BAB III

PERANCANGAN DAN IMPLEMENTASI

Dalam pembuatan aplikasi Tugasku ini penulis menggunakan bahasa pemrograman Java dan menggunakan Eclipse IDE sebagai perangkat lunak pendukung untuk mempermudah pembuatan aplikasi Android dan aplikasi pada Web Server. Dalam proses perancangan pembuatan Aplikasi Tugasku ini terdiri dari beberapa tahap.

3.1 Analisa Program

Pada Analisa Program penulis akan menjelaskan tentang gambaran umum dari aplikasi Tugasku ini. Aplikasi ini diperuntukkan bagi umum untuk mencatat tugas sehari hari dan sebagai pengingat. Penggunaan aplikasi ini bisa di gunakan kapan saja asalkan pengguna sudah mempunyai *smartphone* untuk menginstalnya kedalam sebuah mobile Android yang sesuai dengan versi yang di buat pada aplikasi ini yaitu sistem operasi Android 2.2 dan juga dapat digunakan pada sistem operasi diatasnya.

Fitur utama dari aplikasi Tugasku ini adalah sebagai pengingat tugas tugas di dalam kehidupan sehari hari. Dan juga data bisa di simpan di *cloud*.

3.2 Perancangan Aplikasi

Dalam pembuatan aplikasi ini di bagi menjadi dua langkah utama, yang pertama membuat server yang akan menjadi API di *cloud server* dan sebagai penyimpanan data kemudian langkah berikutnya adalah pembuatan aplikasi di sisi *client* yaitu Aplikasi Android.

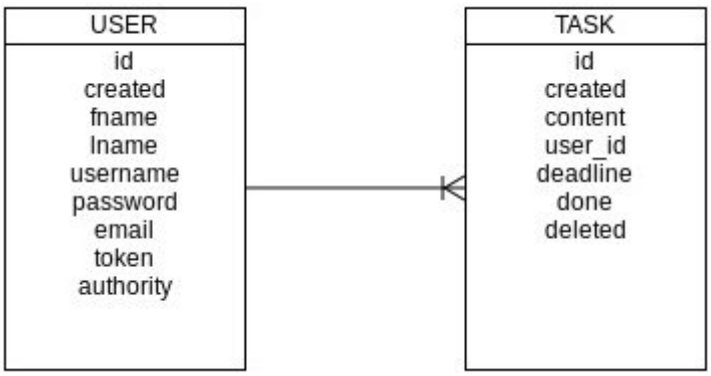
Di dalam pembuatan aplikasi *server* yang pertama penulis akan membuat rancangan database dan, ERD dan Struktur Tabel, kemudian di sisi client Aplikasi Android yang pertama di lakukan adalah membuat rancangan struktur navigasi dan rancangan tampilan (*Mockup*)

3.2.1 Aplikasi *Server*

Di dalam pembuatan aplikasi *web service* di *server cloud* ini penulis perlu membuat rancangan *database* ERD dan Struktur Tabel terlebih dahulu.

3.2.1.1 Rancangan Database

Sebelum membuat *database* dan tabel pada DBMS penulis akan menuliskan rancangan database secara konseptual melalui *Entity Relationship Diagram (ERD)*. Pada penulisan ilmiah ini penulis akan membuat rancangan *Entity Relationship Diagram (ERD)* dimana terdapat 2 buah tabel master . Berikut ini adalah gambar rancangan *ERD*:



Gambar 3.1 *Entity Relationship Diagram*

Pada gambar *ERD* diatas terdapat 2 buah tabel *master user* dan *task*, dan dengan relasi satu *user* dapat mempunyai beberapa *task* (*one to many*)

Setelah relasi antar tabel dibuat, maka langkah selanjutnya adalah mentransformasikan relasi tersebut pada *database*. Berikut ini adalah struktur tabel yang akan dibuat pada *database*.

Tabel 3.1 Struktur Tabel User

Nama Field	Tipe Data	Keterangan
* id	Varchar(50)	Data unik sebagai identifikasi suatu data
created	Timestamp	Menyimpan informasi waktu pada saat data di simpan
fname	Varchar(25)	Menyimpan informasi nama depan pengguna
lname	Varchar(25)	Menyimpan informasi nama belakang pengguna
username	Varchar(25)	Menyimpan informasi id untuk *login* pengguna
password	Varchar(25)	Menyimpan informasi kata sandi pengguna
email	Varchar(25)	Menyimpan informasi *email* pengguna
token	Varchar(50)	Menyimpan *token* yang sudah di dapat ketika *login*
authority	Varchar(10)	Menyimpan *role* dari pengguna

Tabel 3.2 Struktur Tabel Task

Nama Field	Tipe Data	Keterangan
* id	Varchar(50)	Data unik sebagai identifikasi suatu data
created	Timestamp	Menyimpan informasi waktu pada saat data di simpan
content	Varchar(255)	Menyimpan informasi tugas
** user_id	Varchar(50)	Menyimpan informasi identifikasi pengguna
deadline	Timestamp	Menyimpan informasi tanggal jatuh tempo tugas
done	Integer	Menyimpan informasi tugas sudah selesai
deleted	Integer	Menyimpan informasi data telah di hapus

3.2.1.2 Membuat *Database* dan Tabel pada PostgreSQL

Membuat *database* dan tabel pada PostgreSQL bisa dilakukan dengan dua cara, yakni melalui *console* dengan mengetikkan perintah – perintah SQL secara manual atau bisa juga melalui pgAdmin. Pada penulisan ilmiah ini penulis akan membuat *database* dan tabel menggunakan *console*.

```
CREATETABLE etask_user (
        id charactervarying(255)NOTNULL,
        created timestampwithouttimezone,
        email charactervarying(255),
        first_name charactervarying(255),
        last_name charactervarying(255),
        password charactervarying(255),
        token charactervarying(255),
        username charactervarying(255),
        authority charactervarying(255),
        CONSTRAINT etask_user_pkey PRIMARYKEY(id),
        CONSTRAINT etask_user_email_key UNIQUE(email),
        CONSTRAINT etask_user_username_key UNIQUE(username)
)
WITH(
        OIDS=FALSE
);
CREATETABLE task (
        id charactervarying(255)NOTNULL,
        created timestampwithouttimezone,
        content charactervarying(255),
        deadline timestampwithouttimezone,
        deleted boolean,
        done boolean,
        user_id charactervarying(255),
        CONSTRAINT task_pkey PRIMARYKEY(id),
        CONSTRAINT fk3635856d590aaa FOREIGNKEY(user_id)
        REFERENCES etask_user (id)MATCHUnknown
        ONUPDATENO ACTION ONDELETENO ACTION
)
WITH(
        OIDS=FALSE
);
```

3.2.2 Aplikasi Android

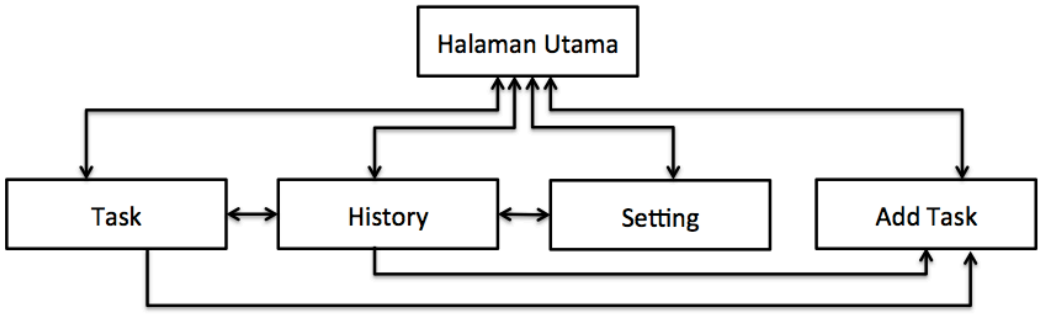
Dalam pembuatan aplikasi Android yang berfungsi sebagai *client* ini terlebih dahulu penulis membuat rancangan struktur navigasi kemudian dilakukan rancangan tampilan aplikasi utama dan pembuatan rancangan tampilan pengaturan.

3.2.2.1 Stuktur Navigasi

Dalam proses perancangan aplikasi, struktur navigasi sangat penting. Struktur navigasi dapat menjelaskan hubungan antar halaman dan juga dapat menjelaskan mengenai alur cerita sebuah program atau aplikasi, juga memberi

kemudahan menganalisa keteraktifan seluruh obyek dan bagaimana pengaruh keinteraktifannya terhadap pengguna. Dalam pembuatan aplikasi ini penulis menggunakan struktur navigasi campuran.

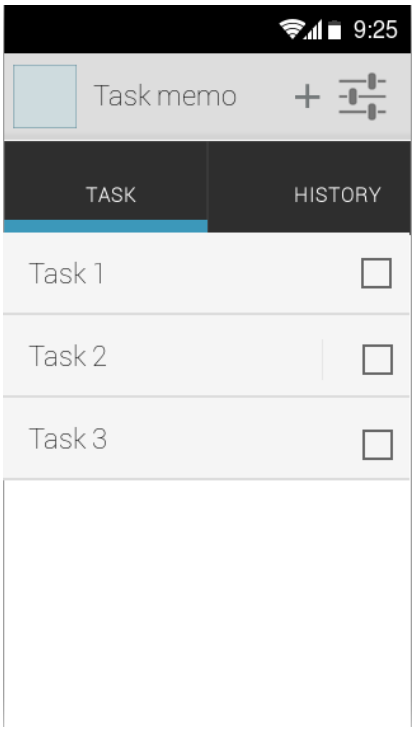
Dalam Struktur navigasi di bawah dapat di lihat ketika kita menjalankan aplikasi tugasku maka aplikasi akan menampilkan halaman utama, di halaman utama ini ada tiga menu yaitu “Task”, “History”, “Setting” dan “Add Task”. Menu Task menampilkan daftar tugas yang telah di input oleh user, menu History di gunakan untuk menampilkan riwayat dari tugas yang telah di kerjakan, kemudian menu setting akan menampilkan halaman pengaturan aplikasi, dan menu Add Task untuk menambahkan tugas.



Gambar 3.2 Struktur Navigasi

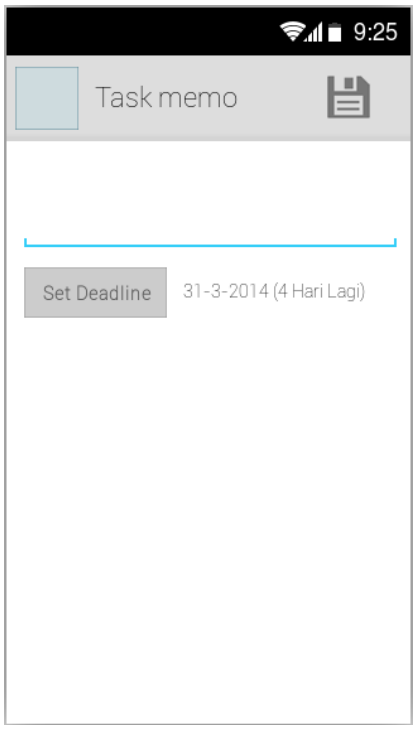
3.2.2.2 Rancangan Tampilan

Rancangan tampilan aplikasi ini terdiri dari rancangan halaman aplikasi utama yang berupa daftar task atau tugas di halaman ini akan menampilkan tugas yang pernah penulis isi dan informasi berapa hari task ini harus selesai (*deadline*). Kemudian di bagian *action bar* terdapat dua menu yaitu menu add untuk menambah tugas dan menu setting untuk pengaturan aplikasi.



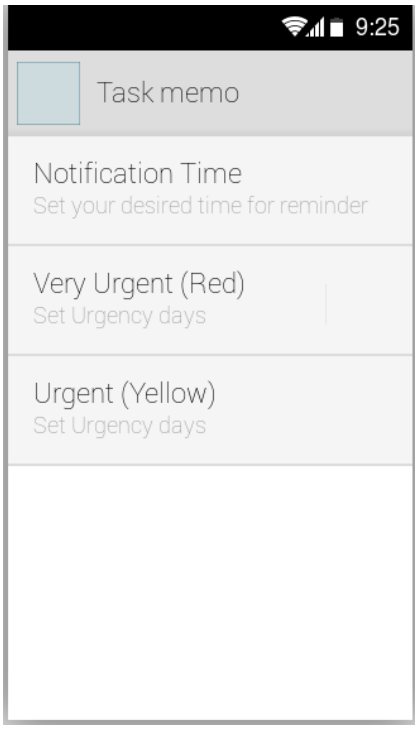
Gambar 3.3 Rancangan Tampilan Dashboard

Untuk tampilan pada halaman Add Task terdapat *form* isian untuk informasi tugas di antaranya adalah konten dan tanggal deadline. *Deadline* ini di gunakan untuk penghitungan berapa hari tugas ini harus selesai yang nantinya di tampilkan pada halaman aplikasi utama.



Gambar 3.4 Rancangan Tampilan Form Tambah Tugas

Kemudian di tampilan setting atau pengaturan terdapat beberapa pilihan pengaturan diantaranya, “Notification Time” yang di fungsinya untuk mengatur pada jam berapa notifikasi akan di tampilkan. Notifikasi ini berisi tentang informasi *deadline* yang terdekat, kemudian ada pengaturan “Urgency”, di pengaturan ini ada dua kategori pertama kategori “Very Urgent” dan “Urgent” di sini penulis bisa mengatur berapa hari sebelum *deadline* di sebut “Very Urgent” atau “Urgent” yang nanti akan di tampilkan warna berbeda di halaman aplikasi utama



Gambar 3.5 Rancangan Tampilan Pengaturan

3.3 Pembuatan Aplikasi

Setelah melalui tahap perancangan aplikasi kemudian lakukan langkah-langkah pembuatan aplikasi sesuai dengan rancangan sistem dan rancangan tampilan yang telah dibuat sebelumnya. Karena sistem yang akan penulis buat ada dua komponen yaitu *Client* dan *Server* maka penulis akan membuat di sisi *server* terlebih dahulu. Tapi sebelum itu penulis akan melakukan meng-install Eclipse dan Plugin-plugin yang di perlukan untuk membuat aplikasi ini.

3.3.1 Meng-install IDE

Langkah-langkah pembuatan aplikasi Aplikasi Tugasku. Pertama-tama diawali dengan meng-*instal* jdk , Proses peng-*instal*-an ini dilakukan pada sistem operasi Centos, jika *software* ini belum ada pada komputer yang digunakan untuk meng-compile skrip *coding* yang dibuat.

