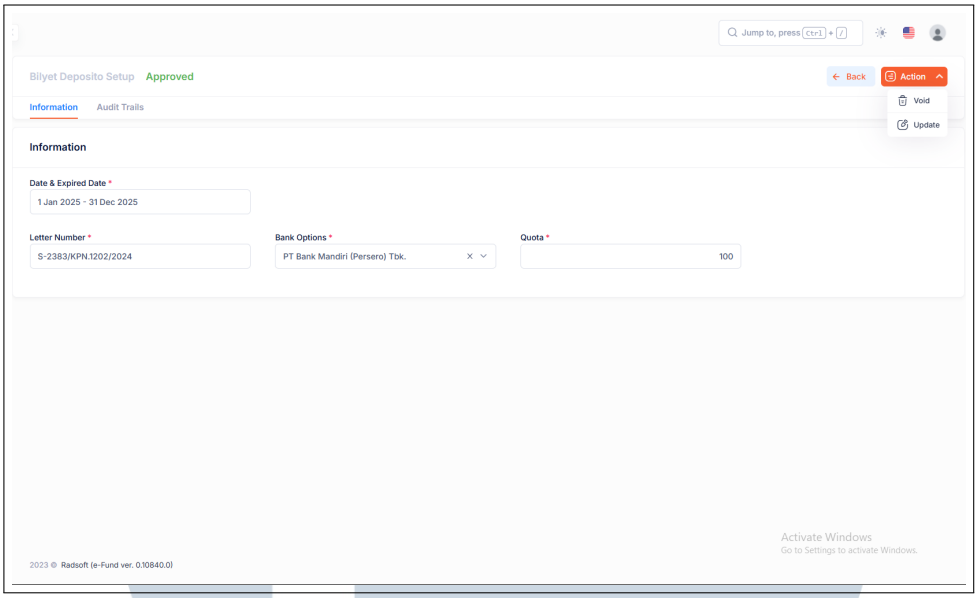
Pada Tabel 3.4 menunjukkan API yang digunakan untuk menambahkan data adalah BilyetDepositoSetupAdd. API ini digunakan untuk menambahkan data baru menggunakan *method POST* dengan *endpoint* /api/bilyet-deposito-setup/. *Request body* harus berisi data baru dalam format *AddBilyetDepositoSetup*. *Response* yang dikembalikan bisa berupa *AckDto* sebagai konfirmasi sukses atau *ValidationProblemDetails* jika validasi gagal. *Permission* yang diperlukan adalah *bilyet-deposito-setup:add*, dengan *status code 200 OK* untuk sukses atau *400 Bad Request* jika terjadi *error*.



Gambar 3.16. Sequence Diagram Add Data

Sequence Diagram pada Gambar 3.16 menggambarkan proses penambahan data baru. Pertama, Pengguna mengirimkan permintaan *POST* dengan data baru yang akan divalidasi terlebih dahulu. Jika validasi gagal, sistem langsung mengembalikan *error 400*. Jika valid, data akan diproses dan disimpan ke *database*. *Respons* sukses akan dikembalikan dengan *status 200* beserta data yang baru dibuat.

4. BilyetDepositoSetupUpdate



Gambar 3.17. Tampilan Update Data BilyetDepositoSetup

Gambar 3.17 merupakan tampilan fitur *update data* pada menu BilyetDepositoSetup. *Update data* dibuat dengan tujuan agar *user* dapat mengubah data yang telah dibuat. *Update* dapat dilakukan dengan cara mengubah *value* pada *field* yang tersedia, lalu *save* data yang telah dirubah dengan memencet tombol *action* disebelah kanan atas.

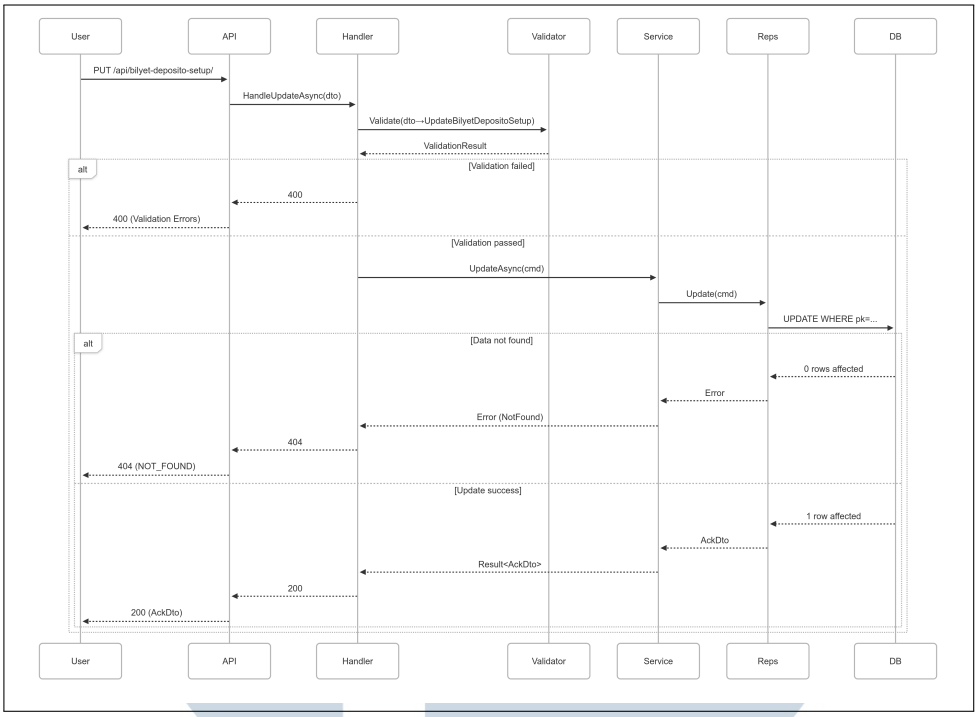
Tabel 3.5. API Update Data

Nama API	BilyetDepositoSetupUpdate
Method	PUT
Endpoint Path	/api/bilyet-deposito-setup/

Request Body/Params	`UpdateBilyetDepositoSetupDto : AddBilyetDepositoSetupDto`
	`{`
	`HistoryPk: long`
	`BilyetDepositoSetupPk: long`
	`}`

Response	AckDto (Konfirmasi sukses) atau ValidationProblemDetails (Jika validasi gagal)
Permission Required	bilyet-deposito-setup:update
Status Codes	200 OK, 400 Bad Request

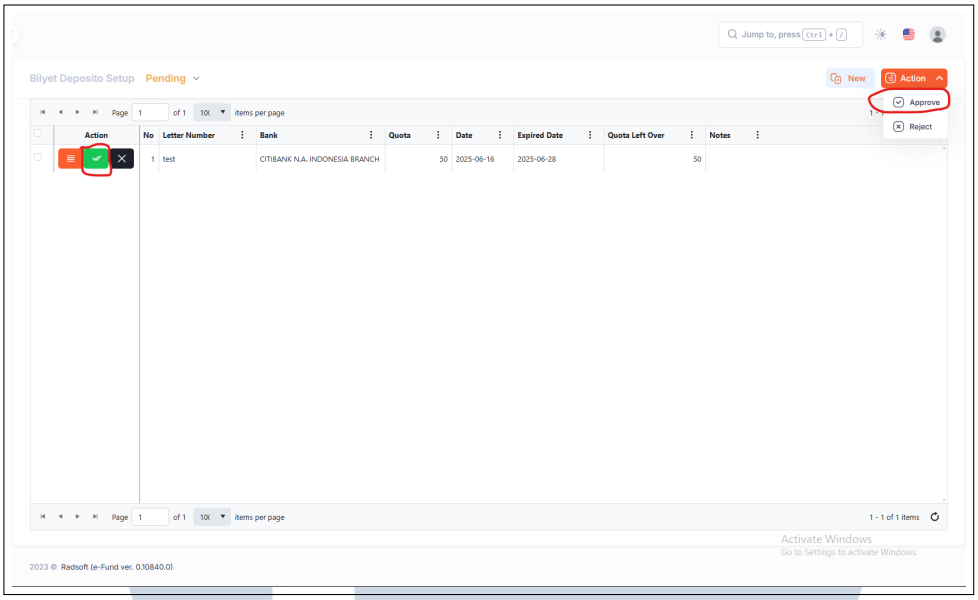
Pada Tabel 3.5 menunjukkan API yang digunakan untuk *update* data adalah BilyetDepositoSetupUpdate. API ini bertujuan untuk memperbarui data yang sudah ada menggunakan *method PUT* dengan *endpoint* /api/bilyet-deposito-setup/. *Request body* harus berisi data yang akan di *update* dalam format *UpdateBilyetDepositoSetup*. *Response* yang dihasilkan berupa *AckDto* untuk konfirmasi sukses atau *ValidationProblemDetails* jika validasi gagal. *Permission* yang diperlukan adalah *bilyet-deposito-setup.update*, dengan *status code 200 OK* atau *400 Bad Request*.



Gambar 3.18. Sequence Diagram Update Data

Sequence Diagram pada Gambar 3.18 menunjukkan alur *update* data. Pertama, pengguna mengirimkan permintaan *PUT* dengan data yang ingin di *update*. Sistem pertama-tama akan memvalidasi *input* data melalui *validator*. Jika validasi gagal, langsung mengembalikan *error 400*. Jika valid, handler akan memproses permintaan update melalui service yang kemudian menyimpan perubahan ke *database*. *Respons* yang dikembalikan bisa berupa *404* jika data tidak ditemukan, atau *200* dengan data terupdate jika berhasil.

5. BilyetDepositoSetupApprove



Gambar 3.19. Tampilan Approve Data BilyetDepositoSetup

Gambar 3.19 merupakan tampilan fitur *approve* pada menu BilyetDepositoSetup. *Approve* dibuat dengan tujuan agar *user* dapat membuat data yang telah dibuat memiliki status *approve*. *Approve* dapat dilakukan dengan memencet tombol pada kolom *action grid* atau dengan memilih *row data* lalu memencet tombol *action update* di sebelah kanan atas.

