Rémi Pourtier
Jean-Philippe Kha

Adaptation des interfaces à l'environnement

Application: Find Your Movie

Technologies: Android & React

Plan

I-Introduction

Pour ce projet, nous avons décidé de développer une application destinée aux utilisateurs qui ne savent pas quel film regarder

Cette application possède des fonctionnalités simples. La page principale contient un film avec la possibilité de sauvegarder le film que l'on a envie de voir ou de le passer. En appuyant sur l'image du film, l'utilisateur a la possibilité de regarder la bande d'annonce, de lire le synopsis, les commentaires positifs et négatifs. Une fois que l'utilisateur a fait son choix, un autre film s'affiche. Il peut consulter quand il le souhaite la liste des films sélectionnés.

Nous avons choisi de confronter deux types d'applications: les applications web et natives. Le but est de comparer les performances de deux langages en terme d'adaptabilité. Pour le développement web, nous avons opté pour React et pour le coté natif, nous avons choisi Android.

Dans ce rapport nous allons vous proposez un tutorial pour ces deux technologies. Nous partageons l'expérience acquise lors de ces deux développement sous la forme d'un tutorial afin d'aider les personnes qui souhaitent utiliser l'une de ces technologies.

Dans la première partie nous allons voir la partie Android. La seconde présentera le développement avec React. En conclusion, nous ferons un bilan des expériences de développement.

II-Développement avec Android

1) Comment l'application a-t-elle été réalisée ?

La réalisation de l'application peut se découper en 4 partie, nous allons tout d'abord voir comment nous créons un menu coulissant de côté("Navigation Drawer"/"Sliding Menu") ensuite nous expliquerons comment nous faisons des onglets coulissant("Sliding Tabs") puis nous verrons comment nous définissons les composants d'un écran afin que celui ci s'adapte à taille de l'écran.
Pour commencer, il faut créer un nouveau projet sous Android Studio et créer un "Blank Activity"

1. Menu coulissant

1.1. Création du thème de notre application

colorPrimaryDark correspond à la couleur qui sera appliqué à la bar de notification
colorPrimary correspond à la couleur qui sera appliqué à la bar d'action
textColorPrimary correspond à la couleur du texte, qui sera appliqué à la bar d'action
windowBackground correspond à la couleur de l'arrière plan par défaut de l'application
navigationBarColor correspond à la couleur de la bar navigation situé au pied de l'application.

Définissons dans un premier temps les couleurs qu'on utilisera, pour cela ouvrez le fichier colors.xml qui se trouve dans res/value

```xml
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <color name="colorPrimary">#4e4e4e</color>
    <color name="colorPrimaryDark">#4e4e4e</color>
    <color name="textColorPrimary">#FFFFFF</color>
    <color name="windowBackground">#FFFFFF</color>
    <color name="navigationBarColor">#000000</color>
    <color name="colorAccent">#ffffff</color>
    <color name="colorSecondary">#BFBFBF</color>
</resources>
```

Ouvrir le fichier style.xml qui se trouve dans le dossier res/values et définissez votre nouveau thème :

```xml
<resources>

    <!-- Base application theme. -->
    <style name="AppTheme" parent="Theme.AppCompat.Light.DarkActionBar">
        <!-- Customize your theme here. -->
    </style>

    <style name="MyMaterialTheme" parent="Theme.AppCompat.Light.DarkActionBar">
        <item name="windowNoTitle">true</item>
        <item name="windowActionBar">false</item>
        <item name="colorPrimary">@color/colorPrimary</item>
        <item name="colorPrimaryDark">@color/colorPrimaryDark</item>
        <item name="colorAccent">@color/colorAccent</item>
        <item name="android:windowActionBarOverlay">true</item>
    </style>

</resources>
```

Ouvrir le fichier dimens.xml situé dans res/values dimens.xml

```xml
<resources>
    <!-- Default screen margins, per the Android Design guidelines. -->
    <dimen name="activity_horizontal_margin">16dp</dimen>
    <dimen name="activity_vertical_margin">16dp</dimen>
    <dimen name="nav_drawer_width">260dp</dimen>
```