Software jdk bisa didapatkan dengan cara men-*download* atau membeli cd aslinya. Namun pada kesempatan ini penulis men-*download* website

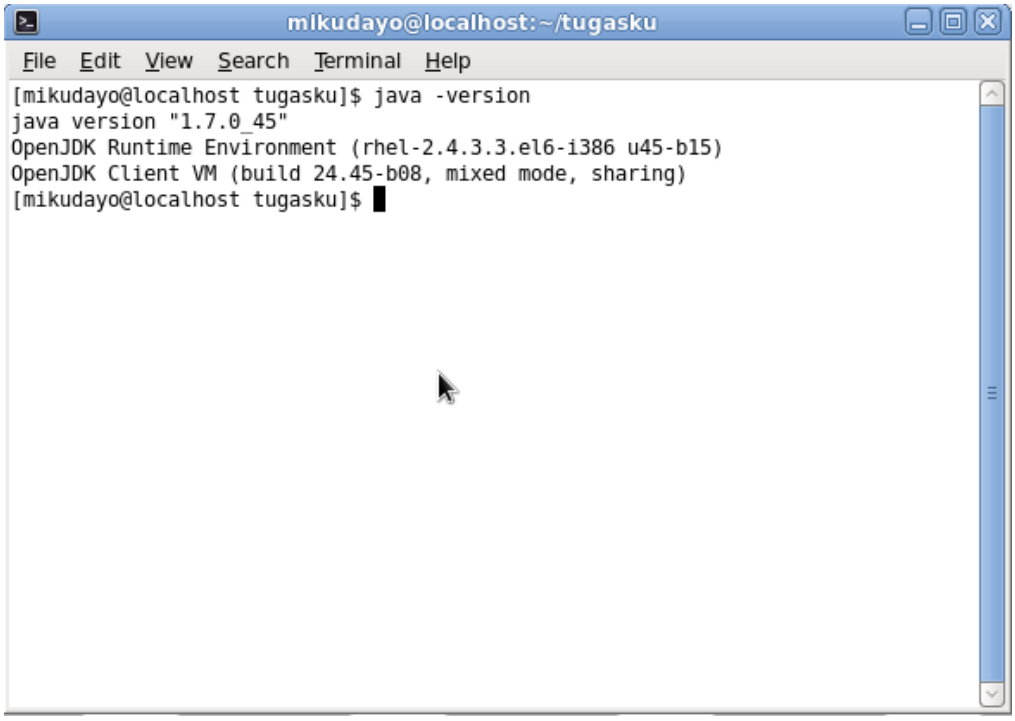
www.oracle.com/technetwork/java/javase/downloads/index.html. Untuk meng-installnya penulis cukup mengeksekusi file binary yang sebelumnya penulis download dengan perintah di konsol

```
rpm -i jdk-7u25-linux-i586.rpm
```

Selanjutnya ketika meng-install selesai penulis bisa cek apakah java sudah terinstall dengan baik di komputer penulis dengan mengeksekusi perintah

```
java -version
```

Jika jdk sudah terinstall dengan baik makan yang akan muncul di konsol adalah seperti di bawah ini



Gambar 3.6 Cek Meng-install JDK Sukses

Kemudian selanjutnya penulis akan meng-*install* Eclipse IDE, untuk mendapatkanya penulis bisa men-*download* dari *website* https://www.eclipse.org/downloads/ dan pilih untuk versi yang "**Eclipse IDE for Java EE Developers"** karena di versi ini merupakan versi terlengkap yang di

dalamnya sudah terdapat *plugin-plugin* yang akan penulis gunakan. Setelah berhasil *download*, meng-install eclipse ini cukup mudah yaitu tunggal ekstrak di tempat yang di inginkan. Untuk menjalankan eclipse ini penulis tinggal mengeksekusi *binary* yang bernama eclipse di dalam archive yang penulis *download* sebelumnya.

Karena eclipse yang penulis download ini berlum terdapat *Plugin* untuk membuat aplikasi android maka penulis perlu meng-*install* secara manual.

1. Di Eclipse pilih menu "Help" > "Install New Software"
2. Klik "Add" di pojok kanan atas
3. Di dalam dialog "Add Repository" masukkan *url* https://dl-ssl.google.com/android/eclipse/ di bagian "Location".
4. Klik "OK"
5. Di dialog "Available Software Dialog", pilih *checkbox* di sebelah "Developer Tools" kemudian klik Next
6. Di dialog selanjutnya lanjutkan dengan klik tombol "Next"
7. Kemudian baca dan centang "License Agreements" kemudian klik "Finish"
8. Setelah meng-install selesai restart Eclipse

Kemudian yang di perlukan selanjutnya adalah *meng-install* sdk android di eclipse

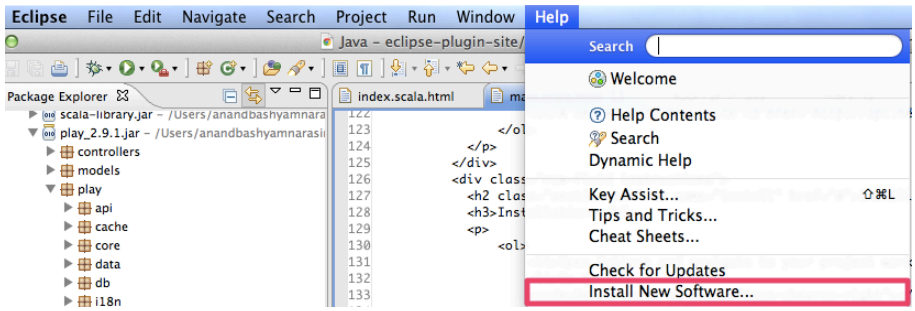
Pertama penulis perlu men-*download* Android SDK dari *website* resmi http://developer.android.com/sdk/index.html dan memilih versi yang android-sdk_r22.3-linux.tgz Kemudian ekstrak di tempat yang di inginkan. Kemudian buka eclipse dan lakukan langkah-langkah berikut

1. Di Eclipse pilih menu "Window" > "Preferences"
2. Di sebelah navigasi kiri pilih "Android"
3. Kemudian masukkan lokasi *sdk* yang sudah penulis ekstrak di bagian "SDK Location", Kemudian klik "Apply" dan "OK"

Setelah selesai untuk meng-*install* yang di butuhkan untuk membuat aplikasi Android selanjutnya penulis akan meng-*install plugin* untuk membuat aplikasi yang dapat di publish ke penyedia layanan cloud yaitu Heroku

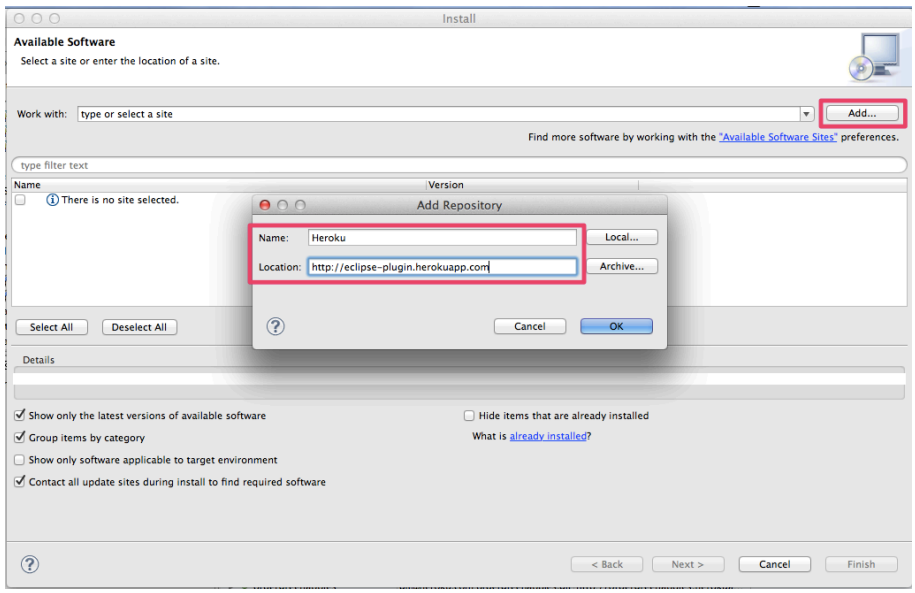
Langkah pertama penulis registrasi akun baru di *website* heroku https://id.heroku.com/signup/devcenter setelah mendapatkan akun penulis lanjutkan untuk meng-*install* di eclipse

1. Di tampilan utama Eclipse pilih menu "Help" > "Install New Software"



Gambar 3.7 Menu "Install New Software"

2. Clik "Add"
3. Masukkan *url* dengan nama Heroku dan https://eclipse-plugin.herokuapp.com/install di bagian "Location" kemudian klik "OK"



Gambar 3.8 Menginputkan link meng-*install* plugin

4. Pilih *Checkbox* di sebelah kiri “Heroku Eclipse Integration”
5. Klik “Next” dua kali
6. Baca “Accept License Agreements” dan kemudian “Finish”
7. *Restart* Eclipse

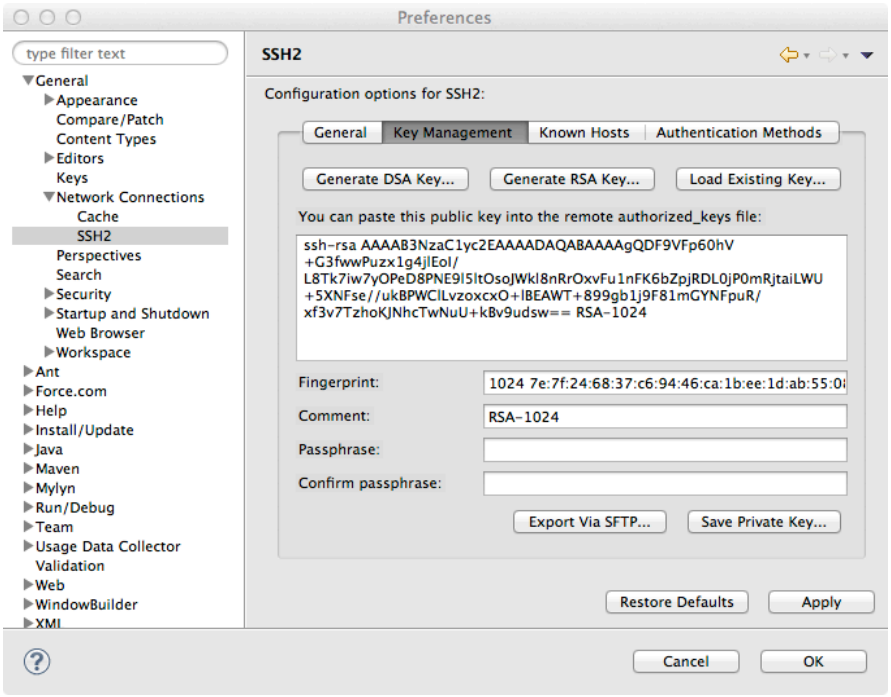
Setelah meng-*install* plugin heroku selesai selanjutnya penulis akan mengkonfigurasi “API Key” untuk heroku agar eclipse penulis terintegrasi dengan Heroku.

1. Di tampilan utama Eclipse pilih “Eclipse” > “Preferences”
2. Pilih Menu “Heroku” di bagian kiri
3. Masukkan *email* dan *password* yang sebelumnya sudah di registrasi
4. Klik “Login”, Jika *login* sukses maka “API Key” akan otomatis terisi.

Membuat *SSH Key*

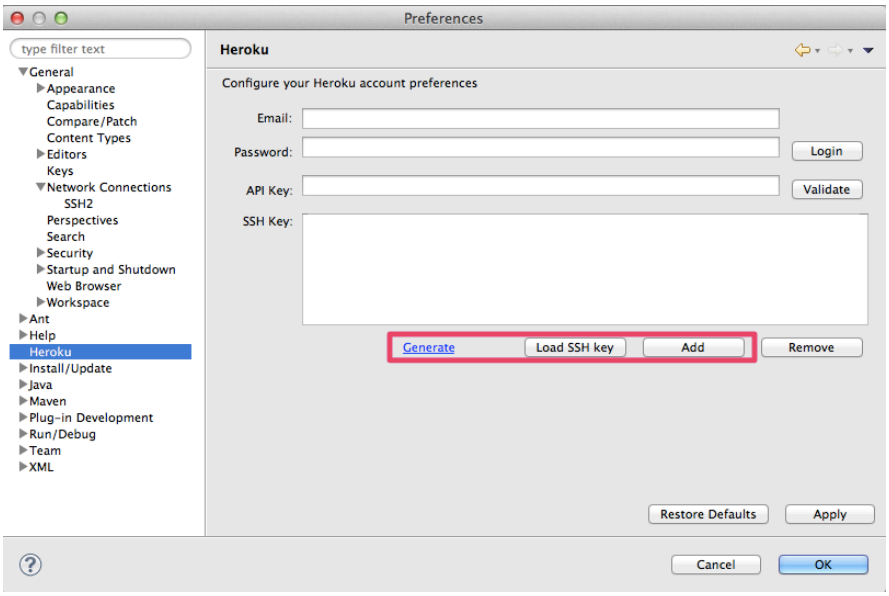
Untuk meng-*upload* aplikasi penulis ke Heroku, *plugin* eclipse Heroku ini menggunakan Git. Jadi selanjutnya penulis akan meng-*install plugin* Git di Eclipse, langkah-langkah untuk meng-*install* ini hampir sama seperti sebelumnya penulis meng-*install plugin* heroku yang membedakan adalah *url* plugin untuk *git* ini adalah http://download.eclipse.org/egit/updates. Sebuah *SSH Key* di butuhkan untuk untuk komunikasi git ke *server* heroku. Penulis membutuhkan beberapa konfigurasi untuk pertama kali membuat aplikasi di Heroku. Pertama yang perlu penulis lakukan adalah meng-*generate* SSH Key baru. Berikut adalah step step untuk mendapatkanya

1. Di Eclipse pilih menu “Eclipse” > “Preferences” > “General Network Connections” > “SSH2”
2. Pilih tab “Key Management”
3. Klik “Generate SSH Key”



Gambar 3.9 Konfigurasi *SSH Key*

4. Klik “Save Private Key”, kemudian Klik “OK”
5. Di Eclipse pilih menu “Eclipse” > “Preference” > “Heroku”
6. Di dalam kolom “SSH Key” klik “Generate”, Kemudian klik “Add” dan “OK”.