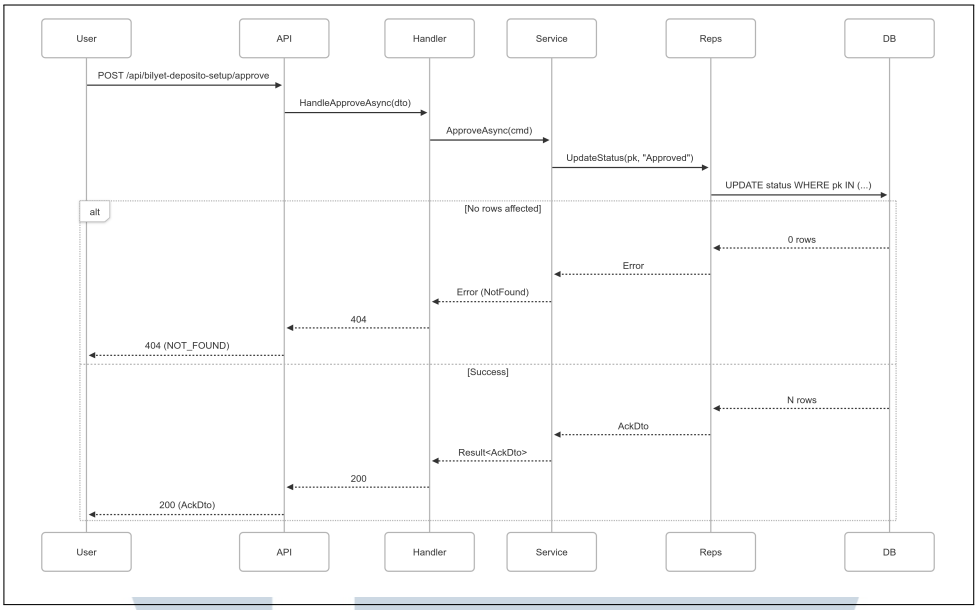
Tabel 3.6. API Approve Data

Nama API	BilyetDepositoSetupApprove
Method	POST
Endpoint Path	/api/bilyet-deposito-setup/approve

Request Body/Params	ApvDataDto
	{
	Pk: long
	HistoryPk: long
	Notes: string
	}

Response	AckDto (Konfirmasi sukses) atau ValidationProblemDetails (Jika validasi gagal)
Permission Required	bilyet-deposito-setup:approve
Status Codes	200 OK, 400 Bad Request

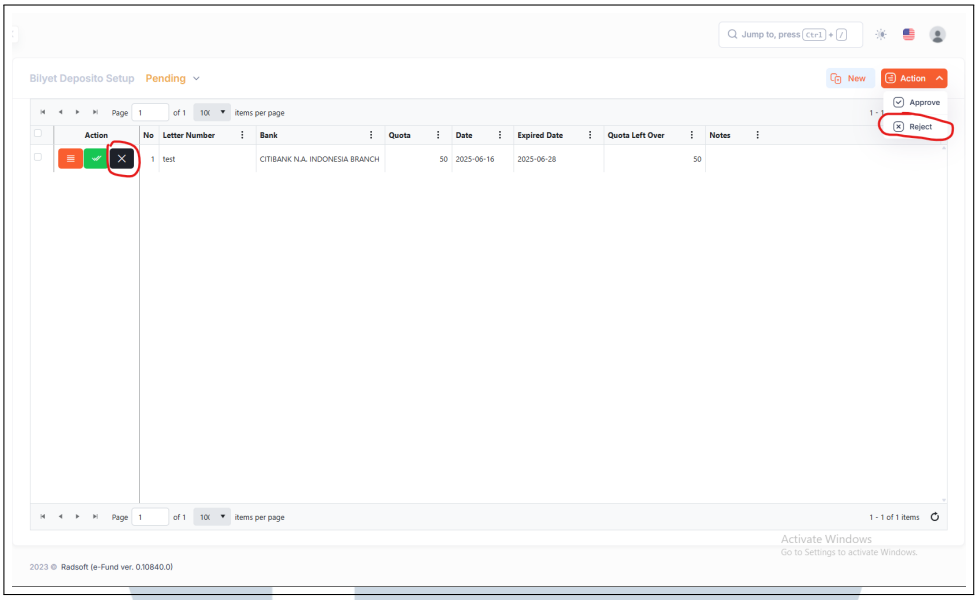
Pada Tabel 3.6 menunjukkan API yang digunakan untuk *approve* data adalah BilyetDepositoSetupApprove. API ini digunakan untuk menyetujui data menggunakan *method POST* dengan *endpoint* /api/bilyet-deposito-setup/approve. *Request body* harus berisi data untuk disetujui. *Response* yang dihasilkan berupa *AckDto* untuk konfirmasi sukses atau *ValidationProblemDetails* jika validasi gagal. *Permission* yang diperlukan adalah *bilyet-deposito-setup.approve*, dengan *status code 200 OK* atau *400 Bad Request*.



Gambar 3.20. Sequence Diagram Approve Data

Sequence Diagram pada Gambar 3.20 menggambarkan proses *approve* data. Alur dimulai ketika pengguna mengirimkan permintaan *POST* ke *endpoint approve* melalui API. Permintaan ini kemudian diproses oleh handler yang memanggil service untuk diproses. Service akan berkomunikasi dengan repository untuk mengupdate status data menjadi 'Approved' di *database*. Jika data tidak ditemukan, sistem akan mengembalikan *respons error 404*, sedangkan jika berhasil akan mengembalikan *respons 200* beserta data konfirmasi.

6. BilyetDepositoSetupReject



Gambar 3.21. Tampilan Reject Data BilyetDepositoSetup

Gambar 3.21 merupakan tampilan fitur *reject* pada menu BilyetDepositoSetup. *Reject* dibuat dengan tujuan agar *user* dapat membuat data yang telah dibuat memiliki status *reject*. Ketika data di *reject*, status data yang sebelumnya *approve* akan kembali ke status *pending*. *Reject* dapat dilakukan dengan memencet tombol pada kolom *action grid* atau dengan memilih *row data* lalu memencet tombol *action update* di sebelah kanan atas.

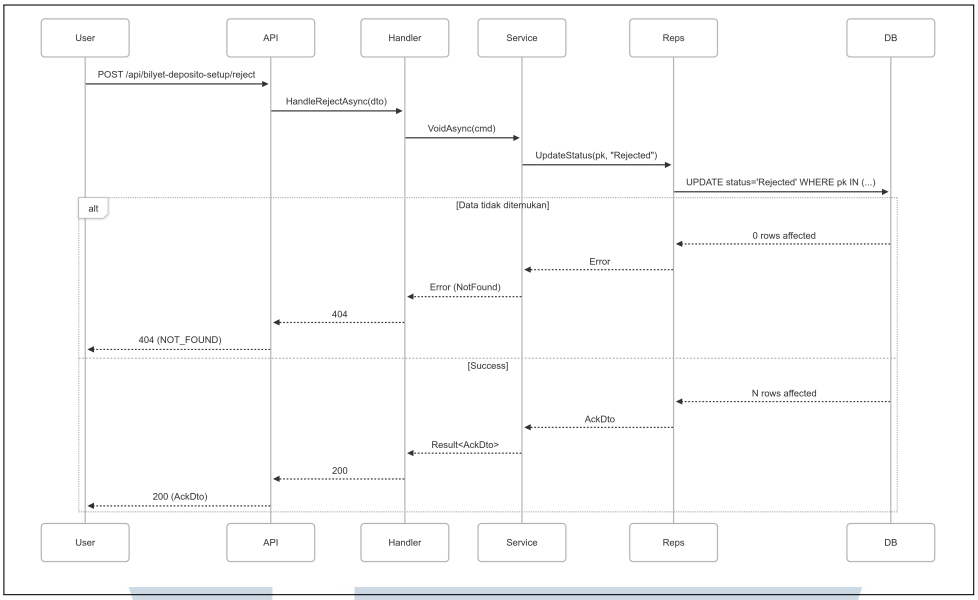
Tabel 3.7. API Reject Data

Nama API	BilyetDepositoSetupReject
Method	POST
Endpoint Path	/api/bilyet-deposito-setup/reject

Request Body/Params	ApvDataDto
	{
	Pk: long
	HistoryPk: long
	Notes: string
	}

Response	AckDto (Konfirmasi sukses) atau ValidationProblemDetails (Jika validasi gagal)
Permission Required	bilyet-deposito-setup:reject
Status Codes	200 OK, 400 Bad Request

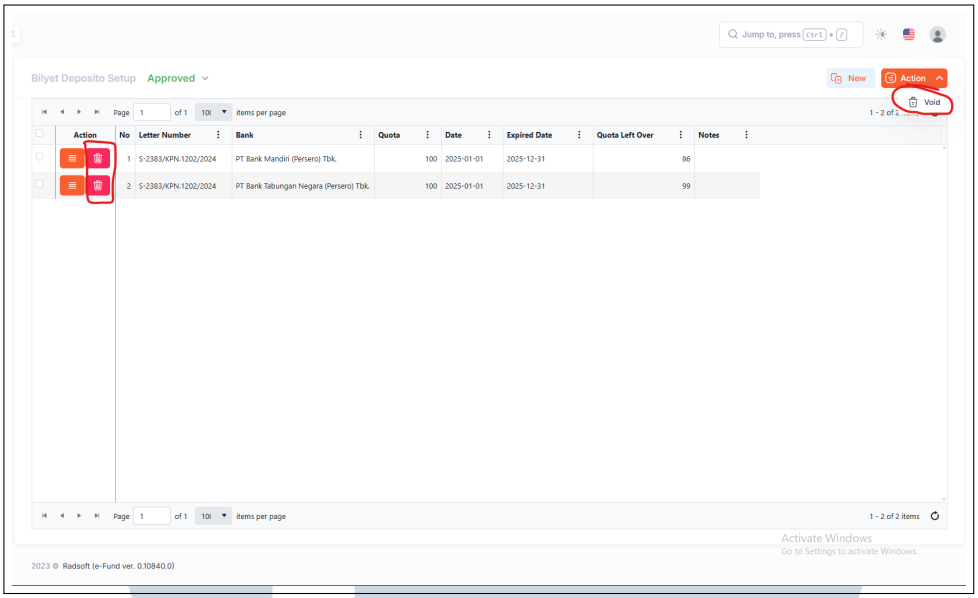
Pada Tabel 3.7 menunjukkan API yang digunakan untuk *reject* data adalah BilyetDepositoSetupReject. API ini berfungsi untuk menolak data menggunakan *method POST* dengan *endpoint* /api/bilyet-deposito-setup/reject. *Request body* harus berisi data untuk penolakan. *Response* yang dihasilkan berupa *AckDto* untuk konfirmasi sukses atau *ValidationProblemDetails* jika validasi gagal. *Permission* yang diperlukan adalah *bilyet-deposito-setupreject*, dengan *status code 200 OK* atau *400 Bad Request*.



