

**MEMBANGUN APLIKASI
WEDDING ORGANIZER BERBASIS WEB
(STUDI KASUS DI LINDA SALON)**

SKRIPSI

Karya tulis sebagai syarat memperoleh
Gelar Sarjana Komputer dari Fakultas Teknologi Informasi
Universitas Bale Bandung

Disusun oleh:

**RIFKI ZULFIQOR
NPM. C1A140012**



**PROGRAM STRATA 1
PROGRAM STUDI TEKNIK INFORMATIKA
FAKULTAS TEKNOLOGI INFORMASI
UNIVERSITAS BALE BANDUNG
BANDUNG
2019**

LEMBAR PERSETUJUAN PENGUJI

MEMBANGUN APLIKASI WEDDING ORGANIZER BERBASIS WEB (STUDI KASUS DI LINDA SALON)

Disusun oleh:

RIFKI ZULFIQOR

NPM.C1A140012

Skripsi ini diterima dan disetujui untuk memenuhi persyaratan mencapai gelar

SARJANA KOMPUTER

Pada

PROGRAM STUDI TEKNIK INFORMATIKA

FAKULTAS TEKNOLOGI INFORMASI

UNIVERSITAS BALE BANDUNG

Baleendah, 2019

Disetujui oleh

Pengaji 1

Pengaji 2

Yudi Herdiana, S.T., M.T.

NIDN.0428027501

Zen Munawar, S.T., M.Kom

NIDN.0422037002

LEMBAR PERSETUJUAN PEMBIMBING

MEMBANGUN APLIKASI WEDDING ORGANIZER BERBASIS WEB (STUDI KASUS DI LINDA SALON)

Disusun oleh:

RIFKI ZULFIQOR

NPM.C1A140012

Skripsi ini diterima dan disetujui untuk memenuhi persyaratan mencapai gelar

SARJANA KOMPUTER

Pada

PROGRAM STUDI TEKNIK INFORMATIKA

FAKULTAS TEKNOLOGI INFORMASI

UNIVERSITAS BALE BANDUNG

Baleendah, 2019

Disetujui oleh

Dosen Pembimbing 1

Dosen Pembimbing 2

Rustiyana, S.T., M.T., M.Pd.

NIDN. 0416017704

Yaya Suharya, S.Kom., M.T.

NIDN. 0407047706

LEMBAR PERSETUJUAN PROGRAM STUDI

MEMBANGUN APLIKASI WEDDING ORGANIZER BERBASIS WEB (STUDI KASUS DI LINDA SALON)

Disusun oleh:

RIFKI ZULFIQOR

NPM.C1A140012

Skripsi ini diterima dan disetujui untuk memenuhi persyaratan mencapai gelar

SARJANA KOMPUTER

Pada

PROGRAM STUDI TEKNIK INFORMATIKA

FAKULTAS TEKNOLOGI INFORMASI

UNIVERSITAS BALE BANDUNG

Baleendah, 2019

Mengesahkan,
Ketua Prodi Teknik Informatika

Yaya Suharya, S.Kom., M.T.
NIDN. 0407047706

LEMBAR PERSETUJUAN FAKULTAS

MEMBANGUN APLIKASI WEDDING ORGANIZER BERBASIS WEB (STUDI KASUS DI LINDA SALON)

Disusun oleh:

RIFKI ZULFIQOR

NPM.C1A140012

Skripsi ini diterima dan disetujui untuk memenuhi persyaratan mencapai gelar

SARJANA KOMPUTER

Pada

PROGRAM STUDI TEKNIK INFORMATIKA

FAKULTAS TEKNOLOGI INFORMASI

UNIVERSITAS BALE BANDUNG

Baleendah,

2019

Mengetahui,

Dekan Fakultas Teknologi dan Informasi

Yudi Herdiana, S.T., M.T.

NIDN.0428027501

ABSTRAK

Resepsi pernikahan adalah proses acara yang membutuhkan persiapan yang total, terkadang baik calon pengantin maupun keluarga akan menghadapi kesulitan, sehingga seringkali dibutuhkan jasa *Wedding Organizer* untuk membantu dalam perencanaan dan pelaksanaan rangkaian acara dengan jadwal dan waktu yang ditentukan. Khususnya di Majalaya, mayoritas pelayanan *Wedding Organizer* masih menggunakan cara lama yakni datang langsung ke kediaman yang punya jasa *Wedding Organizer*. Sehingga diperlukan aplikasi *Wedding Organizer* untuk memudahkan para calon pengantin mendapatkan informasi seputar layanan jasa atau produk sesuai dengan kriteria yang diinginkan.

Maka dari itu, penulis akan meneliti, menganalisis, merancang dan mengimplementasikan aplikasi *Wedding Organizer* berbasis *web* dengan metode pengembangan *Model Driven Development (MDD)* yakni langkah-langkah dari awal sampai akhir pengerjaan skripsi. Melalui aplikasi ini diharapkan para pelanggan bisa memesan paket pernikahan atau produk sesuai dengan kriteria yang di inginkan.

Kata Kunci: Resepsi pernikahan, *Wedding Organizer*, berbasis *web*, *Model Driven Development (MDD)*.

ABSTRACT

Wedding reception is an event process that requires total preparation, sometimes both the bride and groom will face difficulties, so it is often necessary to provide Wedding Organizer services to assist in the planning and implementation of a series of events with a specified schedule and time. Especially in Majalaya, the majority of Wedding Organizer services still use the old method of coming directly to a residence that has a Wedding Organizer service. So we need a Wedding Organizer application to make it easier for prospective brides to get information about service products or products according to the desired criteria.

Therefore, the author will examine, analyze, design and implement a web-based Wedding Organizer application with the method of developing Model Driven Development (MDD), the steps from the beginning to the end of the thesis. Through this application, it is expected that customers can order wedding packages or products according to the desired criteria.

Keywords: Wedding reception, Wedding Organizer, web-based, Model Driven Development (MDD).

KATA PENGANTAR

Dengan mengucapkan puji syukur yang sebesar – besarnya ke hadirat Allah SWT, berkat rahmat-Nya penulis dapat menyelesaikan laporan skripsi sebagai syarat untuk memperoleh gelar Sarjana Komputer pada Program Studi Teknik Informatika di Fakultas Teknologi Informasi Universitas Bale Bandung.

Selama proses penulisan skripsi ini, penulis menemui beberapa hambatan, tetapi dengan terus semangat dan motivasi yang tinggi, penulis akhirnya dapat menyelesaikan skripsi ini. Maka dari itu penulis ingin mengucapkan terima kasih kepada:

1. Bapak Yudi Herdiana, ST. MT. Selaku Dekan Fakultas Teknologi Informasi Universitas Bale Bandung sekaligus merupakan dosen wali bagi penulis.
2. Bapak Yaya Suharya S.Kom, M.T. Selaku Ketua Program Studi Teknik Informatika Fakultas Teknologi Informasi Universitas Bale Bandung dan Dosen Pembimbing 2 yang senantiasa membimbing dan mengarahkan penulis dalam menyelesaikan skripsi ini.
3. Bapak Rustiyana, S.T.,M.T.,M.Pd. sebagai Dosen Pembimbing 1 juga senantiasa membimbing dan mengarahkan penulis dalam menyelesaikan skripsi ini.
4. Kedua orang tua yang telah memberikan dukungan baik moral maupun materi.
5. Teman-teman seperjuangan yang selalu semangat dan mensupport penulis hingga menyelesaikan skripsi ini.
6. Semua pihak yang tidak dapat disebutkan satu persatu namanya, yang telah membantu dalam pelaksanaan penyusunan skripsi.

Semoga Allah SWT, memberikan balasan yang berlipat ganda atas kebaikan yang telah diberikan oleh semua pihak kepada penulis. Aamiin.

Penulis menyadari penyusunan skripsi ini jauh dari kata sempurna, maka dari itu, penulis mengharapkan kritik dan saran yang membangun. Semoga penyusunan skripsi ini dapat bermanfaat bagi semua pihak.

Baleendah, Agustus 2018

Rifki Zulfiqor

DAFTAR ISI

LEMBAR PERSETUJUAN PENGUJI	
LEMBAR PERSETUJUAN PEMBIMBING	
LEMBAR PERSETUJUAN PROGRAM STUDI.....	
LEMBAR PERSETUJUAN FAKULTAS.....	
ABSTRAK	iii
ABSTRACT	iv
KATA PENGANTAR	v
DAFTAR ISI.....	vii
DAFTAR GAMBAR	x
DAFTAR TABEL.....	xi
DAFTAR LAMPIRAN.....	xii
BAB I PENDAHULUAN.....	1
1.1 Lata Belakang.....	1
1.2 Rumusan Masalah	2
1.3 Batasan Masalah.....	2
1.4 Tujuan	3
1.5 Metodologi	3
1.6 Sistematika	5
BAB II TINJAUAN PUSTAKA.....	6
2.1 Landasan Teori.....	6
2.2 Dasar Teori	7
2.2.1 Definisi Aplikasi	7
2.2.2 <i>Web</i>	7
2.2.3 Aplikasi berbasis <i>web</i>	7

2.2.4	<i>Model Driven Development(MDD)</i>	8
2.2.5	UML (<i>Unified Modeling Language</i>).....	10
2.2.6	Basis Data	15
2.2.7	HTML (<i>HyperText Markup Language</i>).....	17
2.2.8	MySQL.....	17
2.2.9	Visual Studio Code	18
2.2.10	Docker	19
2.2.12	CSS (<i>Cascading Style Sheet</i>)	19
BAB III METODOLOGI.....		20
3.1	Metodologi Penelitian	20
3.1.1	<i>Preminilary Investigation</i> (Investigasi awal)	21
3.1.2	<i>Problem Analysis</i> (Analisis masalah)	21
3.1.3	<i>Requirement Analysis</i> (Analisis kebutuhan)	21
3.2	Metodologi Pengembangan Sistem.....	22
3.2.1	<i>Design</i> (Desain).....	22
3.2.2	<i>Construction</i> (Kontruksi/pengkodean).....	23
3.2.3	<i>Implementaton</i> (Implementasi)	23
3.2.4	<i>Dokumentation</i> (Dokumentasi/Laporan Akhir)	23
BAB IV ANALISIS DAN PERANCANGAN		24
4.1	Analisis.....	24
4.1.1	Analisis Masalah	24
4.1.2	Analisis <i>Software</i>	24
4.1.3	Analisis Pengguna.....	25
4.1.4	<i>User Interface</i>	25
4.1.5	Fitur-fitur.....	25
4.1.6	Analisis Data	26

4.2 Perancangan	26
4.2.1 <i>Unified Modeling Language (UML)</i>	27
4.2.1.1 <i>Use case diagram</i>	27
4.2.1.2 <i>Activity diagram</i>	36
4.2.1.3 <i>Class diagram</i>	47
4.3 Perancangan Basis Data	47
4.3.1 Struktur Tabel.....	47
4.3.2 Desain.....	50
4.3.3 Hasil	62
4.3.3.1 Menjalankan sistem.....	62
BAB V KESIMPULAN DAN SARAN.....	72
6.1 Kesimpulan	72
6.2 Saran.....	72
DAFTAR PUSTAKA	73
RIWAYAT HIDUP.....	74
LAMPIRAN	

DAFTAR GAMBAR

Gambar 2.1 <i>Model Driven Development</i>	8
Gambar 3.1 Metodologi penelitian.....	20
Gambar 4.1 <i>Use case diagram</i>	27
Gambar 4.2 <i>Activity diagram login admin</i>	36
Gambar 4.3 <i>Activity diagram menu booking</i>	37
Gambar 4.4 <i>Activity diagram menu user</i>	38
Gambar 4.5 <i>Activity diagram menu produk</i>	39
Gambar 4.6 <i>Activity diagram menu paket nikah</i>	40
Gambar 4.7 <i>Activity diagram setting admin</i>	41
Gambar 4.8 <i>Activity diagram login user</i>	42
Gambar 4.9 <i>Activity diagram pesan paket nikah</i>	43
Gambar 4.10 <i>Activity diagram pesan produk</i>	44
Gambar 4.11 <i>Activity diagram pemesanan</i>	45
Gambar 4.12 <i>Activity diagram setting akun</i>	46
Gambar 4.13 Skema <i>class diagram</i>	47
Gambar 4.14 Rancangan Halaman <i>Login</i>	50
Gambar 4.15 Rancangan <i>Login</i> untuk daftar <i>user</i>	51
Gambar 4.16 Rancangan Halaman <i>dashboard user</i>	51
Gambar 4.17 Rancangan Halaman <i>dashboad user</i> (lanjutan).....	52
Gambar 4.18 Rancangan Halaman produk	52
Gambar 4.19 Rancangan Halaman paket nikah	53
Gambar 4.20 Rancangan Halaman buat <i>booking</i>	53
Gambar 4.21 Rancangan Halaman konfirmasi pesan	54
Gambar 4.22 Rancangan Halaman rincian pemesanan	54

Gambar 4.23 Rancangan Halaman <i>setting</i> akun	55
Gambar 4.24 Rancangan Halaman <i>dashboard</i> admin.....	55
Gambar 4.25 Rancangan Halaman data <i>user</i>	56
Gambar 4.26 Rancangan Halaman edit/lihat <i>user</i>	57
Gambar 4.27 Rancangan Halaman data produk.....	57
Gambar 4.28 Rancangan Halaman tambah produk.....	58
Gambar 4.29 Rancangan Halaman edit/lihat produk	58
Gambar 4.30 Rancangan Halaman data paket <i>wedding</i>	59
Gambar 4.31 Rancangan Halaman tambah paket <i>wedding</i>	59
Gambar 4.32 Rancangan Halaman data <i>booking</i>	60
Gambar 4.33 Rancangan Halaman edit/lihat <i>booking</i>	60
Gambar 4.34 Rancangan Halaman edit/lihat <i>booking</i> (lanjutan)	61
Gambar 4.35 Rancangan Halaman <i>setting</i> admin	61
Gambar 4.36 Halaman <i>dashboard user</i>	62
Gambar 4.37 Halaman <i>dashboard user</i>	62
Gambar 4.38 Halaman <i>login</i>	63
Gambar 4.39 Halaman daftar <i>user</i>	64
Gambar 4.40 Halaman produk	64
Gambar 4.41 Halaman paket nikah	65
Gambar 4.42 Halaman pemesanan.....	65
Gambar 4.43 Halaman pemesanan (lanjutan)	66
Gambar 4.44 Halaman <i>setting user</i>	66

DAFTAR TABEL

Tabel 2.1 Simbol <i>Use case</i> diagram.....	12
Tabel 2.2 Simbol <i>activity</i> diagram	14
Tabel 4.1 Analisis <i>software</i>	24
Tabel 4.2 Analisis data.....	26
Tabel 4.3 Analisis biaya.....	26
Tabel 4.4 Deskripsi admin dan pelanggan	27
Tabel 4.5 <i>Use case</i> admin	28
Tabel 4.6 <i>Use case</i> pelanggan	28
Tabel 4.7 <i>Use case login</i>	29
Tabel 4.8 <i>Use case logout</i>	29
Tabel 4.9 <i>Use case</i> mengelola <i>booking</i>	29
Tabel 4.10 <i>Use case</i> mengelola <i>user</i>	30
Tabel 4.11 <i>Use case</i> mengelola produk	31
Tabel 4.12 <i>Use case</i> mengelola paket pernikahan	32
Tabel 4.13 <i>Use case</i> mengelola <i>setting</i> admin	33
Tabel 4.14 <i>Use case</i> pesan paket pernikahan.....	34
Tabel 4.15 <i>Use case</i> pesan produk.....	34
Tabel 4.16 <i>Use case</i> konfirmasi pemesanan	35
Tabel 4.17 <i>Use case</i> setting akun.....	35
Tabel 4.18 Struktur tabel <i>Booking</i>	48
Tabel 4.19 Struktur tabel <i>Item</i>	48
Tabel 4.20 Struktur tabel <i>PhotoGallery</i>	48
Tabel 4.21 Struktur tabel <i>Product</i>	49
Tabel 4.22 Struktur tabel <i>ProductPhoto</i>	49

Tabel 4.23 Struktur tabel *User* 49

Tabel 4.24 Struktur tabel *UserPhoto* 50

BAB I

PENDAHULUAN

1.1 Latar Belakang

Pernikahan merupakan komitmen jangka panjang dan bersifat sakral, karena mempersatukan dua insan dan dua keluarga yang berbeda, pernikahan juga merupakan ibadah untuk umat Islam. Setiap pasangan tentunya ingin mempersiapkan dengan total dalam acara pesta pernikahan, misalnya persiapan tempat, tata rias, dekorasi, *photografi*, kartu undangan dan lain-lain, maka dari itu tidak sedikit yang memakai jasa *Wedding Organizer*.

Wedding Organizer sendiri merupakan layanan jasa yang membantu dalam persiapan dan pelaksanaan pernikahan. *Wedding Organizer* yang sudah berdiri sejak tahun 2008 yakni Linda Salon merupakan jasa *Wedding Organizer* yang cukup lama yang terjun di bisnis ini, Linda Salon juga menyediakan jasa tata rias, dekorasi, dan alat pesta lainnya yang sesuai dengan keinginan para calon pengantin zaman sekarang.

Dalam penyebaran informasi, Linda Salon menggunakan sosial media untuk memperlihatkan hasilnya dan testimoni dari pelanggan yang menggunakan jasanya, akan tetapi belum memiliki *web* pribadi untuk jangkauan yang lebih luas. Saat ini juga Linda Salon mengalami kendala di antaranya: 1. Kendala saat pengecekan jadwal pernikahan karena masih menggunakan media buku. 2. Pelanggan mengalami kendala untuk mengakses informasi seputar jasa *Wedding Organizer* yang disediakan oleh Linda Salon. 3. Pelanggan mengalami kebingungan untuk menentukan paket pernikahan yang sesuai dengan *budget*.

Berdasarkan permasalahan di atas, maka diperlukan sebuah aplikasi yang dapat memudahkan penjadwalan pernikahan untuk Linda Salon, juga untuk para calon pengantin yang ingin mengakses informasi dan memesan paket pernikahan atau produk sesuai dengan kriteria yang di inginkan

Dalam pelaksanaannya, penulis melakukan penelitian, perancangan dan kemudian mengimplementasikan dalam sebuah aplikasi berbasis *web* dengan judul penelitian yaitu **“MEMBANGUN APLIKASI WEDDING ORGANIZER BERBASIS WEB STUDI KASUS DI LINDA SALON”**.

1.2 Rumusan Masalah

Berdasarkan uraian latar belakang masalah diatas, adapun rumusan masalah adalah :

1. Bagaimana mengelola data jadwal resepsi pernikahan di Linda Salon ?
2. Bagaimana mengelola informasi mengenai *Wedding Organizer* agar dapat di akses dengan mudah ?
3. Bagaimana mengelola informasi *budget* paket resepsi pernikahan atau produk agar dapat di pesan dengan mudah ?

1.3 Batasan Masalah

Adapun aplikasi yang di buat oleh penulis adalah :

- a. Aplikasi yang dibuat hanya dua level pengguna yaitu admin dan pelanggan/*user*.
- b. Pengelolaan data *user/pelanggan* adalah data pelanggan yang sudah terdaftar di database admin.
- c. Pengelolaan data produk adalah data berupa harga dan deskripsi produk yang bisa dipesan.
- d. Pengelolaan paket pernikahan adalah data paket pernikahan yang di pesan oleh pelanggan.

1.4 Tujuan

Tujuan dari penelitian skripsi ini adalah :

1. Mempermudah dan mengorganisir dalam mengelola data pernikahan.
2. Untuk membantu pelanggan dalam memilih jasa *Wedding Organizer* yang terpercaya.
3. Memfasilitasi perkiraan biaya pernikahan untuk para pelanggan/user yang ingin memesan secara online.

1.5 Metodologi

Dalam pengumpulan data dan penyelesaian masalah dalam penelitian, penulis menggunakan metodologi *Model Driven Development (MDD)* yang ditulis oleh The McGraw-Hill Companies. Teknik pengembangan berbasis model (MDD) menekankan gambar model untuk membantu memvisualisasikan dan menganalisis masalah, mendefinisikan kebutuhan bisnis, dan merancang sistem informasi.

Berikut ini gambaran singkat metodologi yang digunakan penulis:

1. *Preminilary Investigation (Investigasi Awal)*

Dalam tahap awal ini penulis melakukan observasi dan pengamatan langsung ke tempat penelitian yang bertempat di Linda Salon sekaligus memohon ijin untuk melakukan penelitian di tempatnya, penulis melakukan pengumpulan data dan wawancara terhadap pihak-pihak yang bersangkutan untuk kebutuhan analisis masalah.

2. *Problem Analisys (Analisis Masalah)*

Penulis melakukan analisis masalah yang terjadi di Linda Salon dalam pengelolaan *wedding organizer* yang sedang berjalan, pemilik berpendapat bahwa ada beberapa kendala pada saat mengelola data pernikahan sehingga di butuhkan penyelesaian masalah yang tepat.

3. *Requirement Analysis (Analisis Kebutuhan)*

Penulis melakukan analisis kebutuhan apa saja yang diperlukan untuk membuat aplikasi yang akan dibangun dan alat/tools yang dibutuhkan dalam pengembangan sistem.

Metode Pengembangan Sistem

Dalam metodologi pengembangan sistem, ada beberapa tahapan yaitu:

1. *Design (Desain/Rancangan)*

Dalam tahap desain/rancangan, penulis membuat rancangan sistem menggunakan Star UML untuk membuat *use case diagram*, *activity diagram* dan *class diagram*. Kemudian membuat rancangan *user interface* penulis menggunakan *Baslamiq Mockup*.

2. *Construction (Konstruksi / penulisan kode)*

Dalam tahap penulisan kode, penulis menggunakan notepad++ dan Visual Studio Code untuk teks editor, XAMPP untuk *localhost* dan bahasa pemrograman PHP dan Bootstrap.

3. *Implementation (Implementasi)*

Dalam tahap implementasi, penulis mengimplementasikan aplikasi berbasis *web* yang di buat dengan cara *upload* ke *hosting web server*.

4. *Documentation (Dokumentasi / Laporan Akhir)*

Dalam tahap ini, penulis melakukan penyelesaian laporan akhir yaitu berupa skripsi yang terdapat dokumentasi.

1.6 Sistematika

Adapun sistematika dari laporan penelitian ini adalah sebagai berikut:

BAB I : PENDAHULUAN

Berisikan latar belakang, rumusan masalah, batasan, rumusan penelitian, penjelasan metodologi yang digunakan dalam pengumpulan data dan penyelesaian penelitian secara singkat dan sistematika yang digunakan dalam penulisan laporan.

BAB II : TINJAUAN PUSTAKA

Berisikan landasan teori yang bersumber dari jurnal-jurnal yang sesuai dengan objek penelitian dan dasar teori yang mendukung dalam penyelesaian permasalahan yang akan di bahas sehingga dapat memecahkan permasalahan yang di hadapi sehingga menjadi referensi saat dilakukannya penelitian.

BAB III : METODOLOGI

Berisikan penjelasan metodologi yang digunakan dan tahapan-tahapan yang dilakukan, penulis menggunakan metode *Model Driven Development(MDD)* pada saat penelitian di Linda Salon, yang selanjutnya dapat diperoleh satu jalan keluar untuk mengatasi masalah yang di hadapi oleh Linda Salon.

BAB IV : ANALISIS DAN PERANCANGAN

Berisikan hasil analisis sistem yang berjalan di Linda Salon dan menjelaskan segala bentuk perancangan sistem yang akan di buat seperti perancangan basis data dan perancangan sistem yang diusulkan. Setelah perancangan yakni menjalankan aplikasi yang di dalamnya berisi *screenshot* tampilan aplikasi dan uraian penggunaan sistem dari hasil pengembangan penelitian yang dilakukan penulis serta melakukan pengujian terhadap aplikasi yang dibuat.

BAB V : KESIMPULAN DAN SARAN

Pada bab terakhir ini, berisikan kesimpulan dari hasil penelitian, saran berisi usulan, memperbaiki kekurangan di dalam aplikasi, dan tindak lanjut oleh siapapun yang ingin mengembangkan skripsi ini di masa yang akan datang.

BAB II

TINJAUAN PUSTAKA

2.1 Landasan Teori

Dalam melakukan penelitian, penulis juga mempelajari referensi dari penelitian yang terdahulu sebagai landasan teori yang melandasi penelitian yang sedang dilakukan. Beberapa landasan teori tersebut yaitu :

1. “APLIKASI FAIRUZ WEDDING ORGANIZER BERBASIS WEB” oleh Rhesa Setya Wijaya. Pada artikel ini aplikasi ini dapat membantu para calon pengantin untuk memilih paket pernikahan dan membantu melakukan reservasi untuk memesan jasa . (Wijaya, 2017)
2. “SISTEM INFORMASI WEDDING PLANNER BERBASIS WEB” oleh Ina Najiah dan Suharyanto. Pada artikel ini disebutkan bahwa Dengan adanya sistem informasi *wedding planner* berbasis *website* ini memberikan gambaran kepada calon pengantin (*member*) untuk merencanakan pernikahannya dengan informasi-informasi yang tersedia di *website wedding planner* ini dan *website* ini membantu *vendor-vendor* atau WO dalam mempromosikan produk-produknya secara *online* dengan cepat dan tanpa biaya. (Najiah & Suharyanto, 2017)
3. “RANCANG BANGUN APLIKASI WEDDING ORGANIZER DI KOTA PONTIANAK BERBASIS WEB” oleh Renny Wulandari Ss, Helen Sasty Pratiwi dan Hafiz Muhardi. Pada artikel ini disebutkan bahwa aplikasi *wedding organizer* di Kota Pontianak mampu memfasilitasi pengguna dengan menyediakan fitur pencarian layanan produk/jasa dan aplikasi *wedding organizer* di Kota Pontianak dapat menjadi media promosi jasa layanan bagi *wedding organizer*. (Wulandari Ss, Pratiwi, & Muhardi, 2017)

2.2 Dasar Teori

2.2.1 Definisi Aplikasi

Menurut Supriyanto (2005,2), Aplikasi adalah program yang memiliki aktivitas pemrosesan perintah yang diperlukan untuk melaksanakan permintaan pengguna dengan tujuan tertentu.

Aplikasi adalah program komputer yang dibuat oleh suatu perusahaan komputer untuk membantu manusia dalam mengerjakan tugas – tugas tertentu, misalnya Ms. Word, Ms. Excel. Aplikasi berbeda dengan sistem operasi (yang menjalankan komputer), *utility* (yang melaksanakan perawatan atau tugas – tugas umum) dan bahasa pemrograman (yang digunakan untuk membuat program – program tertentu).

2.2.2 Web

Web adalah salah satu aplikasi internet yang terdiri dari perangkat lunak, kumpulan protokol dan seperangkat aturan yang memungkinkan kita untuk mengakses informasi di internet. *Web* menggunakan *hypertext* (teks yang terhubung ke teks lainnya) dan mendukung *file multimedia* sehingga dapat digunakan oleh pengguna internet di seluruh dunia. Dengan aplikasi web kita bisa mengkomunikasikan berbagai informasi sekaligus mencari informasi baru di internet. *World Wide Web* ditemukan oleh Tim Berners-Lee, seorang ilmuwan yang bekerja di pusat penelitian fisika CERN. (Sujatmiko, 2012)

2.2.3 Aplikasi berbasis *web*

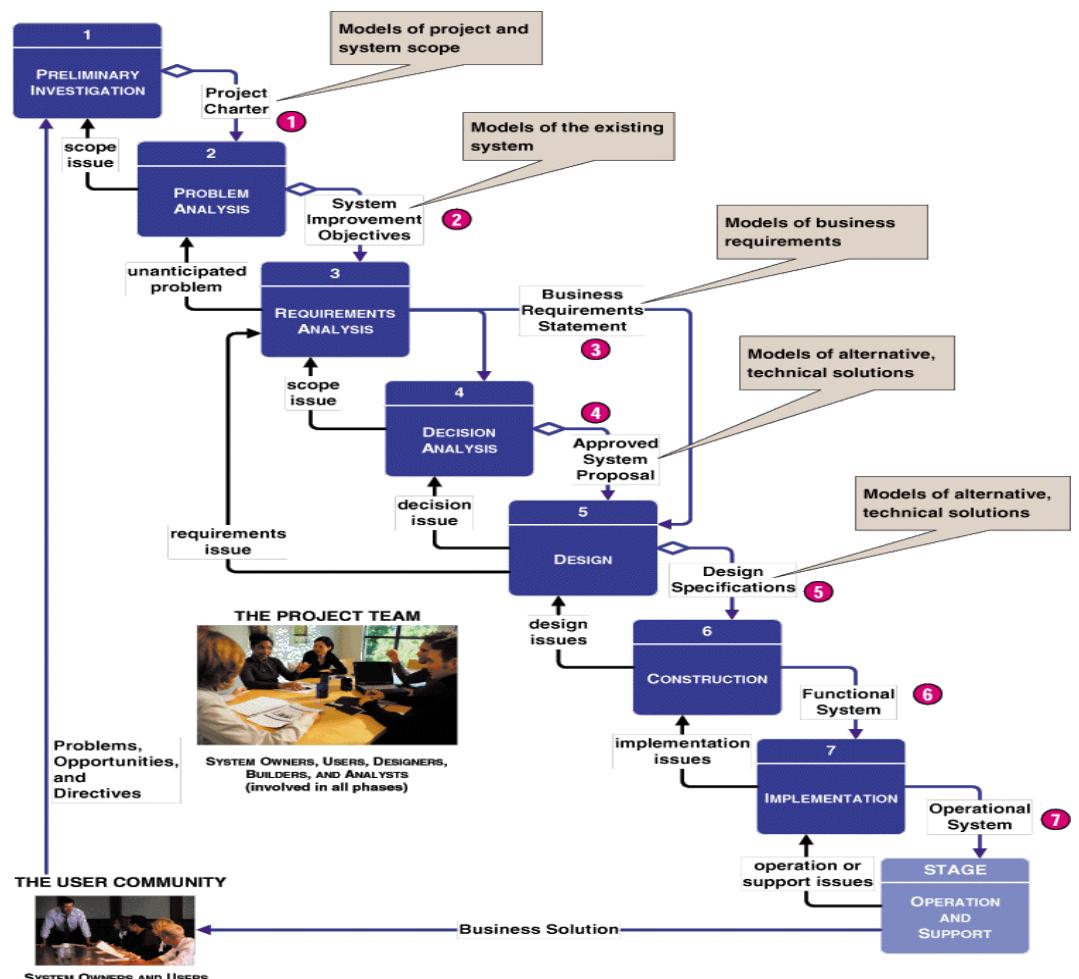
Aplikasi *web* adalah suatu aplikasi yang diakses menggunakan penjelajah *web* (*web browser*) melalui suatu jaringan seperti internet atau intranet. Ia juga merupakan perangkat lunak komputer yang dikodekan dalam bahasa yang didukung penjelajah *web* (*web browser*) seperti HTML, Javascript, AJAX, Java,

dan lain lain serta bergantung pada *browser* tersebut untuk menampilkan aplikasi. (Sujatmiko, 2012)

2.2.4 Model Driven Development (MDD)

Pemodelan adalah tindakan menggambar satu atau lebih representasi grafis (atau gambar) dari suatu sistem. Pemodelan adalah teknik komunikasi berdasarkan pepatah lama, "sebuah gambar bernilai seribu kata."