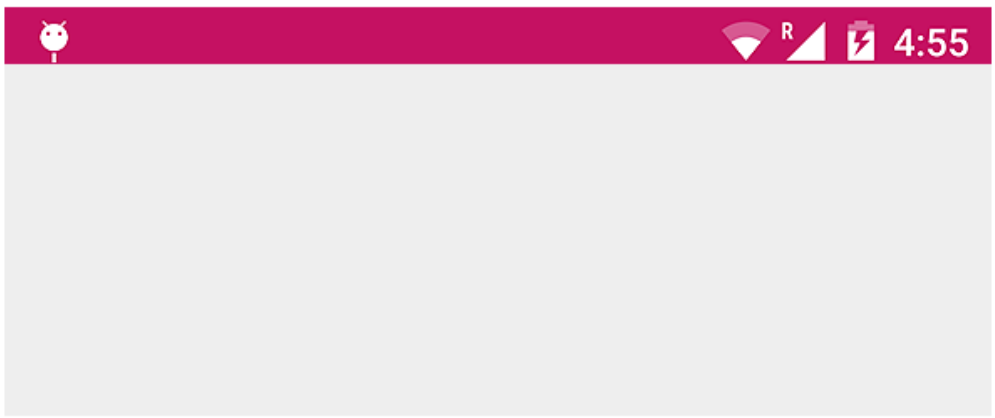
Dans l'AndroidManifest.xml, changez la valeur du thème :

```
android:theme="@style/MyMaterialTheme" >
```

Puis mettez ces valeurs dans le fichier string.xml qui se situe dans le dossier res/values/

```
<resources>
    <string name="app_name">Movie Review App</string>
    <string name="action_settings">Settings</string>
    <string name="action_search">Search</string>
    <string name="drawer_open">Open</string>
    <string name="drawer_close">Close</string>

    <string name="nav_item_home"> Home</string>
    <string name="nav_item_friends">Friends</string>
    <string name="nav_item_list_movie">Films selectionnés</string>

    <!-- navigation drawer item labels  -->
    <string-array name="nav_drawer_labels">
        <item>@string/nav_item_home</item>
        <!--<item>@string/nav_item_friends</item>-->
        <item>@string/nav_item_list_movie</item>
    </string-array>

    <string name="title_list_movie">Films selectionnés</string>
    <string name="title_friends">Friends</string>
    <string name="title_home">Home</string>
    <string name="title_activity_movie_details">MovieDetailsActivity</string>
```

Si vous lancez l'application vous devriez obtenir quelques choses comme çan en fonction de votre couleur que vous aurez défini auparavant :



1.2. Ajout de la bar d'action

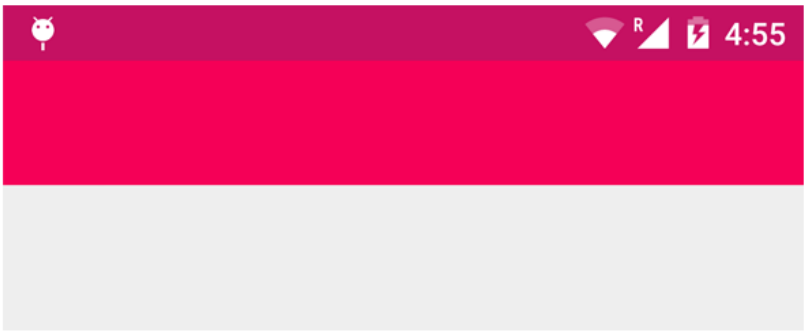
Créez un fichier toolbar.xml dans le répertoire res/layout et ajoutez le code ci-dessous, il permet de définir notre action bar :

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.v7.widget.Toolbar xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:local="http://schemas.android.com/apk/res-auto"
    android:id="@+id/toolbar"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:minHeight="?attr/actionBarSize"
    android:background="?attr/colorPrimary"
    local:theme="@style/ThemeOverlay.AppCompat.Dark.ActionBar"
    local:popupTheme="@style/ThemeOverlay.AppCompat.Light" />
```

Puis ouvrez, votre activity_main.xml puis ajoutez le toolbar que nous venons de créer.