Gambar 3.22. Sequence Diagram Reject Data

Sequence Diagram pada Gambar 3.22 menunjukkan alur *reject* data. Pertama, pengguna mengirimkan permintaan *POST* ke *endpoint* /api/bilyet-deposito-setup/reject melalui API. Permintaan tersebut diteruskan ke *Handler* yang memanggil *method VoidAsync(cmd)* pada *Service*. *Method* yang digunakan untuk reject dan void sama dikarenakan kemiripan fungsi keduanya. Lalu *Service* akan berkomunikasi dengan *Repository* untuk mengupdate status data menjadi ”Rejected” di *database* dengan *query UPDATE status*=”Rejected” WHERE *pk IN* (...). Jika data tidak ditemukan (*0 rows affected*), sistem mengembalikan *error 404*. Jika berhasil (*N rows affected*), *respons 200* (*OK*) dengan konfirmasi *AckDto* dikirim ke pengguna.

7. BilyetDepositoSetupVoid



Gambar 3.23. Tampilan Void Data BilyetDepositoSetup

Gambar 3.23 merupakan tampilan fitur *void* pada menu BilyetDepositoSetup. *Void* dibuat dengan tujuan agar *user* dapat membuat data yang telah dibuat memiliki status *void*. Data yang telah di *void* tidak terhapus secara permanen namun data dibuat memiliki status *void*. *Void* dapat dilakukan dengan memencet tombol pada kolom *action grid* atau dengan memilih *row data* lalu memencet tombol *action update* di sebelah kanan atas.

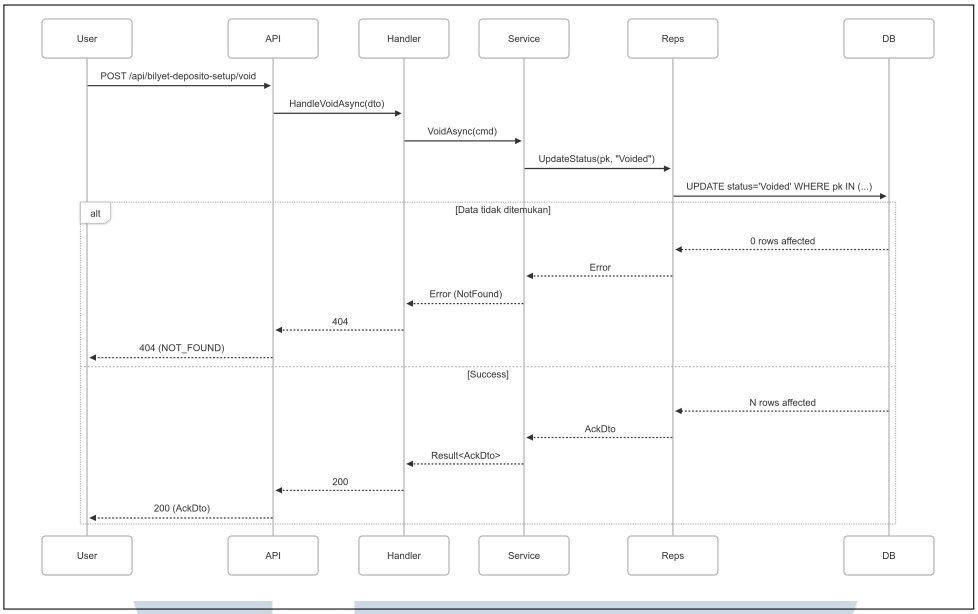
Tabel 3.8. API Void Data

Nama API	BilyetDepositoSetupVoid
Method	POST
Endpoint Path	/api/bilyet-deposito-setup/void

Request Body/Params	ApvDataDto
	{
	Pk: long
	HistoryPk: long
	Notes: string
	}

Response	AckDto (Konfirmasi sukses) atau ValidationProblemDetails (Jika validasi gagal)
Permission Required	bilyet-deposito-setup:void
Status Codes	200 OK, 400 Bad Request

Pada Tabel 3.8 menunjukkan API yang digunakan untuk *void* data adalah BilyetDepositoSetupVoid. API ini digunakan untuk menghapus data menggunakan *method POST* dengan *endpoint* /api/bilyet-deposito-setup/void. *Request body* harus berisi data untuk dihapus. *Response* yang dihasilkan berupa *AckDto* untuk konfirmasi sukses atau *ValidationProblemDetails* jika validasi gagal. *Permission* yang diperlukan adalah *bilyet-deposito-setup:void*, dengan *status code 200 OK* atau *400 Bad Request*.



Gambar 3.24. Sequence Diagram Void Data

Sequence Diagram pada Gambar 3.24 mengilustrasikan proses *void* data. Pertama, pengguna mengirim permintaan *POST* ke *endpoint* /api/bilyet-deposito-setup/void. *Handler* memprosesnya dengan memanggil *method VoidAsync(cmd)* pada *Service*, yang kemudian mengubah status data menjadi ”Voided” di *database* melalui *query UPDATE status* WHERE *pk IN* (...). Jika data tidak ada (*0 rows affected*), sistem menolak permintaan dengan *error 404*. Jika *void* berhasil (*N rows affected*), sistem mengirim *respons 200* (*OK*) beserta objek *AckDto*.

3.3.2 DAPEN BI

DAPEN BI merupakan *platform* investasi yang unggul dalam membantu para investor institusi dari berbagai macam industri baik dari Industri Keuangan Non Bank (IKNB) maupun Perbankan untuk melakukan perencanaan, evaluasi, hingga pengawasan risiko yang berkaitan dengan kegiatan investasi. Pada *project* DAPEN BI ini, tugas yang diberikan serupa dengan LPDP yaitu berupa pemeliharaan dan pengembangan fitur-fitur web baik yang sudah ada maupun yang akan ditambahkan. Sama dengan *Project* LPDP, keahlian teknis yang diperlukan adalah di bidang *Fullstack Development*, khususnya dalam penggunaan bahasa pemrograman *JavaScript*, *CSharp*, pengelolaan *query* basis data dan OOP.

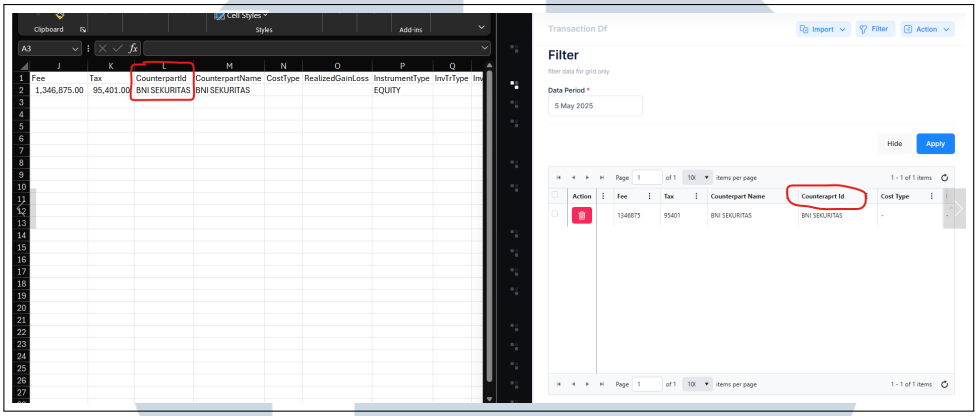
A Usecase Diagram DAPEN BI

Gambar 3.25. Usecase Diagram DAPEN BI

Usecase Diagram Dapenbi pada Gambar 3.25 menggambarkan sistem yang dirancang untuk mengelola data *user* sebagai pemakai web ini. Aktor *admin* dapat melakukan *login* sebelum memulai aplikasi, lalu *admin* juga dapat mengatur pemakaian *user* dengan *management user*. *Management user* terdiri dari *user* untuk siapa pengguna web tersebut dan *roles* untuk mengatur *access* yang dapat dilakukan *user*. *User* dapat melakukan *Get Data*, yang mencakup pengambilan data internal (*Integral Data*) dan eksternal (*Integral Data External*). Selanjutnya, *user* dapat melakukan *Screening Data* yang bertugas menyaring data eksternal. *User* juga dapat *monitoring data* yang di dapat dari hasil perbandingan data antara *data core* dan data yang telah di *screening*. Dan yang terakhir *user* dapat melakukan *Forecasting Data*, di mana sistem melakukan peramalan data dan analisis data.

B Task yang dikerjakan

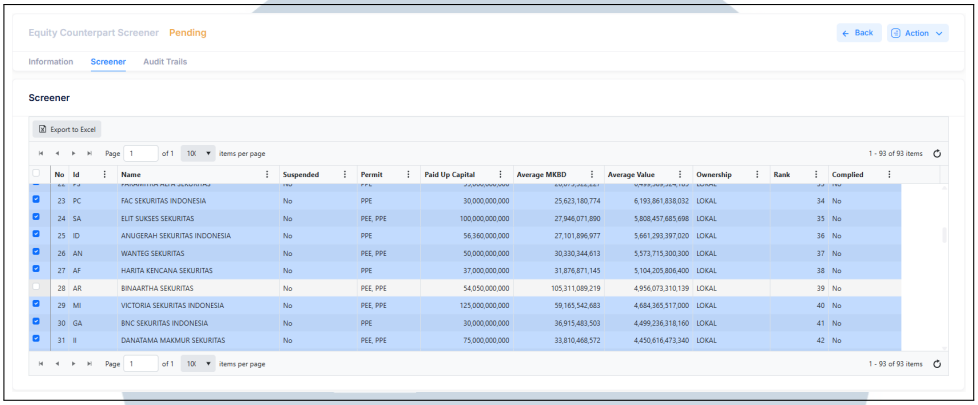
B.1 Perbaikan Template Import KPD pada menu Transaction Df