Teknik pengembangan berbasis model (MDD) menekankan gambar model untuk membantu memvisualisasikan dan menganalisis masalah, mendefinisikan kebutuhan bisnis, dan merancang sistem informasi. berikut ini gambar *Model Driven Development* yang digunakan :



Gambar 2.1 *Model Driven Development*

- a. Analisis dan desain sistem terstruktur - berpusat pada proses
- b. Teknik informasi - berpusat pada data
- c. Analisis dan desain berorientasi obyek - terpusat pada objek (integrasi data dan masalah proses)

Langkah langkah

1. *Preliminary investigation* (investigasi awal)

Tahap ini merupakan tahap awal dari pengembangan sistem. Fase ini berisikan investigasi awal ketika ingin merancang sebuah sistem, seperti wawancara, tinjauan langsung dan mempelajari dokumen perusahaan.

2. Problem *analysis* (Analisis masalah)

Problem *Analysis* ialah menganalisa masalah-masalah yang terdapat di lapangan. Tahap ini merupakan pengembangan dari tahap pertama. Pada tahap ini dilakukan analisis terhadap sistem yang telah ada saat itu. Tahap ini memberikan pemahaman yang lebih dalam bagi tim proyek mengenai permasalahan yang dihadapi. Analisis ini dilakukan untuk menjawab pertanyaan apakah keuntungan yang diperoleh setelah pemecahan masalah lebih besar daripada biaya yang dikeluarkan.

3. *Requirements analysis* (Analisis Kebutuhan)

Requirement Analysis ialah melakukan analisa terhadap kebutuhan perusahaan.. Pekerjaan pada tahap ini adalah mendefinisikan apa saja yang perlu dilakukan oleh system dan apa yang dibutuhkan dan diinginkan oleh pengguna dari sistem baru.

Tahap ini memerlukan perhatian yang besar karena jika terjadi kesalahan dalam menerjemahkan kebutuhan dan keinginan pengguna sistem maka dapat mengakibatkan adanya rasa tidak puas pada sistem final dan perlu diadakan modifikasi yang tentunya akan kembali mengeluarkan biaya.

4. *Decision analysis* (Analisis Keputusan)

Decision Analysis ialah melakukan analisa terhadap keputusan yang akan diambil berdasarkan solusi-solusi yang ditawarkan. Dalam analisis keputusan, umumnya terdapat berbagai alternatif untuk mendesain sistem informasi yang baru.

5. *Design* (Desain)

Setelah diperoleh proposal sistem yang disetujui, maka dapat mulai dilakukan proses desain dari sistem target. Tujuan dari tahap ini adalah untuk mentransformasikan *business requirement statement* menjadi spesifikasi desain untuk proses *construksi*. Dengan kata lain, tahap desain menyatakan bagaimana teknologi akan digunakan dalam sistem yang baru. Tahap ini memerlukan ide dan opini dari pengguna, *vendor*, dan spesialis IT.

6. *Construction* (Kontruksi)

Construction ialah tahapan melaksanakan pengujian pada komponen sistem secara individu dan sistem secara keseluruhan.

7. *Implementation* (Implementasi)

Mengimplementasikan penghubung antara sistem baru dan sistem lama, termasuk instalasi dari *software* yang dibeli atau disewa. Pada tahap ini dilakukan konstruksi basis data, program aplikasi, dan penghubung antara sistem dan pengguna. Beberapa dari komponen ini telah ada sebelumnya. Setelah dilakukan pengujian, maka sistem dapat mulai diimplementasikan.

2.2.5 UML (*Unified Modeling Language*)

UML (*Unified Modeling Language*) adalah bahasa pemodelan untuk sistem atau perangkat lunak yang berparadigma “berorientasi objek”. Pemodelan (*modeling*) sesungguhnya digunakan untuk penyederhanaan permasalahan-

permasalahan yang kompleks sedemikian rupa sehingga lebih mudah dipelajari dan dipahami (Nugroho, 2009) .

Beberapa literature menyebutkan bahwa UML menyediakan sembilan jenis diagram, yang lain menyebutkan delapan karena ada beberapa diagram yang digabung, misanya diagram komunikasi, diagram urutan dan diagram pewaktuan digabung menjadi diagram interaksi. Namun demikian model-model itu dapat dikelompokkan berdasarkan sifatnya yaitu statis atau dinamis. Jenis diagram yang akan digunakan yaitu antara lain:

1. *Class Diagram*

Class diagram adalah diagram yang menunjukkan *class-class* yang ada dari sebuah sistem dan hubungannya secara logika. *Class* diagram digunakan untuk menampilkan kelas-kelas dan paket-paket di dalam sistem.

Class diagram memberikan gambaran sistem secara sistem secara statis dan relasi antar mereka. Biasanya, dibuat beberapa *class* diagram untuk sistem tunggal. Beberapa diagram akan menampilkan *subset* dari kelas-kelas dan relasinya. Dapat dibuat beberapa diagram sesuai dengan yang diinginkan untuk mendapatkan gambaran lengkap terhadap sistem yang dibangun.

2. *Use-Case Diagram*

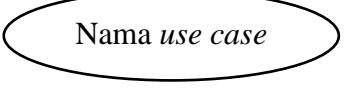
Usecase diagram adalah diagram *usecase* yang digunakan untuk menggambarkan secara ringkas siapa yang menggunakan sistem dan apa saja yang bisa dilakukannya. Diagram *usecase* tidak menjelaskan secara detail tentang penggunaan *usecase*, namun hanya memberi gambaran singkat hubungan antara *usecase*, aktor, dan sistem. Melalui diagram *usecase* dapat diketahui fungsi-fungsi apa saja yang ada pada sistem (Rosa-Salahudin, 2011: 130). Nama suatu *usecase* harus didefinisikan sesimple mungkin dan dapat dipahami.

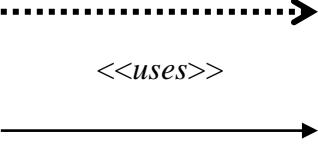
Komponen-komponen yang ada pada *usecase* adalah :

1. Aktor merupakan orang, proses atau sistem lain yang berinteraksi dengan sistem yang akan dibuat. Jadi walaupun simbol aktor dalam diagram *usecase* berbentuk orang, namun aktor belum tentu orang.

2. *Usecase* merupakan fungsionalitas yang disediakan sistem sebagai unit-unit yang saling berinteraksi atau bertukar pesan antar unit maupun aktor.
3. Relasi merupakan hubungan yang terjadi pada sistem baik antar aktor maupun antar *usecase* maupun antara *usecase* dan aktor. Relasi yang digunakan dalam diagram *usecase* antara lain :
 - a. *Assosiation* merupakan relasi yang digunakan untuk menggambarkan interaksi antara *usecase* dan aktor. Asosiasi juga menggambarkan berapa banyak objek lain yang bisa berinteraksi dengan suatu objek atau disebut *multiplicity*.
 - b. *Generalization* merupakan relasi yang menggambarkan *inheritance* baik aktor maupun *usecase*.
 - c. *Dependency* merupakan relasi yang menggambarkan ketergantungan antara *usecase* yang satu dengan *usecase* yang lain. Ada dua macam *dependency* yaitu *include* dan *extends*. *Include* menggambarkan bahwa jalannya suatu *usecase* memicu jalannya *usecase* lain. Misalnya *usecase login* di *include* oleh *usecase* memilih menu, artinya *usecase* memilih menu akan memicu dijalankannya *usecase login*. Sebelum aktor menjalankan *usecase* memilih menu, aktor harus menjalankan *usecase login* dulu. Dalam penggambaran diagram *usecase*, paruh mengarah kepada *usecase* yang di *include*. Sedangkan *extends* menggambarkan bahwa suatu *usecase* dijalankan karena ada persyaratan tertentu dari *usecase* lain. Misal, dalam sebuah sistem user tidak bisa menjalankan *login* sebelum dia mendaftar akun. Dalam diagram *usecase*, *usecase* daftar akun *mengextends* *usecase login*. Artinya aktor harus menjalankan *usecase* daftar akun dulu sebelum menjalankan *usecase login* karena *usecase login* memiliki syarat aktor yang melakukan *login* harus sudah melakukan pendaftaran akun. Arah panah *dependency* mengarah pada *usecase* yang memiliki syarat. (MSDN, n.d).

Tabel 2.1 Simbol *Use case* diagram

Simbol	Deskripsi
<p><i>Use Case</i></p> 	Fungsional yang disediakan sistem sebagai unit – unit yang saling bertukar pesan antar unit / aktor; biasanya dinyatakan dengan menggunakan kata kerja diawali frase nama <i>use case</i>
<p>Aktor / Actor</p>  Nama actor	Orang proses atau sistem lain yang berinteraksi dengan sistem informasi yang akan dibuat di luar sistem informasi yang akan dibuat tersebut, jadi walaupun simbol aktor adalah gambar orang, tapi aktor belum tentu merupakan orang; biasanya menggunakan kata benda di awal frase nama actor
<p>Asosiasi/ association</p> 	Komunikasi antara aktor dan <i>usecase</i> yang berpartisipasi pada <i>usecase</i> atau <i>usecase</i> memiliki interaksi dengan aktor
<p>Ekstensi / extend</p> <p><i><< extend >></i></p> 	Relasi <i>usecase</i> tambahan ke sebuah <i>use case</i> dimana <i>use case</i> yang ditambahkan dapat berdiri sendiri walaupun tanpa <i>use case</i> tambahan itu; biasanya <i>usecase</i> tambahan memiliki nama depan yang sama dengan <i>usecase</i> yang ditambahkan.
<p>Generalisasi / generalization</p> 	Hubungan generalisasi dan spesialis (umum – khusus) antara dua buah <i>use case</i> dimana fungsi yang satu adalah fungsi yang umum dari lainnya.
<p>Menggunakan / include / uses</p> <p><i><<include>></i></p>	Relasi <i>usecase</i> tambahan ke sebuah <i>use case</i> yang ditambahkan memerlukan <i>use</i>

 <i><<uses>></i>	<p><i>case</i> ini untuk menjalankan fungsinya atau sebagai syarat dijalankan <i>usecase</i> ini.</p>
--	---

3. *Activity Diagram*

Activity Diagram adalah diagram yang menggambarkan *workflow* (aliran kerja) atau aktivitas dari sebuah sistem atau proses bisnis. Yang perlu diperhatikan adalah bahwa diagram aktivitas menggambarkan aktivitas sistem bukan apa yang dilakukan aktor, jadi aktivitas yang dapat dilakukan oleh sistem.

Komponen yang ada pada *activity diagram* antara lain :

1. *Activity* atau *state* : Menunjukkan aktivitas yang dilakukan.
2. *Initial activity* atau *initial state* : Menunjukkan awal aktivitas dimulai.
3. *Final Activity* atau *final state* : Menunjukkan bagian akhir dari aktivitas.
4. *Decission* : Digunakan untuk menggambarkan test kondisi untuk memastikan bahwa *control flow* atau *object flow* mengalir lebih ke satu jalur. Jumlah jalur sesuai yang diinginkan.
5. *Merge* : Berfungsi menggabungkan *flow* yang dipecah oleh *decission*.
6. *Synchronization* : Dibagi menjadi 2 yaitu *fork* dan *join*. *Fork* digunakan untuk memecah *behaviour* menjadi *activity* atau *action* yang paralel, sedangkan *join* untuk menggabungkan kembali *activity* atau *action* yang *paralel*.
7. *Swimlanes* : Memecah *activity diagram* menjadi baris dan kolom untuk membagi tanggung jawab obyek-obyek yang melakukan aktivitas.
8. *Transition* : Menunjukkan aktivitas selanjutnya setelah aktivitas sebelumnya.

Tabel 2.2 Simbol *activity diagram*

Simbol	Keterangan
	Titik Awal
	Titik Akhir
	<i>Activity</i>
	Pilihan untuk mengambil keputusan
	<i>Fork</i> ; Digunakan untuk menunjukkan kegiatan yang dilakukan secara <i>parallel</i> atau untuk menggabungkan dua kegiatan <i>parallel</i> menjadi satu
	<i>Rake</i> ; Menunjukkan adanya dekomposisi
	Tanda waktu
	Tanda pengiriman
	Tanda penerimaan
	Aliran akhir

2.2.6 Basis Data

Basis data (*database*) adalah kumpulan informasi yang disimpan didalam komputer secara sistematik sehingga dapat diperiksa menggunakan suatu program komputer untuk memperoleh informasi dari basis data tersebut (Sujatmiko, 2012).

Basis data (*database*) adalah suatu kumpulan data yang disusun dalam bentuk tabel-tabel yang saling berkaitan maupun berdiri sendiri dan disimpan secara bersama-sama pada suatu media. Basis data dapat digunakan oleh satu atau lebih program aplikasi secara optimal, data disimpan tanpa mengalami ketergantungan pada program yang akan menggunakannya (Kadir, 1999).

Adapun komponen – komponen yang ada pada basis data adalah sebagai berikut:

1. *Table*

Tabel adalah kumpulan dari suatu *field* dan *record*. Dalam hal ini biasanya *field* ditunjukan dalam bentuk kolom dan *record* ditunjukan dalam bentuk baris.

2. *Field*

Field adalah sebutan untuk mewakili suatu *record*. Misalnya seorang pegawai dapat dilihat datanya melalui *field* yang diberikan padanya seperti nip, nama, alamat, dan lain-lain.

3. *Record*

Record adalah kumpulan elemen-elemen yang saling berkaitan menginformasikan tentang suatu isi data secara lengkap. Satu *record* mewakili satu data atau informasi tentang seseorang misalnya, nomor daftar, nama pendaftar, alamat, tanggal masuk.

4. *Primary Key*

Primary key adalah suatu kolom (*field*) yang menjadi titik acuan pada sebuah tabel, bersifat unik dalam artian tidak ada satu nilai pun yang sama atau kembar dalam tabel tersebut, dan dalam satu tabel hanya boleh ada satu *primary key*.

5. *Foreign Key*

Foreign key atau disebut juga kunci relasi adalah suatu kolom dalam tabel yang digunakan sebagai “kaitan” untuk melengkapi satu hubungan yang

didapati dari tabel induk, dan biasanya hubungan yang terjalin antar tabel adalah satu ke banyak (*one to many*).

6. *Index*

Index adalah struktur basis data secara fisik, yang digunakan untuk optimalisasi pemrosesan data dan mempercepat proses pencarian data.

2.2.7 HTML (*HyperText Markup Language*)

HTML adalah bahasa komputer yang digunakan untuk membuat sebuah halaman web atau program yang digunakan untuk menulis (membuat) halaman web di internet. Fasilitas / bahasa ini biasanya mempunyai ekstensi .htm, .html atau shtml. Seiring perkembangan kebutuhan HTML saat ini telah memasuki versi 5 dan biasa disebut dengan HTML5. HTML5 adalah sebuah bahasa markah untuk menstrukturkan dan menampilkan isi dari WWW, sebuah teknologi inti dari internet. HTML5 adalah revisi kelima dari HTML (yang pertama kali diciptakan pada tahun 1990 dan versi keempatnya, HTML4 pada tahun 1997) dan hingga bulan juni 2011 masih dalam pengembangan. Tujuan utama pengembangan HTML5 adalah untuk memperbaiki teknologi HTML agar mendukung teknologi multimedia terbaru, mudah dibaca oleh manusia dan mudah dimengerti oleh mesin.

2.2.8 MySQL

MySQL adalah salah satu jenis *database server* yang sangat terkenal dan banyak digunakan untuk membangun aplikasi *web* yang menggunakan *database* sebagai sumber dan pengolahan datanya.

MySQL dikembangkan oleh perusahaan swedia bernama MySQL AB yang pada saat ini bernama Tcx DataKonsult AB sekitar tahun 1994-1995, namun cikal bakal kodennya sudah ada sejak tahun 1979. Awalnya Tcx merupakan perusahaan pengembang *software* dan konsultan database, dan saat ini MySQL sudah diambil alih oleh Oracle Corp.

Kepopuleran MySQL antara lain karena MySQL menggunakan SQL sebagai bahasa dasar untuk mengakses databasenya sehingga mudah untuk digunakan, kinerja *query* cepat, dan mencukupi untuk kebutuhan *database* perusahaan-perusahaan yang berskala kecil sampai menengah, MySQL juga bersifat *open source* (tidak berbayar) .

MySQL merupakan *database* yang pertama kali didukung oleh bahasa pemrograman *script* untuk internet (PHP dan Perl). MySQL dan PHP dianggap sebagai pasangan *software* pembangun aplikasi *web* yang ideal. MySQL lebih sering digunakan untuk membangun aplikasi berbasis web, umumnya pengembangan aplikasinya menggunakan bahasa pemrograman *script* PHP.

MySQL didistribusikan dengan *license open source* GPL (*General Public License*) mulai versi 3.23 pada bulan juni 2000. Software MySQL bisa diunduh melalui *website* resminya di <http://www.MySQL.org> atau di <http://www.mysql.com>.

2.2.9 Visual Studio Code

Visual Studio Code adalah sebuah aplikasi *editor* kode yang tidak hanya tersedia untuk *Windows*, tapi juga tersedia untuk sistem operasi *Linux* dan *Mac OS*. *Visual Studio Code* juga mendukung berbagai jenis bahasa pemrograman. Mulai dari *JavaScript*, *Java*, *PHP*, *C++*, *C#*, *Go*, *JSON*, dan lainnya. Aplikasi *editor* ini bahkan secara otomatis mengidentifikasi jenis bahasa pemrograman yang digunakan dan memberikan variasi warna sesuai dengan setiap fungsi dalam rangkaian kode tersebut.

Fitur menarik di *Visual Studio Code* adalah kemampuan menambah ekstensi. Sehingga para *developer* dapat menambahkan ekstensi agar bisa menggunakan fitur-fitur yang tidak ada di *Visual Studio Code*. Misalnya ekstensi *Reach Native Tools*, yang menyediakan dukungan terhadap *framework React* di *Visual Studio Code*.

2.2.10 Docker

Docker pada awal didirikannya pada tahun 2009 menggunakan nama dotCloud Inc. Namun, pada tahun 2013 dotCloud diubah menggunakan nama *Docker* hingga tulisan ini dibuat.

Docker adalah sebuah aplikasi yang bersifat *open source* yang berfungsi sebagai wadah/*container* untuk mengepak/memasukkan sebuah *software* secara lengkap beserta semua hal lainnya yang dibutuhkan oleh *software* tersebut dapat berfungsi. Pengaturan *software* beserta *file*/hal pendukung lainnya akan menjadi sebuah *Image* (istilah yang diberikan oleh *docker*). Kemudian sebuah instan dari *Image* tersebut kemudian disebut *Container*.

2.2.11 CSS (*Cascading Style Sheet*)

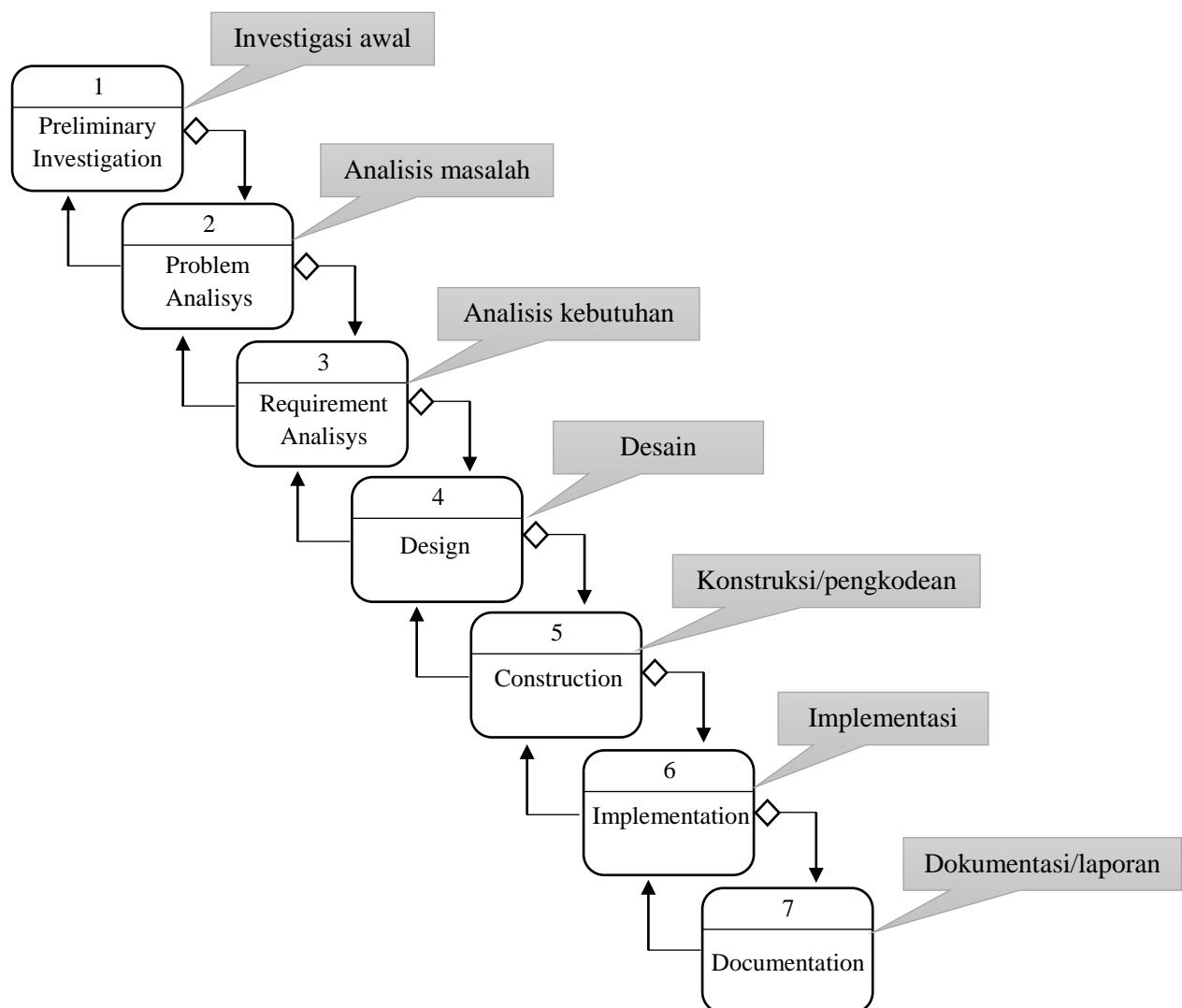
CSS (*Cascading Style Sheet*) adalah kumpulan perintah yang dibentuk dari berbagai sumber yang disusun menurut urutan tertentu sehingga mampu mengatasi konflik gaya / *style*.

BAB III

METODOLOGI

3.1 Metodologi Penelitian

Dalam metodologi penelitian penulis menggunakan Model *Driven Development (MDD)* yang di tulis oleh The McGraw-Hill Companies, berikut tahapan dan gambar skema pengumpulan data dan pengembangan sistem yang di



Gambar 3.1 Metodologi Penelitian

Deskripsi Tahapan penelitian

3.1.1 *Preminilary Investigation (Investigasi awal)*

Dalam investigasi awal, penulis melakukan observasi dan pengamatan di Linda Salon yang beralamat di Jalan Rancajigang RT 01 RW 15 Desa Padamulya Kecamatan Majalaya. Kemudian penulis bertemu dengan pemiliknya yang bernama ibu Karwati dan meminta izin untuk melakukan penelitian di tempatnya. Setelah melakukan izin, penulis melakukan pengumpulan data dengan wawancara kepada pemilik mengenai barang yang disewakan seperti: tenda, panggung, kursi, dekorasi, baju pengantin, dan lain-lain,

Selain itu, sistem yang berjalan saat ini yakni tebagi menjadi 3 tahapan yaitu tahap konsultasi awal, tahap konsultasi lanjutan dan pelaksanaan acara. Adapun tahapan-tahapan konsultasi awal meliputi, konsep tema pernikahan calon pengantin, mengatur anggaran pesta pernikahan sesuai biaya, pengaturan tempat (*indoor* atau *outdoor*), pengaturan posisi (dekorasi, meja, panggung, kursi tamu dan lain-lain), serta memberi masukkan dalam memilih hiburan, photo dan sebagainya.

3.1.2 *Problem Analysis (Analisis masalah)*

Setelah melakukan pengumpulan data dalam investigasi awal, penulis merumuskan masalah-masalah yang ada di Linda Salon diantaranya:

1. Kesulitan mengelola dan mengorganisir data resepsi pernikahan.
2. Pelanggan mengalami kesulitan mengakses atau mendapatkan informasi *Wedding Organizer*.
3. Pelanggan kesulitan membuat perkiraan *budget* resepsi pernikahan.

3.1.3 *Requirement Analysis (Analisis kebutuhan)*

Dalam segi kebutuhan alat/tools yang digunakan dalam pengembangan sistem, penulis menggunakan Laptop Asus X454Y yang dipergunakan untuk

berbagai tahapan dalam penelitian, seperti perancangan sistem, dokumentasi hingga pembuatan laporan baik proposal maupun skripsi.

Sedangkan bahan yang digunakan dalam melaksanakan penelitian ini diantaranya adalah:

1. Docker
2. Visual Studio Code
3. Star UML
4. Balsamiq Mockup
5. Google Chrome

Setelah mengetahui kebutuhan apa saja yang akan digunakan, penulis tidak menerapkan analisis keputusan karena sudah mengetahui apa yang akan dibuat, jadi penulis langsung ke tahap pengembangan sistem yakni tahap desain.

3.2 Metodologi Pengembangan Sistem

3.2.1 *Design* (Desain)

Desain untuk rancangan aplikasi yang akan dibuat penulis berdasarkan hasil analisis yang sudah dilaksanakan sebelumnya. Alat/tools yang akan digunakan dalam tahap rancangan ini yaitu Star UML dan *Balsamiq Mockup*.

Dalam perancangan sistem, penulis menggunakan Star UML. Tahap pertama yaitu membuat *usecase* diagram untuk menggambarkan fungsionalitas utama setiap aktor pada sistem. Selanjutnya membuat *activity* diagram untuk menggambarkan aliran dari suatu aktivitas ke aktivitas lainnya pada sistem. Selanjutnya penulis membuat *class* diagram untuk menggambarkan bagaimana relasi antar model dan fungsi-fungsi dalam model tersebut.

Dalam perancangan *user interface* sistem informasi akademik ini, penulis akan membuat *mockup* dengan menggunakan aplikasi *Balsamiq Mockup*. Penulis akan membuat desain *user interface* dengan memperhatikan desain yang *user friendly*, agar pengguna dapat dengan mudah menggunakan sistem informasi yang dibangun.

3.2.2 *Construction* (Kontruksi/pengkodean)

Dalam pengkodean penulis melakukan secara bertahap dimulai dengan menerjemahkan rancangan dan desain aplikasi yang telah dibuat dalam bentuk mockup sebelumnya ke dalam bahasa pemrograman, selanjutnya menggunakan *visual studio code* sebagai *text editor*, penulis juga menggunakan *Docker* sebagai virtual mesin dan bahasa pemrograman yang digunakan adalah *javascript* dan *graphql*.

3.2.3 *Implementation* (Implementasi)

Dalam hal implementasi penulis mengerjakan dan menerapkan sesuai dengan yang direncanakan yaitu membuat aplikasi *wedding organizer* berbasis *web* di Linda Salon yang sesuai dengan permintaan pemilik dan sesuai data-data yang telah dikumpulkan oleh penulis pada tahap sebelumnya. Penulis juga melakukan pengujian terhadap aplikasi yang di buat memastikan aplikasi berjalan dengan baik dan terbebas dari *error* dan *bugs*.

3.2.4 *Dokumentation* (Dokumentasi /Laporan Akhir)

Dalam tahapan akhir ini penulis melaksanakan penyelesaian laporan akhir yaitu berupa skripsi yang di dalamnya terdapat tahapan-tahapan yang dilakukan dan dokumentasi pada saat merancang aplikasi, implementasi dan penggunaan aplikasi.

BAB IV

ANALISIS DAN PERANCANGAN

4.1 Analisis

4.1.1 Analisis Masalah

Berdasarkan analisis yang dilakukan penulis di Linda Salon, menemui beberapa kendala diantaranya: Kendala saat pengecekan jadwal pernikahan karena masih menggunakan media buku, Pelanggan mengalami kendala untuk mengakses informasi seputar jasa *Wedding Organizer* yang disediakan oleh Linda Salon, Pelanggan mengalami kebingungan untuk menentukan paket pernikahan yang sesuai dengan *budget*.