```
<android.support.v4.widget.DrawerLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/drawer_layout"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:orientation="vertical">

        <LinearLayout
            android:id="@+id/container_toolbar"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:orientation="vertical">

            <include
                android:id="@+id/toolbar"
                layout="@layout/toolbar" />
        </LinearLayout>
```

Compilez et vous devriez avoir :



Pour mieux comprendre la structure de l'application, nous allons rajouter une image "search" qui servira de bouton sur la bar d'action.

Téléchargez le fichier à cette adresse http://api.androidhive.info/images/ic_action_search.png et placez l'image dans un des dossier mipmap.

Puis ouvrez le fichier menu_main.xml qui se situe dans le dossier res/menu et créer un item pour l'image search comme ci-dessous :

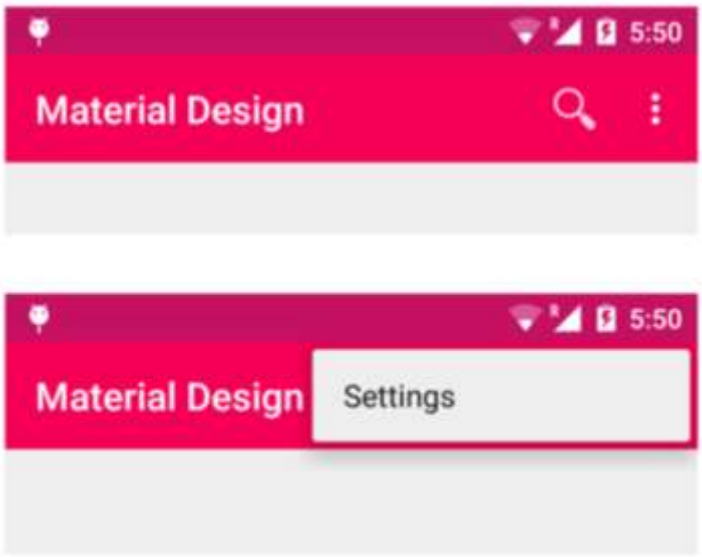
```
<menu xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools" tools:context=".MainActivity">
    <item
        android:id="@+id/action_search"
        android:title="@string/action_search"
        android:orderInCategory="100"
        android:icon="@mipmap/ic_action_search"
        app:showAsAction="ifRoom" />

    <item android:id="@+id/action_settings" android:title="Settings"
        android:orderInCategory="100" app:showAsAction="never" />
</menu>
```

Ouvrez votre fichier MainActivity.java, vérifiez que votre classe hérite de AppCompatActivity. Puis nous allons appeler dans "onCreate" notre bar d'action que nous venons de créer dans notre classe pour qu'il puisse être utilisé. Ensuite nous réécrivons onCreateOptionsMenu() et onOptionsItemSelected().

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    mToolbar = (Toolbar) findViewById(R.id.toolbar);
    setSupportActionBar(mToolbar);
    getSupportActionBar().setDisplayShowHomeEnabled(true);
```

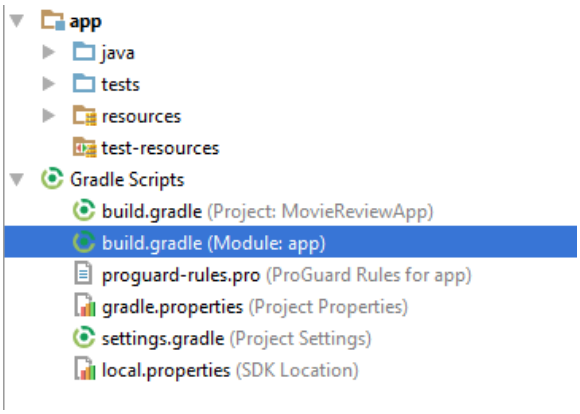
```
@Override
public boolean onCreateOptionsMenu(Menu menu) {
    // Inflate the menu; this adds items to the action bar if it is present.
    getMenuInflater().inflate(R.menu.menu_main, menu);
    return true;
}

@Override
public boolean onOptionsItemSelected(MenuItem item) {
    // Handle action bar item clicks here. The action bar will
    // automatically handle clicks on the Home/Up button, so long
    // as you specify a parent activity in AndroidManifest.xml.
    int id = item.getItemId();

    //noinspection SimplifiableIfStatement
    if (id == R.id.action_settings) {
        return true;
    }

    if(id == R.id.action_search){
        Toast.makeText(getApplicationContext(), "Search action is selected!", Toast.LENGTH_SHORT).show();
        return true;
    }

    return super.onOptionsItemSelected(item);
}
```

Lancez l'application, vous devriez avoir:



1.3. Ajoute du menu coulissant

Dans le build.gradle qui se situe sous le module:app :



Rajoutez les dépendances,les "dependecies" sont des librairies, ci-dessous :

```
dependencies {
    compile fileTree(dir: 'libs', include: ['*.jar'])
    compile 'com.android.support:appcompat-v7:23.0.0'
    compile 'com.android.support:recyclerview-v7:23.0.+'
```

Une fois rajouté, reconstruisez le projet "rebuild project".