Gambar 3.10 *Generate SSH Key*

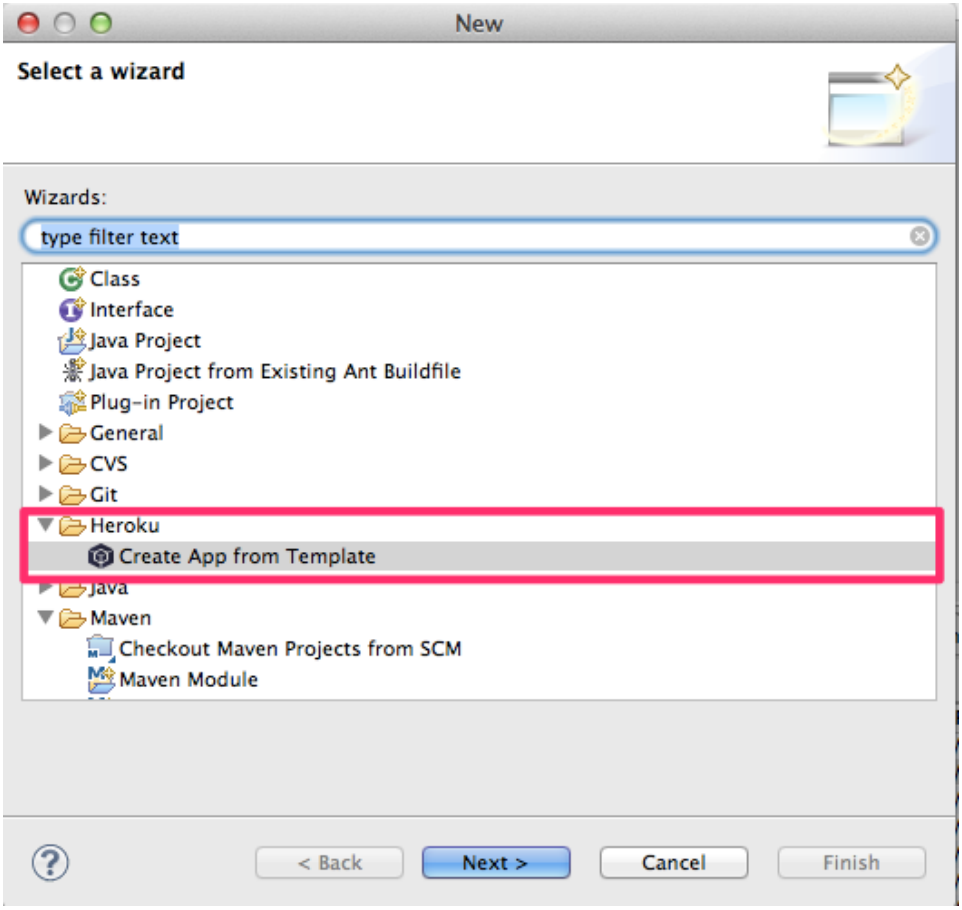
3.3.2 Aplikasi *Server*

Setelah selesai dengan *install*-asi IDE dan *plugin-plugin* langkah selanjutnya adalah membuat aplikasi *server* dimana di tahap ini pertama penulis harus membuat sebuah *database* dan *table* di PostgreSQL dengan *schema* yang sudah penulis buat sebelumnya.

3.3.2.1 Membuat Sring MVC *Project*

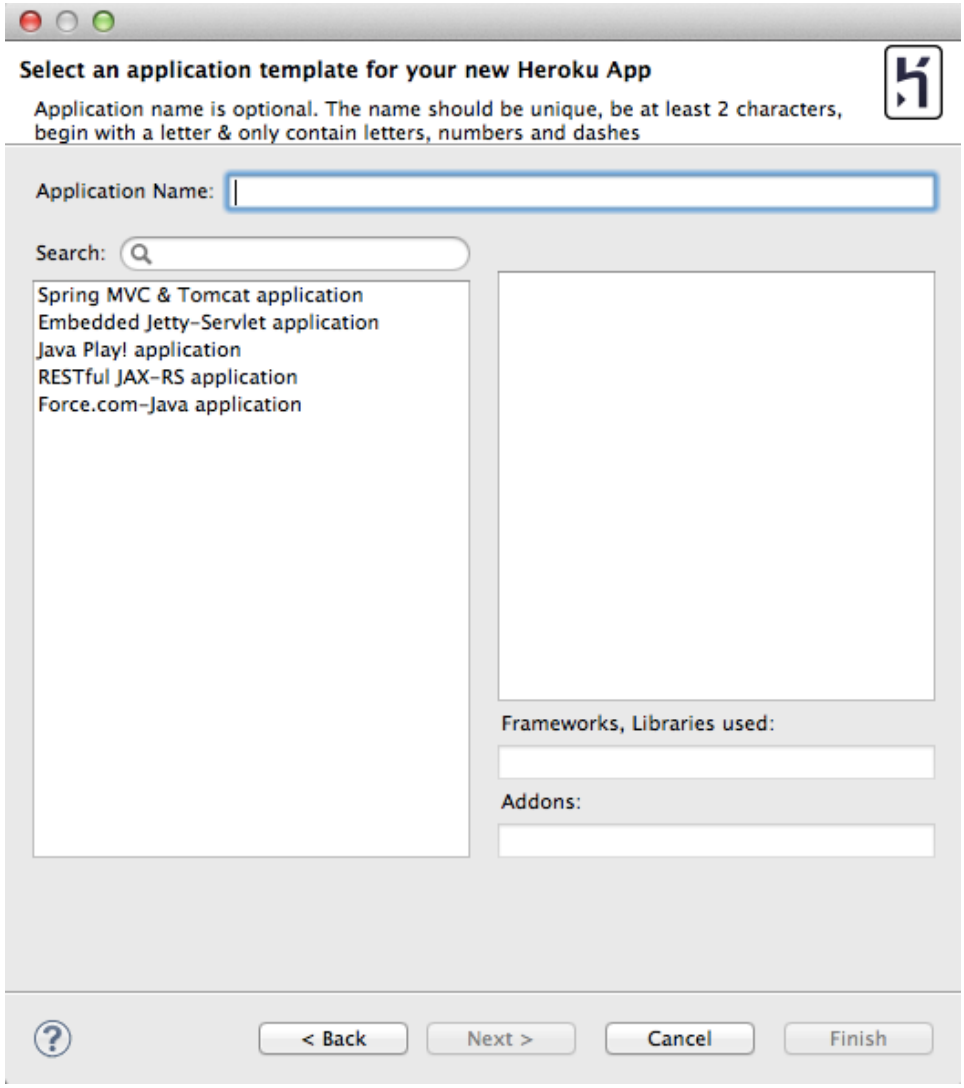
Untuk pertama kali membuat *project* penulis akan menggunakan *template project* yang sudah di sediakan oleh Heroku Plugin Eclipse yang sudah penulis *install* sebelumnya.

1. Buka menu "File" > "New" > "Other" kemudian buka bagian "Heroku"



Gambar 3.11 Membuat *Project* Baru

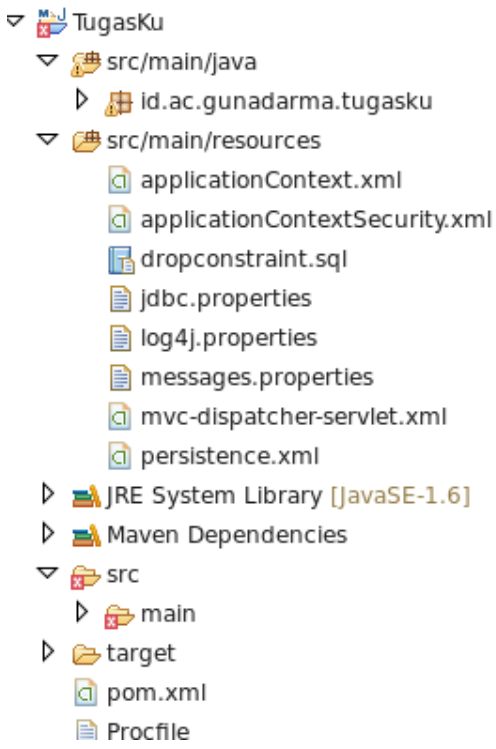
2. Pilih “Create App from Template” kemudian klik “Next”
3. Di dalam nama masukkan “TugasKu”
4. Kemudian di dalam pilihan *template* pilih “Spring MVC & Tomcat Application”



Gambar 3.12 Memilih *Template* Dasar *Project*

5. Klik “Finish” untuk membuat aplikasi dari *template* yang sudah ada.

Ketika selesai maka akan ada project dengan struktur seperti gambar di bawah ini.



Gambar 3.13 Struktur *Project*

3.3.2.2 Implementasi *Data Access Object (DAO) Pattern*

Setelah rancangan konseptual sistem selesai dibuat. Langkah selanjutnya adalah mengimplementasikan rancangan tersebut pada bahasa pemrograman. Bahasa pemrograman yang digunakan adalah Java. Dimana proses peng *coding*-an program menggunakan teknik *Data Access Object (DAO) Pattern* yang merupakan salah satu *design pattern* pada arsitektur Java EE. Ada beberapa tahapan dalam mengimplementasikan *Data Access Object Pattern*. Tahapan – tahapan tersebut adalah :

1. Membuat *class* untuk setiap entitas
2. Membuat *DAO Interface*
3. Membuat *class* implementasi *DAO*

3.3.2.3 Membuat *class* untuk setiap entitas

Entity class merupakan class yang merepresentasikan entitas di dalam database. *Class* ini berfungsi sebagai *setter* dan *getter* di dalam program. Setiap *class* terdiri dari *method – method* yang merepresentasikan *field* di dalam entitas tersebut.

Adapun pada penulisan ilmiah ini penulis hanya menuliskan *script* untuk class entitas Task, sedangkan untuk *class* entitas lainnya bisa dilihat pada lampiran kode program. Berikut adalah *script* untuk *class* entitas Task:

```
@Entity
@Table(name ="task")
publicclass Task extends AbstractBaseEntity {

	privatestaticfinallong serialVersionUID =-
6234938765346643021L;
	private String content;
	private User user;
	private Date deadline;
	privateboolean done;
	privateboolean deleted =false;

	@Column
	public String getContent(){
		return content;
	}

	publicvoid setContent(String content){
	this.content = content;
	}

	@ManyToOne(cascade = CascadeType.ALL)
	public User getUser(){
	return user;
	}

	publicvoid setUser(User user){
	this.user = user;
	}

	@Column
	public Date getDeadline(){
	return deadline;
	}

	publicvoid setDeadline(Date deadline){
	this.deadline = deadline;
	}
```

```
        @Column
        publicboolean isDone(){
        return done;
        }

        publicvoid setDone(boolean done){
        this.done = done;
        }

        @Override
        public String toString(){
        return"Task [content="+ content +", user="+ user +",
deadline="
        + deadline.toString()+", done="+ done +"]";
        }

        @Column
        publicboolean isDeleted(){
        return deleted;
        }

        publicvoid setDeleted(boolean deleted){
        this.deleted = deleted;
        }
}
```

Entitas Task yang terdapat pada *database* memiliki 5 *field*, begitu juga pada saat pembuatan *entity class*. Setiap *class* berisi *method – method* yang sesuai dengan jumlah field. Satu buah field memiliki dua buah *method*, satu *method* sebagai *setter* dan satu *method* sebagai *getter*.

3.3.2.4 Membuat Implementasi *DAO* dan *Interface*

Aplikasi yang penulis buat hanya memiliki logika bisnis berupa melakukan operasi dasar *CRUD (Create, Read, Update, Delete)* pada basis data jadi *service* yang penulis miliki seakan-akan hanya merupakan kepanjangantangan dari *DAO tier* yang telah penulis buat sebelumnya karena memang *DAO tier* yang berurusan langsung dengan basis data bukan *Service tier*. Seperti pada *DAO tier* penulis juga membuat sebuah *interface* dan sebuah kelas implementasi. Berikut adalah contoh *Interface*

```
package id.ac.gunadarma.tugasku.security;

import id.ac.gunadarma.tugasku.model.User;
import id.ac.gunadarma.tugasku.model.Role;
import id.ac.gunadarma.tugasku.model.base.EntityListWrapper;

import java.util.List;

publicinterface UserService {

	publicvoid removeUser(Integer id);

	public List<User> listUser();

	publicvoid addUser(User user);

	publicvoid updateUser(User user);

	public User getUserByID(Integer id);

	public User getUser(String username, String password);

	public User findByName(String username);

	public User updateToken(User user, String generatedToken);

	public User getUserByToken(String accessToken);

	public Role getRoleById(String id);

	public Role getRoleByAuth(String auth);

	public EntityListWrapper<User> all(int max,int page);

}
```

Di bawah ini adalah salah satu contoh Implementasi dari *DAO Service User* yang sudah penulis buat di atas.

```
@Repository
@Transactional(readOnly =true)
publicclass UserServiceImpl implements UserService {
	@PersistenceContext
	private EntityManager entityManager;

	@Transactional(readOnly=false)
	publicvoid addUser(User user){
		try{
			entityManager.persist(user);
		}catch(Exception e){
			e.printStackTrace();
		}
```

```
	}

	public User getUserByID(Integer id){
		Query query = entityManager.createQuery("SELECT u
FROM User u WHERE id = :id", User.class);
		query.setParameter("id", id);
		try{
			return(User) query.getSingleResult();
		}catch(NoResultException e){
			returnnull;
		}
	}
}
```

3.3.2.5 Membuat *API* untuk *security* dan *login*

Karena aplikasi ini mempunyai fitur *multi user* dan dapat di akses oleh beberapa *client* seperti android, maka di sisi *server* penulis membutuhkan sebuah *security handling* yang menangani *user login*. Di dalam pemrograman *web* biasanya penulis hanya perlu membuatuhkan *username* dan *password* untuk mengakses data, tetapi dengan metode *REST API Client - Server* ini penulis menggunakan *Token* untuk *Autorisasi*.