Gambar 3.26. Perubahan setelah counterpartId ditambahkan di excel dan grid

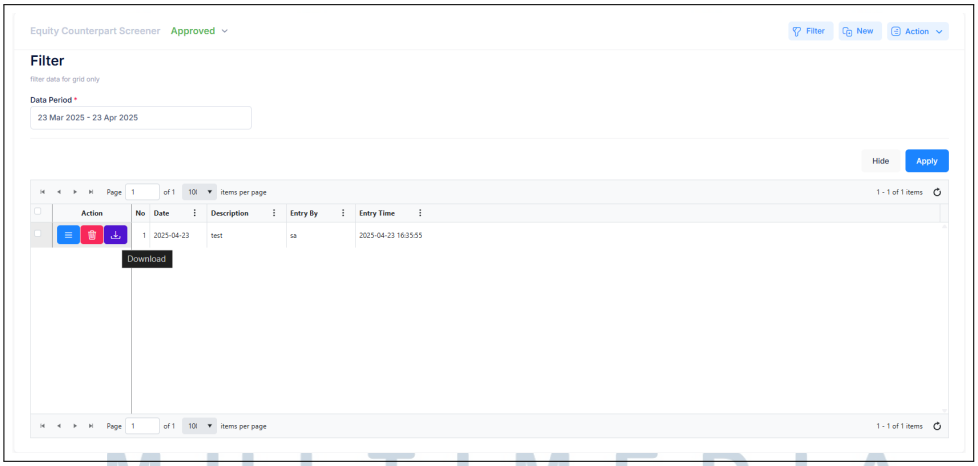
Pada kasus ini, terdapat kebutuhan pada *user* untuk menampilkan kolom '*counterpart*' pada menu *Post Trade Compliance* menggunakan *CounterpartId* pada data *TransactionDf*. Terlihat pada kasus ini, memerlukan *mapping* pada data tersebut, untuk itu pada file *Reps* menu ini terdapat *query* pengambilan data '[*CounterpartId*]' pada tabel *TransactionDf* seperti pada Gambar 3.26. Untuk penempatan *CounterpartId* ditaruh disebelah kiri kolom *CounterpartName*. Setelah pada bagian *backend* telah mengambil data, lalu tambahkan pada *UI script* di dalam KendoGrid kolom *CounterpartId* pada *field*, *orders* untuk urutan penempatan kolom dan *customs* untuk kolom yang ingin ditambahkan dengan judul kolom.

B.2 Perbaikan pada Grid dan Logic Download pada menu Equity Counterpart Screener



Gambar 3.27. Tampilan grid di Menu Equity Counterpart Screener

Pada task ini, terdapat kesalahan *minor* yang membuat *grid* ter-*refresh* sendiri ketika *user* melakukan *scroll* pada *grid*. Untuk kesalahan pertama ini terjadi karena terdapat '*grid.fetch*' pada bagian fungsi *checkbox data* yang diletakkan pada akhir fungsi, maka yang perlu dilakukan adalah menghapus '*grid.fetch*' tersebut agar *user* tidak mengalami *refresh* sendiri. Tampilan *grid* dapat dilihat pada Gambar 3.27.



Gambar 3.28. Fitur download pada grid

Kesalahan kedua terletak pada bagian *checkbox*, ketika *user* ingin melakukan *download* pada *row* tertentu yang telah di *checklist*, file yang ter-*download* adalah semua data bukan data pada *row* yang di *checklist* atau yang di *selected*. Lalu untuk memperbaiki *logic download* pada bagian ini adalah dengan

menambahkan kondisi *where* pada *query* pengambilan data *report* dengan '*IsListed = 1*'. Tampilan fitur *download* dapat dilihat pada Gambar 3.28

3.3.3 DIFA

Tampilan antarmuka DIFA pada versi sebelumnya sudah tidak lagi sesuai dengan perkembangan tren desain digital masa kini. Hal ini disebabkan oleh fakta bahwa mayoritas pengguna awal DIFA berasal dari kalangan klien perusahaan berusia lanjut, di mana fokus pengembangan terdahulu lebih menekankan pada aspek fungsionalitas dibandingkan estetika dan pengalaman pengguna.

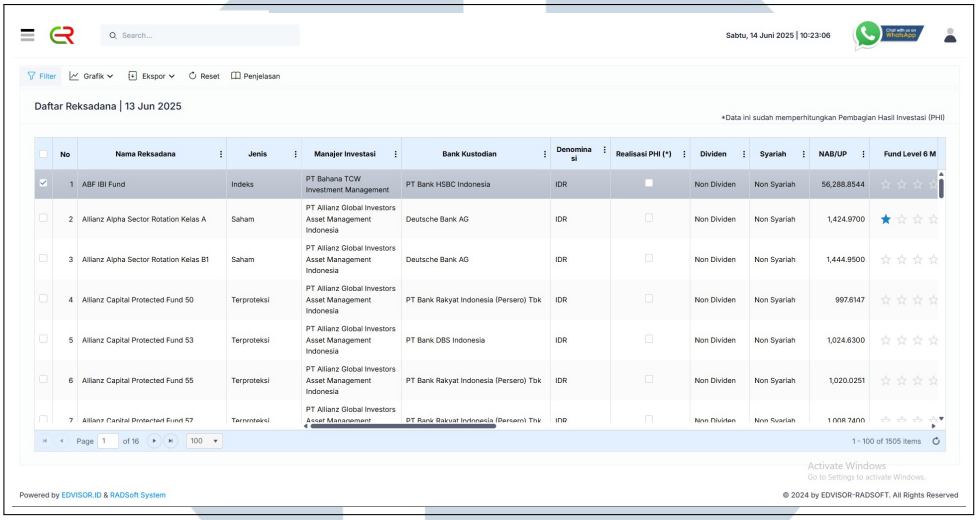
Seiring dengan rencana strategis perusahaan untuk memperluas pasar DIFA ke segmen pengguna retail yang lebih beragam, diperlukan pembaruan terhadap tampilan antarmuka. Pembaruan ini bertujuan untuk menghadirkan desain yang lebih modern, dinamis, dan *user-friendly*, sehingga mampu memenuhi ekspektasi generasi muda yang semakin aktif dalam dunia investasi.

Dengan transformasi ini, diharapkan DIFA dapat lebih mudah diterima, diminati, serta meningkatkan *engagement* dari berbagai kalangan, khususnya generasi muda yang memiliki preferensi terhadap aplikasi dengan tampilan visual menarik, navigasi intuitif, dan pengalaman penggunaan yang nyaman.

Project ini diberikan tugas oleh atasan untuk merombak satu *page* DIFA yaitu *page* daftar reksadana. Pengerjaan DIFA dilakukan dengan beberapa tahap, tahap paling awal adalah melakukan analisis terhadap tampilan dan fungsi DIFA versi lama. Analisis ini bertujuan untuk mengidentifikasi kelemahan dalam aspek desain *UI* seperti keterbacaan, alur navigasi dan estetika, serta *UX* seperti kemudahan penggunaan dalam menggunakan DIFA.

A Breakdown

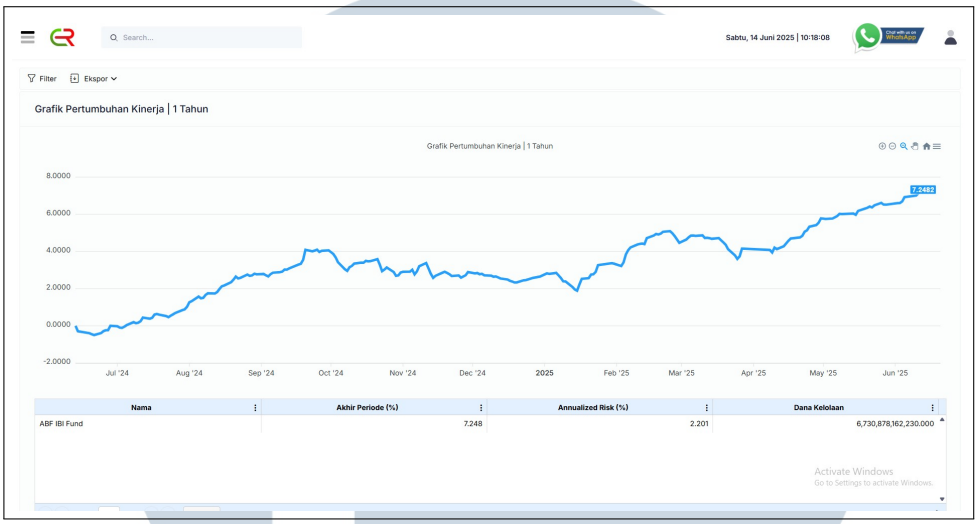
A.1 Dashboard



Gambar 3.29. Tampilan dashboard daftar reksadana

Gambar 3.29 adalah *dasboard* DIFA yang berisi komponen-komponen utama. Pada sisi atas kiri *page* terdapat logo DIFA, *field search* dan tombol *sidebar* yang ketika dibuka akan menampilkan *sidebar*, dibawah komponen tersebut terdapat *toolbar* fitur-fitur yang dapat digunakan oleh *user* seperti *filter*, grafik, ekspor, *reset* dan penjelasan, di sisi tengah *page* terdapat *grid* utama yang menampilkan data-data dari daftar reksadana yang tersedia. Tampilan *dashboard* DIFA pada *page* daftar reksadana menunjukkan bahwa desain *platform* tersebut telah mengalami ketertinggalan dalam mengikuti perkembangan tren desain *web modern*. Hal ini terlihat dari struktur *grid* yang cenderung kaku dan monoton, penggunaan palet warna yang minim dan kurang variatif, serta model tampilan secara keseluruhan yang bersifat konvensional dan kurang menarik secara estetika.