Nous créons ensuite une classe NavDrawerItem qui définira chaque rangée du menu.

```java
/**
 * Created by Jean-Philippe Kha on 30/09/2015.
 */
public class NavDrawerItem {
    private boolean showNotify;
    private String title;


    public NavDrawerItem() {

    }

    public NavDrawerItem(boolean showNotify, String title) {
        this.showNotify = showNotify;
        this.title = title;
    }

    public boolean isShowNotify() { return showNotify; }

    public void setShowNotify(boolean showNotify) { this.showNotify = showNotify; }

    public String getTitle() { return title; }

    public void setTitle(String title) { this.title = title; }
}
```

Nous définissons maintenant la vue qu'aura la rangée dans le menu. Nous mettrons juste un titre, mais il est possible de rajouter une icon ou autre si vous le souhaitez.

Créons dans notre dossier res/layout le fichier nav_drawer_row.xml

```xml
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:clickable="true">

    <TextView
        android:id="@+id/title"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:paddingLeft="30dp"
        android:paddingTop="10dp"
        android:paddingBottom="10dp"
        android:textSize="15dp"
        android:textStyle="bold" />

</RelativeLayout>
```

Nous allons rajouter une icône utilisateur dans notre menu. Téléchargez l'image à cette adresse, si vous n'avez pas d'image. (http://api.androidhive.info/images/ic_profile.png).

Maintenant nous allons créer un autre fichier fragment_navigation_drawer.xml, toujours dans le dossier res/layout. Ce dernier permettra d'afficher ce que nous verrons sur l'application.

fragment_navigation_drawer.xml

```xml
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="@android:color/white">

    <RelativeLayout
        android:id="@+id/nav_header_container"
        android:layout_width="match_parent"
        android:layout_height="140dp"
        android:layout_alignParentTop="true"
        android:background="@color/colorPrimary">

        <ImageView
            android:layout_width="70dp"
            android:layout_height="70dp"
            android:src="@mipmap/ic_profile"
            android:scaleType="fitCenter"
            android:layout_centerInParent="true" />

    </RelativeLayout>

    <android.support.v7.widget.RecyclerView
        android:id="@+id/drawerList"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_below="@id/nav_header_container"
        android:layout_marginTop="15dp" />

</RelativeLayout>
```

Nous allons maintenant adapter les rangées de notre RecyclerView (une liste) avec les rangées, nav_drawer_row.xml précédemment défini. Pour cela nous allons créer une classe java NavigationDrawerAdapter.java

```java
public class NavigationDrawerAdapter extends RecyclerView.Adapter<NavigationDrawerAdapter.MyViewHolder> {
    List<NavDrawerItem> data = Collections.emptyList();
    private LayoutInflater inflater;
    private Context context;

    public NavigationDrawerAdapter(Context context, List<NavDrawerItem> data) {
        this.context = context;
        inflater = LayoutInflater.from(context);
        this.data = data;
    }

    public void delete(int position) {
        data.remove(position);
        notifyItemRemoved(position);
    }

    @Override
    public MyViewHolder onCreateViewHolder(ViewGroup parent, int viewType) {
        View view = inflater.inflate(R.layout.nav_drawer_row, parent, false);
        MyViewHolder holder = new MyViewHolder(view);
        return holder;
    }

    @Override
    public void onBindViewHolder(MyViewHolder holder, int position) {
        NavDrawerItem current = data.get(position);
        holder.title.setText(current.getTitle());
    }
```

```java
    @Override
    public int getItemCount() { return data.size(); }

    class MyViewHolder extends RecyclerView.ViewHolder {
        TextView title;

        public MyViewHolder(View itemView) {
            super(itemView);
            title = (TextView) itemView.findViewById(R.id.title);
        }
    }
```