Jadi Langkah pertama yang akan penulis lakukan adalah membuat *Controller* yang menangani *Login user* dengan membawa *token*.

```
@RequestMapping(value ="/login/token", method =
RequestMethod.POST)
public @ResponseBody ResponseEntity<String> generateToken(
		@RequestParam("username") String username,
		@RequestParam("password") String password)
		throws JsonGenerationException,
JsonMappingException, IOException {
	Map<String, String> map =new HashMap<String, String>();
	User user =null;
	try{
		user = userService.getUser(username, password);
	if(user.getToken()==null){
		String generatedToken =
UUID.randomUUID().toString().replace("-","");
		LOG.info("Generated security token: "+
generatedToken);
		user = userService.updateToken(user,
generatedToken);
	}
		LOG.info("Generated security token already exist: "+
user.getToken());
```

```
            map.put("statusCode","00");
            map.put("statusMessage","Success");
            map.put("secretToken", user.getToken());
      }catch(NoResultException e){
            map.put("statusCode","01");
            map.put("statusMessage","Bad credentials");
            returnnew
ResponseEntity<String>(mapper.writeValueAsString(map),
HttpStatus.UNAUTHORIZED);
      }
      returnnew
ResponseEntity<String>(mapper.writeValueAsString(map),
HttpStatus.OK);
}
```

Di dalam kode di atas penulis menerima *parameter* yang di kirim oleh *client username* dan *password* kemudian dengan menggunakan *dao service* yang telah penulis buat penulis melakukan validasi apakah *username* dan *password* yang di kirim *valid*, kalau *valid* maka *controller* di atas akan meng-*generate* token acak yang akan di kembalikan ke *client*. Kemudian dengan *token* ini aplikasi *client* bisa mengakses data yang ada di *server*.

Langkah selanjutnya adalah membuat *security* untuk melakukan validasi *token* yang di bawa oleh *client* untuk melakukan *request data*.

```
@Override
publicboolean preHandle(HttpServletRequest request,
      HttpServletResponse response, Object handler)throws
Exception {
      String accessToken = request.getHeader("Authorization");
      User user =null;
      try{
            user = userService.getUserByToken(accessToken);
      }catch(NoResultException e){
            e.printStackTrace();
      }
      RequestContextHolder.currentRequestAttributes().setAttribu
te("user", user, RequestAttributes.SCOPE_SESSION);
      if(request.getRequestURI().startsWith("/api/")&& user
==null){

      response.sendError(javax.servlet.http.HttpServletResponse.
SC_UNAUTHORIZED);
            returnfalse;
      }
      returntrue;
}
```

Di dalam kode di atas penulis melakukan validasi ketika *client* mengakses *API* penulis dengan *prefix url api* apakah token yang di bawa setiap *request* adalah *valid*, ketika *valid* maka *request* akan di lanjutkan ke *controller*.

3.3.2.6 Membuat *API Controller* untuk Tugasku

Selanjutnya penulis akan membuat sebuah *Controller* dimana semua *request* yang di butuhkan oleh *client* di proses. Berikut contoh salah satu *controller* untuk menambahkan tugas.

```
@RequestMapping(value ="/api/task/add", method =
RequestMethod.POST)
public @ResponseBody ResponseEntity<String>
addTask(@ModelAttribute Task task)throws
JsonGenerationException, JsonMappingException, IOException {
      Map<String, String> map =new HashMap<String, String>();
      try{
            task.setUser(TokenInterceptor.currentUser());
            LOG.info("Persisting Task ["+task.getId()+"]
"+task.toString());
            if(task.getId()!=null){
                  taskService.add(task);
                  Task t = taskService.get(task.getId());
                  if(!"".equals(task.getContent()))
                        t.setContent(task.getContent());
                  if(task.getDeadline()!=null)
                        t.setDeadline(task.getDeadline());
                  taskService.add(t);
            }else{
                  task.setId(null);
                  taskService.add(task);
            }
            map.put("statusCode","00");
            map.put("statusMessage","Success");
            map.put("id", task.getId());
      }catch(Throwable e){
            map.put("statusCode","02");
            map.put("statusMessage","Persist Error");
            map.put("id","");
            LOG.error("E", e);
      }
      returnnew
ResponseEntity<String>(mapper.writeValueAsString(map),
HttpStatus.OK);
}
```

Di dalam kode di atas *client* akan mengakses API dengan method *POST* dan membawa *parameter* yang ada di *object* Task, kemudian melakukan penyimpanan ke *database* dengan menggunakan *DAO Service* yang telah penulis buat. Setelah penyimpanan data sukses penulis akan mengembalikan *response* sukses berupa *JSON*.

3.3.3 Aplikasi Android

Setelah melalui tahap perancangan aplikasi, meng-*install* Eclipse dan membuat *API server* kemudian lakukan langkah-langkah pembuatan aplikasi sesuai dengan rancangan sistem dan rancangan tampilan yang telah dibuat sebelumnya.

Pembuatan aplikasi android ini dimulai dengan membuat Activity *form* utama di *form* ini yang nanti akan menampilkan daftar tugas dan riwayat kemudian terdapat juga menu "Add Task" dan "Setting". Untuk pembuatan program ini pertama merancang tampilan (*layout)* dan audio yang berada di *folder resource* (res), program dan disimpan dalam bentuk (.xml) dan kemudian dilanjutkan membuat program (.java) yang terdapat di dalam *folder source* (src) *package* aplikasi tersebut.

3.3.3.1 Pembuatan Form Utama

Tampilan awal Dari aplikasi ini yaitu *MainActivity* yang menampilkan daftar tugas dan riwayat, pertama penulis buat rancangan tampilan nya di *layout* XML. Kemudian penulis panggil rancangan yang telah penulis buat di XML ke dalam program java. Berikut listing program activity_main.xml

```
<android.support.v4.view.ViewPagerxmlns:android="http://schemas.
android.com/apk/res/android"
android:id="@+id/pager"
android:layout_width="match_parent"
android:layout_height="match_parent"/>
```

Di activity_main.xml ini penulis menggunakan *widget* ViewPager untuk menampilkan 2 tab Task & History. Kemudian berikut untuk kode java yang terhubung dengan activity_main.xml di atas

```
final ActionBar actionBar = getActionBar();
actionBar.setHomeButtonEnabled(false);
actionBar.setNavigationMode(ActionBar.NAVIGATION_MODE_TABS);
```

Inisialisasi *object* ActionBar di atas adalah untuk mengambil *object* dari *actionbar* android yang kemudian penulis konfigurasi sesuai kebutuhan, disini penulis akan men-*disable* home button dan mengatur mode navigasi dengan *Tab*. Selanjutnya untuk menambahkan *Tab* penulis akan menggunakan kode di bawah ini

```
for(int i =0; i < mAppSectionsPagerAdapter.getCount(); i++){
      actionBar.addTab(actionBar.newTab().setText(
      mAppSectionsPagerAdapter.getPageTitle(i)).setTabListener(t
his));
```

Di perulangan di atas penulis akan mengulang 2x karena penulis telah menset jumlah *tab* ada 2 di method

```
@Override
publicint getCount(){
      return 2;
}
```

Untuk penganturan lengkap pembuatan navigasi berbentuk tab ini ada di *class* AppSectionsPagerAdapter yang ada di MainActivity.java

3.3.3.2 Pembuatan Form Input Task

Pada halaman ini penulis akan membuat 2 inputan, inputan untuk task dan tanggal *deadline*, kemudian ada TextView tambahan untuk menampilkan informasi berapa hari lagi sampai *deadline* dari tanggal *deadline* yang penulis inputkan di *field* sebelumnya. Berikut listing program dari task_form_layout.xml

```
<RelativeLayout
      xmlns:android="http://schemas.android.com/apk/res/android"
      xmlns:tools="http://schemas.android.com/tools"
      android:layout_width="fill_parent"
      android:layout_height="fill_parent"
      android:padding="10dp"
      tools:context=".TaskFormFragmentActivity">
      <EditText
            android:id="@+id/editText1"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_alignParentLeft="true"
            android:layout_alignParentRight="true"
            android:layout_alignParentTop="true"
            android:ems="10"
            android:gravity="top|left"
            android:inputType="textMultiLine"
            android:lines="5"
            android:scrollHorizontally="false"

      android:background="@android:drawable/editbox_background_n
ormal">
      </EditText>

      <Button
            android:id="@+id/button1"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_alignLeft="@+id/editText1"
            android:layout_below="@+id/editText1"
            android:layout_marginTop="10dp"
            android:text="Set Deadline"
            android:onClick="showDatePickerDialog"/>

      <TextView
            android:id="@+id/textView1"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_alignBottom="@+id/button1"
            android:layout_alignRight="@+id/editText1"
            android:layout_alignTop="@+id/button1"
            android:layout_toRightOf="@+id/button1"

      android:textAppearance="?android:attr/textAppearanceMedium
"
            android:text=""
            android:gravity="center_vertical"/>

</RelativeLayout>
```

Widget Button di kode di atas di gunakan untuk menampilkan masukan DatePicker. Kemudian selanjutnya penulis akan membuat *class* yang mengontrol tampilan layout task_form_layout.xml ini.

Setelah seperti biasa menghubungkan *widget* yang ada di xml ke *object* yang ada di *class* TaskFormFragmentActivity berikut kode untuk menyimpan data tugas yang penulis masukkan.

```
privatevoid saveTask(){
      TaskSQLiteHelper db =new TaskSQLiteHelper(this);
      db.addTask(new Task(titleEt.getText().toString(),
date,false,false,""));
      Toast.makeText(this,"Success", Toast.LENGTH_LONG).show();
      setResult(RESULT_OK,null);
      finish();
}
```

Sebelum membuat *method* di atas penulis harus membuat class TaskSQLiteHelper yang di dalamnya terdapat fungsi-fungsi untuk kebutuhan *database* mulai dari *insert*, *update*, *delete* dan *query*.

3.3.3.3 Pembuatan Form Pengaturan

Form pengaturan ini berfungsi untuk menyimpan pengaturan aplikasi, di dalam aplikasi ini penulis akan menambahkan beberapa pengaturan yaitu pengaturan waktu untuk menampilkan notifikasi, kemudian warna berdasarkan jumlah hari *deadline* terdekat. Berbeda dengan *layout* yang sebelumnya *Android Framework* telah menyediakan fitur sendiri untuk membuat *preference*, dan tampilan untuk *preference* ini tidak di letakkan di folder *layout* melainkan di folder xml. Berikut listing program dari preference.xml

```
<PreferenceScreenxmlns:android="http://schemas.android.com/apk/r
es/android">
      <PreferenceCategoryandroid:title="REMINDERS">
            <id.ac.gunadarma.tugasku.ui.TimePreference
                  android:defaultValue="08:00"
                  android:key="notif_time"
                  android:summary="Set your desired time for the
reminder"
                  android:title="Notification Time"/>
      </PreferenceCategory>
      <PreferenceCategoryandroid:title="URGENCY">
```

```
            <id.ac.gunadarma.tugasku.ui.NumberPreference
                  android:defaultValue="3"
                  android:key="very_urgent"
                  android:summary="Set urgency days"
                  android:title="Very Urgent (Red)"/>
            <id.ac.gunadarma.tugasku.ui.NumberPreference
                  android:defaultValue="7"
                  android:key="urgent"
                  android:summary="Set urgency days"
                  android:title="Urgent (Yellow)"/>
      </PreferenceCategory>
</PreferenceScreen>
```

Tag yang di gunakan juga berbeda dengan *widget* yang sebelumnya penulis buat di *layout*. *PreferenceScreen* ini sebagai *container* utama, kemudian di dalamnya di bagi menjadi dua kelompok. *TimePreference* ini adalah class custom untuk menampilkan masukan tanggal di preference karena di Android belum neyediakan untuk fitur DatePicker di *preference*, begitu juga dengan NumberPreference ini adalah *class custom* untuk menampilkan *input* jumlah hari. Untuk *preference* ini penulis tidak perlu membuat class *Activity* seperti biasa, penulis hanya membuat *class Fragment* seperti di bawah ini.

```
publicclass PrefsFragment extends PreferenceFragment {
      @Override
      publicvoid onCreate(Bundle savedInstanceState){
            super.onCreate(savedInstanceState);
            addPreferencesFromResource(R.xml.preferences);
      }
}
```

Kemudian untuk memanggil *class fragment* ini penulis menggunakan getFragmentManager().beginTransaction().replace(android.R.id.*content*,**new** PrefsFragment()).commit();

3.3.3.4 Membuat *Database SQLite*

Untuk membuat *database* di android penulis akan menggunakan implementasi dari *class* SQLiteOpenHelper di dalam *class* ini terdapat fungsi-fungsi *Create*, *Read*, *Update* dan *Delete*. Pertama penulis mendefinisikan struktur database yang akan penulis buat.

```
publicstaticfinalint DATABASE_VERSION =1;
publicstaticfinal String DATABASE_NAME ="TaskDB";
publicstaticfinal String TABLE_TASK ="task";
publicstaticfinal String KEY_ID ="id";
publicstaticfinal String KEY_TITLE ="title";
publicstaticfinal String KEY_DEADLINE ="deadline";
publicstaticfinal String KEY_DONE ="done";
publicstaticfinal String KEY_DELETED ="deleted";
publicstaticfinal String KEY_REMOTE_ID ="remote";
```

Di *database* ini penulis akan membuat *table* "Task" dengan kolom id, title, deadline, done, deleted, remote. *Table* ini nanti yang akan penulis gunakan untuk kebutuhan pencocokan data dengan data yang ada di server yang telah penulis buat sebelumnya. Kemudian di *method onCreate* penulis akan mengisikan kode untuk membuat tabel "Task".

```
String CREATE_TASK_TABLE =
      "CREATE TABLE "+TABLE_TASK+" ( "
      +KEY_ID+" INTEGER PRIMARY KEY AUTOINCREMENT, "
      +KEY_TITLE+" TEXT, "
      +KEY_DEADLINE+" INTEGER, "
      +KEY_DONE+" INTEGER,"
      +KEY_DELETED+" INTEGER,"
      +KEY_REMOTE_ID+" TEXT)";
db.execSQL(CREATE_TASK_TABLE);
```

Kemudian selanjutnya penulis akan melengkapi *method-method* untuk kebutuhan ke *database*.

```
publicvoid addTask(Task task){
      Log.d("addTask", task.toString());
      SQLiteDatabase db =this.getWritableDatabase();
      ContentValues values =new ContentValues();
      values.put(KEY_TITLE, task.getTitle());
      Calendar to = Calendar.getInstance();
      to.setTime(new Date(task.getDeadline()));
      values.put(KEY_DEADLINE, to.getTimeInMillis());
      values.put(KEY_DONE, task.isDone()?1:0);
      values.put(KEY_DELETED, task.isDeleted()?1:0);
      values.put(KEY_REMOTE_ID, task.getRemoteId());
      db.insert(TABLE_TASK,null, values);
      db.close();
}
```

Di kode di atas adalah contoh untuk *insert data* ke dalam tabel yang sudah di buat sebelumnya.