4.1.2 Analisis Software

Berikut adalah spesifikasi *software* minimum yang digunakan dalam pembuatan aplikasi.

Tabel 4.1 Analisis *Software*

SOFTWARE	KEGUNAAN
Visual studio code versi 1.19.2	<i>text editor</i> yang mudah dan membantu dalam penulisan code
Google chrome versi 68	<i>browser</i> yang digunakan sebagai pembuka aplikasi yang di buat
Docker	Virtual mesin
OS Windows 10	<i>System operasi</i> yang digunakan
Astah Community	Pembuatan UML
Baslamiq Mockup	Mendesain rancangan awal aplikasi

4.1.3 Analisis Pengguna

Penganalisaan pengguna adalah yang berkaitan dengan yang akan memakai aplikasi ini, pengguna aplikasi adalah admin dan para pelanggan yang ingin mengetahui informasi seputar Linda Salon. Aplikasi ini memberikan kemudahan untuk penggunanya dalam mengolah dan menginput data, juga desain aplikasi ini memiliki kemudahan dalam pengoperasianya sehingga pengguna tidak kebingungan saat menjalankannya.

4.1.4 *User Interface*

User interface bertujuan untuk membuat aplikasi yang dibuat menjadi mudah di pahami admin atau pelanggan. *User interface* yang dibangun juga harus memperhatikan desain yang user *friendly*, jadi pada saat admin atau pelanggan yang menggunakan aplikasi ini dapat di operasikan dengan mudah sehingga tidak kebingungan pada saat menggunakan aplikasi. Selain desain yang simpel juga harus memperhatikan warna *background* yang digunakan, karena jika menggunakan warna yang terlalu cerah maka membuat mata cepat lelah dan tidak konsentrasi, maka dari itu aplikasi ini hanya sedikit penggunaan warna yang cerah.

4.1.5 Fitur-fitur

Fitur-fitur yang dibuat dalam aplikasi ini dimaksudkan untuk para pengguna lebih mudah mengelola sebuah data. Berikut adalah fitur-fitur yang disediakan oleh aplikasi ini:

1. Pengelolaan paket nikah, untuk menampilkan paket-paket dan mengedit paket nikah.
2. Pengelolaan pelanggan, untuk menampilkan pelanggan yang terdaftar dan mengedit pelanggan.
3. Pengelolaan produk, untuk menampilkan produk yang di sediakan dan mengedit produk.
4. Menu pendaftaran, untuk mendaftar sebagai pelanggan yang terdaftar di data Salon Linda.

5. Menu booking, untuk menampilkan item yang sudah di booking oleh pelanggan/user.
6. Menu pemesanan, untuk menampilkan produk yang dibooking dan pengeditan pemesanan.
7. Menu *setting* pelanggan, untuk mengedit akun pelanggan.
8. Menu *setting* admin, untuk mengedit akun admin.

4.1.6 Analisis Data

Berikut adalah analisis data berupa data masukkan, proses dan keluaran yang menunjang aplikasi ini.

Tabel 4.2 Tabel Analisis Data

Masukkan	Proses	Keluaran
Pelanggan mengisi data dan <i>password</i> pada saat <i>login</i>	Data pelanggan di simpan di <i>database</i>	Data pelanggan
Pelanggan memilih tanggal booking/ pernikahan	Aplikasi menganalisa jadwal	Bila kosong bisa di <i>booking</i> atau bila isi maka pilih tanggal yang lain
Pelanggan memilih paket pernikahan	Aplikasi menghitung biaya sesuai dengan pilihan paket pelanggan	Rincian biaya di tampilkan

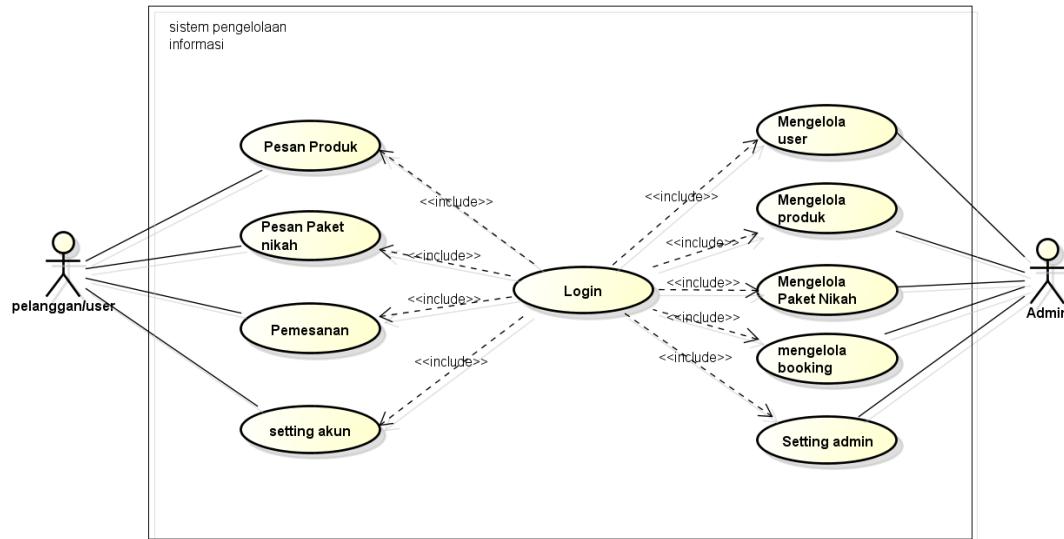
4.2 Perancangan

Pada skripsi ini perancangan model yang digunakan penulis adalah *use case* diagram, *scenario use case*, *activity* diagram dan *class* diagram.

4.2.1 Unified Modeling Language (UML)

4.2.1.1 Use case diagram

Use case diagram di buat untuk menunjukkan fungsionalitas utama dari setiap level *user* pada aplikasi yang digambarkan dengan *actor*. Terdapat 2 *actor* dalam *usecase* diagram ini yaitu admin dan pelanggan, juga 10 *usecase* utama.



Gambar 4.1 *Use case* Diagram

Berikut adalah deskripsi pendefinisian admin dan pelanggan pada perancangan aplikasi *wedding organizer* berbasis *web*:

Tabel 4.4 Deskripsi Admin dan Pelanggan

No	Aktor	Deskripsi
1.	Admin	Admin bertugas untuk mengelola data pelanggan, data barang dan data <i>booking</i> pada aplikasi
2.	Pelanggan	Pelanggan adalah pengguna aplikasi yang bisa melihat daftar menu dan memesan bila sudah <i>login</i>

a. *Use case* Aplikasi

Berikut adalah deskripsi pendefinisian *Use case* pada perancangan aplikasi *wedding organizer* berbasis *web* :

Tabel 4.5 *Use case Admin*

No	Use Case	Deskripsi
1.	<i>Login/Logout</i>	Merupakan proses untuk melakukan <i>Login</i> (masuk) dan <i>Logout</i> (keluar) dari <i>system</i> oleh admin
2.	Mengelola <i>Booking</i>	Merupakan pengeditan dan penghapusan data <i>booking</i> .
3.	Mengelola <i>user</i>	Merupakan pengeditan dan penghapusan data <i>user/pelanggan</i> .
4.	Mengelola paket nikah	Merupakan penambahan, pengeditan dan penghapusan data paket nikah.
5.	Mengelola produk	Merupakan penambahan, pengeditan dan penghapusan data produk.
6.	<i>Setting</i> admin	Merupakan pengeditan dan penghapusan data admin.

Tabel 4.6 *Use case Pelanggan*

No	Use Case	Deskripsi
1.	Pendaftaran Pelanggan	Merupakan proses penginputan data sebagai pelanggan yang terdaftar.
2.	Melihat paket nikah	Merupakan pilihan paket yang disediakan dan bisa memesan bila sudah daftar menjadi pelanggan atau login.
3.	Melihat produk	Merupakan pilihan produk yang di sediakan dan bisa memesan bila sudah daftar menjadi pelanggan atau login.
4.	Melihat pemesanan	Merupakan menu yang berisi pesanan yang sudah dibooking oleh pelanggan.
5.	<i>Setting</i> akun	Merupakan pengeditan pada <i>password</i> maupun data pelanggan

Tabel 4.7 *Use case Login*

Nama Use case : Login	
Aksi Aktor	Reaksi Sistem
Skenario Normal	
1. Memasukkan <i>Username</i> dan <i>Password</i>	
	2. Memeriksa <i>valid</i> tidaknya data yang dimasukan
	3. Masuk ke aplikasi Linda Salon Berbasis Web
Skenario Alternatif	
1. Memasukkan <i>Username</i> dan <i>Password</i>	
	2. Memeriksa <i>valid</i> tidaknya data yang dimasukkan
	3. Menampilkan pesan <i>Login</i> tidak <i>valid</i>
4. Memasukkan <i>Username</i> dan <i>Password</i> yang <i>valid</i>	
	5. Memeriksa <i>valid</i> tidaknya data yang dimasukan
	6. Masuk ke aplikasi Linda Salon Berbasis Web

Tabel 4.8 *Use case Logout*

Nama Use case : Logout	
Aksi Aktor	Reaksi Sistem
Skenario Normal	
1. Memilih menu <i>Logout</i>	2. Melakukan <i>Logout</i>

b. *Use case* untuk admin

Tabel 4.9 *Use case* Mengelola Booking

Nama <i>Use case</i> : Mengelola Booking	
Aksi Aktor	Reaksi Sistem
Skenario Normal	
	1. Memeriksa <i>Login</i>
2. Masuk ke menu <i>booking</i>	
	3. Menampilkan menu <i>booking</i> .
4. Mengubah status dan memasukkan deskripsi lalu klik <i>Button Save</i> .	
	5. Data tersimpan
Skenario Alternatif	
	1. Memeriksa <i>Login</i>
2. Masuk ke menu <i>booking</i>	
	3. Menampilkan menu <i>booking</i> .
4. Mengubah status dan deskripsi lalu klik <i>Button cancel</i> .	
	5. Tidak menyimpan data yang telah dibuat
	6. Kembali ke menu <i>booking</i> .

Tabel 4.10 *Use case* Mengelola User

Nama <i>Use case</i> : Mengelola user	
Aksi Aktor	Reaksi Sistem
Skenario Normal	
	1. Memeriksa <i>Login</i>
2. Masuk ke menu <i>user</i>	
	3. Menampilkan menu <i>user</i>
4. Memilih edit/lihat <i>user</i>	
	5. Menampilkan menu edit/lihat <i>user</i>
6. Mengubah foto atau data	

pada halaman edit/lihat <i>user</i> lalu klik <i>Button Save.</i>	
	7. Data tersimpan
Skenario Alternatif	
2. Masuk ke menu <i>user</i>	1. Memeriksa <i>Login</i>
4. Memilih edit/lihat <i>user</i>	3. Menampilkan menu <i>user</i> .
6. Mengubah foto atau data pada halaman edit/lihat <i>user</i> lalu klik <i>Button cancel.</i>	5. Menampilkan menu edit/lihat <i>user</i>
	7. Tidak menyimpan data yang telah dibuat
	8. Kembali ke menu pelanggan.

Tabel 4.11 *Use case* Mengelola Produk

Nama Use case : Mengelola produk	
Aksi Aktor	Reaksi Sistem
Skenario Normal	
	1. Memeriksa <i>Login</i>
2. Masuk ke menu produk	2. Menampilkan menu produk
4. Memilih tambah produk	3. Menampilkan menu produk
6. Memasukkan foto dan deskripsi pada halaman tambah barang lalu klik <i>Button Save.</i>	4. Menampilkan menu tambah produk
	5. Menampilkan menu tambah produk
	6. Data tersimpan
Skenario Alternatif	
2. Masuk ke menu produk	1. Memeriksa <i>Login</i>
4. Memilih tambah produk	2. Menampilkan menu produk.
	3. Menampilkan menu tambah produk

6. Memasukkan foto dan deskripsi pada halaman tambah barang lalu klik <i>Button cancel.</i>	
	7. Tidak menyimpan data yang telah dibuat
	8. Kembali ke menu produk

Tabel 4.12 *Use case* Mengelola Paket Nikah

Nama <i>Use case</i> : Mengelola paket nikah	
Aksi Aktor	Reaksi Sistem
Skenario Normal	
	1. Memeriksa <i>Login</i>
2. Masuk ke menu paket	
	3. Menampilkan menu paket
4. Memilih tambah paket	
	5. Menampilkan menu tambah paket
6. Memasukkan foto dan deskripsi pada halaman tambah paket lalu klik <i>Button Save.</i>	
	7. Data tersimpan
Skenario Alternatif	
	1. Memeriksa <i>Login</i>
2. Masuk ke menu paket	
	3. Menampilkan menu paket
4. Memilih tambah paket	
	5. Menampilkan menu tambah paket
6. Memasukkan foto dan deskripsi pada halaman tambah paket lalu klik <i>Button cancel.</i>	
	7. Tidak menyimpan data yang telah dibuat
	8. Kembali ke menu paket pernikahan

Tabel 4.13 *Use case* Mengelola *Setting Admin*

Nama <i>Use case</i> : Mengelola <i>setting admin</i>	
Aksi Aktor	Reaksi Sistem
Skenario Normal	
	1. Memeriksa <i>Login</i>
2. Masuk ke menu <i>setting</i>	
	3. Menampilkan menu <i>setting</i>
4. Memasukkan foto atau data pada halaman <i>setting admin</i> lalu klik <i>Button Save</i> .	
	5. Data tersimpan
Skenario Alternatif	
	1. Memeriksa <i>Login</i>
2. Masuk ke menu <i>setting</i>	
	3. Menampilkan menu <i>setting</i>
4. Memasukkan foto atau data pada halaman <i>setting admin</i> lalu klik <i>Button cancel</i> .	
	5. Tidak menyimpan data yang telah dibuat
	6. Kembali ke menu <i>setting</i>

c. *Use case* untuk pelanggan

Tabel 4.14 *Use case* Pesan Paket Nikah

Nama <i>Use case</i> : pesan paket nikah	
Aksi Aktor	Reaksi Sistem
Skenario Normal	
	1. Memeriksa <i>Login</i>
2. Pilih menu paket nikah	
	3. Menampilkan menu paket nikah
4. Pilih tombol <i>booking</i> dan pilih tanggal <i>booking</i> lalu klik buat	
5. Pilih paket yang di inginkan	
	6. Paket yang sudah di pilih

	masuk ke keranjang pemesanan dan tersimpan
Skenario Alternatif	
1. Memeriksa <i>Login</i>	
2. Pilih menu paket nikah	
	3. Menampilkan menu paket nikah.
4. Pilih tombol <i>booking</i> dan pilih tanggal <i>booking</i> lalu klik <i>cancel</i>	
	5. Tidak menyimpan data yang telah dibuat
	6. Kembali ke menu paket

Tabel 4.15 *Use case* Pesan Produk

Nama <i>Use case</i> : Pesan produk	
Aksi Aktor	Reaksi Sistem
Skenario Normal	
	1. Memeriksa <i>Login</i>
2. Pilih menu produk	
	3. Menampilkan menu produk
4. Pilih tombol <i>booking</i> dan pilih tanggal <i>booking</i> lalu klik buat	
5. Pilih produk yang diinginkan	
	6. Produk yang sudah di pilih masuk ke keranjang pemesanan dan tersimpan
Skenario Alternatif	
	1. Memeriksa <i>Login</i>
2. Pilih menu produk	
	3. Menampilkan menu produk
4. Pilih tombol <i>booking</i> dan pilih tanggal <i>booking</i> lalu klik <i>cancel</i>	
	5. Tidak menyimpan data yang telah dibuat
	6. Kembali ke menu produk

Tabel 4.16 *Use case* Konfirmasi Pemesanan

Nama <i>Use case</i> : Konfirmasi pemesanan	
Aksi Aktor	Reaksi Sistem
Skenario Normal	
	1. Memeriksa <i>Login</i>
2. Pilih menu pemesanan	
	3. Menampilkan pesanan yang sudah di pilih dalam keranjang pemesanan
4. Memilih pesanan dan konfirmasi	
	5. Konfirmasi di terima oleh admin
	6. Data tersimpan
Skenario Alternatif	
	1. Memeriksa <i>Login</i>
2. Masuk ke menu pemesanan	
	3. Menampilkan menu pemesanan
4. Tidak memilih pesanan dan klik <i>Button cancel</i>	
	5. Tidak menyimpan data yang telah dibuat
	6. Kembali ke menu pemesanan

Tabel 4.17 *Use case* Setting Akun

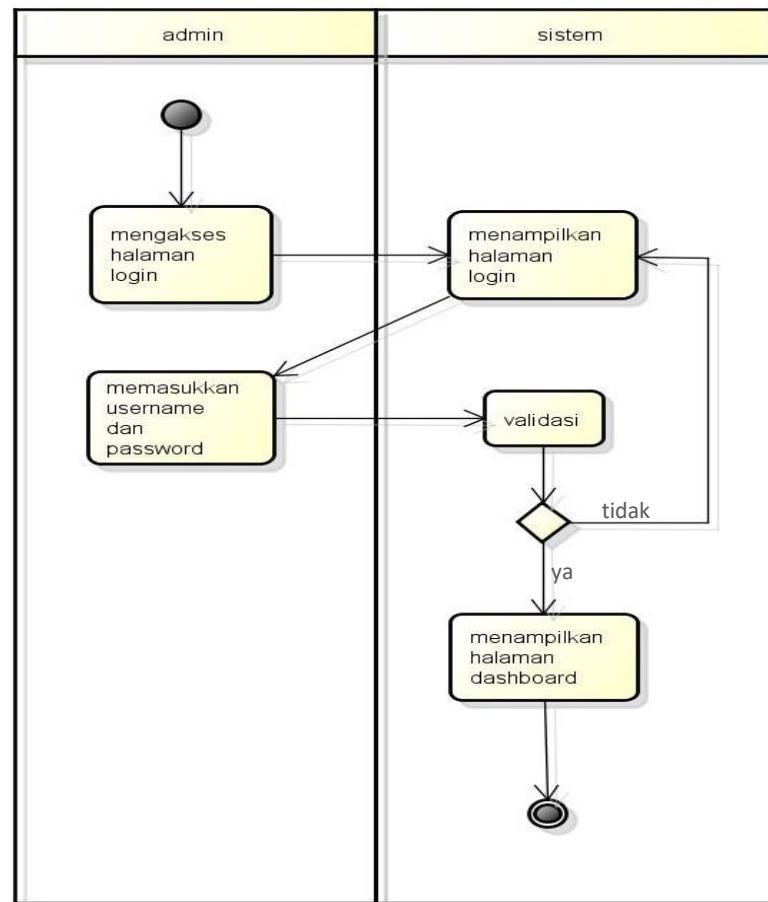
Nama <i>Use case</i> : Setting akun	
Aksi Aktor	Reaksi Sistem
Skenario Normal	
	1. Memeriksa <i>Login</i>
2. Masuk ke menu setting akun	
	3. Menampilkan menu akun
4. Mengedit data akun dan klik <i>Button Save</i>	
	5. Data tersimpan
Skenario Alternatif	

	1. Memeriksa <i>Login</i>
2. Masuk ke menu setting	
	3. Menampilkan menu akun
4. Mengedit data akun dan klik <i>Button cancel</i>	
	5. Tidak menyimpan data yang telah dibuat
	6. Kembali ke menu akun

4.2.1.2 Activity diagram

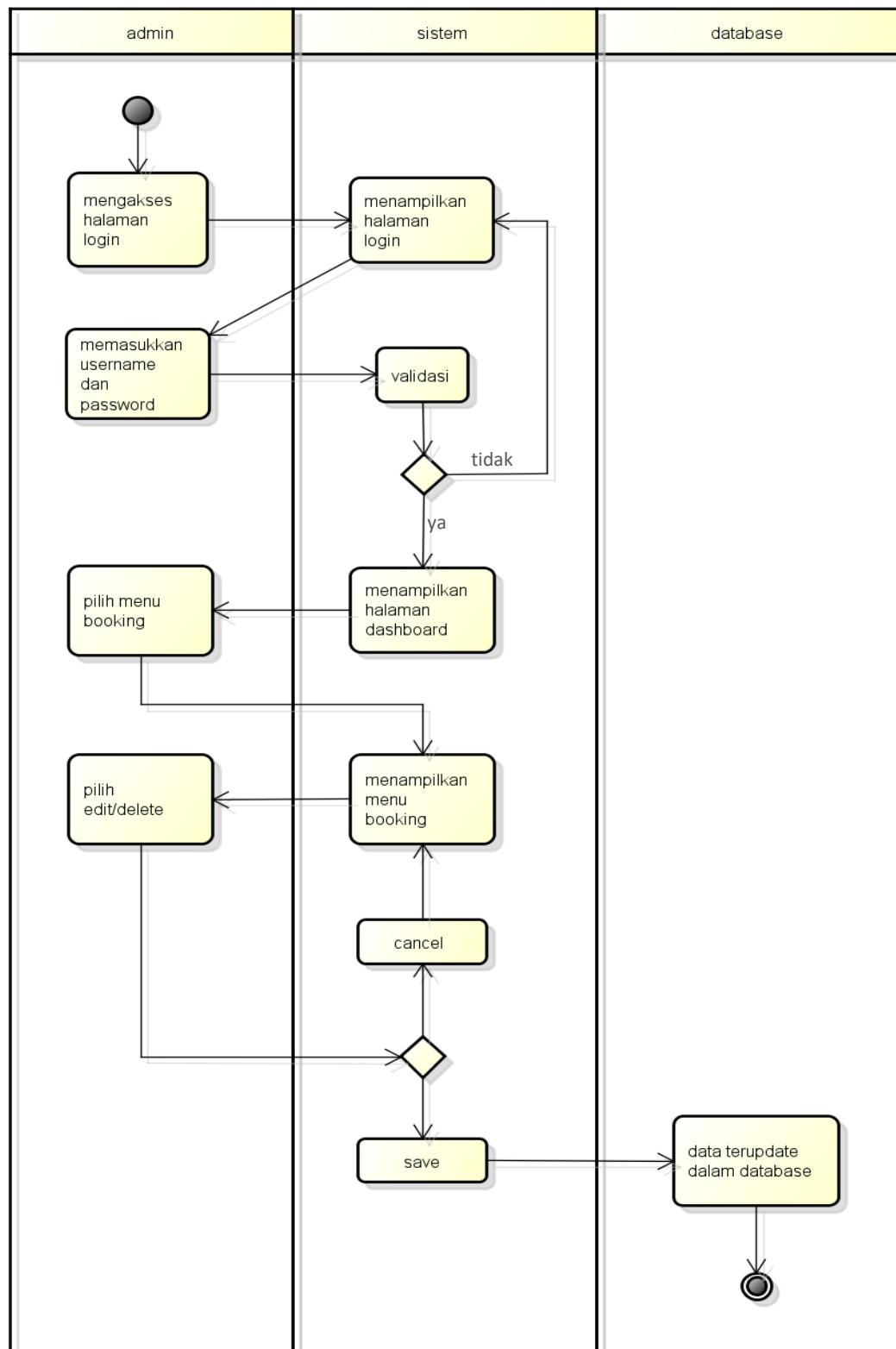
Activity diagram ini dibuat untuk menunjukkan aktivitas yang dilakukan admin dan pelanggan serta timbal balik yang dilakukan aplikasi terhadap user secara otomatis. Activity diagram untuk admin adalah sebagai berikut:

1. Activity diagram login admin



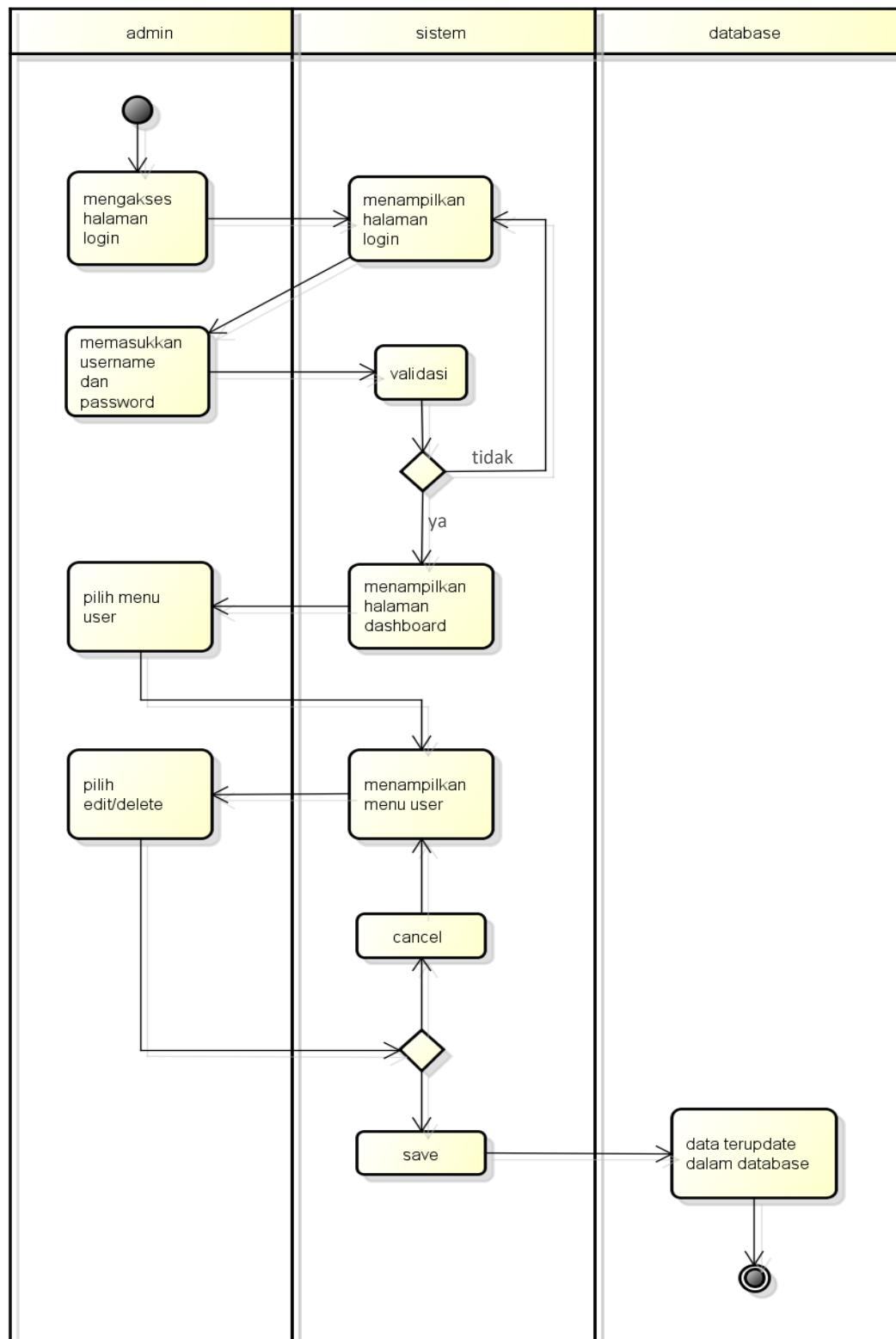
Gambar 4.2 activity diagram login admin

2. Activity diagram menu booking



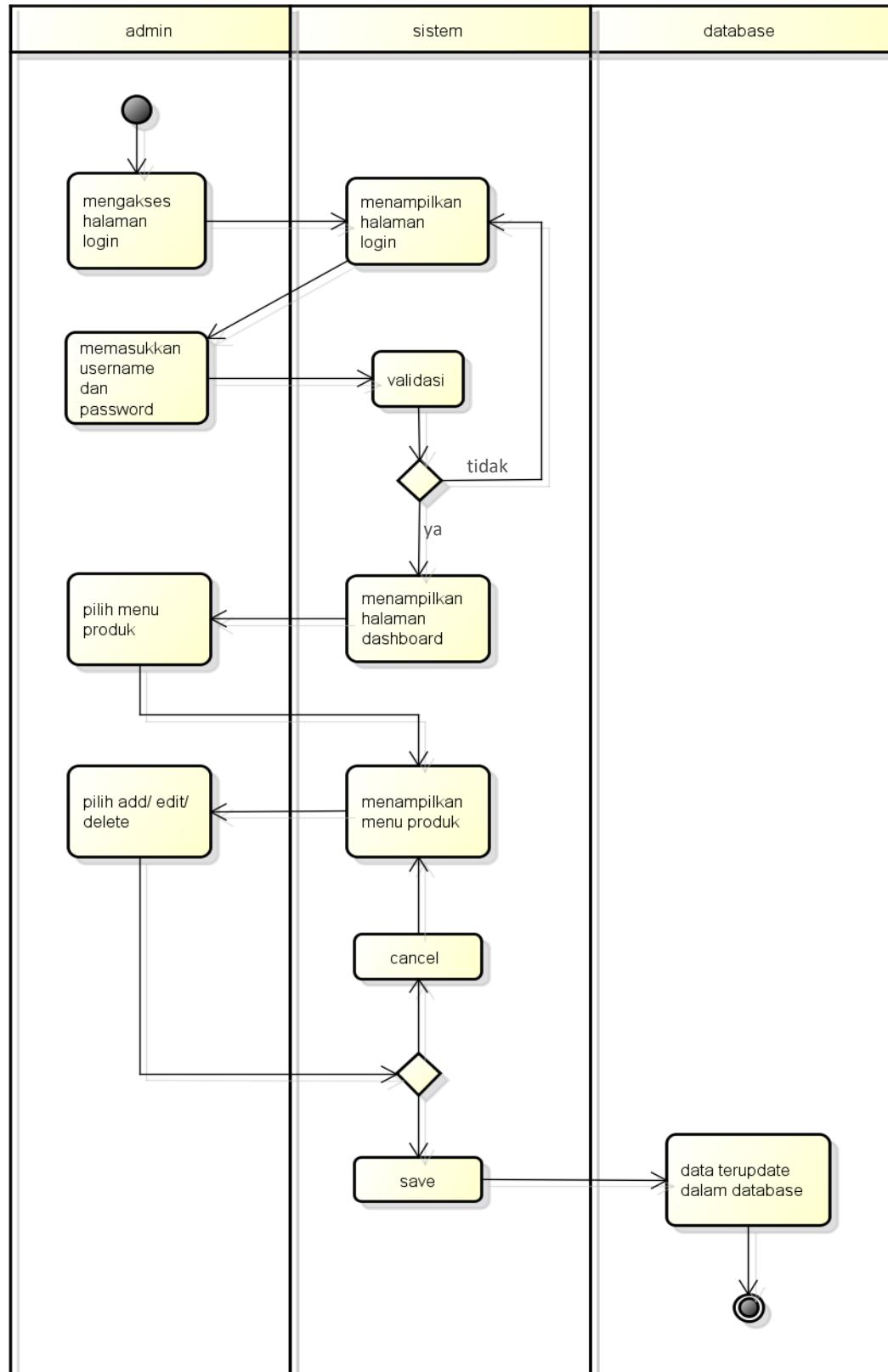
Gambar 4.3 activity diagram menu booking