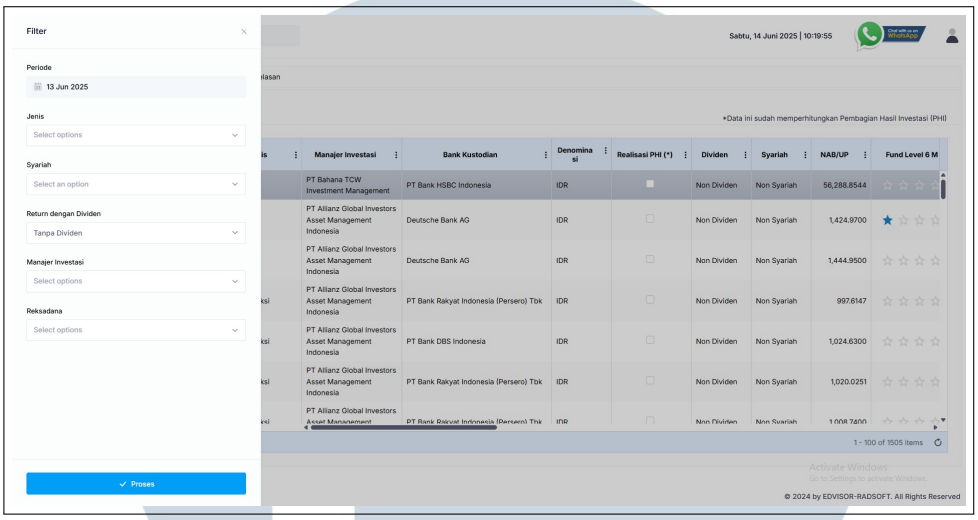
A.2 Chart



Gambar 3.30. Tampilan page chart daftar reksadana

Komponen yang terdapat pada *page chart* mirip dengan *page dashboard*. Perbedaan dengan *page chart* adalah *toolbar* fitur yang disediakan hanya dua yaitu *filter* dan ekspor seperti pada Gambar 3.30, lalu pada sisi tengah terdapat *chart* dari *data grid* serta terdapat *grid* dibawah *chart* untuk menampilkan isi *detail data chart*. Tampilan *page chart* daftar reksadana menggunakan tema terang dengan latar belakang putih polos sebagai dasar utama *page*. Grafik pertumbuhan kinerja dibuat dengan garis berwarna biru sederhana. Meskipun penggunaan tema terang bertujuan untuk menjaga keterbacaan, secara keseluruhan desain ini justru memberikan kesan monoton dan kurang dinamis. Palet warna yang digunakan terbatas, tanpa adanya variasi kontras yang mampu menarik perhatian pengguna pada area-area penting.

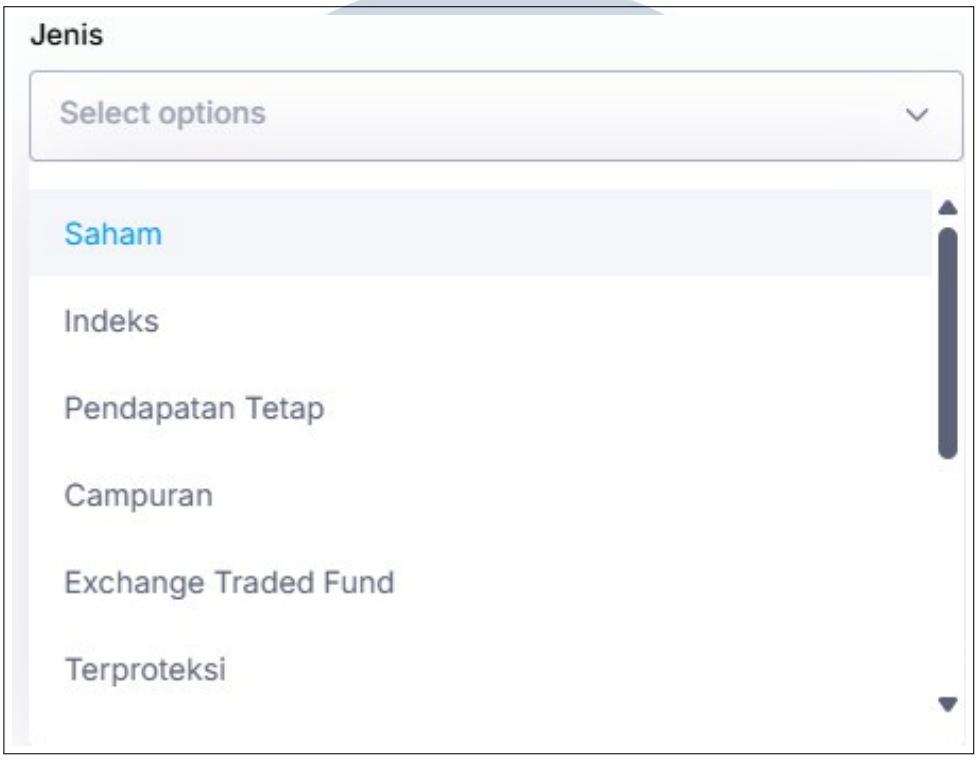
A.3 Filter



Gambar 3.31. Tampilan fitur filter daftar reksadana

Fitur *filter* pada *page* daftar reksadana terdapat beberapa *option* yang menggunakan *design grid* sederhana dengan garis pembatas, tetapi tampilannya terlalu padat dan monoton karena kurangnya variasi warna atau *shading* untuk membedakan baris atau kolom penting. *Design* ini mengandalkan fungsi dibandingkan estetika, sehingga terkesan kaku dan kurang menarik secara visual. Tampilan fitur *filter* dapat dilihat pada Gambar 3.31.

A.4 Dropdown

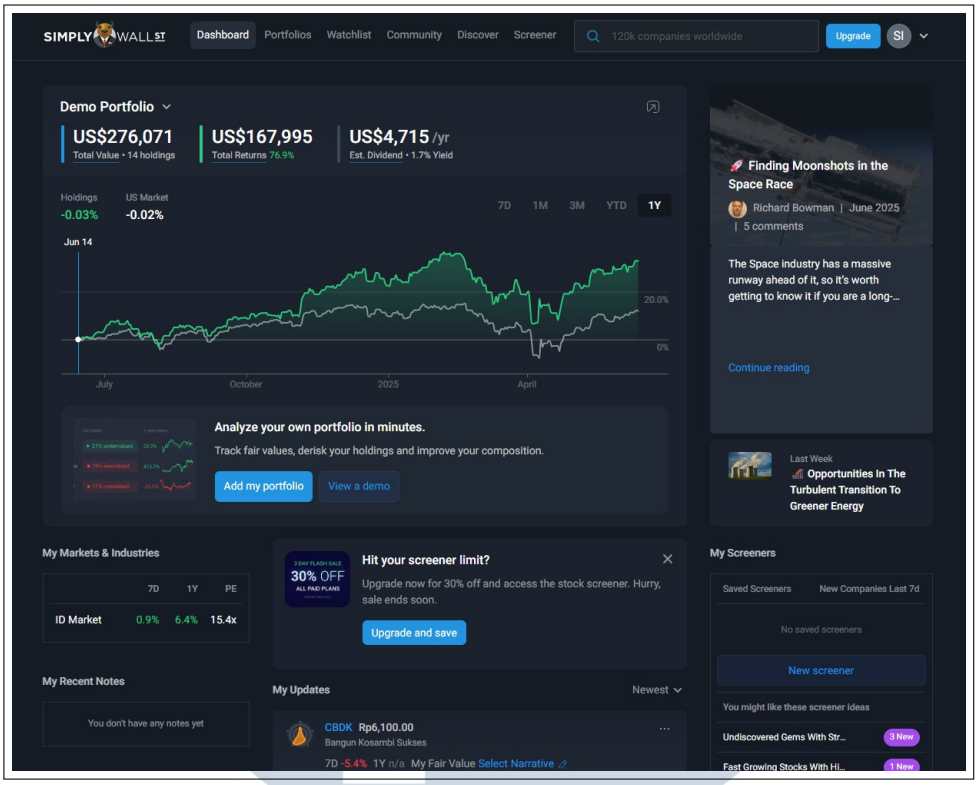


Gambar 3.32. Tampilan dropdown filter daftar reksadana

Komponen *dropdown* pada *filter* daftar reksadana menampilkan daftar pilihan investasi dengan desain yang sangat minimalis. Tampilannya menggunakan latar putih polos dengan teks hitam sederhana, tanpa elemen *visual* pendukung seperti *border*, atau efek interaktif. Meskipun fungsional, desain ini terlihat datar dan kurang menarik di mata anak muda. Tampilan *dropdown* dapat dilihat pada Gambar 3.32

B Analisis Web Investasi Sejenis

Step selanjutnya adalah mengumpulkan referensi dari berbagai *platform* investasi digital yang telah lebih dulu dikenal luas dan memiliki reputasi baik. Tujuan pencarian referensi ini adalah untuk memahami pengelolaan informasi finansial dan implementasi tampilan modern yang relevan dengan kebutuhan pengguna masa kini.



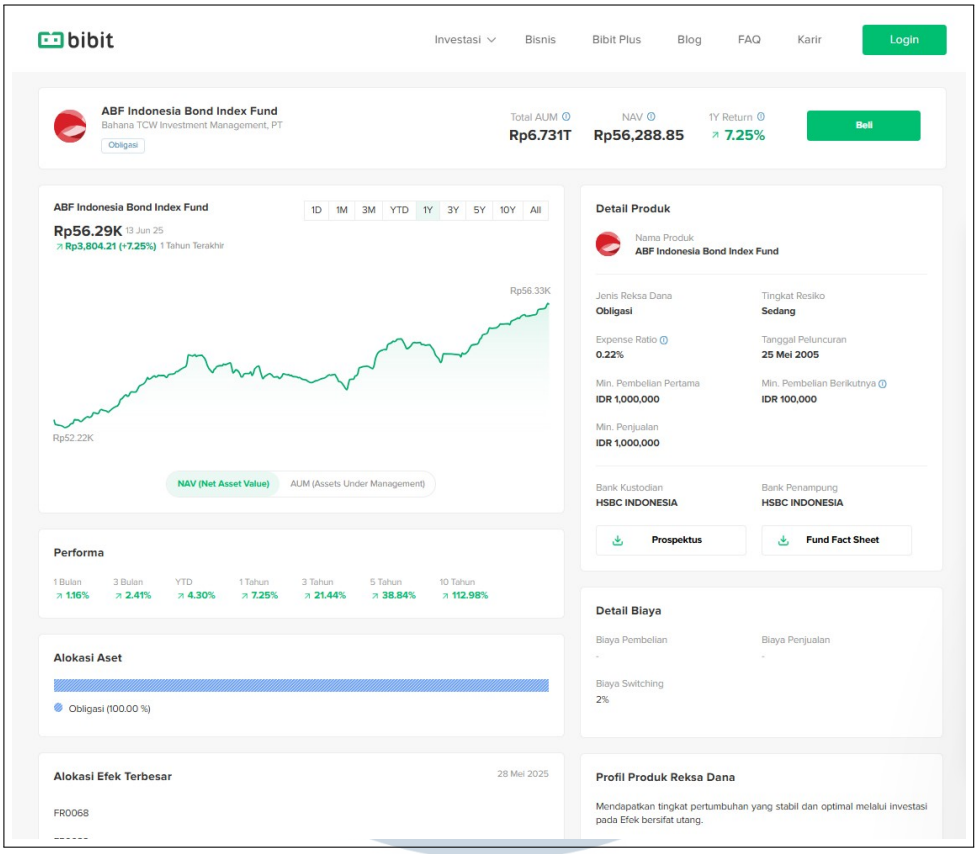
Gambar 3.33. Tampilan web Simplywall

Salah satu referensi utama dalam perancangan ulang *page* daftar reksadana adalah *website Simplywall.st*, seperti ditunjukkan pada Gambar 3.33. *Simplywall.st* merupakan *platform* investasi yang dikenal secara global karena kemampuannya dalam menyajikan data keuangan dan portofolio secara visual, interaktif, dan terstruktur.