3.3.3.5 Membuat *Sync Adapter*

SyncAdapter adalah sebuah fitur yang di sediakan oleh Android yang akan mengatur *background sync* yang artinya mengambil atau mengirim data di proses dalam, biasanya di gunakan untuk mencocokkan data yang ada di android ke *server*. Fitur ini terdaftar di dalam *SyncManager* yang mengatur jalanya sinkronisasi. Berikut adalah keuntungan menggunakan *SyncAdapter*

1. Tampilan, semua *SyncAdapter* di android dapat di akses di menu "Settings", di bawah "Account". Di sini pengguna di berikan beberapa pengaturan seperti mematikan sinkronisasi.
2. Mekanisme *Retry*, sebuah *SyncAdapter* terdapat implementasi untuk mengulang kembali sinkronisasi jiga sinkronisasi yang di lakukan sebelumnya gagal.

Berikut adalah langkah untuk mengimplementasikan *SyncAdapter* di Android.

1. Membuat "Account Authenticator" dan "ContentProvider" di sini terdapat fungsi untuk membuat akun di android dan mengatur manipulasi data.
2. Membuat *class SyncAdapter* disini penulis akan membuat logika untuk sinkronisasi data ke *server*.
3. Membuat *SyncService service* yang menjalankan *SyncAdapter*.

Sebuah *SyncAdapter* akan mengakses data yang terdapat di lokal, yang disediakan oleh *ContentProvider* dan mengakses *server* dengan mengirimkan *token* yang di dapat dari *Authenticator*. Berikut adalah listing program untuk mendapatkan *token* dari *server*, fungsi ini terdapat di *class TaskServerAuthenticate*

```
@Override
public String userSignIn(String username, String password,
String authType)throws Exception {
      DefaultHttpClient httpClient =new DefaultHttpClient();
      String url ="http://tugasku.herokuapp.com/login/token";
      HttpPost httpPost =new HttpPost(url);

      List<NameValuePair> nameValuePairs =new
ArrayList<NameValuePair>();
```

```
      nameValuePairs.add(new BasicNameValuePair("username",
username));
      nameValuePairs.add(new BasicNameValuePair("password",
password));

      httpPost.setEntity(new
UrlEncodedFormEntity(nameValuePairs));

      String authtoken =null;
      try{
            HttpResponse response =
httpClient.execute(httpPost);
            String responseString =
EntityUtils.toString(response.getEntity());
            CredentialsResponse registerResponse =new
Gson().fromJson(responseString, CredentialsResponse.class);
            if(registerResponse.statusCode !=0){
                  thrownew Exception("Error signing-in
["+registerResponse.statusCode+"] - "+
registerResponse.statusMessage);
            }else{
                  authtoken = registerResponse.secretToken;
            }
            Log.d("Task","Response:
"+registerResponse.toString());
      }catch(IOException e){
            e.printStackTrace();
      }
      return authtoken;
}
```

Di baris kode di atas pertama penulis akan *login* ke *server* dengan membawa parameter *username* dan *password* dengan menggunakan HttpPost kemudian mem-*parsing response* yang di terima, jika *login* sukses maka response yang di terima adalah *token* yang kemudian di gunakan untuk mengakses *data* di *server* tanpa harus *login* terlebih dahulu.

Di dalam *content provider* ini terdapat fungsi yang menjembatani antara *SyncAdapter* dengan *database*, karena penulis telah membuat implementasi *Sqliteopenhelper* maka akan lebih mudah membuat *Contentprovider* ini, berikut contoh *method* untuk *insert* data di *Contentprovider* ke *database* melalui *SqliteOpenHelper*

```
@Override
public Uri insert(Uri uri, ContentValues values){
      SQLiteDatabase db = dbHelper.getWritableDatabase();
```

```
        int token = URI_MATCHER.match(uri);
        switch(token){
                case PATH_TOKEN:{
                long id =
db.insert(TaskSQLiteHelper.TABLE_TASK,null, values);
                if(id !=-1)

        getContext().getContentResolver().notifyChange(uri,null);
                return
CONTENT_URI.buildUpon().appendPath(String.valueOf(id)).build();
                }
                default:{
                        thrownew UnsupportedOperationException("URI:
"+ uri +" not supported.");
                }
        }
}
```

Setelah selesai maka penulis akan membuat *class SyncAdapter* yang mengatur *logika* untuk sinkronisasi

Fungsi utama di dalam *class* ini terdapat di *method* onPerformSync *method* ini yang akan di jalankan ketika tiba jadwal untuk sinkronisasi yang telah di jadwalkan oleh sistem.

```
@Override
publicvoid onPerformSync(Account account, Bundle extras, String
authority,
        ContentProviderClient provider, SyncResult syncResult){
        StringBuilder sb =new StringBuilder();
        if(extras !=null){
                for(String key : extras.keySet()){
                        sb.append(key +"["+ extras.get(key)+"] ");
                }
        }
        Log.d(TAG,"onPerformSync for account["+ account.name +"],
Extras: "+ sb.toString());
        try{
                String authToken =
mAccountManager.blockingGetAuthToken(account,
AccountManager.KEY_AUTHTOKEN,true);
                String userObjectId =
mAccountManager.getUserData(account,
AccountGeneral.USERDATA_USER_OBJ_ID);
                TaskSQLiteHelper sqLiteHelper =new
TaskSQLiteHelper(getContext());
                Log.d(TAG,"onPerformSync ["+ authToken +"].
userObjectId: "+ userObjectId);
                List<id.ac.gunadarma.tugasku.helper.dao.Task>
remoteTaskList = Api.getTaskList(authToken);
```

```
            Log.d(TAG,"onPerformSync, Remote Task: "+
remoteTaskList.toString());
            ArrayList<Task> localTaskList =new
ArrayList<Task>();
            Cursor curTvShows =
provider.query(TaskContentProvider.CONTENT_URI,null,null,null,nu
ll);
            if(curTvShows !=null){
                  while(curTvShows.moveToNext()){

      localTaskList.add(Task.fromCursor(curTvShows));
                  }
                  curTvShows.close();
            }
            Log.d(TAG,"onPerformSync, Local Task: "+
localTaskList.toString());
            ArrayList<Task> taskToRemote =new ArrayList<Task>();
            for(Task localTask : localTaskList){
                  if("".equals(localTask.getRemoteId()))
                        taskToRemote.add(localTask);
            }
            Log.d(TAG,"onPerformSync, Task to Remote: "+
taskToRemote.toString());
            ArrayList<Task> taskToLocal =new ArrayList<Task>();
            for(id.ac.gunadarma.tugasku.helper.dao.Task
remoteTask : remoteTaskList){
                  if(!sqLiteHelper.isSynced(remoteTask.id))
                  taskToLocal.add(remoteTask.getTaskLocal());
            }
            Log.d(TAG,"onPerformSync, Task to Local: "+
taskToLocal.toString());
            if(taskToRemote.size()==0){
                  Log.d("Task", TAG +"> No local changes to
update server");
            }else{
                  Log.d("Task", TAG +"> Updating remote server
with local changes");
                  for(Task remoteTask : taskToRemote){
                        Log.d("Task", TAG +"> Local -> Remote
["+ remoteTask.getId()+"]");
                        String id =
Api.postTask(remoteTask.getTaskRemote(), authToken);
                        if(!"".equals(id)){

      sqLiteHelper.markSynced(remoteTask.getId(), id);
                        }
                  }
            }
            if(taskToLocal.size()==0){
                  Log.d("Task", TAG +"> No server changes to
update local database");
            }else{
                  Log.d("Task", TAG +"> Updating local database
with remote changes");
                  int i =0;
                  ContentValues showsToLocalValues[]=new
```

```
ContentValues[taskToLocal.size()];
                  for(Task localTask : taskToLocal){
                        Log.d("Task", TAG +"> Remote -> Local
["+ localTask.getRemoteId()+"]");
                        showsToLocalValues[i++]=
localTask.getContentValues();
                  }

      provider.bulkInsert(TaskContentProvider.CONTENT_URI,
showsToLocalValues);
            }
            Log.d("Task", TAG +"> Finished.");

      }catch(OperationCanceledException e){
            e.printStackTrace();
      }catch(IOException e){
            syncResult.stats.numIoExceptions++;
            e.printStackTrace();
      }catch(AuthenticatorException e){
            syncResult.stats.numAuthExceptions++;
            e.printStackTrace();
      }catch(Exception e){
            e.printStackTrace();
      }
}
```

Di dalam method di atas pertama penulis mengambil *token* yang sudah penulis dapatkan pada proses *login* sebelumnya kemudian mengambil *data* dari *server* dan dari lokal. Kemudian mencocokan keduanya *data* mana yang harus di *upload* ke *server* atau sebaliknya data mana yang harus di *insert* ke lokal *database* agar data bisa sama antara *server* dan *lokal*.

Setelah selesai membuat *SyncAdapter* selanjutnya penulis akan medaftarkan *SyncService* agar *SyncAdapter* yang telah penulis buat di kenali oleh sistem.

```
<providerandroid:authorities="id.ac.gunadarma.tugasku.provider"
      android:name=".db.TaskContentProvider"
      android:label="@string/provider_name"
      android:exported="false"/>

<serviceandroid:name=".sync.TaskSyncService"android:exported="tr
ue">
      <intent-filter>
            <actionandroid:name="android.content.SyncAdapter"/>
      </intent-filter>
      <meta-
dataandroid:name="android.content.SyncAdapter"android:resource="
```

```
@xml/sync_adapter"/>
</service>

<serviceandroid:name=".account.TaskAuthenticatorService">
      <intent-filter>

      <actionandroid:name="android.accounts.AccountAuthenticator
"/>
      </intent-filter>
      <meta-
dataandroid:name="android.accounts.AccountAuthenticator"android:
resource="@xml/authenticator"/>
</service>
```

Kode *xml* di atas adah konfigurasi agar *Service* yang telah penulis buat di kenali oleh android yang kemudian akan di jalankan otomatis.

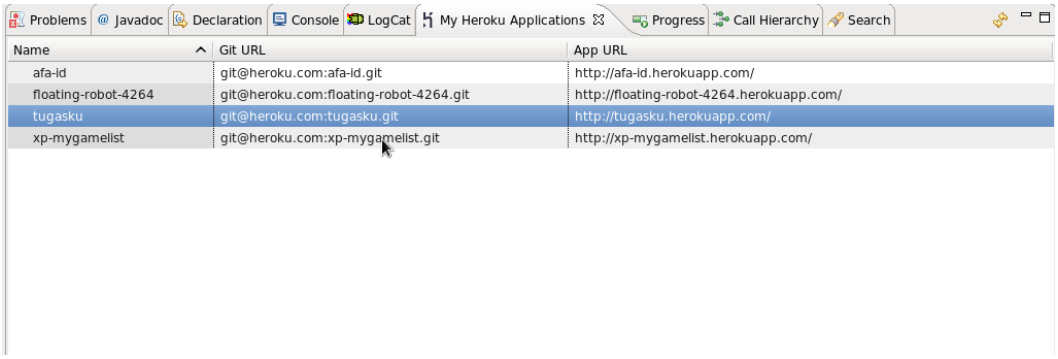
3.4 *Upload* Aplikasi

Heroku adalah layanan *cloud*, tepatnya *Platform as a service (PaaS)*. Heroku membuat penulis dapat fokus hanya untuk pembuatan aplikasi tanpa mengkhawatirkan masalah *server* seperti meng-*install* dan *maintenance* yang semuanya sudah diatur oleh Heroku. Selain itu, penulis dapat membuat akun gratis di Heroku. Sebuah *free account* di Heroku dapat *hosting* maksimal 5 aplikasi.

Di samping itu, Heroku mempunyai konsep *Add-ons,* dimana penulis dapat menambah berbagai macam teknologi tambahan seperti PostgreSQL, MongoDB, MySQL, Redis, Hadoop, dan masih banyak lagi, ke dalam aplikasi penulis yang Sedang berjalan secara *on the fly*. Bahkan beberapa *add-ons* tersebut ada yang gratis. Heroku juga mempunyai Heroku Toolbelt, yaitu semacam aplikasi *console (CLI)* untuk konfigurasi aplikasi yang penulis di Heroku. Penulis dapat menggunakan Heroku Toolbelt untuk *login*, membuat aplikasi, meng-*upload*, memasang *addons*, dll.