3. Activity diagram menu user



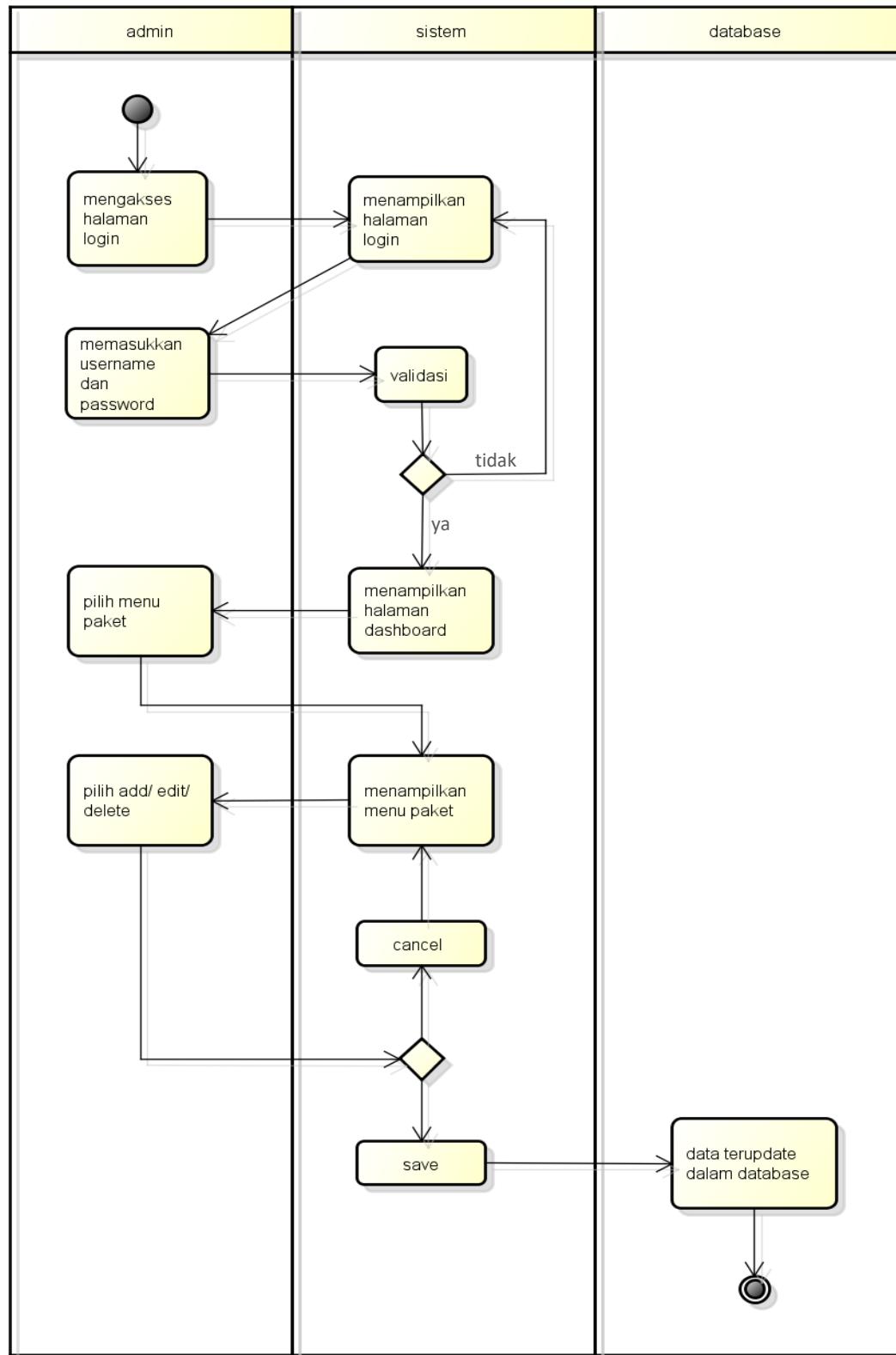
Gambar 4.4 *activity diagram menu user*

4. Activity diagram menu produk



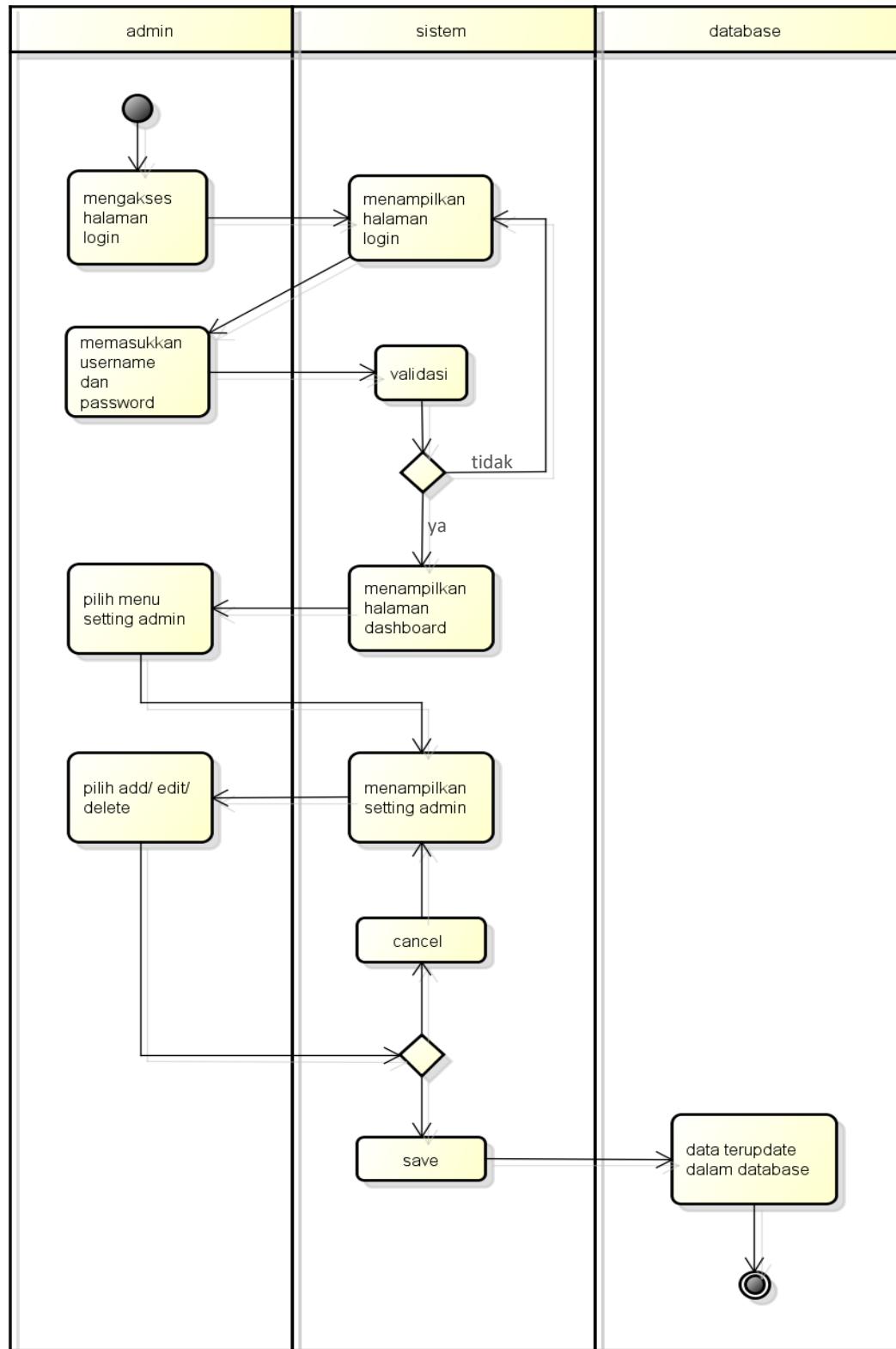
Gambar 4.5 *activity diagram* menu produk

5. *Activity diagram menu paket nikah*



Gambar 4.6 *activity diagram menu paket nikah*

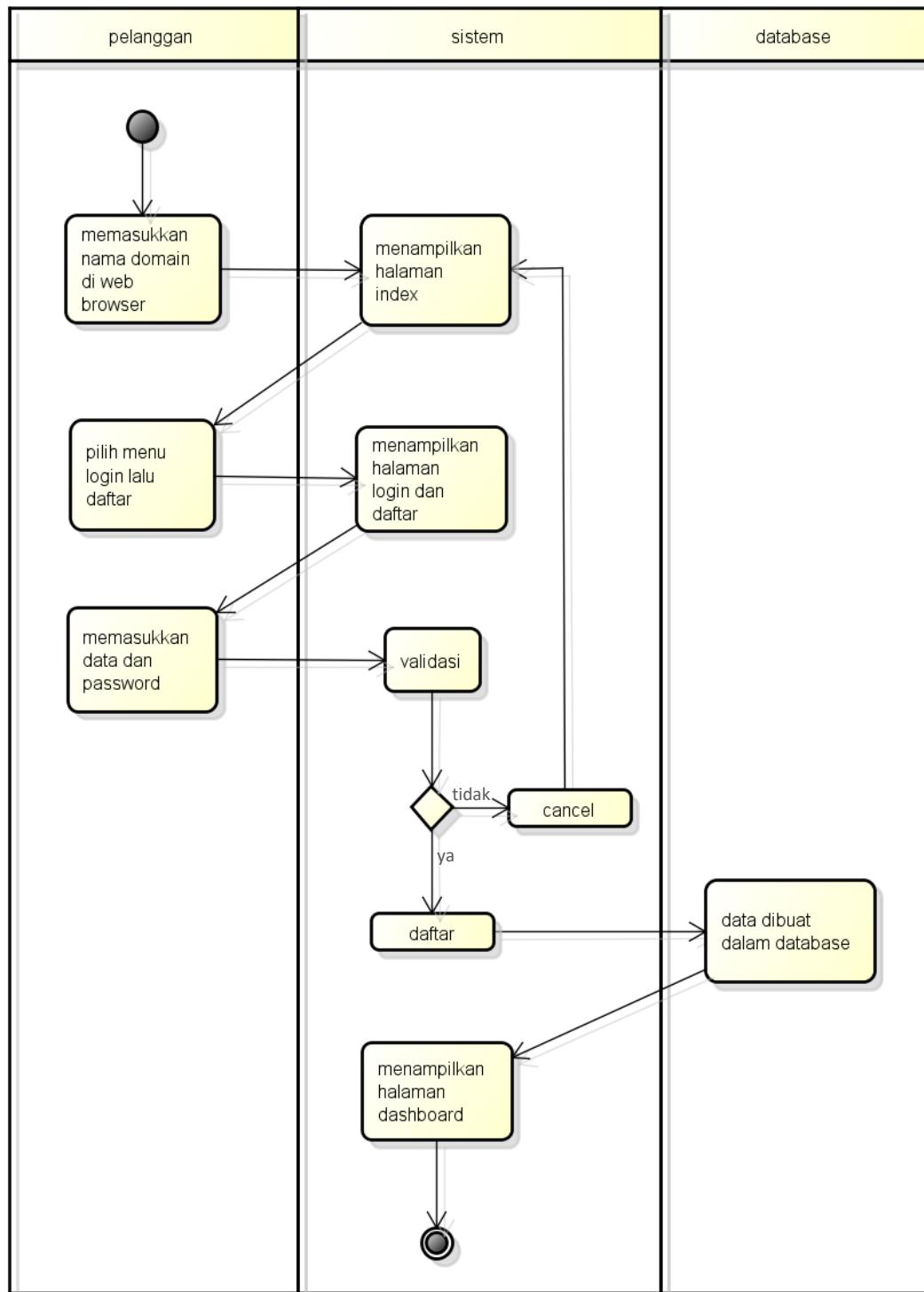
6. Activity diagram setting admin



Gambar 4.7 *activity diagram setting admin*

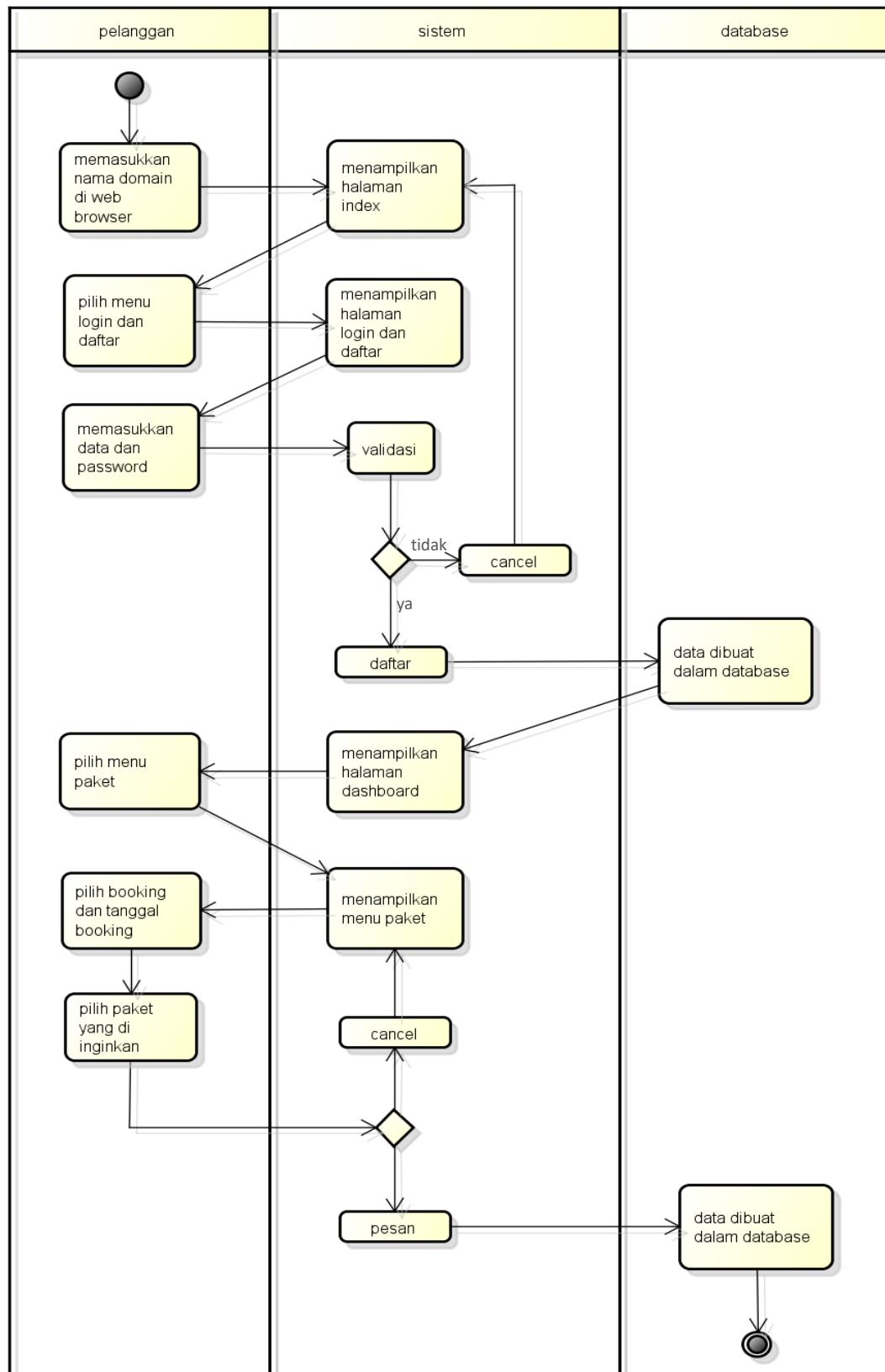
Activity diagram untuk pelanggan adalah sebagai berikut:

1. Activity diagram login user



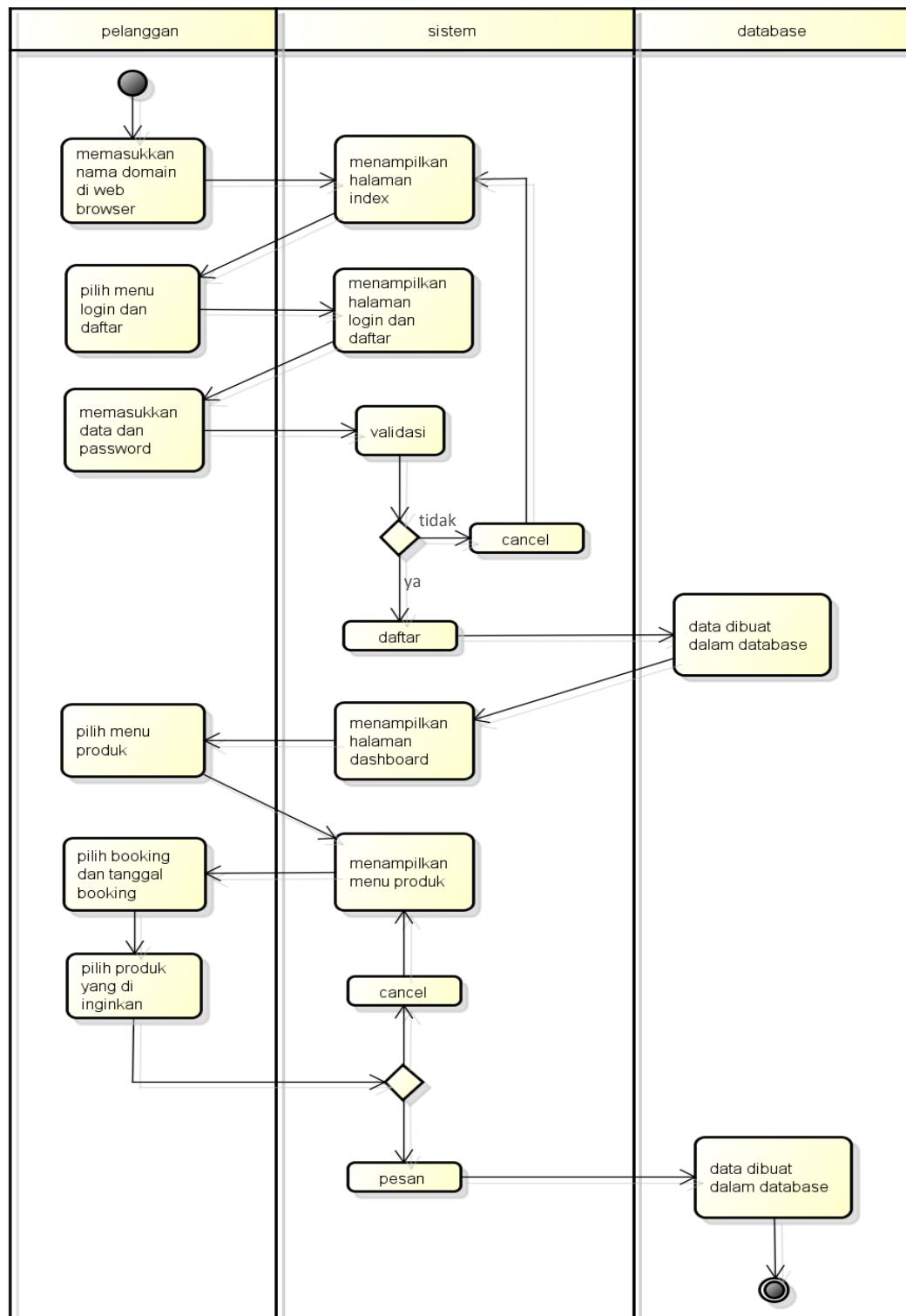
Gambar 4.8 activity diagram login user

2. Activity diagram pesan paket nikah



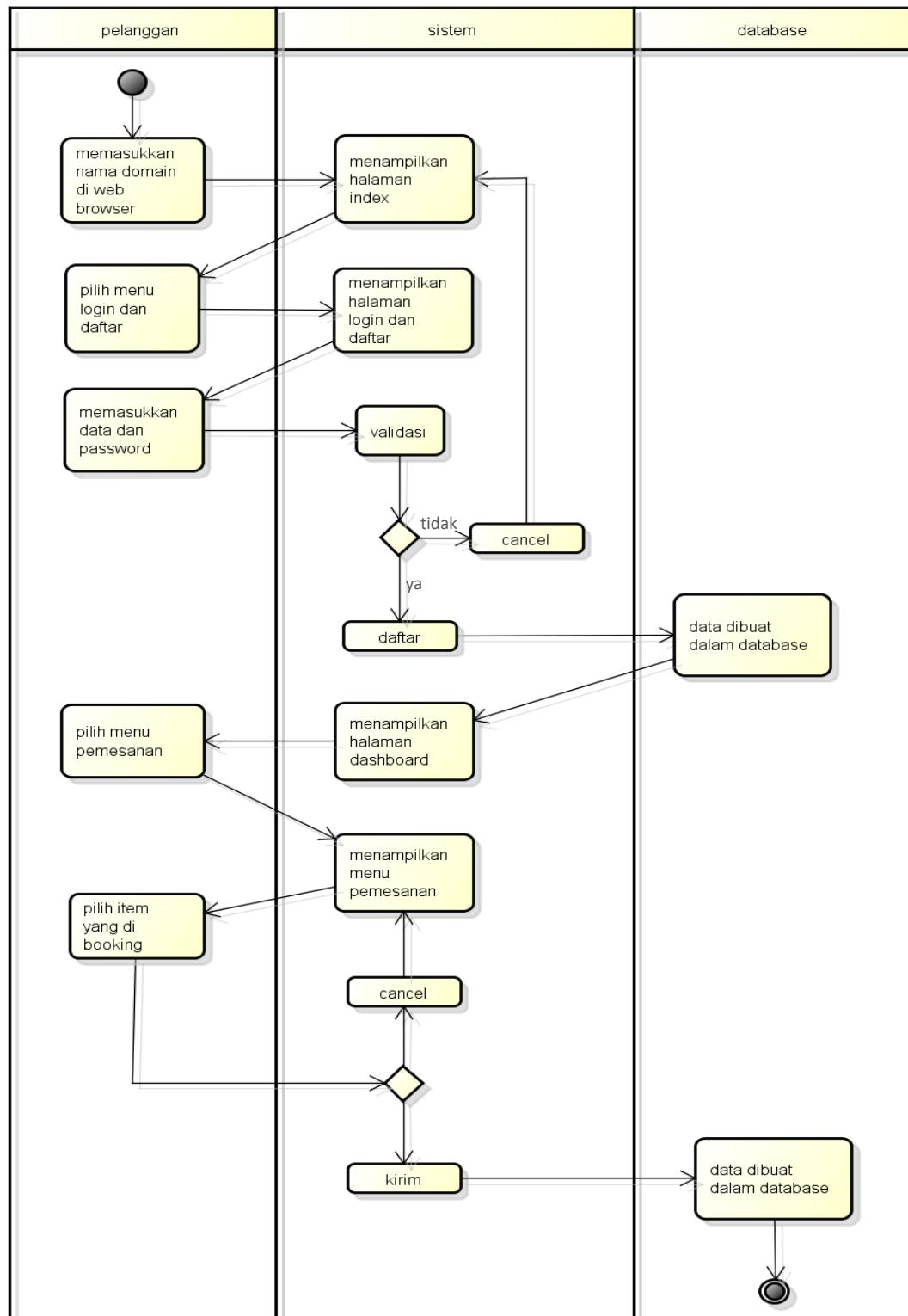
Gambar 4.9 *activity diagram pesan paket nikah*

3. Activity diagram pesan produk



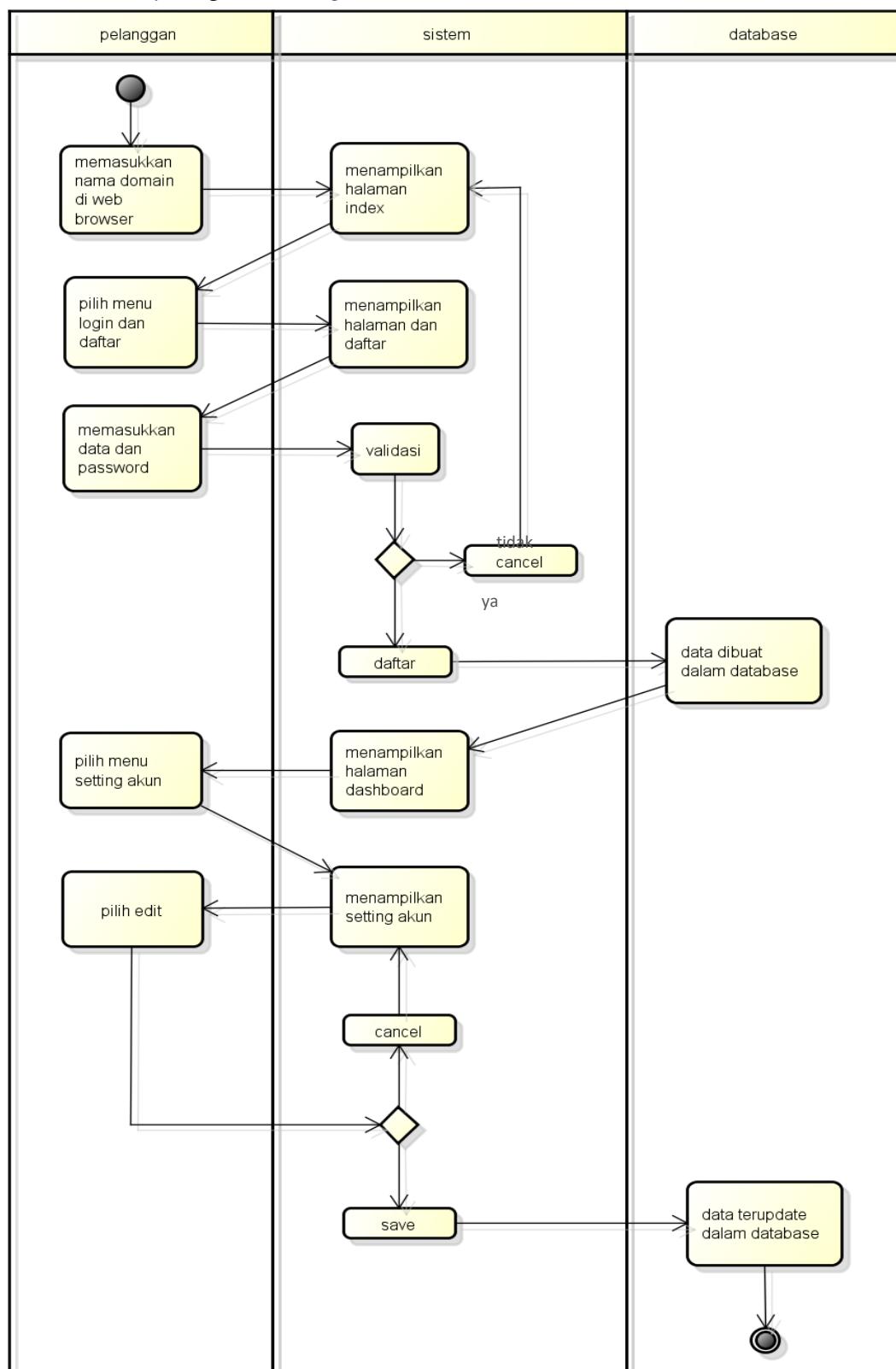
Gambar 4.10 *activity diagram pesan produk*

4. Activity diagram pemesanan



Gambar 4.11 activity diagram pemesanan

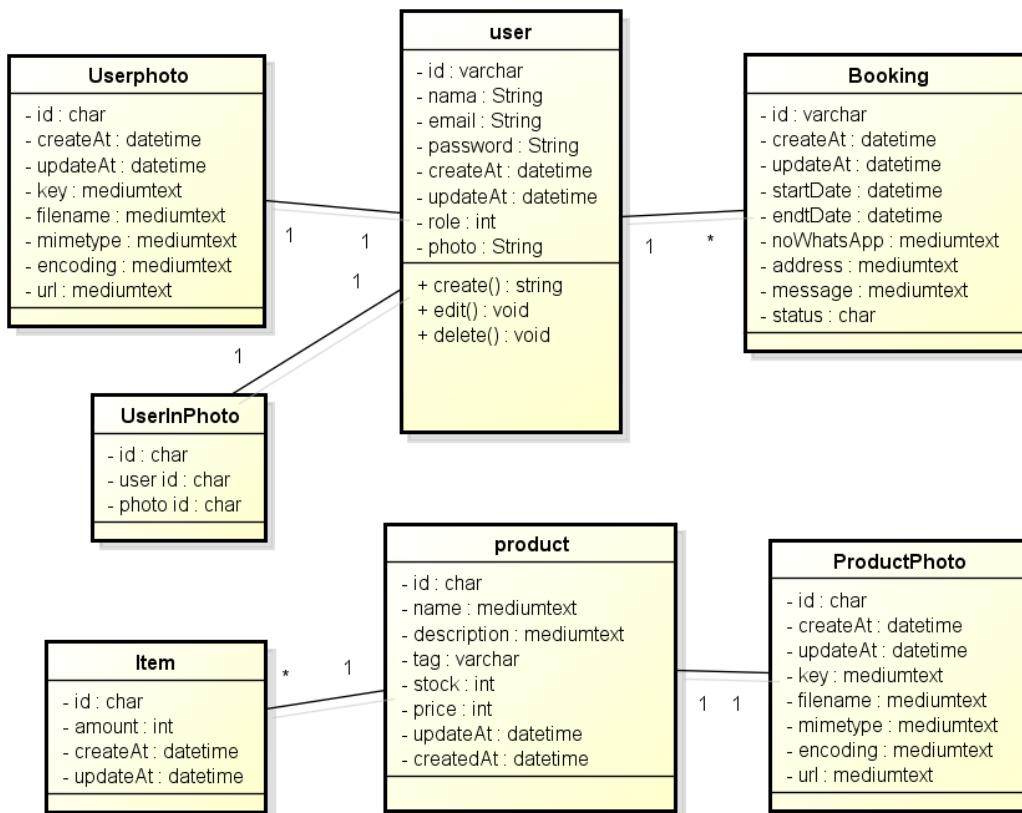
5. Activity diagram *setting akun*



Gambar 4.12 *activity diagram setting akun*

4.2.1.3 Class diagram

Class diagram menunjukkan hubungan yang terjadi pada database aplikasi. Hubungan antara tabel-tabel yang saling berkolaborasi dalam mengelola data di Aplikasi Wedding Organizer Linda Salon. Berikut skemanya:



Gambar 4.13 skema class diagram

4.3 Perancangan Basis Data

Basis data dibutuhkan untuk menyiapkan semua data-data pokok yang dibutuhkan untuk dijadikan informasi yang ditampilkan pada aplikasi yang dibuat.