Une fois fait nous créons une classe java FragmentDrawer.java, qui hérite de fragment, qui permettra d'instancier le menu déroulant.

```java
/**
 * Created by Jean-Philippe Kha on 30/09/2015.
 */
import ...

public class FragmentDrawer extends Fragment {

    private static String TAG = FragmentDrawer.class.getSimpleName();

    private RecyclerView recyclerView;
    private ActionBarDrawerToggle mDrawerToggle;
    private DrawerLayout mDrawerLayout;
    private NavigationDrawerAdapter adapter;
    private View containerView;
    private static String[] titles = null;
    private FragmentDrawerListener drawerListener;

    public FragmentDrawer() {

    }

    public void setDrawerListener(FragmentDrawerListener listener) {
        this.drawerListener = listener;
    }

    public static List<NavDrawerItem> getData() {
        List<NavDrawerItem> data = new ArrayList<>();


        // preparing navigation drawer items
        for (int i = 0; i < titles.length; i++) {
            NavDrawerItem navItem = new NavDrawerItem();
            navItem.setTitle(titles[i]);
            data.add(navItem);
        }
        return data;
    }
```

```java
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);

    // drawer labels
    titles = getActivity().getResources().getStringArray(R.array.nav_drawer_labels);
}

@Override
public View onCreateView(LayoutInflater inflater, ViewGroup container,
                         Bundle savedInstanceState) {
    // Inflating view layout
    View layout = inflater.inflate(R.layout.fragment_navigation_drawer, container, false);
    recyclerView = (RecyclerView) layout.findViewById(R.id.drawerList);

    adapter = new NavigationDrawerAdapter(getActivity(), getData());
    recyclerView.setAdapter(adapter);
    recyclerView.setLayoutManager(new LinearLayoutManager(getActivity()));
    recyclerView.addOnItemTouchListener(new RecyclerTouchListener(getActivity(), recyclerView, new ClickListener() {
        @Override
        public void onClick(View view, int position) {
            drawerListener.onDrawerItemSelected(view, position);
            mDrawerLayout.closeDrawer(containerView);
        }

        @Override
        public void onLongClick(View view, int position) {

        }
    }));

    return layout;
}
```

```java
public void setUp(int fragmentId, DrawerLayout drawerLayout, final Toolbar toolbar) {
    containerView = getActivity().findViewById(fragmentId);
    mDrawerLayout = drawerLayout;
    mDrawerToggle = new ActionBarDrawerToggle(getActivity(), drawerLayout, toolbar, "Open", "Close") {
        @Override
        public void onDrawerOpened(View drawerView) {
            super.onDrawerOpened(drawerView);
            getActivity().invalidateOptionsMenu();
        }

        @Override
        public void onDrawerClosed(View drawerView) {
            super.onDrawerClosed(drawerView);
            getActivity().invalidateOptionsMenu();
        }

        @Override
        public void onDrawerSlide(View drawerView, float slideOffset) {
            super.onDrawerSlide(drawerView, slideOffset);
            toolbar.setAlpha(1 - slideOffset / 2);
        }
    };

    mDrawerLayout.setDrawerListener(mDrawerToggle);
    mDrawerLayout.post(() -> { mDrawerToggle.syncState(); });

}

public static interface ClickListener {
    public void onClick(View view, int position);

    public void onLongClick(View view, int position);
}
```

```
static class RecyclerTouchListener implements RecyclerView.OnItemTouchListener {

    private GestureDetector gestureDetector;
    private ClickListener clickListener;

    public RecyclerTouchListener(Context context, final RecyclerView recyclerView, final ClickListener clickListener) {
        this.clickListener = clickListener;
        gestureDetector = new GestureDetector(context, new GestureDetector.SimpleOnGestureListener() {
            @Override
            public boolean onSingleTapUp(MotionEvent e) { return true; }

            @Override
            public void onLongPress(MotionEvent e) {
                View child = recyclerView.findChildViewUnder(e.getX(), e.getY());
                if (child != null && clickListener != null) {
                    clickListener.onLongClick(child, recyclerView.getChildPosition(child));
                }
            }
        });
    }

    @Override
    public boolean onInterceptTouchEvent(RecyclerView rv, MotionEvent e) {

        View child = rv.findChildViewUnder(e.getX(), e.getY());
        if (child != null && clickListener != null && gestureDetector.onTouchEvent(e)) {
            clickListener.onClick(child, rv.getChildPosition(child));
        }
        return false;
    }

    @Override
    public void onTouchEvent(RecyclerView rv, MotionEvent e) {
    }

    @Override
    public void onRequestDisallowInterceptTouchEvent(boolean disallowIntercept) {

    }

    public interface FragmentDrawerListener {
        public void onDrawerItemSelected(View view, int position);
    }
}
```

Dans cette classe, nous définissons tout les comportements du menu en fonction des gestes de l'utilisateur.