Ketika penulis membuat *project* aplikasi *web* mengunakan Heroku *Plugin* di eclipse, maka akan otomatis aplikasi penulis akan di buatkan git *repository* di lokal dan di *server* heroku. *Project* ini bisa penulis lihat di folder

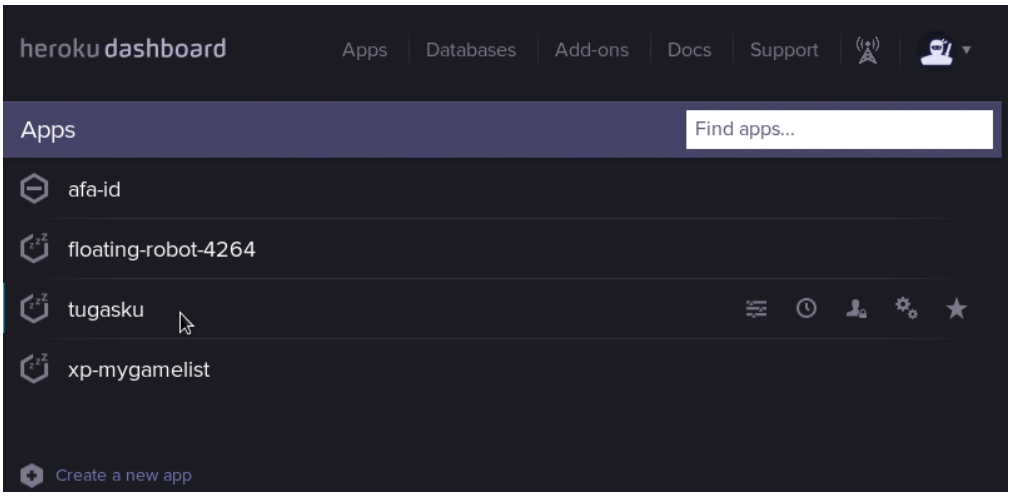
~/git/heroku/{nama-project-penulis}. Kemudian untuk melihat aplikasi apa saja yang sudah penulis buat penulis dapat melihat di eclipse bagian “My Heroku Applications”



Gambar 3.14 Daftar Heroku Project

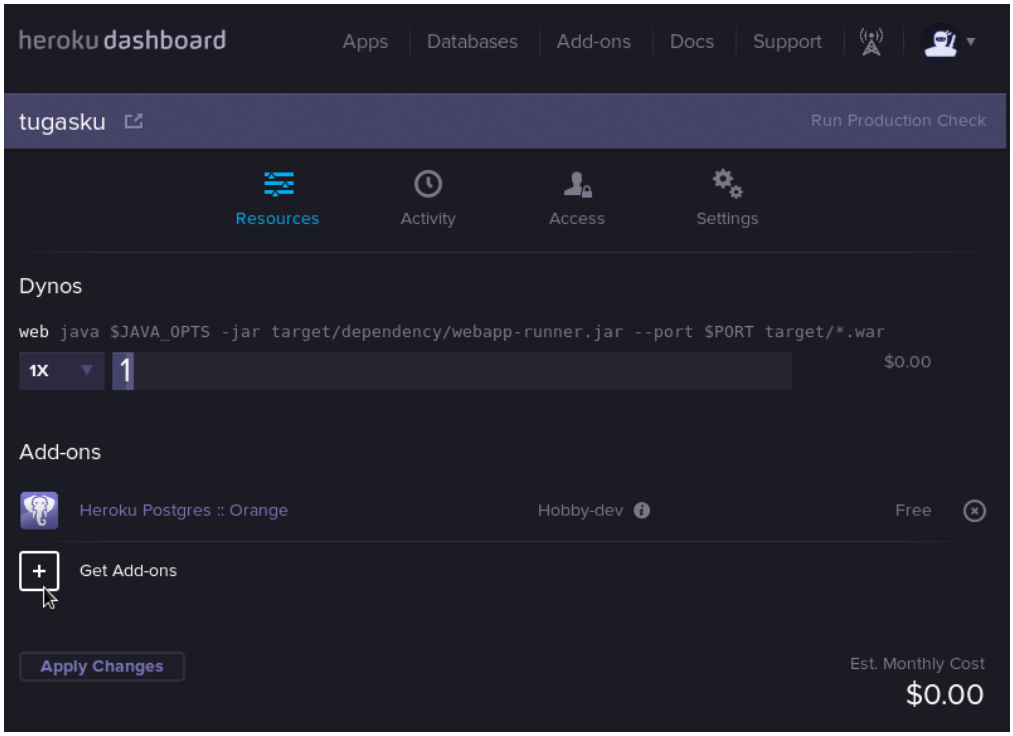
Pertama sebelum penulis meng-*upload* aplikasi ke Heroku *Cloud* penulis harus melakukan berberapa konfigurasi terlebih dahulu, konfigurasi pertama adalah penulis akan menambahkan *plugin database* PostgreSQL ke dalam aplikasi penulis. Berikut langkah-langkahnya

1. *Login* ke halaman dashboard heroku.
2. Pilih aplikasi yang akan penulis tambahkan *database* PostgreSQL



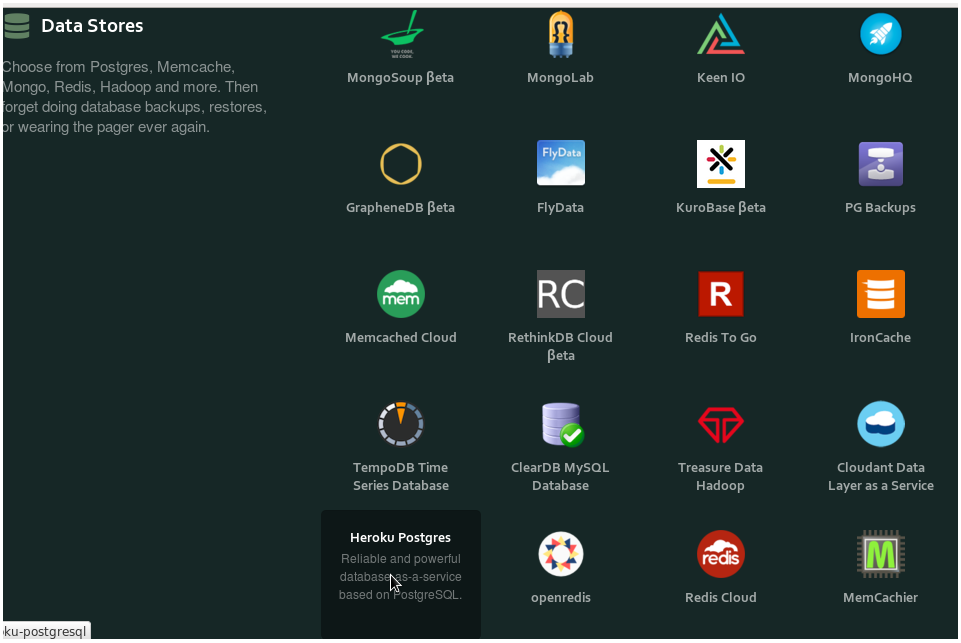
Gambar 3.15 Daftar Heroku *Project* di Heroku *Server*

3. Di Bagian konfigurasi aplikasi pilih menu “Get Add-Ons”



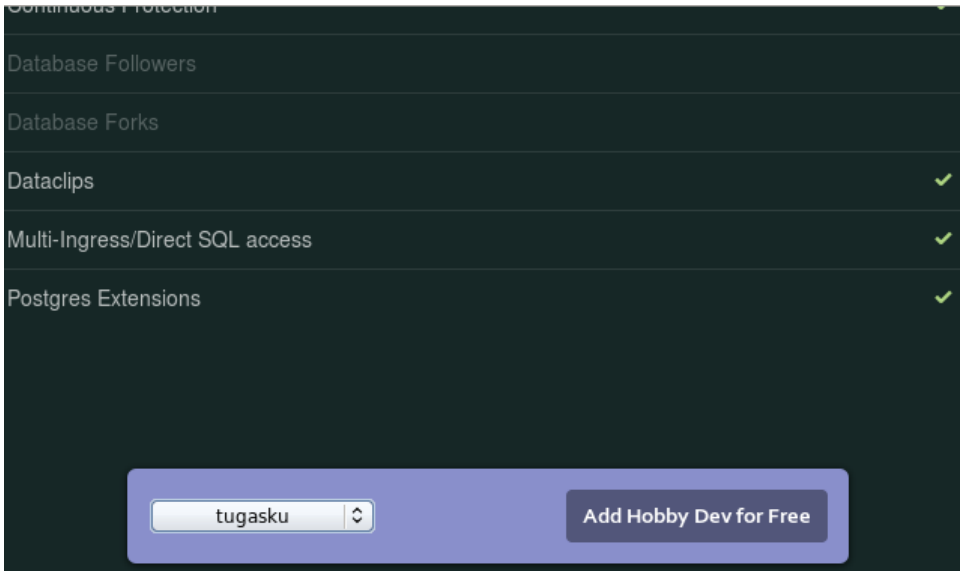
Gambar 3.16 Menu Menambahkan *Add On*

4. di halaman ini penulis dapat memilih *Add-ons* sesuai yang penulis butuhkan. Karena aplikasi penulis akan menggunakan *database* postgresql maka di sini penulis akan memilih *add-ons* Heroku Postgres.



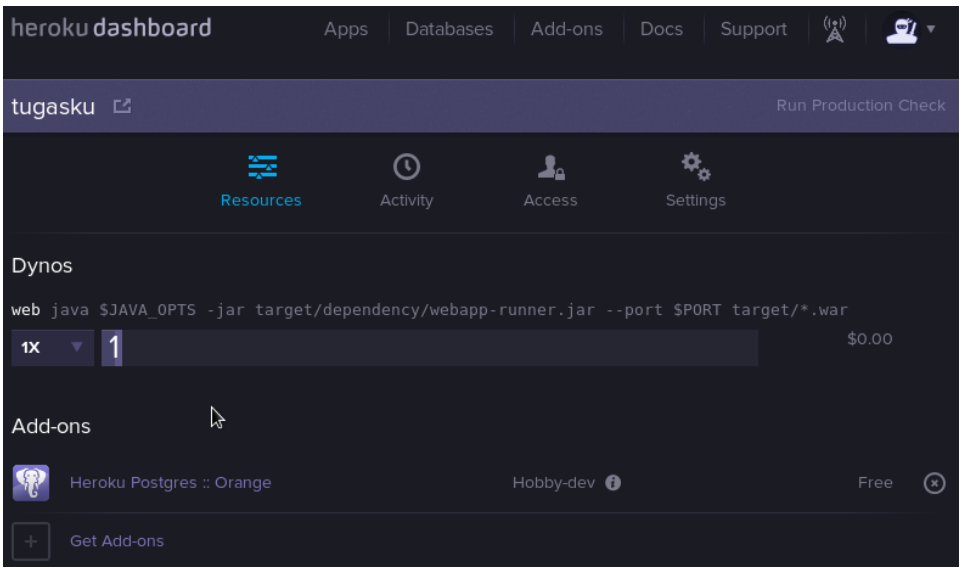
Gambar 3.17 Daftar *Add On*

5. Setelah itu penulis akan masuk ke halaman "pricing plans", di sini penulis bisa memilih paket sesuai kebutuhan. Untuk penulisan ini penulis akan menggunakan plan "Free" atau gratis, kemudian di bawah ada pilihan aplikasi mana yang akan penulis tambahkan *Add-on* ini. Setelah selesai klik "Add Hobby Dev for Free".



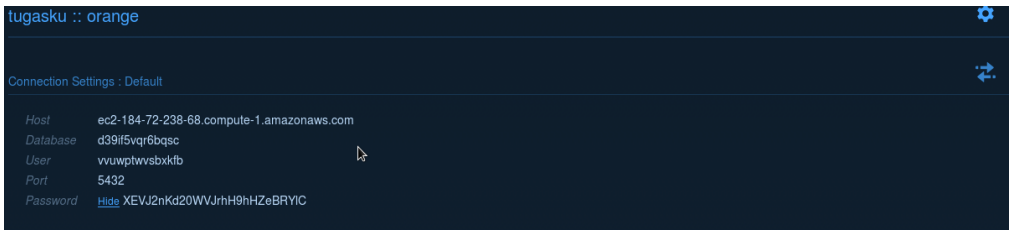
Gambar 3.18 Memasang *Database*

6. Setelah sukses menambahkan *add-on* maka secara otomatis Heroku akan membuatkan *database* di *server* Heroku, untuk melihat lebih detail informasi *database* penulis bisa mengakses ke *dashboard* di bagian *add-on* klik "Heroku Postgres"



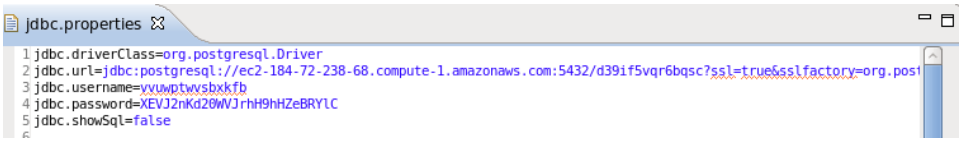
Gambar 3.19 Menu Membuka Informasi Database

7. di halaman ini adalah letak di mana konfigurasi untuk *database* penulis. selanjutnya penulis akan memasang konfigurasi ini ke aplikasi penulis



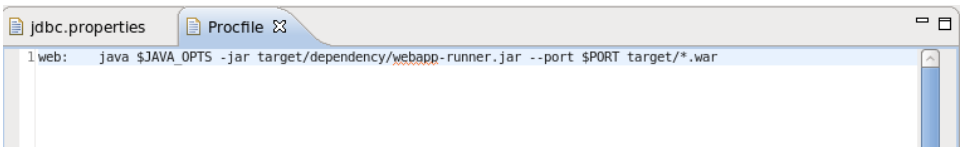
Gambar 3.20 Detail Database

8. di file konfigurasi *jdbc.properties* di project penulis masukkan konfigurasi *database* seperti di step sebelumnya



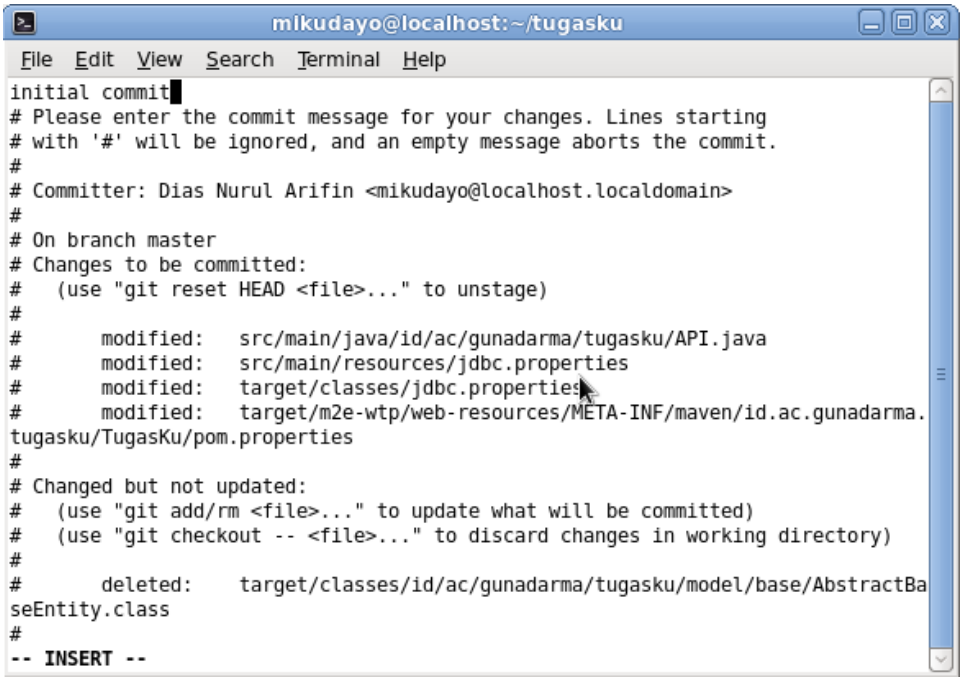
Gambar 3.21 Konfigurasi *Database* di *Project*

9. setelah konfigurasi *database* selesai selanjutnya penulis akan mengenalkan kepada Heroku aplikasi yang telah penulis buat adalah aplikasi *web* dan *file* mana yang akan di *upload* di *file Procfile*



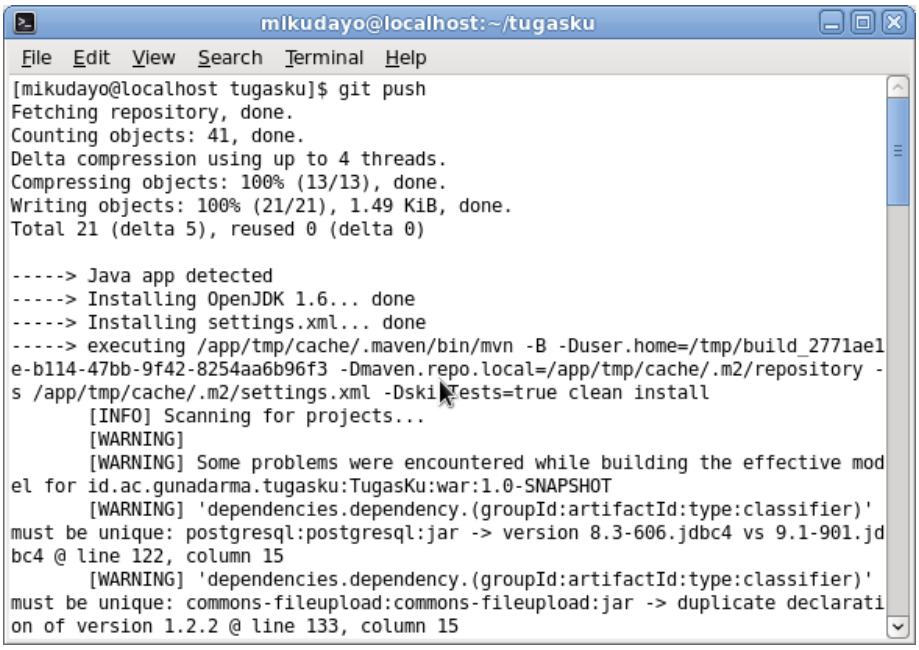
Gambar 3.22 Konfigurasi *Procfile*

10. Selanjutnya penulis akan meng-*upload* dan menjalankan aplikasi. Masuk ke *folder* aplikasi penulis dan *commit* dengan perintah "git commit".