4.3.1 Struktur Tabel

Struktur tabel dilakukan untuk mengetahui struktur basis data yang akan dibuat pada aplikasi sebagai media penyimpanan data agar dapat diakses dengan mudah dan cepat. Adapun struktur rancangan basis data yang akan dibuat pada aplikasi ini adalah sebagai berikut :

Tabel 4.18 Struktur Tabel *Booking*

Field	Type	Null	Key	Default	Extra
id	char(25)	NO	PRI	NULL	
createdAt	datetime(3)	NO		NULL	
updateAt	datetime(3)	NO		NULL	
startDate	datetime(3)	NO		NULL	
endDate	Datetime(3)	NO		NULL	
noWhatsApp	mediumtext	YES		NULL	
address	mediumtext	YES		NULL	
message	mediumtext	YES		NULL	
status	Varchar(191)	NO		NULL	

Tabel 4.19 Struktur Tabel *Item*

Field	Type	Null	Key	Default	Extra
id	char(25)	NO	PRI	NULL	
amount	int(11)	NO		NULL	
updateAt	datetime(3)	NO		NULL	
createdAt	datetime(3)	NO		NULL	

Tabel 4.20 Struktur Tabel PhotoGallery

Field	Type	Null	Key	Default	Extra
id	char(25)	NO	PRI	NULL	
key	mediumtext	NO		NULL	
filename	mediumtext	NO		NULL	
mimetype	mediumtext	NO		NULL	
encoding	mediumtext	NO		NULL	
url	mediumtext	NO		NULL	
updateAt	datetime(3)	NO		NULL	
createdAt	datetime(3)	NO		NULL	

Tabel 4.21 Struktur Tabel *Product*

Field	Type	Null	Key	Default	Extra
Di	char(25)	NO	PRI	NULL	
name	mediumtext	NO		NULL	
description	mediumtext	NO		NULL	
Tag	varchar(191)	NO		NULL	
stock	int(11)	NO		NULL	
price	int(11)	NO		NULL	
updateAt	datetime(3)	NO		NULL	
createdAt	datetime(3)	NO		NULL	

Tabel 4.22 Struktur Tabel *ProductPhoto*

Field	Type	Null	Key	Default	Extra
id	char(25)	NO	PRI	NULL	
createdAt	datetime(3)	NO		NULL	
updateAt	datetime(3)	NO		NULL	
Key	mediumtext	NO		NULL	
filename	mediumtext	YES		NULL	
mimetype	mediumtext	NO		NULL	
encoding	mediumtext	NO		NULL	
url	mediumtext	NO	UNI	NULL	

Tabel 4.23 Struktur Tabel *User*

Field	Type	Null	Key	Default	Extra
id	char(25)	NO	PRI	NULL	
name	mediumtext	NO		NULL	
email	mediumtext	NO	UNI	NULL	
password	mediumtext	NO		NULL	
createdAt	datetime(3)	NO		NULL	

updateAt	datetime(3)	NO		NULL	
role	varchar(191)	NO		NULL	

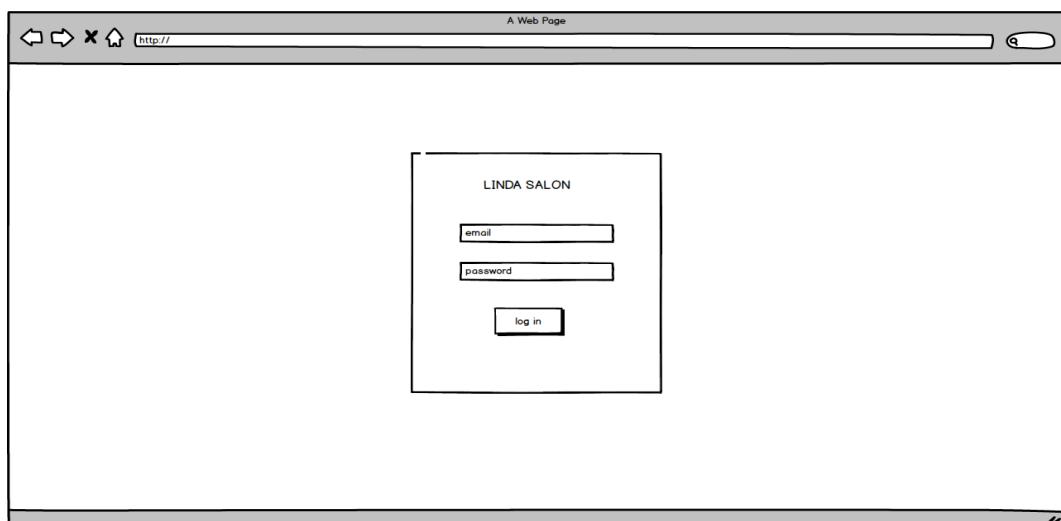
Tabel 4.24 Struktur Tabel *UserPhoto*

Field	Type	Null	Key	Default	Extra
id	char(25)	NO	PRI	NULL	
createdAt	datetime(3)	NO		NULL	
updateAt	datetime(3)	NO		NULL	
key	mediumtext	NO		NULL	
filename	mediumtext	YES		NULL	
mimetype	mediumtext	NO		NULL	
encoding	mediumtext	NO		NULL	
url	mediumtext	NO	UNI	NULL	

4.3.2 Desain

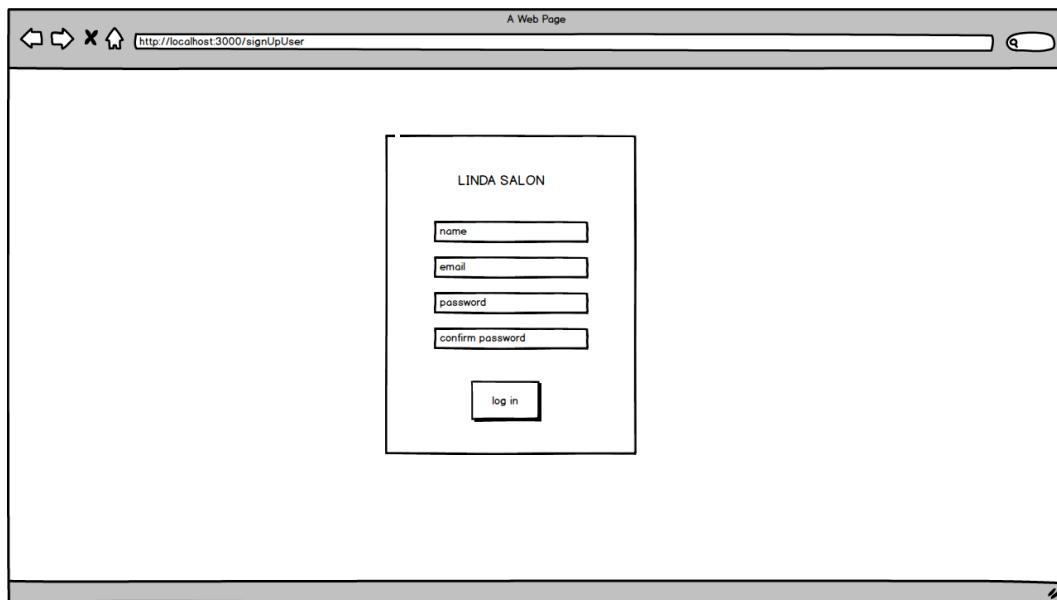
Untuk perencanaan desain antar muka dari aplikasi Wedding Organizer dibuat dalam bentuk mockup dengan menggunakan aplikasi balsamiq. Berikut adalah desain antar muka untuk aplikasi Wedding Organizer ini.

a. Rancangan Login



Gambar 4.14 Rancangan Halaman Login

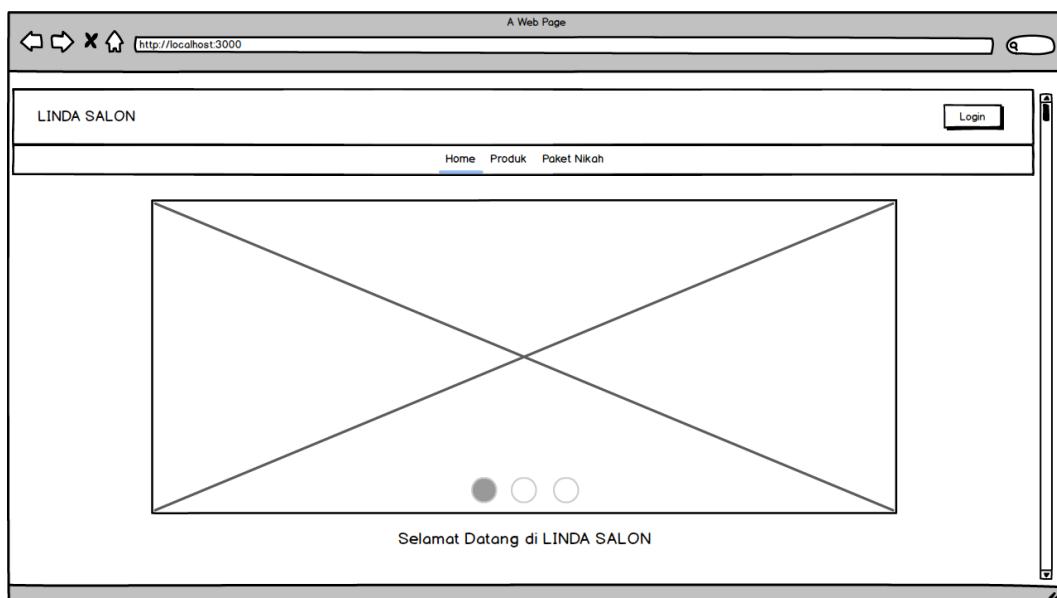
Keterangan di atas merupakan rancangan tampilan login untuk Admin maupun User yang ingin masuk ke Aplikasi.



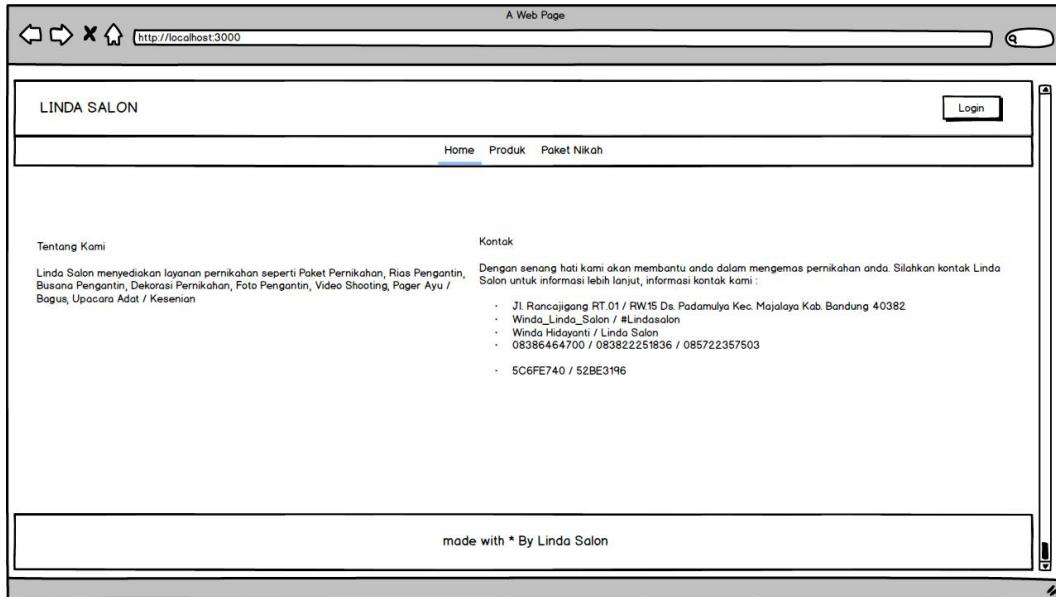
Gambar 4.15 Rancangan login untuk daftar *user*

Rancangan halaman ini untuk user yang belum punya akun dan ingin mendaftar sebagai pelanggan/member.

b. Rancangan menu utama *user*

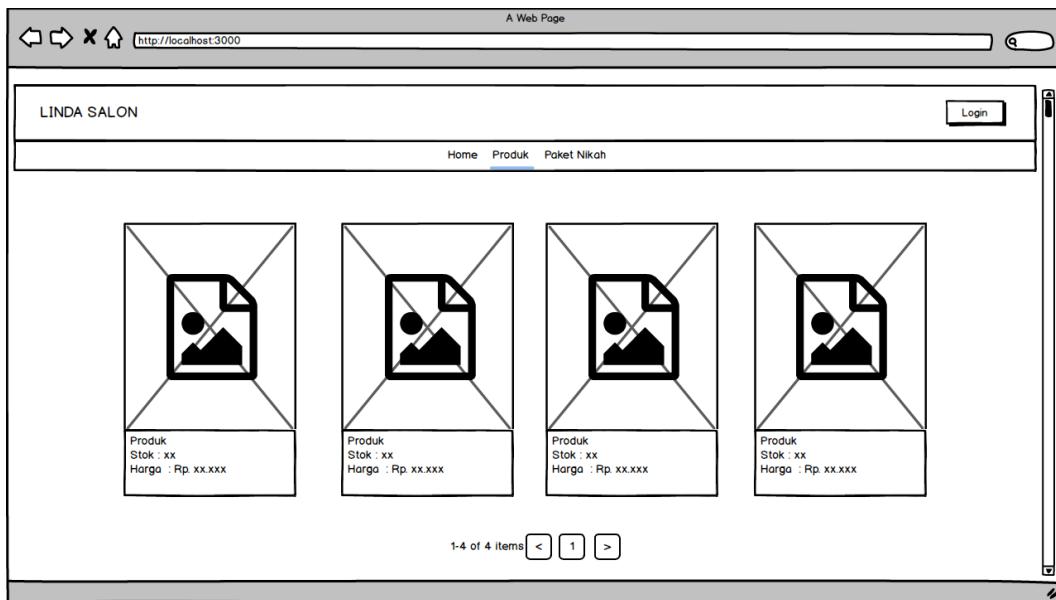


Gambar 4.16 Rancangan Halaman *dashboard user*



Gambar 4.17 Rancangan Halaman *dashboard user* (lanjutan)

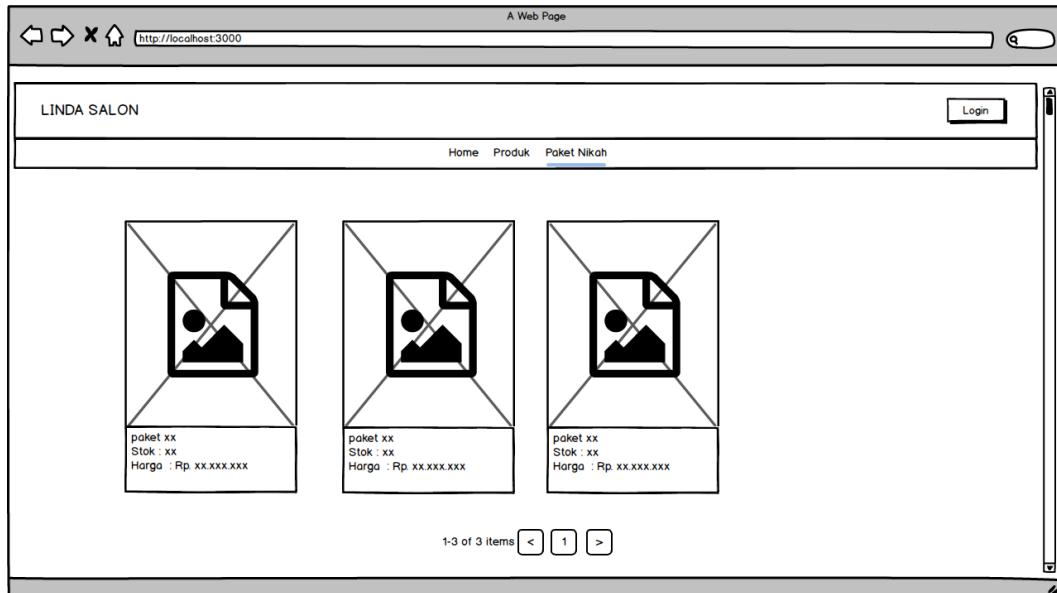
c. Rancangan menu produk



Gambar 4.18 Rancangan Halaman produk

Halaman produk menampilkan produk-produk yang bisa kita pesan setelah login, terdiri dari deskripsi produk, stok dan harga.

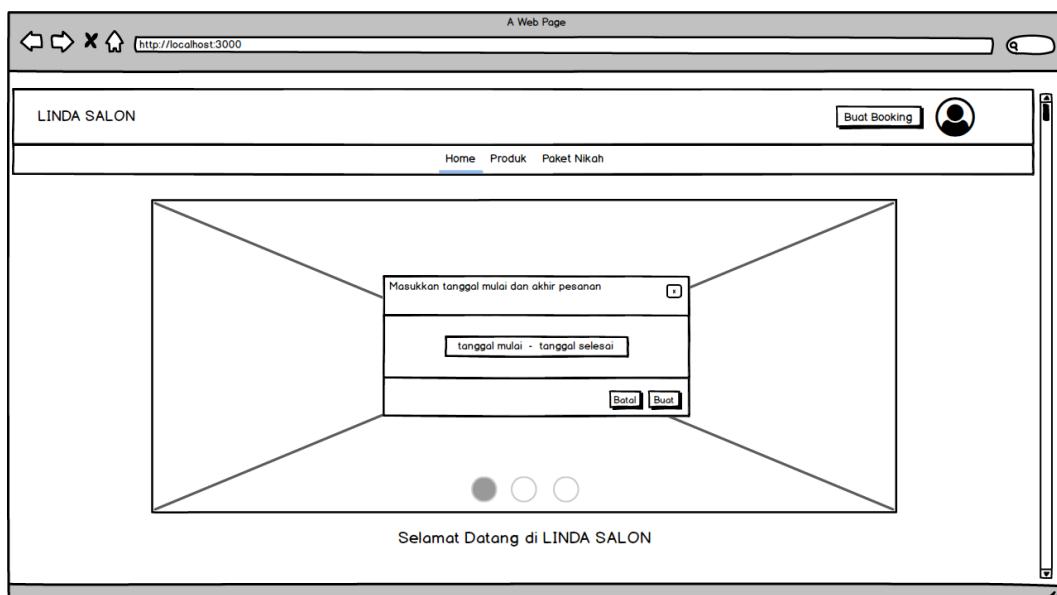
d. Rancangan menu paket nikah



Gambar 4.19 Rancangan Halaman paket nikah

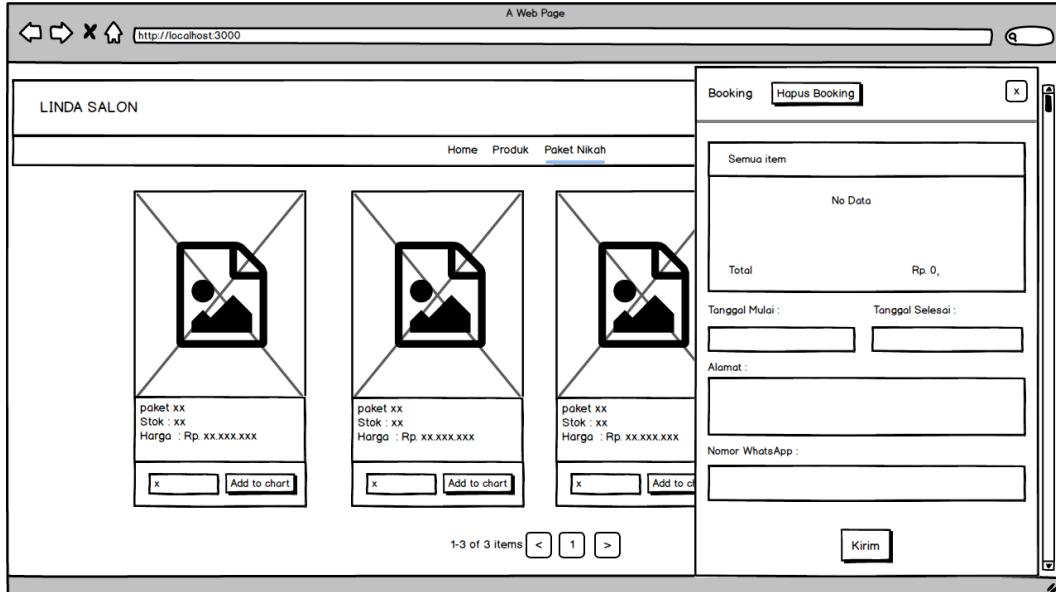
Halaman paket nikah menampilkan paket-paket yang bisa kita pesan setelah login, bedanya dengan produk yakni paket nikah lebih kumplit dan memudahkan dalam menentukan budget pernikahan. Dalam menu paket nikah terdiri dari deskripsi produk, stok dan harga.

e. Rancangan menu buat *booking*



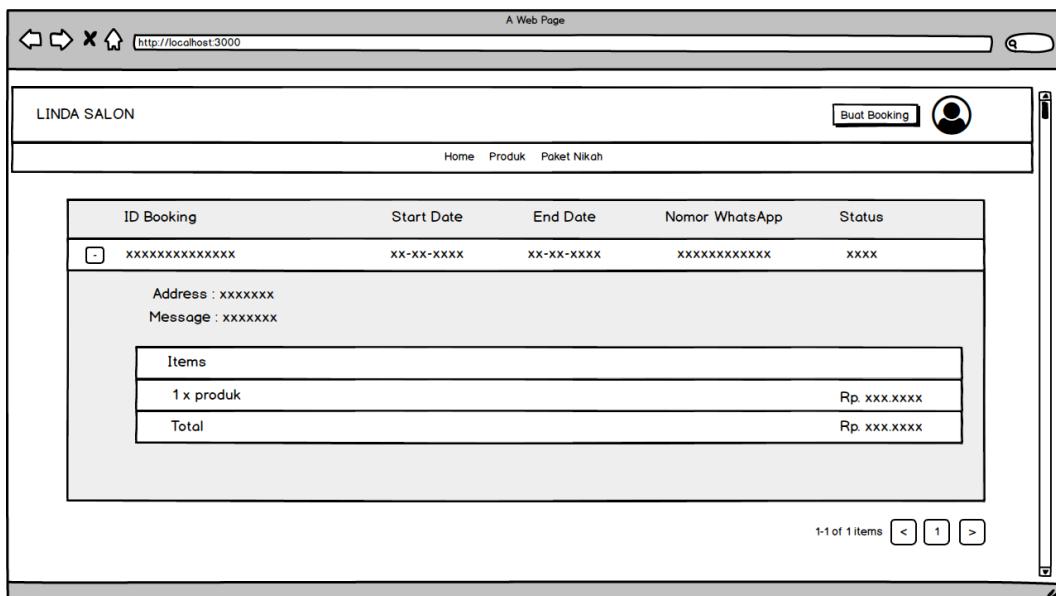
Gambar 4.20 Rancangan Halaman buat *booking*

Halaman buat booking menampilkan form untuk memasukkan tanggal mulai dan tanggal akhir membooking, setelah itu klik tombol buat maka menu produk dan paket nikah bisa di pesan.



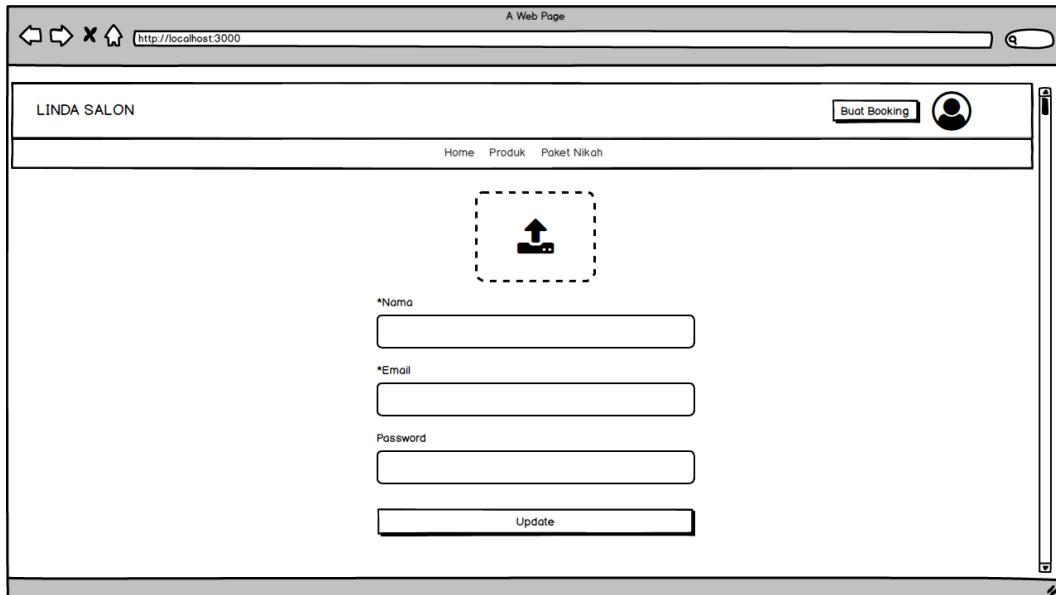
Gambar 4.21 Rancangan Halaman konfirmasi pesanan

Halaman konfirmasi pesanan menampilkan item yang di booking juga form untuk memasukkan alamat dan no whatsapp, setelah itu klik tombol kirim maka pesanan pelanggan akan di proses.