Tampilan antarmuka *Simplywall.st* menggunakan tema gelap yang modern dan estetik, sesuai dengan tren desain saat ini. Penggunaan latar belakang gelap yang dipadukan dengan warna cerah pada grafik dan informasi penting memberikan kontras visual yang kuat. Hal ini tidak hanya meningkatkan keterbacaan, tetapi juga memudahkan pengguna dalam memahami data secara cepat.

Dari segi struktur konten, informasi seperti portofolio, analisis pasar, dan berita investasi ditampilkan secara rapi dan mudah diakses. Grafik kinerja investasi diletakkan di bagian tengah, dikelilingi oleh *panel* tambahan seperti daftar pasar dan rekomendasi.

Dari hasil analisis ini, diperoleh banyak inspirasi yang dapat diterapkan pada perancangan ulang *page* daftar reksadana, terutama dalam penggunaan tema gelap, penyajian data secara grafis serta penataan *layout* yang terstruktur.



Gambar 3.34. Tampilan web Bibit

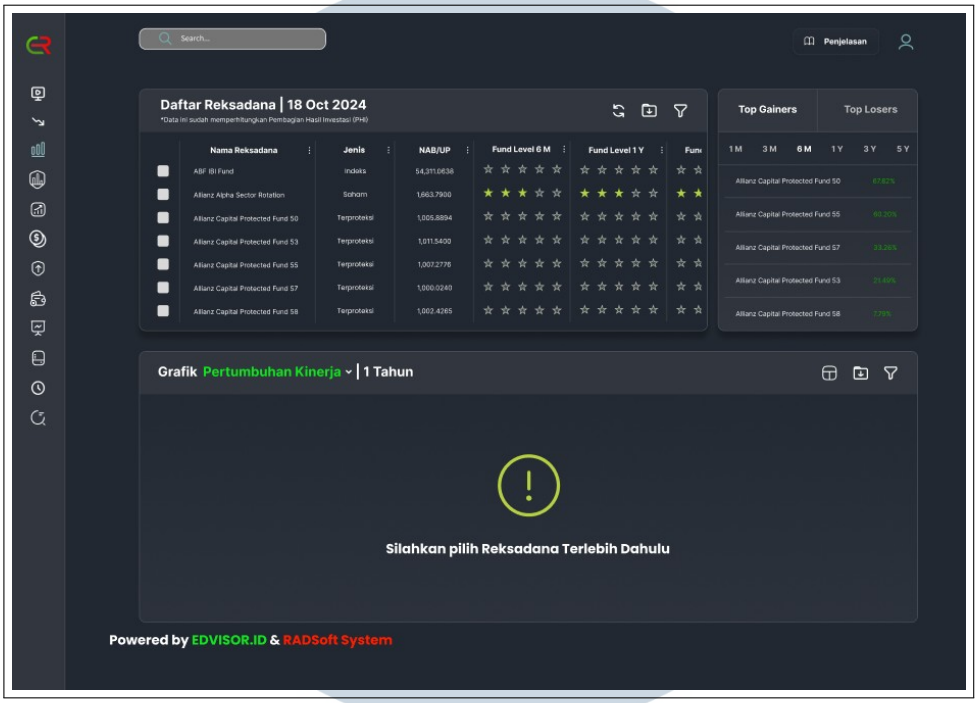
Referensi kedua berasal dari tampilan web Bibit (Gambar 3.34), yang mengusung antarmuka bersih, modern, dan responsif. Dominasi warna putih dengan aksen hijau khas *brand* digunakan untuk menyorot elemen penting seperti tombol aksi dan data kinerja. Struktur *layout* yang rapi, penggunaan komponen *modular* seperti grafik performa, tab navigasi, dan kartu informasi menjadikan tampilan mudah dibaca, cepat diakses, serta efisien untuk dikembangkan lebih lanjut.

Analisis terhadap desain Bibit memberikan banyak inspirasi dalam merancang ulang *page* daftar reksadana, khususnya dalam menciptakan tata letak yang terstruktur, ringan, dan ramah pengguna. Pendekatan *modular* serta pemilihan warna yang tepat membantu menciptakan fokus visual dan kemudahan interaksi, sesuai dengan kebutuhan pengguna modern.

C Design dan Prototyping

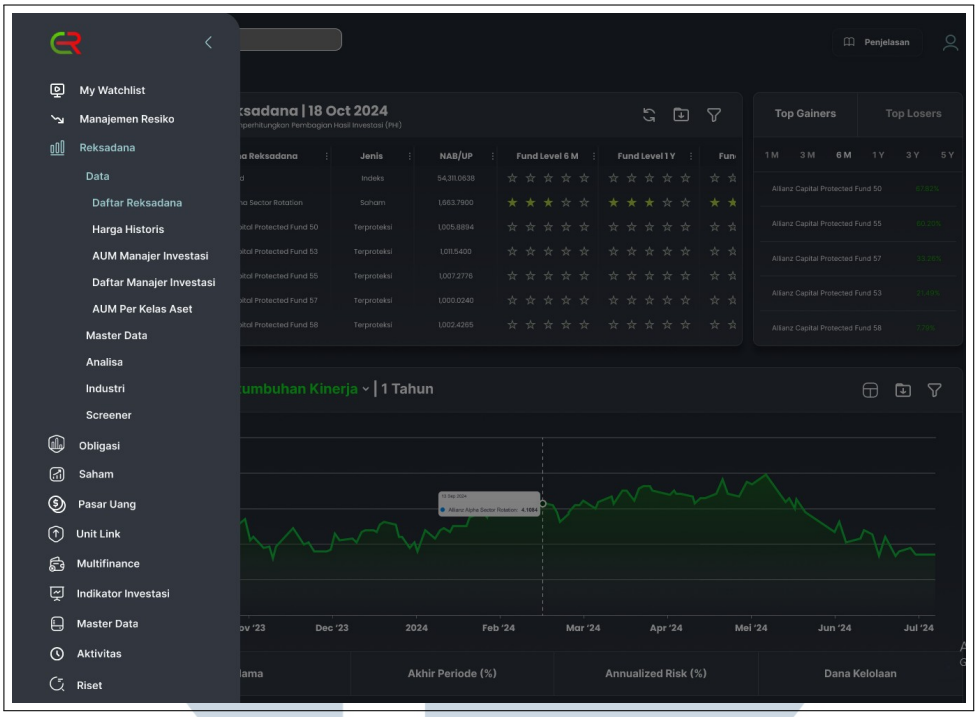
Tahap selanjutnya membuat *design* dan *prototyping* dengan Figma. *Design* dan *Prototyping* dibuat untuk memberikan gambaran kepada atasan untuk rencana

desain yang akan dibuat pada page daftar reksadana mulai dari desain dan juga *prototyping*.



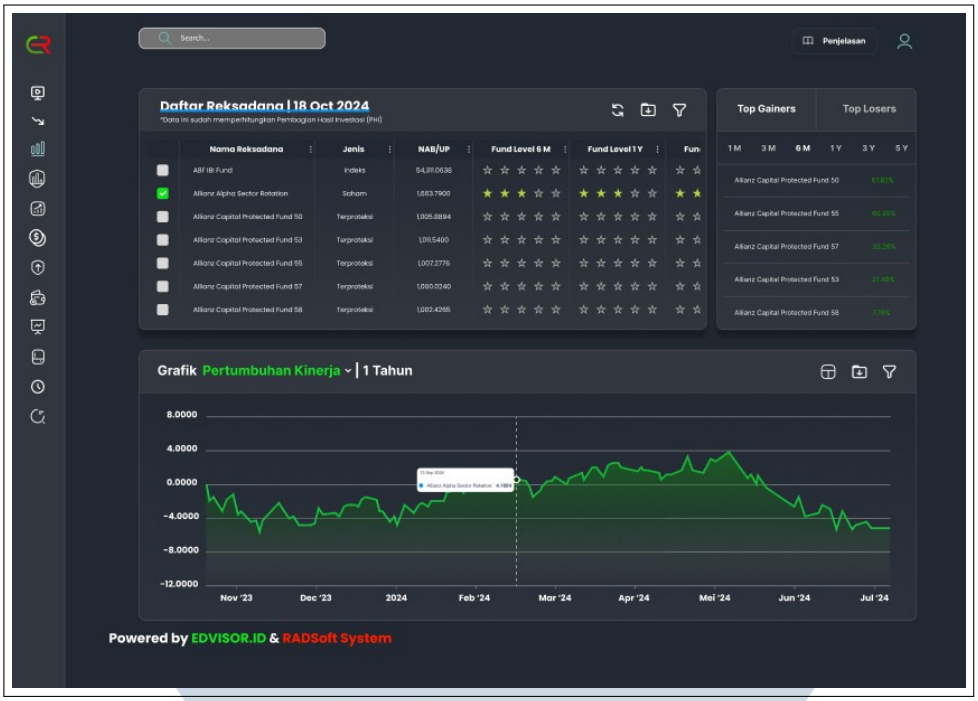
Gambar 3.35. Tampilan desain dan prototype dashboard daftar reksadana

Desain *dashbboard* daftar reksadana pada Gambar 3.35 dibuat dengan tema gelap dan menjadi *main color* pada desain ini. *Sidebar* dibuat dengan lebih minimalis dengan memunculkan sedikit logo-logo menu di sisi kiri *page* dan apabila kursor diarahkan ke sisi tersebut, *sidebar* akan terbuka. Bagian atas *page* dibuat *simple* dengan beberapa komponen saja seperti *search*, penjelasan dan juga *profile*. Lalu untuk isi *page, layout* dibuat dengan berbagai *card* yang diisi dengan *grid* dan juga *Chart*. Terdapat sebuah *Grid* tambahan untuk menampilkan *list Top Gainers* dan *Top Losers*. Tampilan *grid* dibuat dengan modern dan minimalis yang dimana tampilan *grid* dibuat tidak terlalu kotak dan pemilihan warna yang menyesuaikan dengan warna tema. Fitur-fitur*grid* dan *chart* diletakkan di *header card* dalam bentuk *logo* bertujuan agar *user* mengetahui fitur tersebut dipergunakan untuk *grid* atau *chart*.