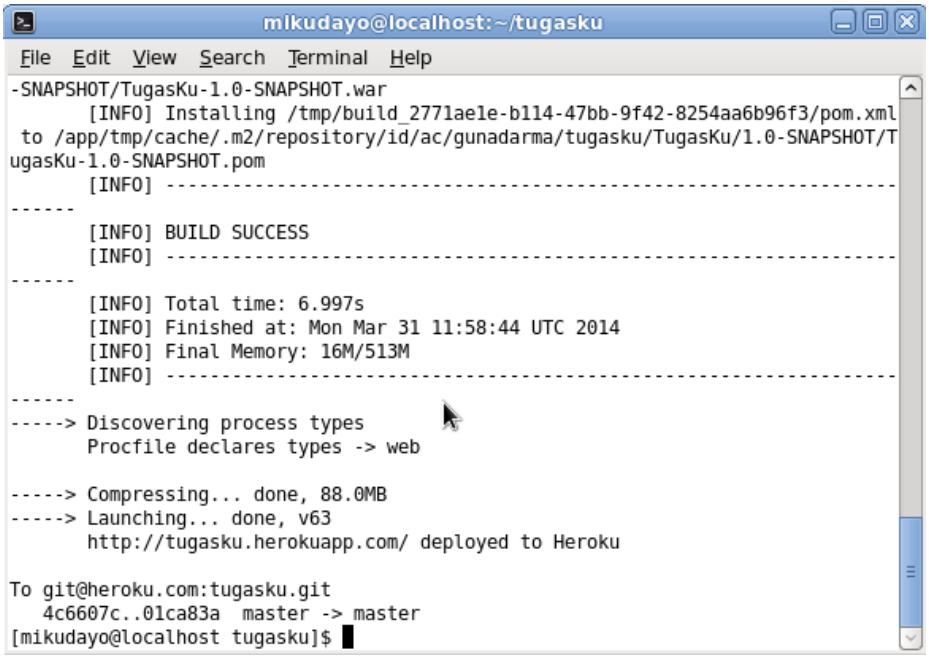


Gambar 3.23 *Commit Project*

11. Setelah semua listing program di *commit* ke *repository* lokal maka selanjutnya penulis akan melakukan *push* ke *repository* yang ada di heroku *server* dengan perintah "git push". Setelah sukses *push* ke *reposiory server* maka heroku akan otomatis men-*compile* dan meng-*upload* aplikasi penulis.



Gambar 3.24 Proses *Compile* aplikasi



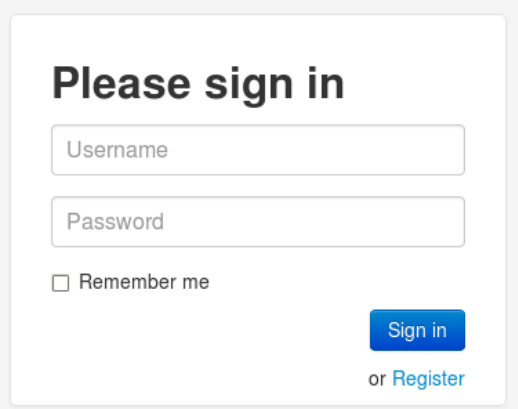
Gambar 3.25 Proses *Upload* aplikasi

3.5 Demo Aplikasi

Setelah pembuatan program aplikasi maka dilakukan uji coba dengan menggunakan ponsel berbasis Android dengan versi minimal Android 4.0 (Ice Cream Sandwich). Untuk melakukan uji coba ini penulis menggunakan sistem operasi berbasis android 4.3 (Jelly Bean). Spesifikasi ponsel yang digunakan sebagai berikut :

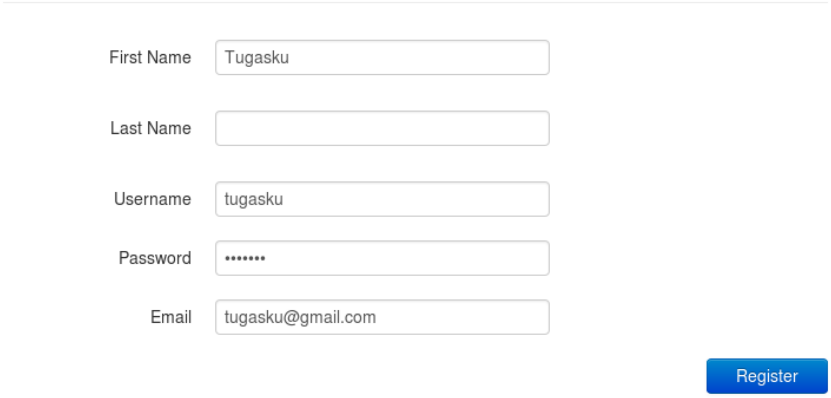
Layar	- TFT capacitive touchscreen, 16M colors - 1080 x 1920 pixels, 5.0 inches (~441 ppi pixel density)
Memory	- 16 GB, 2 GB RAM
Fitur	- Sistem Operasi 4.3 (Jelly Bean) - Prosessor Quad-core 1.5 GHz Krait - Sensor Accelerometer, gyro, proximity, compass

Untuk menggunakan aplikasi Tugasku ini langkah pertama adalah daftar akun di *server* yang sudah penulis *upload* di Heroku. Di dalam penulisan ini *URL* aplikasi yang sudah di *upload* adalah http://tugasku.herokuapp.com setelelah mengakses *url* tersebut maka akan tampil halaman *login*.



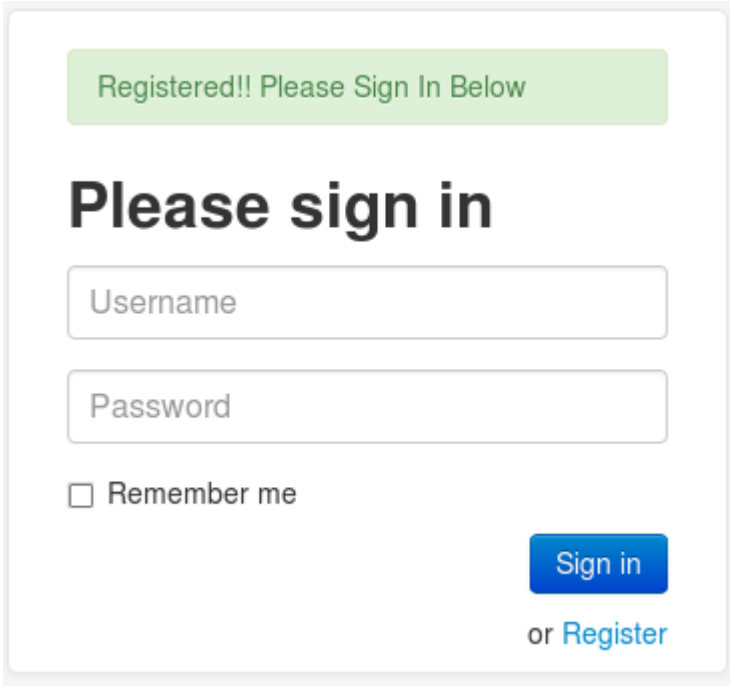
Gambar 3.26 Halaman *Login* Aplikasi *Web*

Untuk memperoleh akun klik link “Register”.

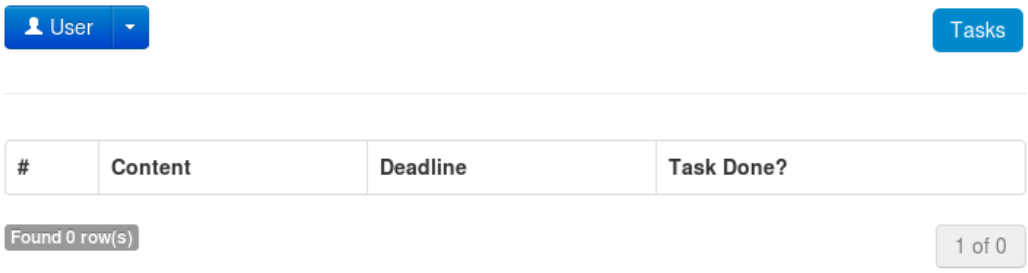


Gambar 3.27 Halaman *Register* Aplikasi *Web*

Setelah selesai mengisi data akun, klik tombol “Register”



Gambar 3.28 Halaman Sukses Registrasi

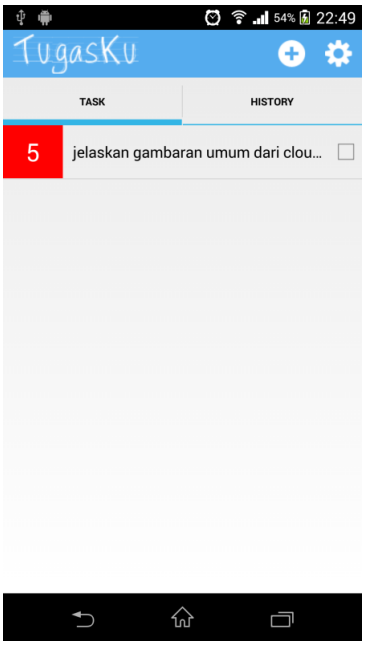


Gambar 3.29 Halaman *Dashboard* Aplikasi Web

Setelah berhasil register akun, selanjutnya penulis bisa langsung *login* dengan akun yang telah penulis buat

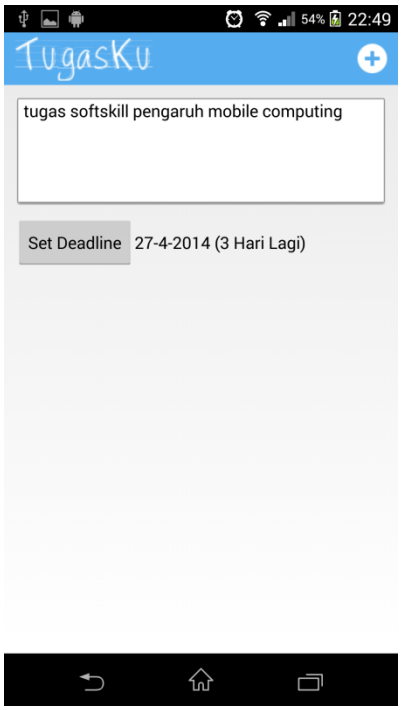
Ketika penulis login ke dalam aplikasi penulis akan di di arahkan ke halaman *index* dimana halaman ini yang nantinya akan menampilkan tugas yang telah penulis masukkan di aplikasi android. Untuk selanjutnya penulis akan menjalankan aplikasi di android.

Ketika penulis membuka aplikasi android Tugasku ini akan langsung menampilkan data tugas penulis, dan data riwayat dari tugas yang sudah selesai.



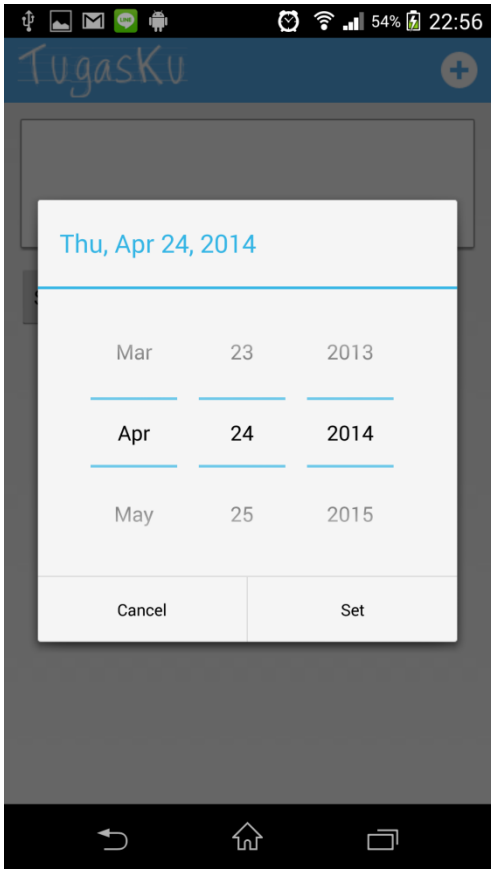
Gambar 3.30 Halaman *Dashboard* Aplikasi Android

Untuk menambahkan tugas penulis bisa mengakses dengan menu tambah di pojok kiri atas.