Gambar 4.22 Rancangan Halaman rincian pemesanan

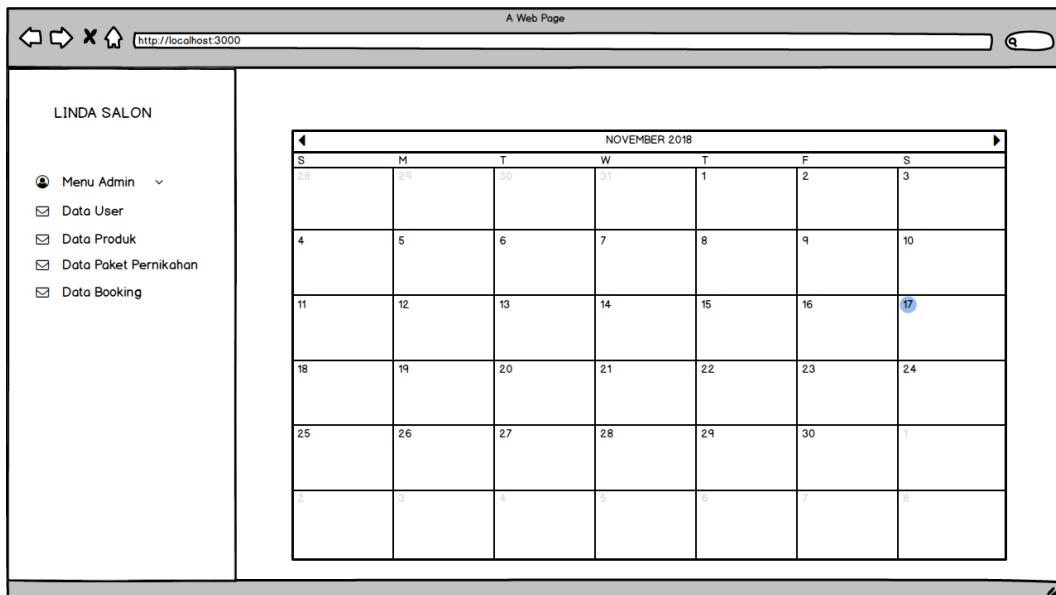
f. Rancangan menu *setting* akun



Gambar 4.23 Rancangan Halaman *setting* akun

Halaman *setting* akun untuk mengubah foto profil, nama, *email* dan password pelanggan.

g. Rancangan menu admin



Gambar 4.24 Rancangan Halaman dashboard admin

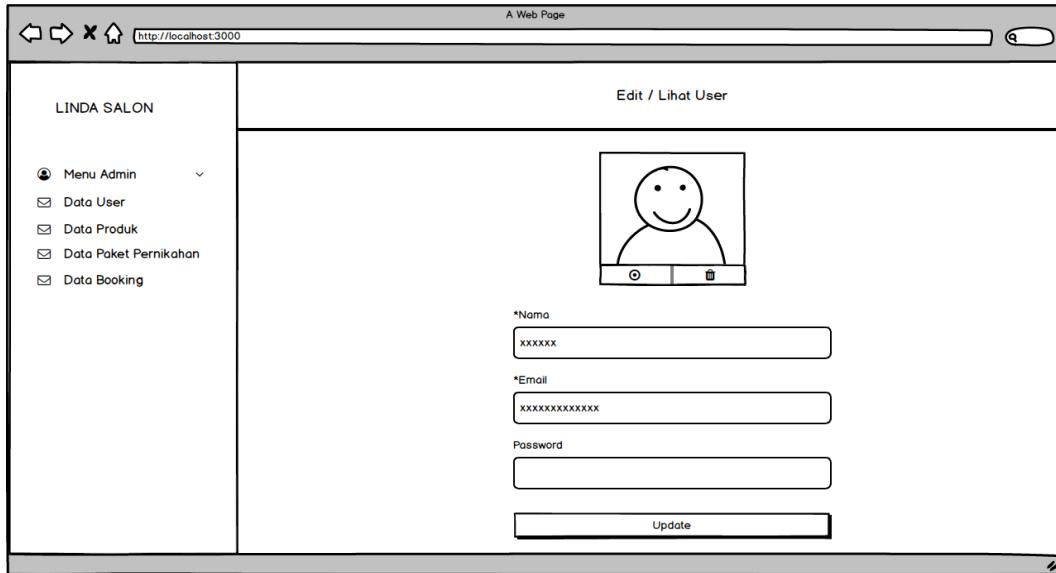
Dashboard admin merupakan halaman utama ketika admin berhasil login, dan menampilkan kalender yang sudah berisi catatan pelanggan yang ingin booking. Menu yang terdapat di halaman admin diantaranya:

1. Menu data *user*
2. Menu data produk
3. Menu data paket nikah
4. Menu data *booking*
5. Menu *setting* akun admin

h. Rancangan data *user*

A screenshot of a web-based application interface titled "Data User". The interface has a sidebar on the left containing the "LINDA SALON" logo and a "Menu Admin" section with options: Data User (selected), Data Produk, Data Paket Pernikahan, and Data Booking. The main content area is titled "Data User" and displays a table of user data. The table has columns for "Id", "User Booking", "Nomor WhatsApp", and "Status". There are two rows of data, each with a checkbox next to the "Id" column. At the top right of the main area are buttons for "Pilih Aksi" and "Jalankan". At the bottom right are navigation links: "1-2 of 2 items", "< 1 >", and a double arrow icon.

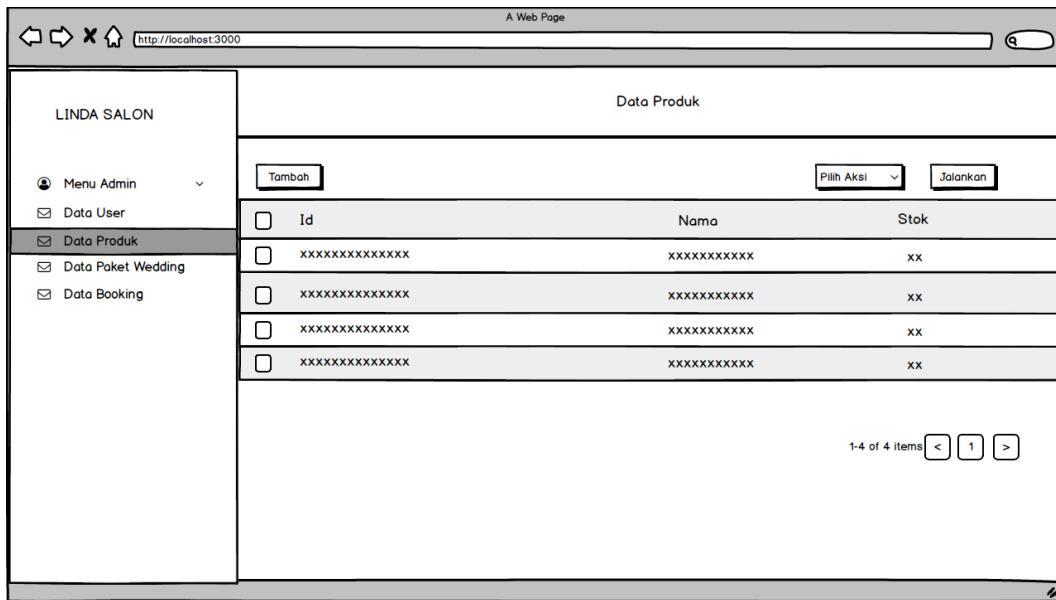
Gambar 4.25 Rancangan Halaman data *user*



Gambar 4.26 Rancangan Halaman edit/lihat user

Halaman data user menampilkan data user atau pelanggan, terdiri dari id, nama user, nomor whatsapp, status, dan pilih aksi. Di dalam menu edit/lihat user terdiri dari form nama,email dan password serta tombol update.

i. Rancangan data produk



Gambar 4.27 Rancangan Halaman data produk

Halaman data produk terdiri dari id, nama dan stok, selain itu ada juga tombol tambah untuk menambahkan produk dan tombol pilih aksi untuk edit/lihat produk.

A wireframe diagram of a web page titled "Tambah Produk". The page has a header "A Web Page" and a URL "http://localhost:3000". On the left is a sidebar with "LINDA SALON" and a menu: "Menu Admin", "Data User", "Data Produk" (selected), "Data Paket Wedding", and "Data Booking". The main content area is titled "Tambah Produk". It contains a dashed box for file upload, fields for "Nama", "Harga", "Stok", and "Deskripsi", and a "Update" button.

Gambar 4.28 Rancangan Halaman tambah produk

Halaman tambah produk terdiri dari upload gambar, form nama, harga, stok dan deskripsi untuk uraian produk serta tombol *update*.

A wireframe diagram of a web page titled "Edit/Lihat Produk". The layout is similar to the previous one, with a sidebar for "LINDA SALON" and a "Data Produk" menu item selected. The main content area shows a preview image of a smiling person, fields for "Nama" (xxxxxx), "Harga" (xxxxxx), "Stok" (xx), and "Deskripsi" (xxxxxxxx xxxxxxxx xxxx), and an "Update" button.

Gambar 4.29 Rancangan Halaman edit/lihat produk

Halaman edit/lihat produk terdiri dari mengganti gambar, form nama, harga, stok dan deskripsi untuk uraian produk serta tombol *update*.

j. Rancangan data paket nikah

A screenshot of a web-based application interface titled "Data Paket Wedding". The left sidebar, titled "LINDA SALON", contains a menu with "Menu Admin" and several "Data" options, where "Data Paket Wedding" is currently selected. The main content area displays a table with three rows of data. Each row has a checkbox column, an "Id" column containing "xxxxxxxxxxxx", a "Nama" column containing "xxxxxxxxxxxx", and a "Stok" column containing "xx". Above the table are buttons for "Tambah" (Add), "Pilih Aksi" (Select Action), and "Jalankan" (Run). Below the table is a pagination control showing "1-3 of 3 items" with buttons for navigation.

Gambar 4.30 Rancangan Halaman data paket *wedding*

Halaman paket nikah menampilkan data paket nikah, terdiri dari id, nama dan stok, selain itu ada juga tombol tambah untuk menambahkan paket dan tombol pilih aksi untuk edit/lihat paket.

A screenshot of a web-based application interface titled "Tambah Paket Wedding". The left sidebar, titled "LINDA SALON", contains a menu with "Menu Admin" and several "Data" options, where "Data Paket Wedding" is currently selected. The main content area features a large dashed box with an upward arrow icon, likely for file upload. Below this are four input fields labeled "*Nama", "*Harga", and "*Stok", followed by a larger "Deskripsi" field. At the bottom is a single "Update" button.

Gambar 4.31 Rancangan Halaman tambah paket *wedding*

Halaman tambah produk terdiri dari *upload* gambar, form nama, harga, stok dan deskripsi untuk uraian paket serta tombol *update*.

k. Rancangan data *Booking*

A screenshot of a web application interface titled "Data Booking". The left sidebar shows a navigation menu with "LINDA SALON" at the top, followed by "Menu Admin" and several "Data" options: "Data User", "Data Produk", "Data Paket Pernikahan", and "Data Booking". The "Data Booking" option is highlighted with a gray background. The main content area is titled "Data Booking" and contains a table with two rows of booking data. The columns are labeled "ID Booking", "User Booking", "Nomor WhatsApp", and "Status". Both rows show placeholder values like "xxxxxxxxxxxx" for ID, "xxxxxxxxxx" for User Booking, "xxxxxxxxxx" for Nomor WhatsApp, and "xxxx" for Status. At the bottom right of the table, it says "1-2 of 2 items" with navigation buttons (< 1 >).

Gambar 4.32 Rancangan Halaman data *booking*

A screenshot of a web application interface titled "Edit/Lihat Booking". The left sidebar shows a navigation menu with "LINDA SALON" at the top, followed by "Menu Admin" and several "Data" options: "Data User", "Data Produk", "Data Paket Wedding", and "Data Booking". The "Data Booking" option is highlighted with a gray background. The main content area is titled "Edit/Lihat Booking" and contains a form with five input fields. The first field is a dropdown labeled "*Status" with "xxxxxx" selected. The second field is a text input labeled "Tanggal Mulai" with "xxxxxx" in it. The third field is a text input labeled "Tanggal Selesai" with "xxxxxx" in it. The fourth field is a text input labeled "Alamat" with "xxxxxxxx xxxxxxxx xxxx" in it. The fifth field is a text input labeled "User yang booking" with "xxxxxx" in it.

Gambar 4.33 Rancangan Halaman *edit/lihat booking*

LINDA SALON

Menu Admin

- Data User
- Data Produk
- Data Paket Wedding
- Data Booking**

Edit/Lihat Booking

Hak Akses Customer
xxxxxx

No Whatsapp
xxxxxxxxxx

Pesan
xxxxxx

Items	
1x produk	Rp. xxxxxxxx,
Total	Rp. xxxxxxxx,

Update

Gambar 4.34 Rancangan Halaman *edit/lihat booking* (lanjutan)

1. Rancangan *setting admin*

LINDA SALON

Menu Admin

- Data User
- Data Produk
- Data Paket Wedding
- Data Booking

Setting Admin

*Nama
xxxxxx

*Email
xxxxxx

*Password

Update

Gambar 4.35 Rancangan Halaman *setting admin*

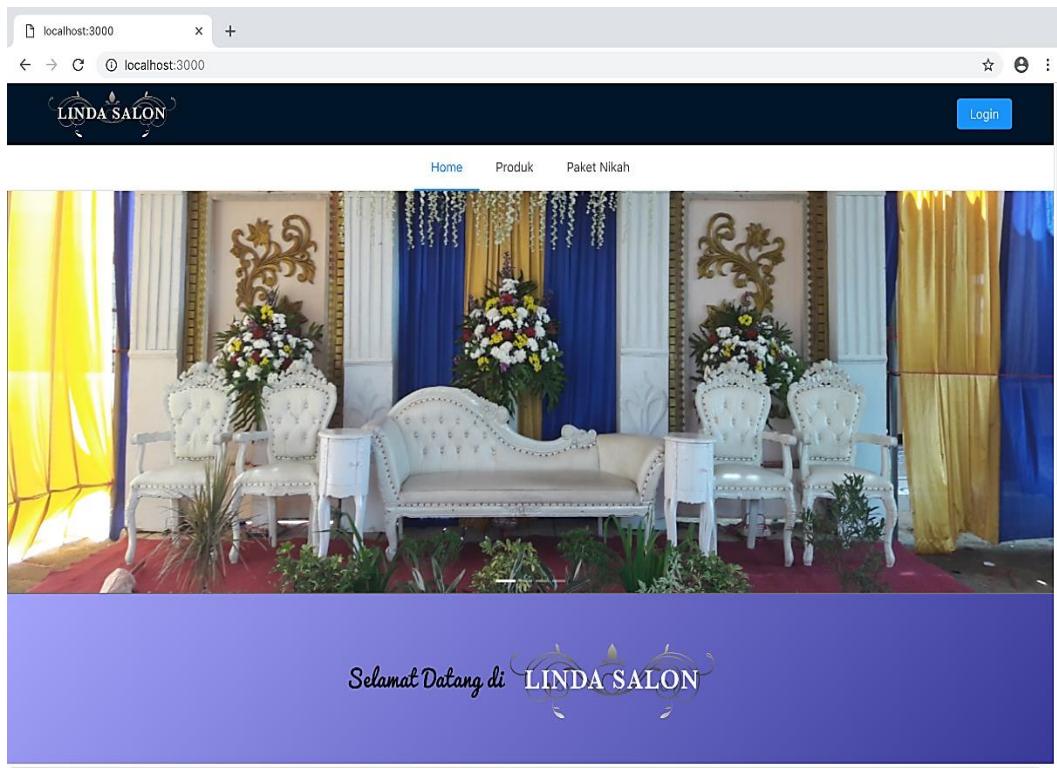
4.3.3 Hasil

4.3.3.1 Menjalankan Sistem

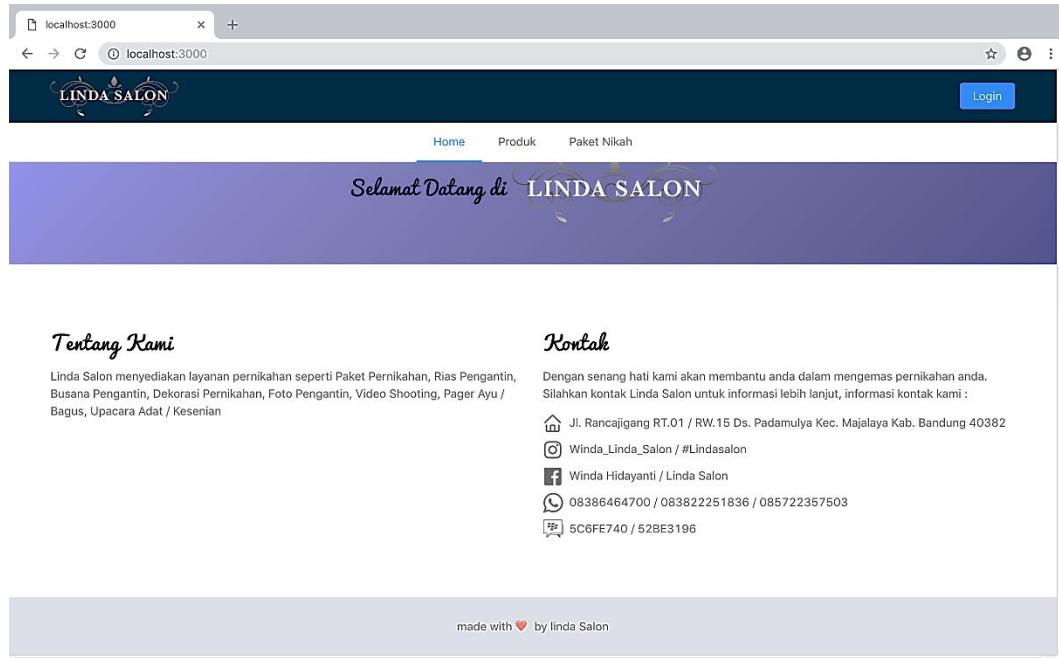
Aplikasi yang di buat penulis adalah aplikasi e-commerce tetapi hanya sampai pemesanan saja, tidak sampai ke transaksi pembayaran dan lain-lain. Berikut adalah tampilan aplikasi yang sudah jadi untuk user dan admin:

a. Dashboard user

Halaman dashboard menampilkan modul-modul yang ada pada aplikasi Linda salon dan semua hanya bisa di lihat saja. Jika ingin memesan produk atau paket nikah user harus daftar dan login terlebih dahulu.



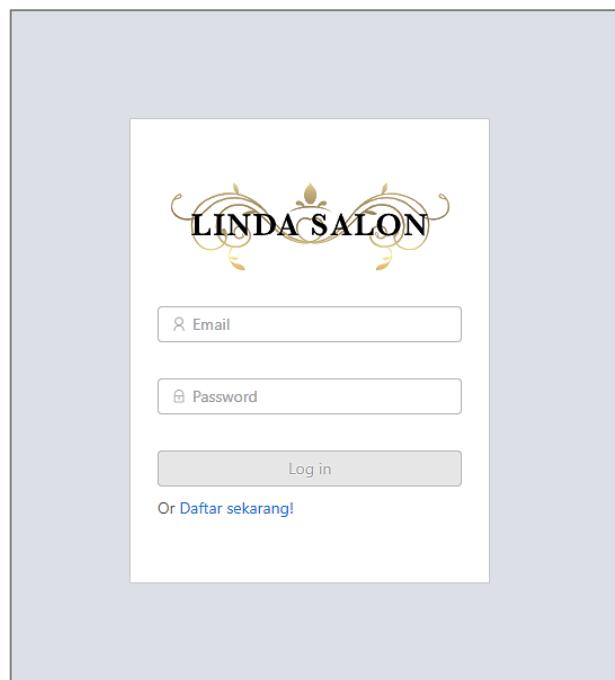
Gambar 4.36 Halaman *Dashboard user*



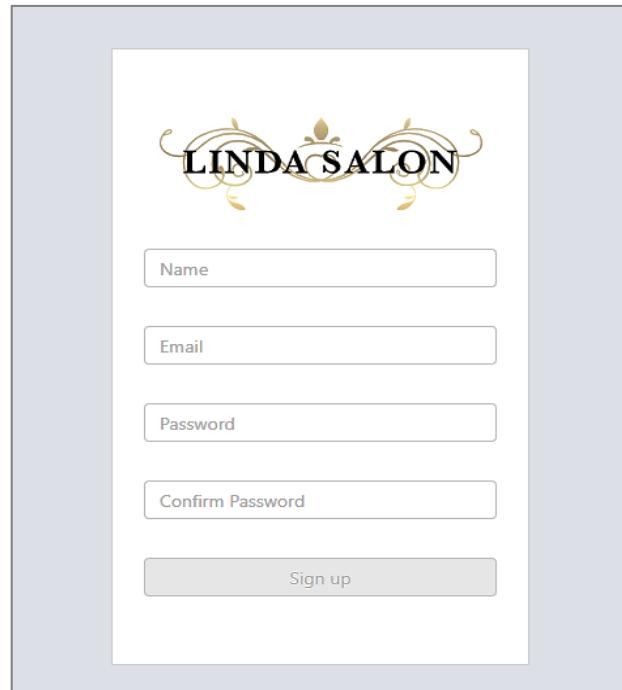
Gambar 4.37 Halaman *Dashboard user* (lanjutan)

b. Tampilan Halaman Login

Halaman login merupakan hak akses untuk user yang ingin memesan produk dan hak akses masuk admin untuk masuk ke halaman admin.



Gambar 4.38 Halaman *login*



Gambar 4.39 Halaman Daftar user

c. Halaman produk

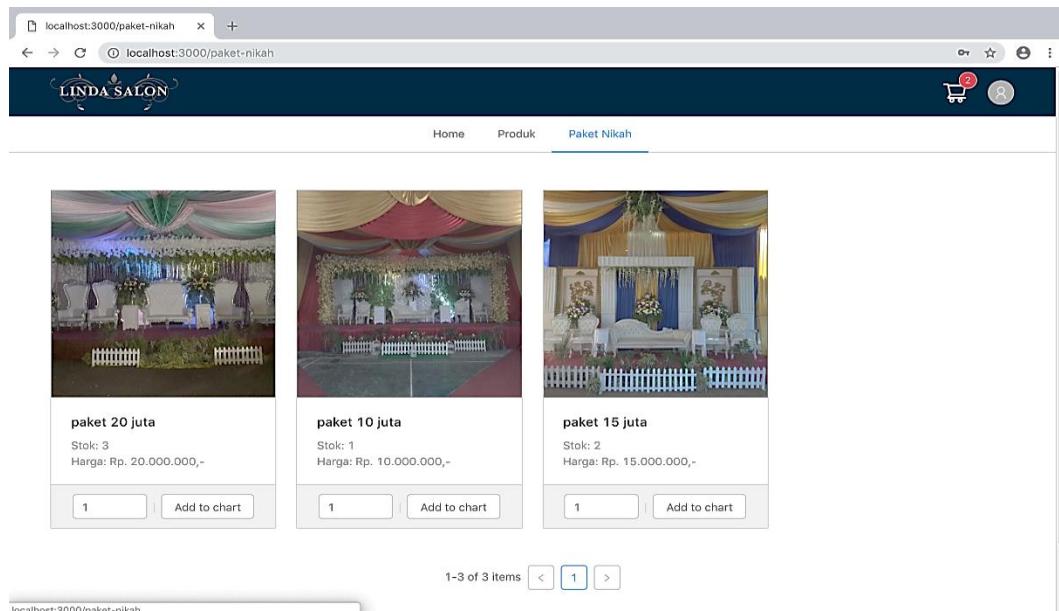
Halaman produk berisi produk yang bisa di booking setelah login dan berisi deskripsi gambar, stok dan harga

Produk	Stok	Harga	Quantity	Action
baju pengantin gold	1	Rp. 100.000,-	1	Add to chart
baju pengantin biru tua	1	Rp. 100.000,-	1	Add to chart
baju pengantin biru	1	Rp. 100.000,-	1	Add to chart
baju pengantin pink	1	Rp. 100.000,-	1	Add to chart

Gambar 4.40 Halaman produk

d. Halaman paket nikah

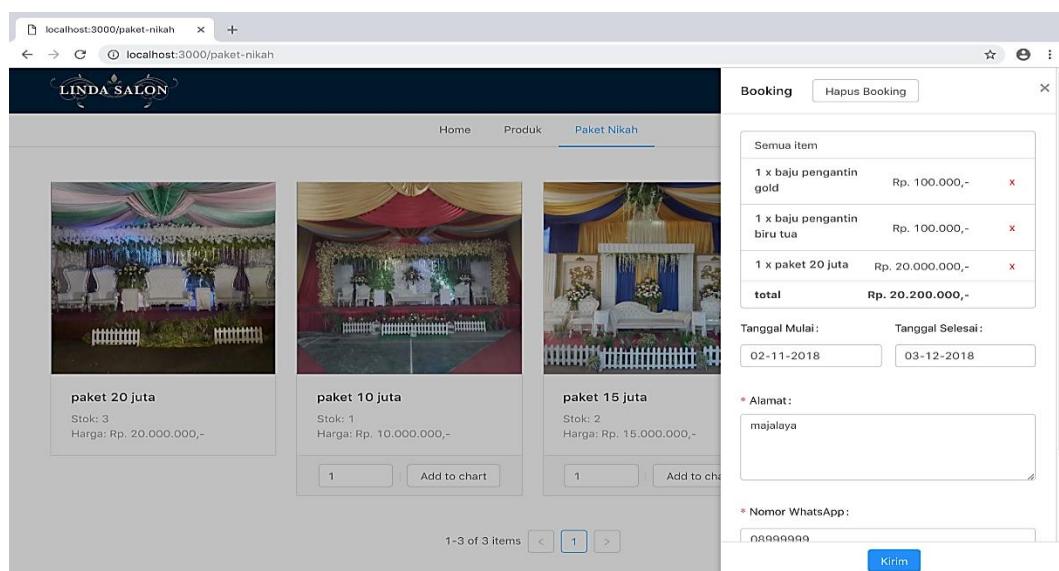
Halaman paket nikah berisi paket-paket yang di sediakan sesuai dengan harga yang telah di tentukan



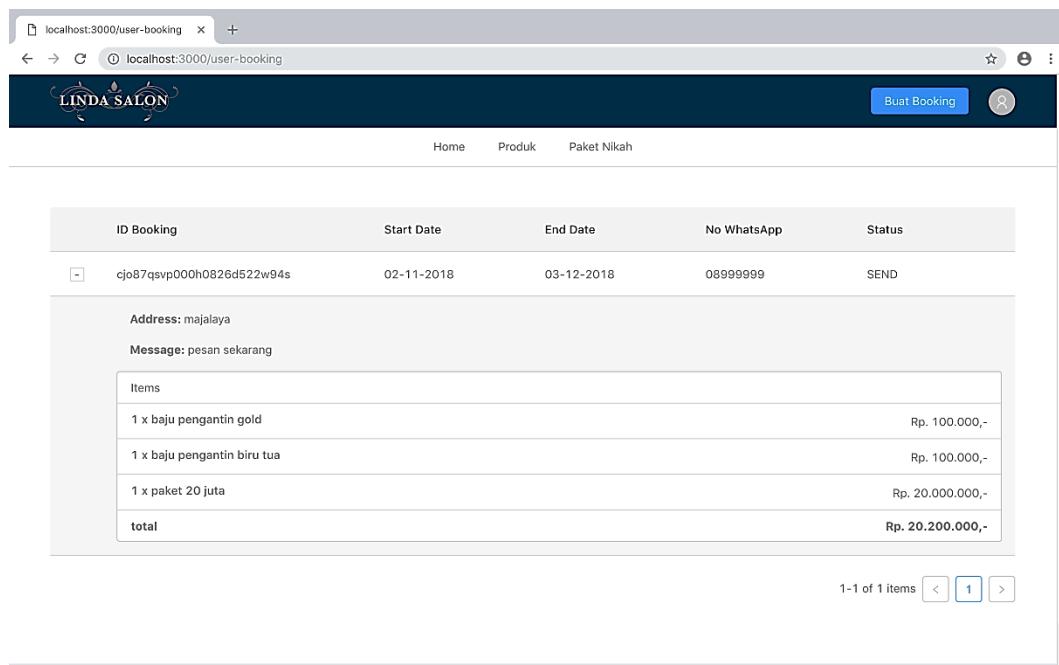
Gambar 4.41 Halaman paket nikah

e. Halaman pemesanan

Pemesanan berisi deskripsi *item* yang sudah di *booking* dan akan di konfirmasi.