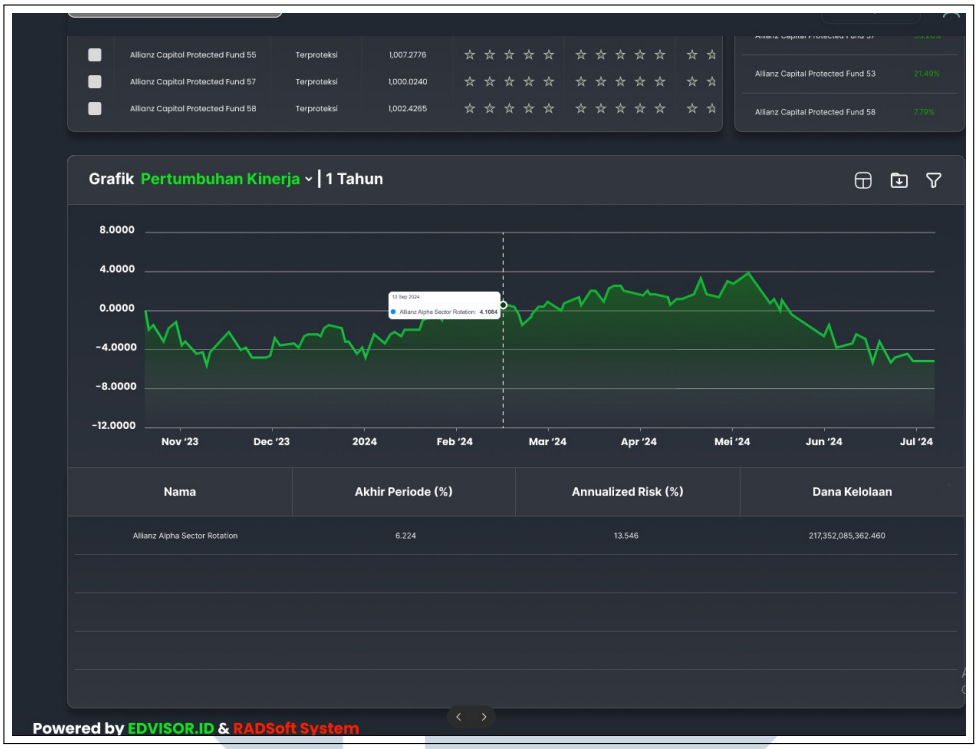
Gambar 3.36. Tampilan desain dan prototype sidebar daftar reksadana

Sidebar page daftar reksadana pada Gambar 3.36 dibuat dengan pendekatan modern yang mengusung tema *dark mode* untuk memberikan kesan profesional sekaligus meningkatkan kenyamanan visual. Tata letak navigasi disusun secara vertikal dan tersegmentasi berdasarkan kategori fungsi seperti Reksadana, Saham, Obligasi, dan lainnya, sehingga memudahkan pengguna dalam mengakses fitur secara cepat dan terarah. Penambahan ikon yang konsisten dan minimalis memperkuat identifikasi menu tanpa mengganggu kesederhanaan tampilan.



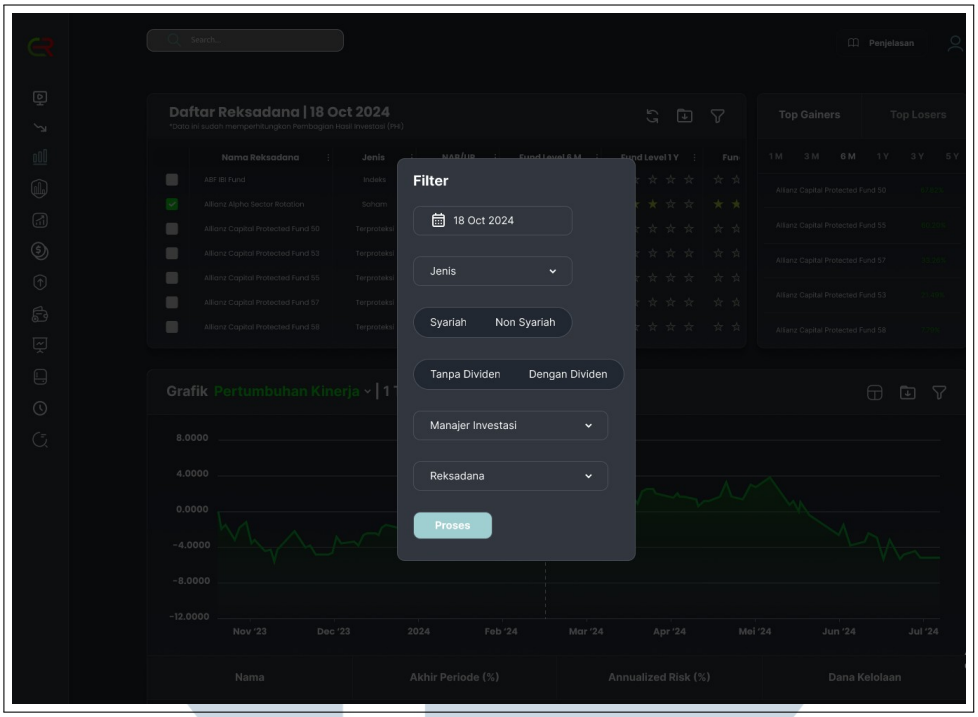
Gambar 3.37. Tampilan desain dan prototype chart daftar reksadana

Grafik pada Gambar 3.37 dibuat dengan menggunakan garis berwarna hijau terang di atas latar gelap untuk menunjukkan perubahan nilai investasi dari waktu ke waktu, sehingga menciptakan kontras visual yang jelas dan mudah dibaca. Di bagian atas grafik terdapat *dropdown* grafik yang memungkinkan pengguna memilih jenis data kinerja yang ingin ditampilkan. *Tooltips* juga aktif ketika kursor diarahkan ke titik data tertentu, menampilkan informasi *detail* seperti nama reksadana dan nilai pada tanggal spesifik.



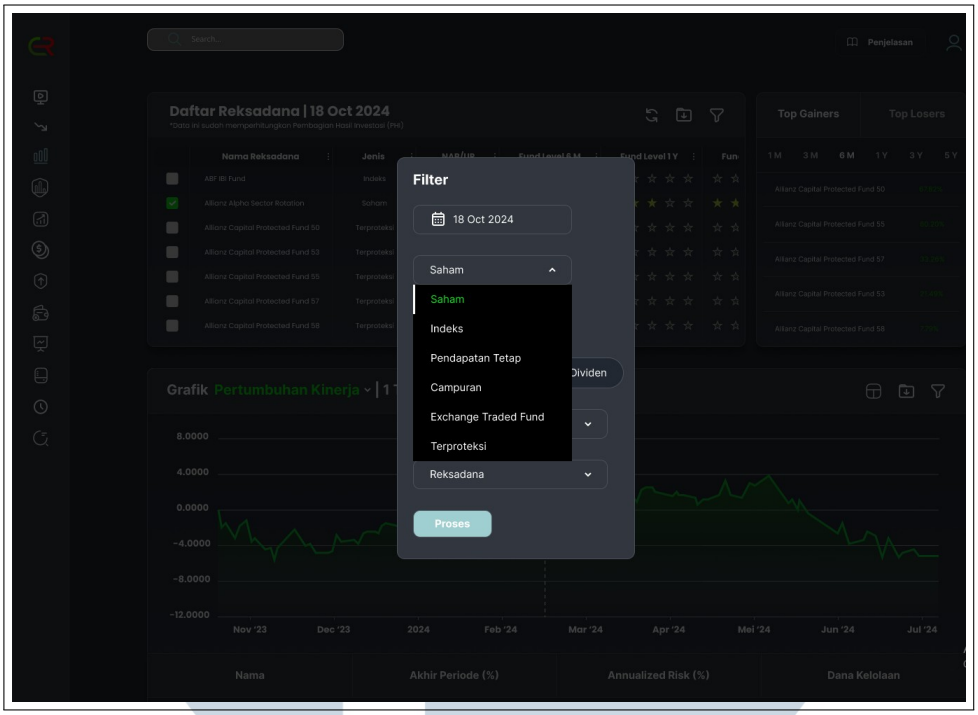
Gambar 3.38. Tampilan desain dan prototype grid detail chart daftar reksadana

Grid detail pada Gambar 3.38 yang terletak di bawah grafik menampilkan informasi tambahan yang bersifat kuantitatif untuk mendukung analisis *visual*. Data ini membantu pengguna memahami performasi investasi tidak hanya dari segi pertumbuhan, tetapi juga dari sisi stabilitas dan skala aset yang dikelola. Desain tabel dibuat tetap konsisten dengan tema gelap keseluruhan, menggunakan tipografi yang jelas dan pemisahan kolom yang rapi untuk menjaga keterbacaan. Peletakan yang langsung terhubung dengan grafik di atas memungkinkan pengguna melakukan korelasi antara *visual* tren dan angka-angka penting tanpa harus berganti tampilan, sehingga meningkatkan efisiensi interaksi dan pengalaman pengguna.



Gambar 3.39. Tampilan desain dan prototype fitur filter daftar reksadana

Fitur *filter* pada Gambar 3.39 disusun dalam bentuk *modal* yang ringkas dan intuitif, memungkinkan pengguna menyaring data reksadana berdasarkan beberapa kriteria utama. *Modal filter* tampil melayang di atas konten utama dengan latar gelap transparan, menciptakan fokus visual yang jelas. Elemen-elemen seperti *date picker, dropdown*, dan tombol *switch* memiliki sudut membulat dan *padding* yang cukup, memberikan kesan ramah pengguna. Tata letak yang vertikal dan teratur mempermudah navigasi antar pilihan *filter*, sedangkan tombol Proses berwarna terang memberikan titik perhatian yang kuat di akhir alur interaksi. Pengguna dapat memilih tanggal referensi data, jenis reksadana (seperti saham, indeks, dsb.), status syariah, distribusi dividen, manajer investasi, serta nama produk reksadana tertentu.