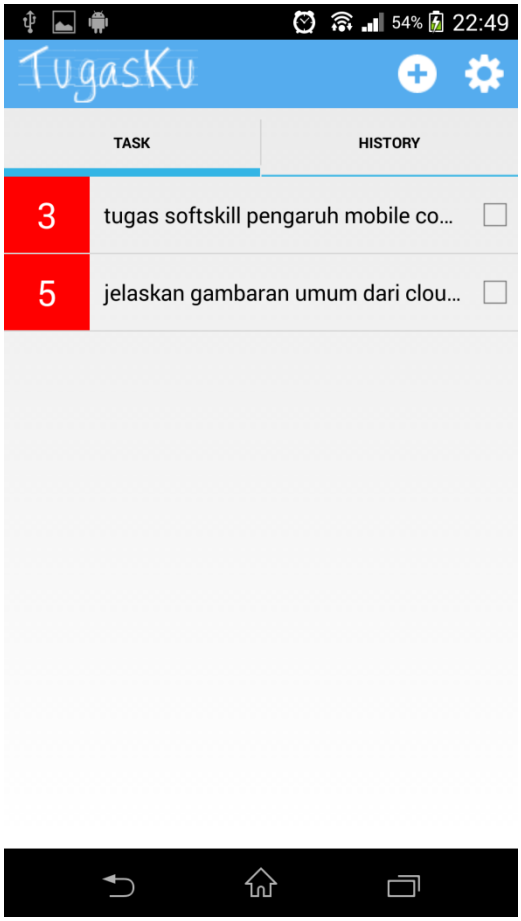
Gambar 3.31 Halaman *Form* Tambah Tugas Aplikasi Android

Setelah tampil form isi data, penulis bisa mengatur *deadline* dengan tombol “Set Deadline”



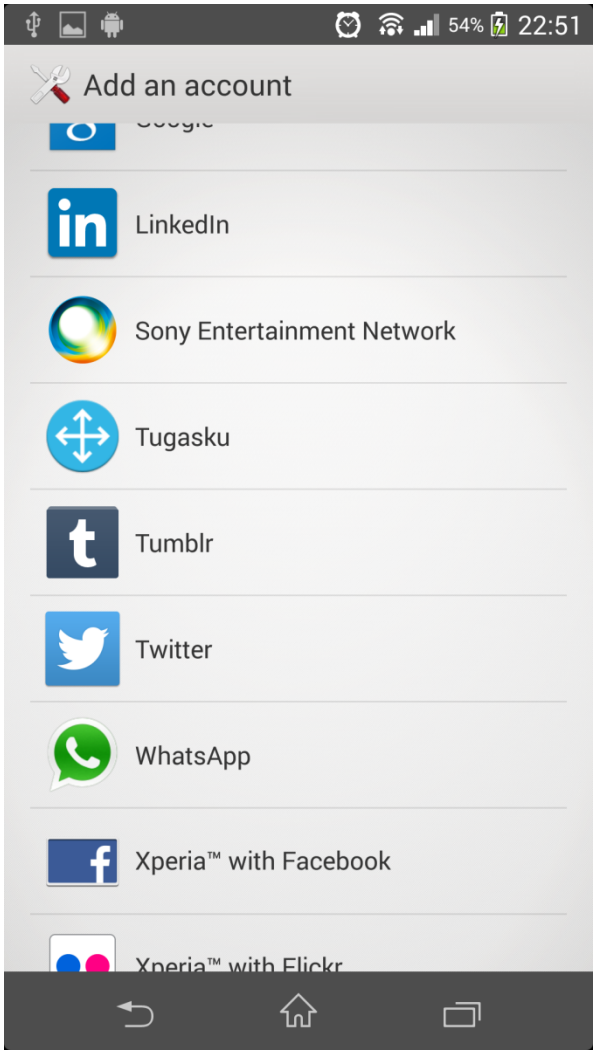
Gambar 3.32 Memasukkan tanggal *deadline*

Kemudian pilih pilih menu “save” di pojok kiri atas. Setelah selesai maka penulis akan di arahkan ke halaman utama dengan tambahan data yang baru di masukkan.



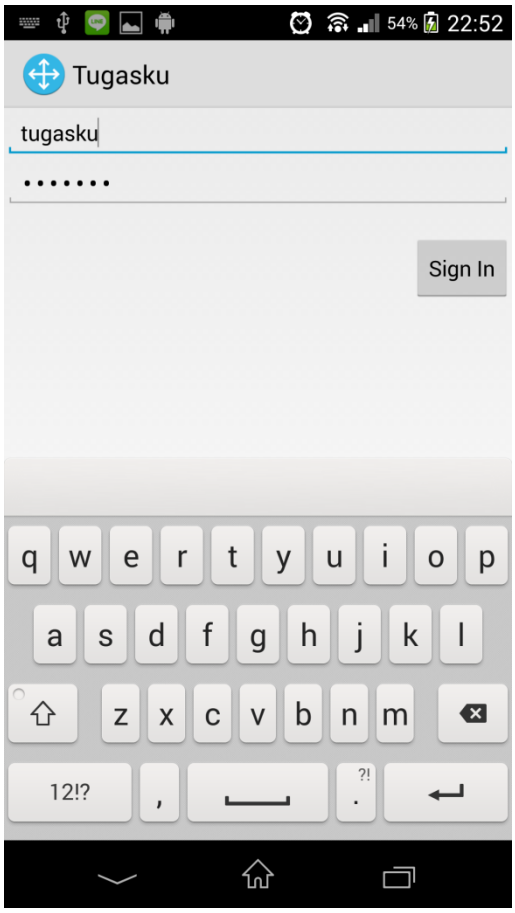
Gambar 3.33 Tugas Berhasil di Tambahkan

Untuk mengaktifkan sinkronisasi data ke *server cloud* yang sebelumnya sudah di *upload*. Di android penulis perlu *login* dengan akun yang sudah teregister. Di handphone Android masuk ke menu “Settings” > “Account”



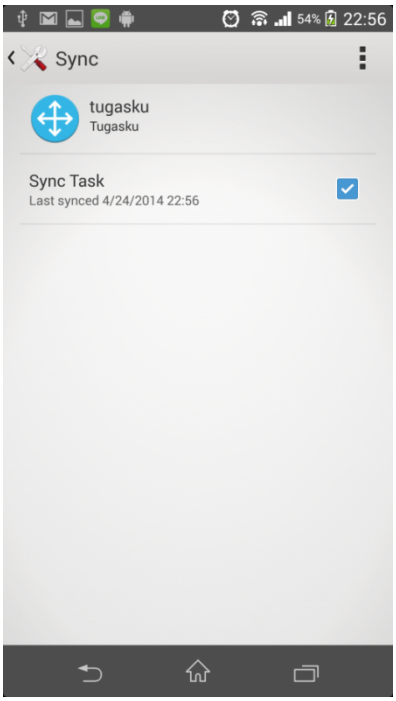
Gambar 3.34 Daftar Akun Sinkronisasi di Android

Kemudian pilih account “Tugasku”



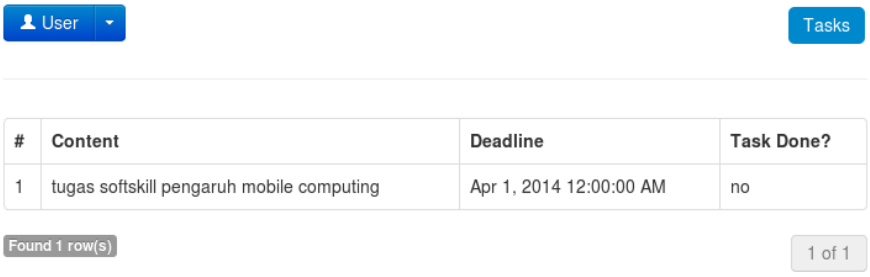
Gambar 3.35 Login Akun Sinkronisasi

Setalah itu penulis akan di minta *login* dengan akun yang sudah terdaftar. Ketika *login* telah berhasil maka data yang telah penulis isi di lokal akan otomatis tersinkronisasi dengan *data* di *server*.



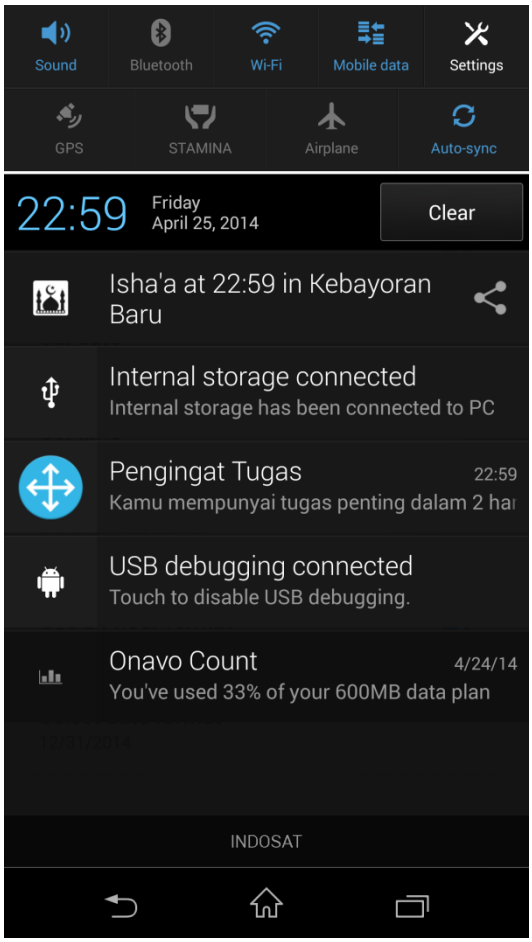
Gambar 3.36 Sinkronisasi Berhasil

Setelah status sudah selesai tersinkronisasi penulis bisa cek di *server* dengan masuk ke aplikasi Tugasku versi *web*.



Gambar 3.37 Data tugas berhasil di sinkronisasi ke *Server*

Di aplikasi Tugasku ini terdapat notifikasi dimana setiap waktu yang di kondigurasi di menu pengaturan akan menampilkan notifikasi setiap hari untuk mengingatkan pengguna.



Gambar 3.38 Notifikasi Setiap Hari

3.5.1 *System Internal Testing*

System Internal Testing bertujuan untuk memastikan bahwa aplikasi Tugasku berjalan dengan baik atau tidak dan sesuai rancangan yang ada. Selain itu hasil yang didapat menjadi bahan evaluasi sebelum pemakai akhir melakukan test terhadap aplikasi Tugasku. Ada beberapa hal yang diamati dalam proses uji coba ini :

Tabel 3.3 Hasil Internal Testing

No	Uji Coba	Keterangan	Hasil
1	Struktur navigasi	Struktur navigasi sesuai dengan struktur yang dirancang.	OK
2	Tampilan desain menu utama	Desain sesuai dengan rancangan menu utama yang ada	OK
3	Aplikasi versi *Web*	Aplikasi *server* berjalan baik dan dapat di akses melalui *internet*	OK
4	Registrasi akun	Dapat register di aplikasi versi *web*	OK
5	*Login* di aplikasi versi *Web*	Dapat *login* dan masuk ke dalam *dashboard* di aplikasi versi *web*	OK
6	Menu tambah tugas	Menu menambahkan tugas berjalan dengan baik	OK
7	Tab Menu Task	Halaman depan menampilkan daftar tugas yang sudah di isikan	OK
8	Tab Menu History	Tab history dapat menampilkan riwayat tugas	OK
9	Fitur Sinkronisasi	Sinkronisasi berjalan dengan baik, data dapat di kirim ke server dan sebaliknya	OK
10	Fitur Notifikasi	Notifikasi setiap hari berjalan dengan baik	OK

3.5.2 *User Acceptance Test*

Penulis melakukan pengujian kepada pengguna akhir *(end user)* untuk mendapatkan informasi mengenai aplikasi Tugasku. Jenis pengujian ini memberikan pengguna akhir keyakinan bahwa aplikasi yang disampaikan kepada mereka memenuhi persyaratan mereka. Untuk itu penulis memberikan tiga pertanyaan yang berkaitan dengan desain, kemudahan dan manfaat berbentuk kuesioner kepada 10 respoden mahasiswa serta meminta penilaiannya. Penilaian kuesioner dengan menggunakan Skala *Likert* yang mempunyai bobot nilai pada masing-masing jawaban pertanyaan. Bobot nilai pada masing-masing pertanyaan sebagai berikut :

Tabel 3.4 Tabel Skor

Skala Likert	Bobot Nilai
Sangat Setuju	4
Setuju	3
Kurang Setuju	2
Tidak Setuju	1

Cara menghitung skor dan presentase penggolongan skor penilaian adalah sebagai berikut :

a. Cara Menghitung Skor

Skor = frekwensi x bobot nilai

Jumlah skor = jumlah skor skala penilaian 1 sampai dengan 4

b. Cara penghitungan presentase penggolongan skor penilaian

Penggolongan skor penilaian dilakukan berdasarkan skor ideal, dimana repondennya berjumlah 10 orang, maka :

Skor Ideal (skor tertinggi) = 10 x bobot nilai tertinggi

Skor terendah = 10 x bobot nilai terendah

Sehingga persentase penggolongan skor penilaian adalah :

$$\frac{\text{Jumlah Skor}}{\textit{SkorIdeal}} \times 100\%$$

Sedangkan kriteria interpretasi skor berdasarkan persentase kelompok responden :

76% – 100% = sangat baik/sangat menarik/sangat setuju/ sangat efektif

51% – 75% = baik/menarik/setuju/efektif

26% – 50% = kurang baik/kurang menarik/kurang setuju/kurang efektif

0% – 25% = tidak baik/tidak menarik/tidak setuju/efektif

Berikut ini merupakan tabel hasil kuesioner yang telah diisi oleh 10 responden :

Tabel 3.5 Hasil Menghitung Skor

No	Pertanyaan	4	3	2	1	Persentase
1	Bagaimana desain dari aplikasi Tugasku?	7	1	2	0	87.5%
2	Apakah aplikasi Tugasku Mudah di gunakan?	9	1	0	0	97.5%
3	Apakah aplikasi Tugasku dapat membantu anda untuk mengingat tugas sehari-hari anda?	9	1	0	0	97.5%
4	Apakah Fitur notifikasi berguna bagi anda?	8	1	1	0	92.5%
5	Apakah fitur upload di cloud berguna bagi anda?	5	2	0	3	72.5%
6	Apakah tampilan daftar tugas yang ada di halaman utama sudah informatif ?	8	2	0	0	95%