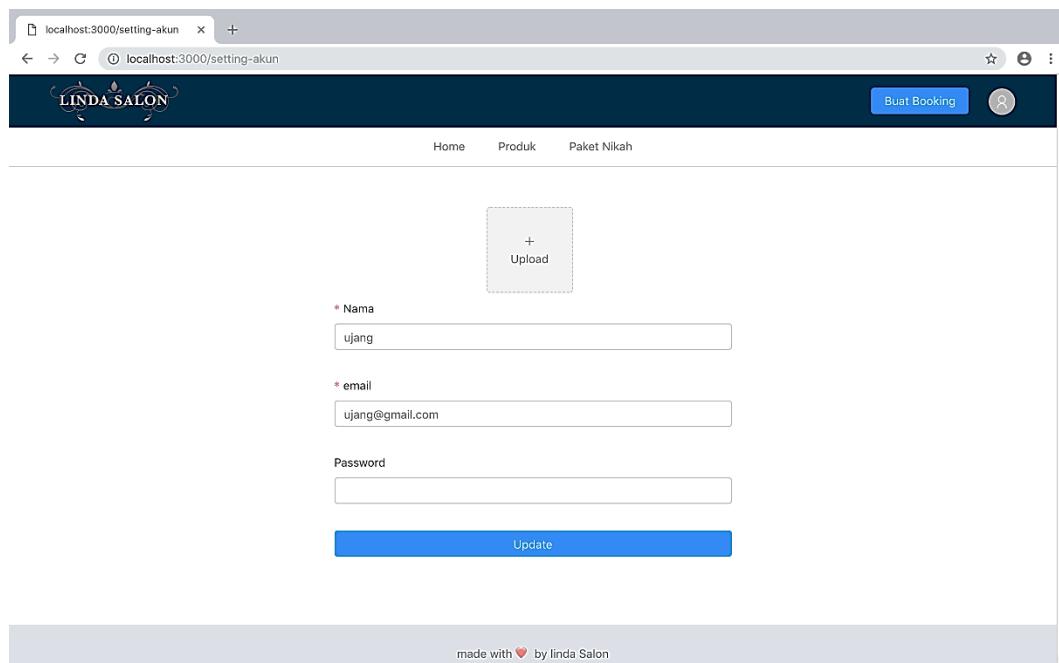
Gambar 4.42 Halaman pemesanan



Gambar 4.43 Halaman pemesanan

f. Halaman *setting user*

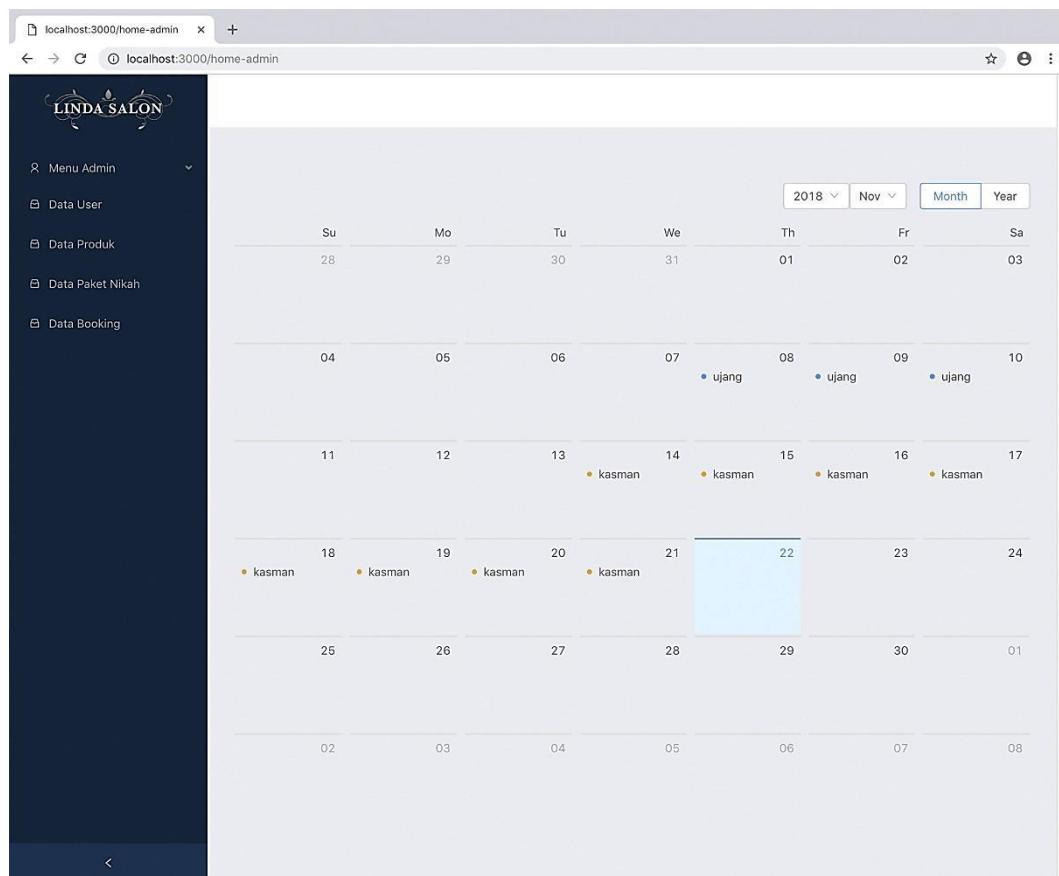
Halaman *setting user* berisi pengaturan foto profil, nama dan *password*



Gambar 4.44 Halaman *setting user*

g. Halaman *Dashboard* admin

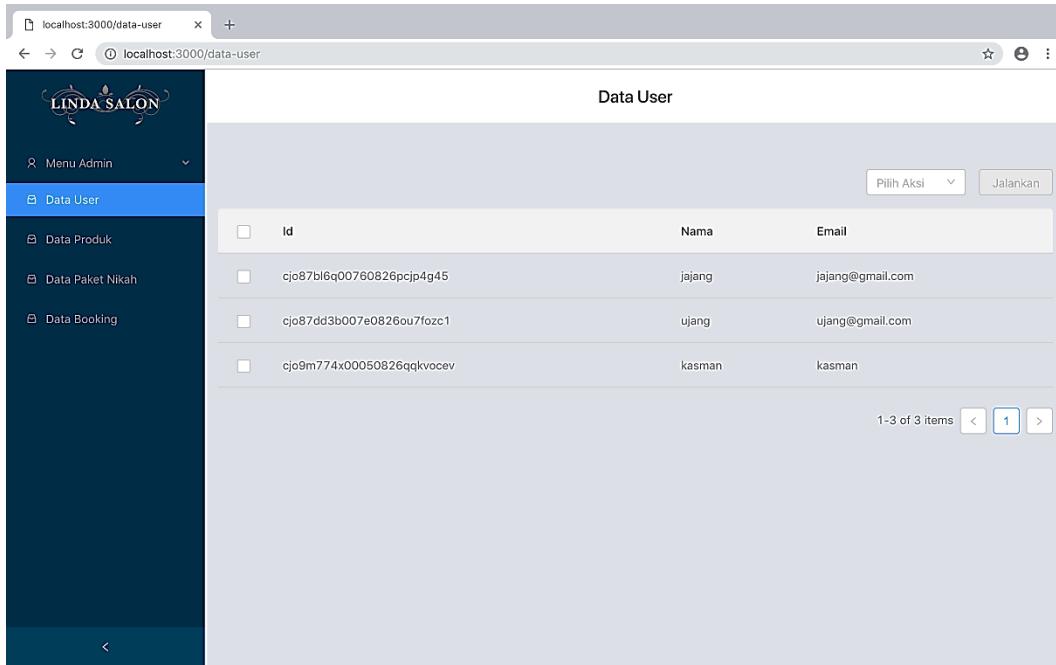
Halaman *dashboard* admin adalah halaman utama *login* yang terdiri dari menu pelanggan, produk, paket nikah, pemesanan dan *setting*.



Gambar 4.45 Halaman *dasboard* admin

h. Halaman menu *user*

Halaman ini berisi daftar *user* yang sudah terdaftar dan memiliki akun yang bisa login dan memesan paket dan produk.



The screenshot shows a web application interface for managing user data. On the left, there is a dark sidebar with a logo for 'LINDA SALON' and a navigation menu containing 'Menu Admin', 'Data User' (which is highlighted in blue), 'Data Produk', 'Data Paket Nikah', and 'Data Booking'. The main content area has a title 'Data User' at the top right. Below it is a table with three columns: 'Id', 'Nama', and 'Email'. The table contains three rows of data:

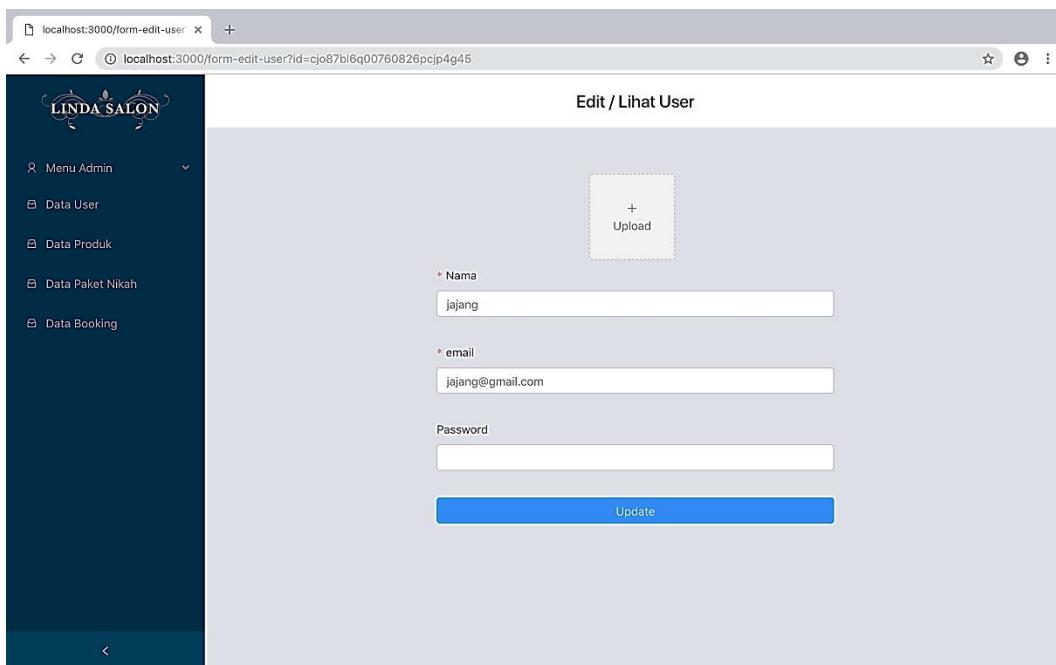
Id	Nama	Email
cjo87bl6q00760826pcjp4g45	jajang	jajang@gmail.com
cjo87dd3b007e0826ou7fozc1	ujang	ujang@gmail.com
cjo9m774x00050826qqkvocev	kasman	kasman

At the bottom right of the table, there is a pagination indicator showing '1-3 of 3 items' with buttons for navigating between pages.

Gambar 4.46 Halaman menu *user*

i. Halaman edit *user*

Halaman edit *user* berisi edit foto profil, nama, email dan *password*

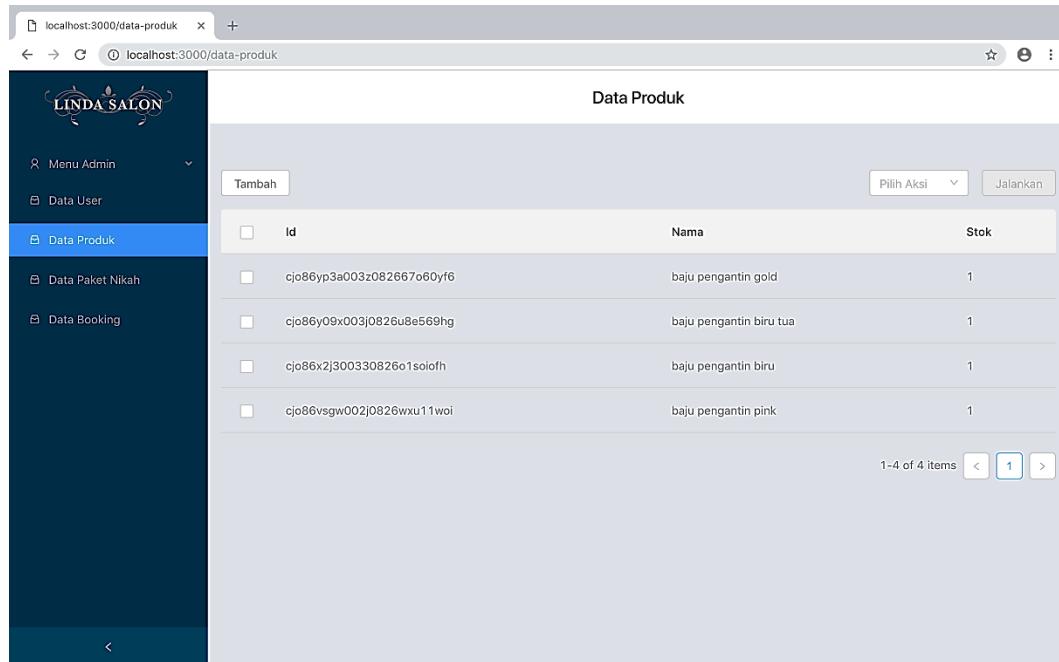


The screenshot shows the 'Edit / Lihat User' page. It features a sidebar with the same 'LINDA SALON' logo and menu as the previous page. The main form area has a title 'Edit / Lihat User'. It includes a file upload field labeled '+ Upload' with a dashed box placeholder. Below it are input fields for 'Nama' (containing 'jajang'), 'email' (containing 'jajang@gmail.com'), and 'Password'. A large blue 'Update' button is located at the bottom of the form.

Gambar 4.47 Halaman menu *user*

j. Halaman menu data produk

Halaman data produk berisi nama produk, harga dan stok



The screenshot shows a web application interface for managing products. On the left, there is a dark sidebar with a logo 'LINDA SALON' at the top and a navigation menu containing 'Menu Admin', 'Data User', 'Data Produk' (which is highlighted in blue), 'Data Paket Nikah', and 'Data Booking'. The main content area has a title 'Data Produk' and a sub-section 'Tambah'. It displays a table with four rows of product data:

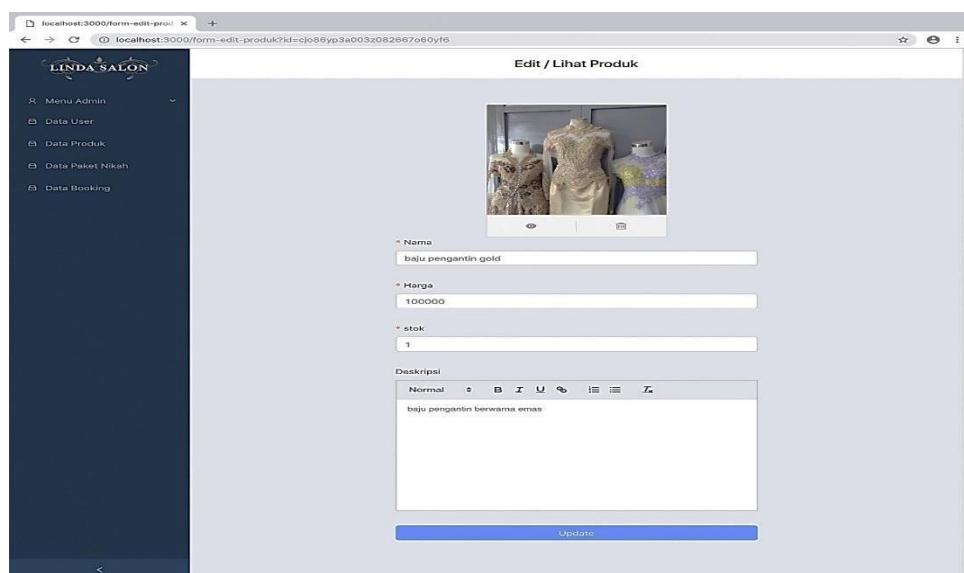
<input type="checkbox"/>	Id	Nama	Stok
<input type="checkbox"/>	cjo86yp3a003z082667o60yf6	baju pengantin gold	1
<input type="checkbox"/>	cjo86y09x003j0826u8e569hg	baju pengantin biru tua	1
<input type="checkbox"/>	cjo86x2j300330826o1soiofh	baju pengantin biru	1
<input type="checkbox"/>	cjo86vsgw002j0826wxu11woi	baju pengantin pink	1

At the bottom right of the table, it says '1-4 of 4 items' with navigation arrows. There are also buttons for 'Pilih Aksi' and 'Jalankan'.

Gambar 4.48 Halaman menu data produk

k. Halaman edit produk

Halaman edit produk berisi edit foto produk, nama, stok dan deskripsi produk



The screenshot shows a 'Edit / Lihat Produk' page. At the top, there is a thumbnail image of three traditional wedding dresses. Below the image, there are input fields for 'Nama' (name) containing 'baju pengantin gold', 'Harga' (price) containing '100000', and 'stok' (stock) containing '1'. Below these fields is a rich text editor titled 'Deskripsi' (Description) with the text 'baju pengantin berwarna emas'. At the bottom of the form is a blue 'Update' button.

Gambar 4.49 Halaman menu data produk

i. Halaman menu *booking*

Halaman menu *booking* menampilkan *user* yang membooking produk/paket dan di proses oleh admin

The screenshot shows a web application interface for managing bookings. On the left, there is a dark sidebar with a logo for 'LINDA SALON' and a navigation menu containing 'Menu Admin', 'Data User', 'Data Produk', 'Data Paket Nikah', and 'Data Booking'. The 'Data Booking' option is highlighted with a blue background. The main content area has a title 'Data Booking' at the top. Below it is a table with the following data:

<input type="checkbox"/>	Id	User Booking	No WhatsApp	Status
<input type="checkbox"/>	cjo9qbopu004o0a26jacvsh0	ujang	123	SEND
<input type="checkbox"/>	cjo9miwv1001o0826rr9quh2l	ujang	0982928222	PROCESS
<input type="checkbox"/>	cjo9m7sfv000d08269dz86l2a	kasman	091818181	SEND

At the bottom of the table, there is a message '1-3 of 3 items' followed by navigation buttons: '<', '1' (which is highlighted in blue), and '>'.

Gambar 4.50 Halaman menu data produk

j. Halaman edit *booking*

Halaman edit *booking* berisi data user yang booking dan perubahan status oleh admin

LINDA SALON

Edit / Lihat Booking

Status: SEND

Tanggal Mulai: 18-12-2018

Tanggal Selesai: 20-12-2018

Alamat Tujuan: jl. Rancasari RT.02 / 15 Desa Padamulya Kecamatan Majalaya

User yang booking: ujang

Hak akses user: CUSTOMER

No WhatsApp: 123

Pesan: qq

No WhatsApp: 123

Items	Rp. 10.000.000,-
1 x paket 10 Juta	Rp. 10.000.000,-
total	Rp. 10.000.000,-

Update

Gambar 4.51 Halaman *edit booking*

k. Halaman *setting admin*

Halaman setting admin berisi pengaturan foto profil, nama dan password

LINDA SALON

Setting Admin

Nama: Admin

Email: admin@gmail.com

Password:

Update

Gambar 4.52 Halaman *edit booking*

BAB V

KESIMPULAN DAN SARAN

5.1 Kesimpulan

Berdasarkan analisis, observasi dan perancangan yang telah dilakukan mengenai Aplikasi Wedding Organizer di Linda Salon dapat dicapai kesimpulan sebagai berikut :

- a. Dengan dibangunnya aplikasi ini dapat memudahkan pelanggan dalam mengakses informasi seputar jasa wedding organizer khususnya yang di kelola oleh Linda Salon.
- b. Aplikasi Wedding Organizer Berbasis Web yang sudah dibuat. Diharapkan aplikasi ini dapat membantu pelanggan dalam proses reservasi dan bisa memangkas biaya.

5.2 Saran

Saran yang diharapkan dari hasil analisis, observasi dan perancangan yang telah dicapai maupun untuk proses pengembangannya mengenai Aplikasi Wedding Organizer di Linda Salon. Di masa yang akan datang maka dapat diberikan saran-saran sebagai berikut :

- a. Perlu penyempurnaan pada aplikasi ini baik tampilan maupun fitur-fitur, dari fitur pemesanan
- b. Perlu adanya penambahan fitur, seperti : fitur pencarian, pilihan transaksi pembayaran, dan hak akses vendor sebagai mitra Linda Salon yang ingin menawarkan jasa atau produknya kepada pelanggan.

DAFTAR PUSTAKA

- Najiah, I., & Suharyanto. (2017). Sistem Informasi Wedding Planner Berbasis Web. *Jurnal Ilmu Pengetahuan dan Teknologi Komputer*.
- Nugroho, A. (2009). *Rekayasa Perangkat Lunak Menggunakan UML & Java*. Yogyakarta: Andi Offset.
- Raharjo, B., Heryanto, I., & RK, E. (2012). *Modul Pemrograman Web HTML, PHP & MySQL*. Bandung: Modula.
- Sujatmiko, E. (2012). *Kamus Teknologi Informasi dan Komunikasi*. Surakarta: Aksarra Sinergi Media.
- Wijaya, R. S. (2017). Aplikasi Fairuz Wedding Organizer Berbasis Web Based Application Fairuz Wedding Organizing. *e-Proceeding of Applied Science*.
- Wulandari Ss, R., Pratiwi, H. S., & Muhardi, H. (2017). Rancang Bangun Aplikasi Wedding Organizer Di Kota Pontianak Berbasis Web. *Jurnal Sistem dan Teknologi Informasi*.
- Rosa, A., & Shalahuddin, M. (2011). *Modul Pembelajaran Rekayasa Perangkat Lunak*. Bandung:Modula.
- Whittenn, J., & Bentley, L. (2007). *System Analysis dan Design Method*. McGraw-Hill.

RIWAYAT HIDUP

i

LAMPIRAN

I. Database

Datamodel.graphql

```
type User {
    id: ID! @unique
    name: String!
    email: String! @unique
    password: String!
    createdAt: DateTime!
    updatedAt: DateTime!
    photo: UserPhoto @relation(name: "UserInPhoto", onDelete: CASCADE)
    role: Role! @default(value: "CUSTOMER")
    bookings: [Booking!]! @relation(name: "BookingInUser", onDelete:
CASCADE)
}

type Booking {
    id: ID! @unique
    createdAt: DateTime!
    updatedAt: DateTime!
    startDate: DateTime!
    endDate: DateTime!
    noWhatsApp: String
    address: String
    message: String
    items: [Item!]! @relation(name: "BookingInItem", onDelete: CASCADE)
    status: BookingStatus! @default(value: "CREATED")
    userBooking: User! @relation(name: "BookingInUser", onDelete:
SET_NULL)
}

type Item {
    id: ID! @unique
    amount: Int!
    product: Product @relation(name: "ProductInItem", onDelete: SET_NULL)
    booking: Booking @relation(name: "BookingInItem", onDelete: SET_NULL)
}

type Product {
    id: ID! @unique
    name: String!
    description: String!
    tag: ProductTag!
    stock: Int!
    photo: ProductPhoto @relation(name: "ProductInPhoto", onDelete:
CASCADE)
    item: [Item!]! @relation(name: "ProductInItem", onDelete: CASCADE)
    price: Int!
}

type UserPhoto {
    id: ID! @unique
```

```

createdAt: DateTime!
updatedAt: DateTime!
key: String!
filename: String
mimetype: String!
encoding: String!
url: String! @unique
userPhoto: User @relation(name: "UserInPhoto", onDelete: SET_NULL)
}

type ProductPhoto {
  id: ID! @unique
  createdAt: DateTime!
  updatedAt: DateTime!
  key: String!
  filename: String
  mimetype: String!
  encoding: String!
  url: String! @unique
  Product: Product @relation(name: "ProductInPhoto", onDelete: SET_NULL)
}

type PhotoGallery {
  id: ID! @unique
  key: String!
  filename: String!
  mimetype: String!
  encoding: String!
  url: String!
}

enum Role {
  CUSTOMER
  ADMIN
}

enum ProductTag {
  BARANG,
  PAKETWEDDING
}

enum BookingStatus {
  CREATED,
  SEND,
  PROCCESSED,
  SUCCESS,
}

```

Index.ts

```

import { GraphQLServer } from 'graphql-yoga'
import { Prisma } from './generated/prisma'
import resolvers from './resolvers'
import ultimateSchemaString from './schema'
import options from './options-server'

```

```

import permission from './middleware/permissions'
import persistedQueriesMiddleware from
'./middleware/persistedQueriesMiddleware'

const server = new GraphQLServer({
  typeDefs: ultimateSchemaString,
  resolvers,

  middlewares: [
    permission
  ],

  context: req => ({
    ...req,
    db: new Prisma({
      endpoint: process.env.PRISMA_ENDPOINT, // the endpoint of the
      Prisma API (value set in `.`env`)
      debug: true, // log all GraphQL queries & mutations sent to the
      Prisma API
      secret: process.env.PRISMA_SECRET, // only needed if specified in
      `database/prisma.yml` (value set in `.`env`)
    }),
  }),
})

server.express.get('/graphql', persistedQueriesMiddleware)
// server.express.post('/graphql', persistedQueriesMiddleware)
server.start(options, () => console.log(`Server is running on
http://localhost:4000`))

```

II. Page

App.tsx

```

import App, { Container, AppComponentProps } from 'next/app'
import { ApolloProvider } from 'react-apollo'
import withApollo from '../lib/withApollo'
import { ApolloClient } from 'apollo-boost'

type Apollo = {
  apolloClient: ApolloClient<any>
} & AppComponentProps

class MyApp extends App<Apollo> {
  render() {
    const { Component, pageProps, apolloClient } = this.props;

    return (
      <Container>
        <ApolloProvider client={apolloClient}>
          <Component {...pageProps} />
        </ApolloProvider>
      </Container>
    );
  }
}

```

```

}

export default withApollo(MyApp as typeof MyApp)

```

Document.tsx

```

import Document, { Head, Main, NextScript } from 'next/document'

export default class MyDocument extends Document {
  render() {
    return (
      <html>
        <Head>
          <link rel="stylesheet" href="/_next/static/style.css" />
        </Head>
        <body>
          <Main />
          <NextScript />
        </body>
      </html>
    )
  }
}

```

Data-booking.tsx

```

import { Component } from 'react'
import checkLoggedIn from 'lib/checkLoggedIn'
import { NextContextNewContext } from 'lib/withApollo'
import { MeQuery } from 'types/schema-types'
import { AdminLayoutWithProvider, AdminLayoutConsumer } from
'layout/layoutAdmin'
import redirect from 'lib/redirect'
import TableBooking from 'components/tableBooking'
import '../styles/index.less'

type Props = {
  loggedInUser: MeQuery
}

class DataBooking extends Component<Props> {
  static async getInitialProps (context: NextContextNewContext) {
    const { loggedInUser } = await checkLoggedIn(context.apolloClient)

    if (loggedInUser && loggedInUser.me && loggedInUser.me.role ===
"ADMIN") {
      return {
        loggedInUser
      }
    } else {
      redirect(context, '/')
    }
  }
}

```

```

    render() {
      const { loggedInUser } = this.props
      return (
        <AdminLayoutWithProvider
          idUser={loggedInUser && loggedInUser.me ? loggedInUser.me.id : undefined}
          nameHeader="Data Booking"
        >
        <AdminLayoutConsumer>
          {({ appProvider } => {
            if (appProvider) {
              return (
                <TableBooking
                  productPerPage={12}
                />
              )
            }
          })}
        </AdminLayoutConsumer>
      </AdminLayoutWithProvider>
    )
  }
}

export default DataBooking

```

Data-paket-wedding.tsx

```

import { Component } from 'react'
import TableProducts from 'components/tableProducts'
import checkLoggedIn from 'lib/checkLoggedIn'
import { NextContextNewContext } from 'lib/withApollo'
import { MeQuery } from 'types/schema-types'
import { AdminLayoutWithProvider, AdminLayoutConsumer } from
'layout/layoutAdmin'
import redirect from 'lib/redirect'
import '../styles/index.less'

type Props = {
  loggedInUser: MeQuery
}

class DataPaketWedding extends Component<Props> {
  static async getInitialProps (context: NextContextNewContext) {
    const { loggedInUser } = await checkLoggedIn(context.apolloClient)

    if (loggedInUser && loggedInUser.me && loggedInUser.me.role ===
"ADMIN") {
      return {
        loggedInUser
      }
    } else {
      redirect(context, '/')
    }
  }
}

```

```

    render() {
      const { loggedInUser } = this.props
      return (
        <AdminLayoutWithProvider
          idUser={loggedInUser && loggedInUser.me ? loggedInUser.me.id : undefined}
          nameHeader="Data Paket Nikah"
        >
        <AdminLayoutConsumer>
          {({ appProvider } => {
            if (appProvider) {
              return (
                <TableProducts
                  productPerPage={12}
                  tag="PAKETWEDDING"
                />
              )
            }
          })}
        </AdminLayoutConsumer>
      </AdminLayoutWithProvider>
    )
  }
}

export default DataPaketWedding

```

Data-products.tsx

```

import { Component } from 'react'
import TableProducts from 'components/tableProducts'
import checkLoggedIn from 'lib/checkLoggedIn'
import { NextContextNewContext } from 'lib/withApollo'
import { MeQuery } from 'types/schema-types'
import { AdminLayoutWithProvider, AdminLayoutConsumer } from
'layout/layoutAdmin'
import redirect from 'lib/redirect'
import '../styles/index.less'

type Props = {
  loggedInUser: MeQuery
}

class DataProduk extends Component<Props> {
  static async getInitialProps (context: NextContextNewContext) {
    const { loggedInUser } = await checkLoggedIn(context.apolloClient)

    if (loggedInUser && loggedInUser.me && loggedInUser.me.role ===
"ADMIN") {
      return {
        loggedInUser
      }
    } else {
      redirect(context, '/')
    }
  }
}

```

```

    render() {
      const { loggedInUser } = this.props
      return (
        <AdminLayoutWithProvider
          idUser={loggedInUser && loggedInUser.me ? loggedInUser.me.id : undefined}
          nameHeader="Data Produk"
        >
          <AdminLayoutConsumer>
            {( appProvider ) => {
              if (appProvider) {
                return (
                  <TableProducts
                    productPerPage={12}
                    tag="BARANG"
                  />
                )
              }
            }}
          </AdminLayoutConsumer>
        </AdminLayoutWithProvider>
      )
    }
  }

export default DataProduk

```

Data-user.tsx

```

import { Component } from 'react'
import checkLoggedIn from 'lib/checkLoggedIn'
import { NextContextNewContext } from 'lib/withApollo'
import { MeQuery } from 'types/schema-types'
import { AdminLayoutWithProvider, AdminLayoutConsumer } from
'layout/layoutAdmin'
import TableUsers from 'components/tableUsers'
import redirect from 'lib/redirect'
import '../styles/index.less'

type Props = {
  loggedInUser: MeQuery
}

class DataUser extends Component<Props> {
  static async getInitialProps (context: NextContextNewContext) {
    const { loggedInUser } = await checkLoggedIn(context.apolloClient)

    if (loggedInUser && loggedInUser.me && loggedInUser.me.role ===
"ADMIN") {
      return {
        loggedInUser
      }
    } else {
      redirect(context, '/')
    }
  }
}

```

```

    }

    render() {
      const { loggedInUser } = this.props
      return (
        <AdminLayoutWithProvider
          idUser={loggedInUser && loggedInUser.me ? loggedInUser.me.id : undefined}
          nameHeader="Data User"
        >
        <AdminLayoutConsumer>
          {( appProvider ) => {
            if (appProvider) {
              return (
                <TableUsers
                  userPerPage={12}
                  role="CUSTOMER"
                />
              )
            }
          }}
        </AdminLayoutConsumer>
      </AdminLayoutWithProvider>
    )
  }
}

export default DataUser

```

Form-edit-booking.tsx

```

import { Component } from 'react'
import checkLoggedIn from 'lib/checkLoggedIn'
import { NextContextNewContext } from 'lib/withApollo'
import { MeQuery, BookingQuery, BookingQueryVariables } from
'types/schema-types'
import { Row } from 'antd'
import BOOKING_QUERY from 'queries/booking/booking.graphql'
import FormEditBooking from 'components/formEditBooking'
import { AdminLayoutWithProvider, AdminLayoutConsumer } from
'layout/layoutAdmin'
import { Query } from 'react-apollo'
import redirect from 'lib/redirect'
import '../styles/index.less'

type Props = {
  loggedInUser: MeQuery
  queryId: string
}

class QueryBooking extends Query<BookingQuery, BookingQueryVariables>
{}

class FormEditBookingComp extends Component<Props> {
  static async getInitialProps (context: NextContextNewContext) {
    const { loggedInUser } = await checkLoggedIn(context.apolloClient)
  }
}