Gambar 3.40. Tampilan desain dan prototype dropdown filter daftar reksadana

Tampilan *dropdown* pada Gambar 3.40 dirancang sederhana dengan latar belakang gelap dan teks kontras yang jelas untuk memudahkan pembacaan. Interaksi *dropdown* ini memungkinkan pengguna memilih dengan cepat kategori reksadana yang diinginkan, sehingga hasil pencarian menjadi lebih relevan sesuai preferensi dan strategi investasi masing-masing pengguna.

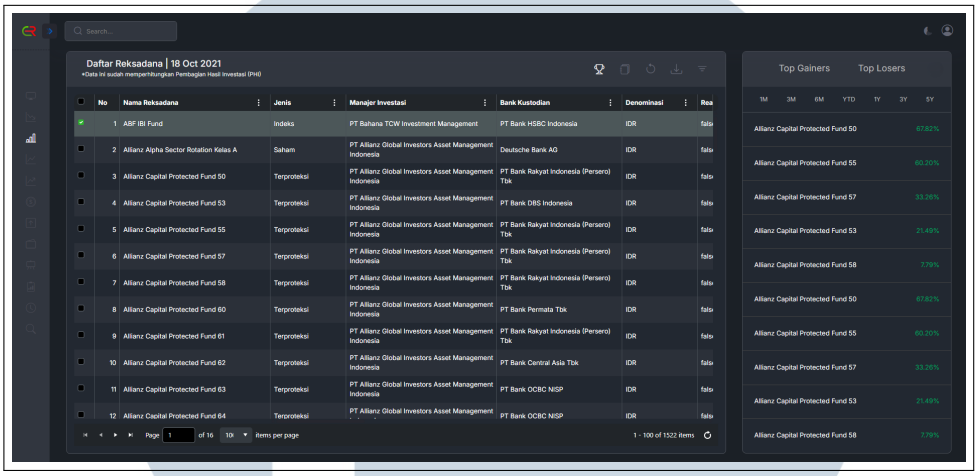
D Presentation

Tahapan terakhir dari tahap-tahap sebelumnya yaitu presentasi ke atasan. Setelah semua *design* dan *prototype* selesai, atasan akan menilai dan memberikan masukkan seperti kebutuhan pengguna dan juga kebutuhan akan yang akan datang sehingga *design* tersebut fleksibel ketika terdapat tambahan-tambahan menu pada DIFA. Apabila atasan sudah menyetujui *design* tersebut baru tahap implementasi dapat dilaksanakan, namun apabila belum, *feedback* dari atasan akan diperbaiki dan di implementasikan terlebih dahulu.

E Implementasi

Setelah semua tahapan diatas terpenuhi, barulah pada tahap ini dimulai mengimplementasikan hasil *design* dan *prototype* dalam bentuk *coding* yang

nantinya akan menghasilkan sebuah *web page* daftar reksadana. Berbagai macam bahasa yang digunakan yaitu HTML, CSS, dan *JQuery* dengan tambahan bantuan dari *framework* dan *library* seperti *Bootstrap*, KendoGrid, dan *ApexChart*.

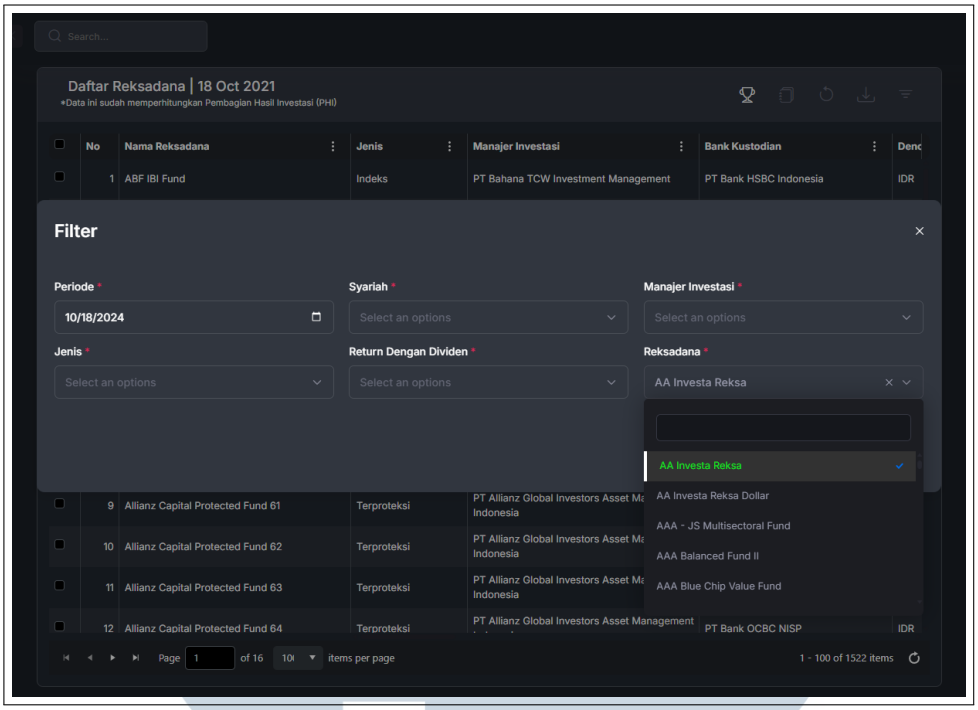


Gambar 3.41. Tampilan implementasi dashboard daftar reksadana

Pada *Page Menu* Daftar Reksadana terdapat beberapa komponen *header* yang berisi fitur utama dari *page* ini yaitu, *Grid List, Chart, Filter, Grid Gainers and Losers, Export to Excel*, Penjelasan dan *Refresh*. Fitur *Grid List* berfungsi untuk melihat list-list reksadana yang ada. Data di dalam *Grid* diambil dari API yang akan diteruskan ke *Backend* yang berisi *query*, lalu *Backend* akan membawa data tersebut kembali ke *Frontend* yang akan mengisi *data source* dari *Grid* tersebut untuk ditampilkan pada saat *user* meng-*load Page* Daftar Reksadana. *Grid Top Gainers* dan *Losers* berfungsi untuk melihat *list Top* Reksadana, namun isi dari data *grid* tersebut masih sebuah *hardcode* dikarenakan atasan menunda ide *project* ini untuk sementara. Fitur *Export to Excel* berfungsi untuk mengekspor data dari *grid* ke *excel*. Fitur ini dibuat menggunakan *library Kendo* yang memang digunakan untuk membawa semua data dari Kendogrid ke *Excel*. Fitur penjelasan berisi sebuah *link* yang mengarah ke *Youtube* dengan isi *video tutorial* menggunakan web ini, sedangkan *Refresh* berfungsi untuk melakukan *fetch* pada *grid*, sehingga apabila terdapat perubahan *Grid* menampilkan data yang paling ter-*update*. Tampilan *dashboard* dapat dilihat pada Gambar 3.41.

Gambar 3.42. Tampilan implementasi chart daftar reksadana

Tampilan *chart* dibuat dengan *design* yang lebih modern dan *simple*. Fitur *chart* dibuat agar *user* dapat melihat visualisasi dari data reksadana. Cara kerja *chart* adalah dengan memilih data yang ingin di visualisasikan datanya dari *Grid List* sehingga dari *backend* menarik data lewat API untuk mengambil data yang dipilih, lalu terdapat *dropdown* pada *card header* yang berfungsi untuk memilih jenis *chart*. Setelah memilih jenis *chart*, *chart* di *generate* menggunakan *library* dari Kendo berdasarkan jenis *chart* dan data pilihan dari API. Fitur ini merupakan fitur yang sulit untuk dibuat dikarenakan kompleksitasnya. Fitur ini menampilkan beberapa jenis *chart* dengan *data real-time*, tergantung pada seleksi *row* di *grid Kendo UI* yang memiliki 1.600+ kolom. Pembuatan fitur ini harus memastikan perubahan di satu bagian (misal: *filter*) tidak merusak logika *chart* atau *grid*. Tampilan *chart* dapat dilihat pada Gambar 3.42.



Gambar 3.43. Tampilan implementasi filter daftar reksadana

Fitur *filter* dibuat di tengah *Page* di atas tampilan *Grid List* dengan *dropdown* dan *option* yang dapat dipilih *user* untuk mem-*filter* data reksadana. *Design* dari *filter* juga minimalis dan modern dengan warna netral dan gelap yang nyaman untuk di mata kebanyakan anak muda disertai dengan warna hijau terang untuk menarik fokus mata *user* pada *aspect* yang penting. Lalu isi dari *dropdown* dan *option* tersebut bisa dari *hardcode* ataupun API. Ketika fitur *filter* di klik, maka API akan mengambil data dari API, lalu opsi yang dipilih *user* akan masuk ke dalam "*model*". *Model* dibawa ke *backend* untuk diproses melalui API. Kemudian ketika *user* klik tombol proses, data yang telah di *filter* dengan isi *model* akan dibawa ke *Frontend Grid*. Tampilan *filter* dapat dilihat pada Gambar 3.43.

3.4 Kendala dan Solusi yang Ditemukan

Kendala yang ditemukan adalah kesulitan dalam memahami bahasa pemrograman *Vue.js* karena ini merupakan pengalaman pertama dalam menggunakannya. Tantangan juga muncul saat melakukan *breakdown project* serta saat mengimplementasikan berbagai logika dalam kode yang sedang dikembangkan. Solusi yang didapatkan seperti *mentor* dan atasan selalu memberikan dukungan dalam penyelesaian *project*. Karena masih mengalami kendala dalam memahami *Vue.js*, atasan menyarankan untuk menggunakan bahasa

pemrograman yang lebih *familiar*, sehingga diputuskan untuk menggunakan *jQuery*. Selain itu, atasan juga membantu dalam proses *breakdown project*, sementara *mentor* secara konsisten memberikan arahan terkait logika yang perlu diterapkan dalam pengembangan. Meskipun kasus yang dihadapi masih cukup rumit, dengan solusi dari *mentor* dan atasan dapat memberikan gambaran solusi terhadap halangan tersebut.