```

```
const queryId = context.query.id

if (loggedInUser && loggedInUser.me && loggedInUser.me.role === "ADMIN") {
    return {
        loggedInUser,
        queryId: queryId
    }
} else {
    redirect(context, '/')
}

render() {
    const { loggedInUser, queryId } = this.props
    return (
        <AdminLayoutWithProvider
            idUser={loggedInUser && loggedInUser.me ? loggedInUser.me.id : undefined}
            nameHeader="Edit / Lihat Booking"
        >
        <AdminLayoutConsumer>
            {({ appProvider }) => {
                if (appProvider) {
                    return (
                        <Row
                            type="flex"
                            justify="center"
                            align="middle"
                        >
                            <QueryBooking
                                query={BOOKING_QUERY}
                                variables={{ id: queryId }}
                            >
                                {({ data, error }) => {
                                    if (!error && data && data.booking) {
                                        return (
                                            <FormEditBooking
                                                data={data.booking}
                                            />
                                        )
                                    } else {
                                        return null
                                    }
                                }}
                            </QueryBooking>
                        </Row>
                    )
                }
            }}
        </AdminLayoutConsumer>
        </AdminLayoutWithProvider>
    )
}
}
```

```
export default FormEditBookingComp
```

Form-edit-product.tsx

```
import { Component } from 'react'
import checkLoggedIn from 'lib/checkLoggedIn'
import { NextContextNewContext } from 'lib/withApollo'
import { MeQuery, ProductQuery, ProductQueryVariables } from
'types/schema-types'
import { Row } from 'antd'
import PRODUCT_QUERY from 'queries/product/product.graphql'
import FormEditProduct from 'components/formEditProduct'
import { AdminLayoutWithProvider, AdminLayoutConsumer } from
'layout/layoutAdmin'
import { Query } from 'react-apollo'
import redirect from 'lib/redirect'
import '../styles/index.less'

type Props = {
  loggedInUser: MeQuery
  queryId: string
}

class QueryProduct extends Query<ProductQuery, ProductQueryVariables> {}

class FormEditProduk extends Component<Props> {
  static async getInitialProps (context: NextContextNewContext) {
    const { loggedInUser } = await checkLoggedIn(context.apolloClient)
    const queryId = context.query.id

    if (loggedInUser && loggedInUser.me && loggedInUser.me.role ===
"ADMIN") {
      return {
        loggedInUser,
        queryId: queryId
      }
    } else {
      redirect(context, '/')
    }
  }

  render() {
    const { loggedInUser, queryId } = this.props
    return (
      <AdminLayoutWithProvider
        idUser={loggedInUser && loggedInUser.me ? loggedInUser.me.id :
undefined}
        nameHeader="Edit / Lihat Produk"
      >
        <AdminLayoutConsumer>
          {( appProvider ) => {
            if (appProvider) {
              return (
                <Row
```

```

        type="flex"
        justify="center"
        align="middle"
      >
      <QueryProduct
        query={PRODUCT_QUERY}
        variables={{{
          id: queryId
        }}}
      >
        {({ data, error }) => {
          if (!error && data && data.product) {
            return (
              <FormEditProduct
                data={data.product}
                tag="BARANG"
              />
            )
          }
        }}
      </QueryProduct>
    </Row>
  )
)
}
</AdminLayoutConsumer>
</AdminLayoutWithProvider>
)
}
}

export default FormEditProduk

```

Form-edit-user.tsx

```

import { Component } from 'react'
import checkLoggedIn from 'lib/checkLoggedIn'
import { NextContextNewContext } from 'lib/withApollo'
import { MeQuery, UserQuery, UserQueryVariables } from 'types/schema-types'
import { Row } from 'antd'
import USER_QUERY from 'queries/user/user.graphql'
import FormEditUser from 'components/formEditUser'
import { AdminLayoutWithProvider, AdminLayoutConsumer } from 'layout/layoutAdmin'
import { Query } from 'react-apollo'
import '../styles/index.less'

type Props = {
  loggedInUser: MeQuery
  queryId: string
}

class QueryUser extends Query<UserQuery, UserQueryVariables> {}

class FormEditUserQ extends Component<Props> {

```

```
static async getInitialProps (context: NextContextNewContext) {
  const { loggedInUser } = await checkLoggedIn(context.apolloClient)
  const queryId = context.query.id
  if (loggedInUser && loggedInUser.me && loggedInUser.me.role !== "ADMIN") {

  }

  return {
    loggedInUser,
    queryId: queryId
  }
}

render() {
  const { loggedInUser, queryId } = this.props
  return (
    <AdminLayoutWithProvider
      idUser={loggedInUser && loggedInUser.me ? loggedInUser.me.id : undefined}
      nameHeader="Edit / Lihat User"
    >
    <AdminLayoutConsumer>
      {({ appProvider }) => {
        if (appProvider) {
          return (
            <Row
              type="flex"
              justify="center"
              align="middle"
            >
              <QueryUser
                query={USER_QUERY}
                variables={{
                  id: queryId
                }}
              >
                {({ data, error }) => {
                  if (!error && data && data.user) {
                    return (
                      <FormEditUser
                        data={data.user}
                      />
                    )
                  } else {
                    return null
                  }
                }}
              </QueryUser>
            </Row>
          )
        }
      }
    </AdminLayoutConsumer>
  </AdminLayoutWithProvider>
)
}
```

```

}

export default FormEditUserQ

```

Form-setting-admin.tsx

```

import { Component } from 'react'
import checkLoggedIn from 'lib/checkLoggedIn'
import { NextContextNewContext } from 'lib/withApollo'
import { MeQuery } from 'types/schema-types'
import { Row } from 'antd'
import FormSettingAccount from 'components/formSettingAccount'
import { AdminLayoutWithProvider, AdminLayoutConsumer } from
'layout/layoutAdmin'
import redirect from 'lib/redirect'
import '../styles/index.less'

type Props = {
  loggedInUser: MeQuery
}

class FormSettingAdmin extends Component<Props> {
  static async getInitialProps (context: NextContextNewContext) {
    const { loggedInUser } = await checkLoggedIn(context.apolloClient)

    if (loggedInUser && loggedInUser.me && loggedInUser.me.role ===
"ADMIN") {
      return {
        loggedInUser,
      }
    } else {
      redirect(context, '/')
    }
  }

  render() {
    const { loggedInUser } = this.props
    return (
      <AdminLayoutWithProvider
        idUser={loggedInUser && loggedInUser.me ? loggedInUser.me.id : undefined}
        nameHeader="Setting Admin"
      >
        <AdminLayoutConsumer>
          {( appProvider ) => {
            if (appProvider) {
              return (
                <Row
                  type="flex"
                  justify="center"
                  align="middle"
                >
                  <FormSettingAccount idUser={loggedInUser.me.id} />
                </Row>
              )
            }
          }}
```

```

        }
      </AdminLayoutConsumer>
    </AdminLayoutWithProvider>
  )
}

export default FormSettingAdmin

```

Form-tambah-paket-wedding.tsx

```

import { Component } from 'react'
import FormAddProduct from 'components/formAddProduct'
import checkLoggedIn from 'lib/checkLoggedIn'
import { NextContextNewContext } from 'lib/withApollo'
import { MeQuery } from 'types/schema-types'
import { Row } from 'antd'
import { AdminLayoutWithProvider, AdminLayoutConsumer } from
'layout/layoutAdmin'
import redirect from 'lib/redirect'
import '../styles/index.less'

type Props = {
  loggedInUser: MeQuery
}

class FormTambahPaketWedding extends Component<Props> {
  static async getInitialProps (context: NextContextNewContext) {
    const { loggedInUser } = await checkLoggedIn(context.apolloClient)

    if (loggedInUser && loggedInUser.me && loggedInUser.me.role ===
"ADMIN") {
      return {
        loggedInUser
      }
    } else {
      redirect(context, '/')
    }
  }

  render() {
    const { loggedInUser } = this.props
    return (
      <AdminLayoutWithProvider
        idUser={loggedInUser && loggedInUser.me ? loggedInUser.me.id : undefined}
        nameHeader="Tambah Paket Wedding"
      >
        <AdminLayoutConsumer>
          {( appProvider ) => {
            if (appProvider) {
              return (
                <Row
                  type="flex"
                  justify="center"
                  align="middle"

```

```

        >
          <FormAddProduct
            tag="PAKETWEDDING"
          />
        </Row>
      )
    }
  }
</AdminLayoutConsumer>
</AdminLayoutWithProvider>
)
}
}

export default FormTambahPaketWedding

```

Form-tambah-product.tsx

```

import { Component } from 'react'
import FormAddProduct from 'components/formAddProduct'
import checkLoggedIn from 'lib/checkLoggedIn'
import { NextContextNewContext } from 'lib/withApollo'
import { MeQuery } from 'types/schema-types'
import { Row } from 'antd'
import { AdminLayoutWithProvider, AdminLayoutConsumer } from
'layout/layoutAdmin'
import redirect from 'lib/redirect'
import '../styles/index.less'

type Props = {
  loggedInUser: MeQuery
}

class FormTambahProduk extends Component<Props> {
  static async getInitialProps (context: NextContextNewContext) {
    const { loggedInUser } = await checkLoggedIn(context.apolloClient)

    if (loggedInUser && loggedInUser.me && loggedInUser.me.role ===
"ADMIN") {
      return {
        loggedInUser
      }
    } else {
      redirect(context, '/')
    }
  }

  render() {
    const { loggedInUser } = this.props
    return (
      <AdminLayoutWithProvider
        idUser={loggedInUser && loggedInUser.me ? loggedInUser.me.id :
undefined}
        nameHeader="Tambah Produk"
      >

```

```

        <AdminLayoutConsumer>
        {(
            appProvider ) => {
            if (appProvider) {
                return (
                    <Row
                        type="flex"
                        justify="center"
                        align="middle"
                    >
                        <FormAddProduct
                            tag="BARANG"
                        />
                    </Row>
                )
            }
        })
    </AdminLayoutConsumer>
    </AdminLayoutWithProvider>
)
}
}

export default FormTambahProduk

```

home-admin.tsx

```

import { Component } from 'react'
import checkLoggedIn from 'lib/checkLoggedIn'
import { NextContextNewContext } from 'lib/withApollo'
import { MeQuery } from 'types/schema-types'
import { AdminLayoutWithProvider, AdminLayoutConsumer } from
'layout/layoutAdmin'
import redirect from 'lib/redirect'
import { Row } from 'antd'
import '../styles/index.less'
import FullCalendar from 'components/fullCalendar';

type Props = {
    loggedInUser: MeQuery
}

class HomeAdmin extends Component<Props> {
    static async getInitialProps (context: NextContextNewContext) {
        const { loggedInUser } = await checkLoggedIn(context.apolloClient)

        if (loggedInUser && loggedInUser.me && loggedInUser.me.role ===
"ADMIN") {
            return {
                loggedInUser
            }
        } else {
            redirect(context, '/')
        }
    }
}

```

```

    render() {
      const { loggedInUser } = this.props
      return (
        <AdminLayoutWithProvider
          idUser={loggedInUser && loggedInUser.me ? loggedInUser.me.id : undefined}
          nameHeader=""
        >
          <AdminLayoutConsumer>
            {({ appProvider }) => {
              if (appProvider) {
                return (
                  <Row
                    type="flex"
                    justify="center"
                    align="middle"
                    style={{
                      width: "100%",
                      height: "100%"
                    }}
                  >
                    <FullCalendar />
                  </Row>
                )
              }
            }}
          </AdminLayoutConsumer>
        </AdminLayoutWithProvider>
      )
    }
  }

export default HomeAdmin

```

Index.tsx

```

import React from 'react'
import checkLoggedIn from 'lib/checkLoggedIn'
import { NextContextNewContext } from 'lib/withApollo'
import { MeQuery } from 'types/schema-types'
import { LindaSalonIconWhiteLarge } from 'components/icons/linda-salon-icon'
import { UserLayoutWithProvider, UserLayoutConsumer } from 'layout/layoutUser'
import { Layout, Row, Col, Icon, Carousel } from 'antd'
import WhatsApp from 'components/icons/whatsapp'
import Bbm from 'components/icons/bbm'
import '../styles/index.less'

type Props = {
  loggedInUser: MeQuery,
}

class Index extends React.Component<Props> {
  static async getInitialProps (context: NextContextNewContext) {
    const { loggedInUser } = await checkLoggedIn(context.apolloClient)
  }
}

```

```
        return {
          loggedInUser
        }
      }

render() {
  const { loggedInUser } = this.props
  return (
    <>
    <UserLayoutWithProvider
      idUser={loggedInUser && loggedInUser.me ? loggedInUser.me.id : undefined}
    >
      <UserLayoutConsumer>
        {({ appProvider }) => {
          if ( appProvider ) {
            return (
              <>
                <Layout.Content style={{ marginTop: 110 }}>
                  <Carousel autoplay>
                    <div className="img-cover img-1" />
                    <div className="img-cover img-2" />
                    <div className="img-cover img-3" />
                    <div className="img-cover img-4" />
                </Carousel>
                <Row
                  type="flex"
                  justify="center"
                  align="middle"
                  className="slide-1"
                  style={{
                    marginBottom: 12
                  }}
                >
                  <h2 className="head-1">Selamat Datang di</h2>
                  <LindaSalonIconWhiteLarge />
                </Row>

                <Row
                  type="flex"
                  gutter={24}
                  justify="center"
                  style={{
                    padding: "64px 50px"
                  }}
                >
                  <Col
                    span={12}
                  >
                    <h1 className="heading-admin">Tentang
Kami</h1>
                    <p>Linda Salon menyediakan layanan pernikahan seperti Paket Pernikahan, Rias Pengantin, Busana Pengantin, Dekorasi Pernikahan, Foto Pengantin, Video Shooting, Pager Ayu / Bagus, Upacara Adat / Kesenian </p>
                  </Col>
                  <Col

```

```

        span={12}
      >
        <h1 className="heading-admin">Kontak</h1>
        <p>Dengan senang hati kami akan membantu anda
dalam mengemas pernikahan anda. Silahkan kontak Linda Salon untuk
informasi lebih lanjut, informasi kontak kami :</p>
        <div className="kontak-wrapper">
          <span className="kontak-icon">
            <Icon className="icon" type="home" style={{ fontSize: 24 }}/>
            Jl. Rancajigang RT.01 / RW.15 Ds.
Padamulya Kec. Majalaya Kab. Bandung 40382
          </span>
        </div>
        <div className="kontak-wrapper">
          <span className="kontak-icon">
            <Icon className="icon" type="instagram" style={{ fontSize: 24 }}/>
            Winda_Linda_Salon / #Lindasalon
          </span>
        </div>
        <div className="kontak-wrapper">
          <span className="kontak-icon">
            <Icon className="icon" type="facebook" style={{ fontSize: 24 }}/>
            Winda Hidayanti / Linda Salon
          </span>
        </div>
        <div className="kontak-wrapper">
          <span className="kontak-icon">
            <WhatsApp />
            <p style={{ margin: "0 0 0 8px" }}>08386464700 / 083822251836 / 085722357503</p>
          </span>
        </div>
        <div className="kontak-wrapper">
          <span className="kontak-icon">
            <Bbm />
            <p style={{ margin: "0 0 0 8px" }}>5C6FE740 / 52BE3196</p>
          </span>
        </div>
      </Col>
    </Row>
  </Layout.Content>
</>
)
}
}
</UserLayoutConsumer>
</UserLayoutWithProvider>
</>
)
}
}

export default Index

```

Paket-nikah.tsx

```
import React from 'react'
import ProductLists from 'components/productsList'
import checkLoggedIn from 'lib/checkLoggedIn'
import { NextContextNewContext } from 'lib/withApollo'
import { PageInfo } from 'utilities/pageInfo'
import { MeQuery } from 'types/schema-types'
import { UserLayoutWithProvider, UserLayoutConsumer } from
'layout/layoutUser'
import { Layout } from 'antd'
import '../styles/index.less'

type Props = {
  loggedInUser: MeQuery,
  pageInfo: PageInfo
}

const PRODUCT_PER_PAGE = 12

class PaketNikah extends React.Component<Props> {
  static async getInitialProps (context: NextContextNewContext) {
    const { loggedInUser } = await checkLoggedIn(context.apolloClient)
    return {
      loggedInUser
    }
  }

  render() {
    const { loggedInUser } = this.props
    return (
      <>
        <UserLayoutWithProvider
          idUser={loggedInUser && loggedInUser.me ? loggedInUser.me.id : undefined}>
          <UserLayoutConsumer>
            {({ appProvider }) => {
              if ( appProvider ) {
                return (
                  <Layout.Content style={{ marginTop: 110, padding:
"50px 50px", background: "#FFF" }}>
                    <ProductLists
                      productPerPage={PRODUCT_PER_PAGE}
                      tag="PAKETWEDDING"
                      dataFragment={appProvider.dataUserForProductFilter}
                      />
                    </Layout.Content>
                )
              }
            }}
          </UserLayoutConsumer>
        </UserLayoutWithProvider>
      </>
    )
  }
}
```

```

        )
    }
}

export default PaketNikah

```

produk.tsx

```

import React from 'react'
import ProductLists from 'components/productsList'
import checkLoggedIn from 'lib/checkLoggedIn'
import { NextContextNewContext } from 'lib/withApollo'
import { MeQuery } from 'types/schema-types'
import { UserLayoutWithProvider, UserLayoutConsumer } from
'layout/layoutUser'
import { Layout } from 'antd'
import '../styles/index.less'

type Props = {
    loggedInUser: MeQuery,
}

const PRODUCT_PER_PAGE = 12

class Produk extends React.Component<Props> {
    static async getInitialProps (context: NextContextNewContext) {
        const { loggedInUser } = await checkLoggedIn(context.apolloClient)
        return {
            loggedInUser
        }
    }

    render() {
        const { loggedInUser } = this.props
        return (
            <>
                <UserLayoutWithProvider
                    idUser={loggedInUser && loggedInUser.me ? loggedInUser.me.id
: undefined}
                >
                    <UserLayoutConsumer>
                        {({ appProvider }) => {
                            if ( appProvider ) {
                                return (
                                    <Layout.Content style={{ marginTop: 110, padding:
"50px 50px", background: "#FFF" }}>
                                        <ProductLists
                                            productPerPage={PRODUCT_PER_PAGE}
                                            tag="BARANG"
                                            dataFragment={appProvider.dataUserForProductFilter}
                                            />
                                    </Layout.Content>
                                )
                            }
                        }}
                
```

```

        </UserLayoutConsumer>
    </UserLayoutWithProvider>
    </>
)
}
}

export default Produk

```

Setting-akun.tsx

```

import React from 'react'
import checkLoggedIn from 'lib/checkLoggedIn'
import { NextContextNewContext } from 'lib/withApollo'
import { MeQuery } from 'types/schema-types'
import { UserLayoutWithProvider, UserLayoutConsumer } from
'layout/layoutUser'
import FormSettingAccount from 'components/formSettingAccount'
import redirect from 'lib/redirect'
import { Row, Layout } from 'antd'
import '../styles/index.less'

type Props = {
    loggedInUser: MeQuery,
}

class SettingAkun extends React.Component<Props> {
    static async getInitialProps (context: NextContextNewContext) {
        const { loggedInUser } = await checkLoggedIn(context.apolloClient)
        if (loggedInUser && loggedInUser.me) {
            return {
                loggedInUser,
            }
        } else {
            redirect(context, '/user-login')
        }
    }

    render() {
        const { loggedInUser } = this.props
        return (
            <>
            <UserLayoutWithProvider
                idUser={loggedInUser && loggedInUser.me ? loggedInUser.me.id : undefined}>
            </UserLayoutWithProvider>
            <UserLayoutConsumer>
                {( appProvider ) => {
                    if ( appProvider && appProvider.idUser ) {
                        return (
                            <Layout.Content style={{ marginTop: 110, padding:
                            "50px 50px", background: "#FFF" }}>
                                <Row type="flex" justify="center">
                                    <FormSettingAccount idUser={loggedInUser.me.id} />
                                </Row>
                            </Layout.Content>
                        )
                    }
                }}
            </UserLayoutConsumer>
        )
    }
}

```

```

        )
    }

        </UserLayoutConsumer>
        </UserLayoutWithProvider>
        </>
    )
}
}

export default SettingAkun

```

Singup-user.tsx

```

import React, { Component } from 'react'
import { message, Form, Input, Button, Layout } from 'antd'
import { FormComponentProps } from 'antd/lib/form'
import { graphql, compose, MutationFunc, DataValue } from 'react-apollo'
import { singUpMutation, singUpMutationVariables, MeQuery } from
'types/schema-types'
import SINGUP_QUERY from 'queries/user/signUpM.graphql'
import ME_QUERY from 'queries/user/meQ.graphql'
import Router from 'next/router'
import { LindaSalonIcon } from 'components/icons/linda-salon-icon'

const FormItem = Form.Item;
const { Content } = Layout

type PropSignUp = {
  meQ: DataValue<MeQuery>,
  singUpMutation: MutationFunc<singUpMutation,
  singUpMutationVariables> | undefined
} & FormComponentProps

const hasErrors = fieldsError => {
  return Object.keys(fieldsError).some(field => fieldsError[field]);
}

class SignUpUser extends Component<PropSignUp, {}> {

  state = {
    confirmDirty: false
  }

  componentDidMount() {
    // To disabled submit button at the beginning.
    this.props.form.validateFields()
  }

  handleSubmit = (event: React.FormEvent<HTMLFormElement>) => {
    const { form, singUpMutation } = this.props
    event.preventDefault()
    form.validateFields((err, values) => {
      if (!err && singUpMutation) {

```

```

singUpMutation({
  variables: {
    name: values.name,
    email: values.email,
    password: values.password
  }
}).then(result => {
  if (result) {
    localStorage.setItem('token', result.data.signup.token)
    Router.push('/')
  }
}).catch(error => message.error(error.message))
}
})
}

compareToFirstPassword = (rule, value, callback) => {
  console.log(rule)
  const form = this.props.form;
  if (value && value !== form.getFieldValue('password')) {
    callback('Two passwords is inconsistent!');
  } else {
    callback();
  }
}

validateToNextPassword = (rule, value, callback) => {
  console.log(rule)
  const form = this.props.form;
  if (value && this.state.confirmDirty) {
    form.validateFields(['confirm'], { force: true } as any);
  }
  callback();
}

render() {
  const { getFieldDecorator, getFieldsError, getFieldError,
  isFieldTouched } = this.props.form
  const nameError = isFieldTouched('name') && getFieldError('name')
  const emailError = isFieldTouched('email') &&
getFieldError('email')
  const passwordError = isFieldTouched('password') &&
getFieldError('password')
  const confirmPasswordError = isFieldTouched('confirmPassword') &&
getFieldError('confirmPassword')

  return (
    <Layout className="login-cover">
      <Content className="content-login">
        <Layout className="login-wrapper">
          <LindaSalonIcon />
        </Layout>
        <Form layout="vertical" onSubmit={this.handleSubmit}>
          <FormItem
            validateStatus={nameError ? 'error' : 'success'}
            help={nameError || ''}
          >

```

```
{
  getFieldDecorator('name', {
    rules: [
      { required: true, message: 'Please input your
name!' }
    ]
  ))(
    <Input type="text" placeholder="Name" />
  )
}
</FormItem>
<FormItem
  validateStatus={emailError ? 'error' : 'success'}
  help={emailError || ''}
>
{
  getFieldDecorator('email', {
    rules: [
      { required: true, message: 'Please input your
email!' }
    ]
  ))(
    <Input type="text" placeholder="Email" />
  )
}
</FormItem>
<FormItem
  validateStatus={passwordError ? 'error' : 'success'}
  help={passwordError || ''}
>
{
  getFieldDecorator('password', {
    rules: [
      { required: true, message: 'Please input your
password!', },
      { validator: this.validateToNextPassword }
    ]
  ))(
    <Input type="password" placeholder="Password" />
  )
}
</FormItem>
<FormItem
  validateStatus={confirmPasswordError ? 'error' :
'success'}
  help={confirmPasswordError || ''}
>
{
  getFieldDecorator('confirmPassword', {
    rules: [
      { required: true, message: 'Please confirm your
password!', },
      { validator: this.compareToFirstPassword }
    ]
  ))(
    <Input type="password" placeholder="Confirm

```

```

        Password" />
      )
    }
  </FormItem>
  <FormItem>
    <Button type="primary" htmlType="submit"
disabled={hasErrors(getFieldsError())} className="button-submit-
login">
      Sign up
    </Button>
  </FormItem>
  </Form>
</Content>
</Layout>
)
}
}

const SignUpUserUpWrapper = Form.create({})(SignUpUser)

const WithSingUpMutation = compose(
  graphql<{}, MeQuery, {}, {}>(ME_QUERY, {
    name: 'meQ',
    options: {
      fetchPolicy: "network-only"
    },
  }),
  graphql<{}, singUpMutation, singUpMutationVariables,
{}>(SINGUP_QUERY, {
    name: 'singUpMutation'
  })
)(SignUpUserUpWrapper)

export default WithSingUpMutation

```

User-booking.tsx

```

import React from 'react'
import BookingList from 'components/bookingList'
import checkLoggedIn from 'lib/checkLoggedIn'
import { NextContextNewContext } from 'lib/withApollo'
import { MeQuery } from 'types/schema-types'
import { Layout } from 'antd'
import { UserLayoutWithProvider, UserLayoutConsumer } from
'layout/layoutUser'
import '../styles/index.less'

type Props = {
  loggedInUser: MeQuery,
}

const BOOKING_PER_PAGE = 12

class UserBooking extends React.Component<Props> {
  static async getInitialProps (context: NextContextNewContext) {
    const { loggedInUser } = await checkLoggedIn(context.apolloClient)
  }
}

```

```

        return {
          loggedInUser
        }
      }

      render() {
        const { loggedInUser } = this.props
        return (
          <>
            <UserLayoutWithProvider
              idUser={loggedInUser && loggedInUser.me ? loggedInUser.me.id : undefined}
            >
              <UserLayoutConsumer>
                {({ appProvider }) => {
                  if ( appProvider && appProvider.idUser ) {
                    return (
                      <Layout.Content style={{ marginTop: 110, padding: "50px 50px", background: "#FFF" }}>
                        <BookingList
                          bookingPerPage={BOOKING_PER_PAGE}
                          idUser={appProvider.idUser}
                        />
                      </Layout.Content>
                    )
                  }
                }}
              </UserLayoutConsumer>
            </UserLayoutWithProvider>
          </>
        )
      }
    }

    export default UserBooking
  
```

User-login.tsx

```

import React, { Component } from 'react'
import { message, Layout, Form, Icon, Input, Button } from 'antd'
import { graphql, compose, MutationFunc, DataValue } from 'react-apollo'
import { LindaSalonIcon } from 'components/icons/linda-salon-icon'
import { FormComponentProps } from 'antd/lib/form'
import { LoginMutation, LoginMutationVariables, MeQuery } from 'types/schema-types'
import ME from 'queries/user/meQ.graphql'
import LOGIN from 'queries/user/loginM.graphql'
import cookie from 'cookie'
import Router from 'next/router'
import '../styles/index.less'

const { Content } = Layout
const FormItem = Form.Item

type PropLogin = {
  
```

```

    meQ: DataValue<MeQuery>,
    loginMutation: MutationFunc<LoginMutation, LoginMutationVariables> |
undefined
} & FormComponentProps

const hasErrors = fieldsError => {
  return Object.keys(fieldsError).some(field => fieldsError[field]);
}

class UserLogin extends Component<PropLogin, {}>{

  componentDidMount() {
    // To disabled submit button at the beginning.
    this.props.form.validateFields();
  }

  private handleSubmit = (event: React.FormEvent<HTMLFormElement>) =>
{
  const { form, loginMutation } = this.props
  event.preventDefault();
  form.validateFields((err, values) => {
    if (!err && loginMutation) {
      loginMutation({
        variables: {
          email: values.email,
          password: values.password
        }
      }).then(result => {
        if (result) {
          document.cookie = cookie.serialize('token',
result.data.login.token, {
            maxAge: 30 * 24 * 60 * 60 // 30 days
          })
          localStorage.setItem('token', result.data.login.token)
          if (result.data.login.user.role === 'ADMIN') {
            Router.push('/home-admin')
          } else {
            Router.push('/')
          }
        }
      }).catch(error => message.error(error.message))
    }
  });
}

  render() {
    const { getFieldDecorator, getFieldsError, getFieldError,
isFieldTouched } = this.props.form

    const userNameError = isFieldTouched('email') &&
getFieldError('email')
    const passwordError = isFieldTouched('password') &&
getFieldError('password')

    return (
      <Layout className="login-cover">

```

```

<Content className="content-login">
  <Layout className="login-wrapper">
    <LindaSalonIcon />
  </Layout>
  <Form layout="vertical" onSubmit={this.handleSubmit}>
    <FormItem
      validateStatus={userNameError ? 'error' : 'success'}
      help={userNameError || ''}
    >

      {getFieldDecorator('email', {
        rules: [{ required: true, message: 'Please input your
email!' }],
      })(

        <Input prefix={<Icon type="user" style={{ color:
'rgba(0,0,0,.25)' }} />} placeholder="Email" />
      )}
    </FormItem>
    <FormItem
      validateStatus={passwordError ? 'error' : 'success'}
      help={passwordError || ''}
    >

      {getFieldDecorator('password', {
        rules: [{ required: true, message: 'Please input your
Password!' }],
      })(

        <Input prefix={<Icon type="lock" style={{ color:
'rgba(0,0,0,.25)' }} />} type="password" placeholder="Password" />
      )}
    </FormItem>
    <FormItem>
      <Button type="primary" htmlType="submit"
disabled={hasErrors(getFieldsError())} className="button-submit-
login">
        Log in
      </Button>
      <div className="register-wrapper">
        Or <a onClick={() => Router.push('/signup-user')}>
          Daftar sekarang!
        </a>
      </div>
    </FormItem>
  </Form>
</Content>
</Layout>
)
}
}

const WithLoginMutation = compose(
  graphql<{}, LoginMutation, LoginMutationVariables, {}>(LOGIN, {
    name: 'loginMutation',
  }),
  graphql<{}, MeQuery, {}, {}>(ME, {
    name: 'meQ',
    options: {
      fetchPolicy: "network-only"
    }
  })
)

```

```
  },
  Form.create({})
)(UserLogin)

export default WithLoginMutation
```