Dans le activity_main.xml, nous pouvons alors écrire notre menu déroulant.

```
    <fragment
        android:id="@+id/fragment_navigation_drawer"
        android:name="com.polytechnicesophia.jpkha.moviereviewapp.FragmentDrawer"
        android:layout_width="260dp"
        android:layout_height="match_parent"
        android:layout_gravity="start"
        app:layout="@layout/fragment_navigation_drawer"
        tools:layout="@layout/fragment_navigation_drawer" />

</android.support.v4.widget.DrawerLayout>
```

Et le définir dans notre MainActivity.java notre DrawerLayout

```
public class MainActivity extends AppCompatActivity implements FragmentDrawer.FragmentDrawerListener {

    private static String TAG = MainActivity.class.getSimpleName();

    private Toolbar mToolbar;
    private FragmentDrawer drawerFragment;
    private Fragment currentFragment;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        mToolbar = (Toolbar) findViewById(R.id.toolbar);
        setSupportActionBar(mToolbar);
        getSupportActionBar().setDisplayShowHomeEnabled(true);
        drawerFragment = (FragmentDrawer)
                getSupportFragmentManager().findFragmentById(R.id.fragment_navigation_drawer);
        drawerFragment.setUp(R.id.fragment_navigation_drawer, (DrawerLayout) findViewById(R.id.drawer_layout), mToolbar);
        drawerFragment.setDrawerListener(this);
        // display the first navigation drawer view on app launch
        displayView(0);
    }

    @Override
    public void onDrawerItemSelected(View view, int position) { displayView(position); }

    private void displayView(int position) {
        currentFragment = null;
        //String title = getString(R.string.app_name);
        String title = "";
        switch (position) {
            case 0:
                currentFragment = new HomeFragment();
                title = "Movie Review App";
                break;
            case 1:
                currentFragment = new MovieListFragment();
                title ="Films selectionnés";
                break;
            default:
                break;
        }

        if (currentFragment != null) {
            FragmentManager fragmentManager = getSupportFragmentManager();
            FragmentTransaction fragmentTransaction = fragmentManager.beginTransaction();
            fragmentTransaction.replace(R.id.container_body, currentFragment);
            fragmentTransaction.commit();

            // set the toolbar title
            getSupportActionBar().setTitle(title);
        }
    }
}
```

1.4. Implémentation de fragments au menu

Pour compléter cette première partie nous allons créer les fragments pour pouvoir naviguer. Définissons home_fragment.xml dans un premier temps et FragmentHome.java dans un second temps. Les autres fragments sont construits de la même façon.

FRAGMENT_HOME.XML

```xml
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    tools:context="info.androidhive.materialdesign.activity.HomeFragment">


    <TextView
        android:id="@+id/label"
        android:layout_alignParentTop="true"
        android:layout_marginTop="100dp"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:gravity="center_horizontal"
        android:textSize="45dp"
        android:text="HOME"
        android:textStyle="bold"/>

    <TextView
        android:layout_below="@id/label"
        android:layout_centerInParent="true"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:textSize="12dp"
        android:layout_marginTop="10dp"
        android:gravity="center_horizontal"
        android:text="Edit fragment_home.xml to change the appearance" />

</RelativeLayout>
```

et le FragmentHome :

HOMEFRAGMENT.JAVA

```java
import android.app.Activity;
import android.os.Bundle;
import android.support.v4.app.Fragment;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;


public class HomeFragment extends Fragment {

    public HomeFragment() {
        // Required empty public constructor
    }

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

    }

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
                             Bundle savedInstanceState) {
        View rootView = inflater.inflate(R.layout.fragment_home, container, false);


        // Inflate the layout for this fragment
        return rootView;
    }

    @Override
    public void onAttach(Activity activity) {
        super.onAttach(activity);
    }

    @Override
    public void onDetach() {
        super.onDetach();
    }
}
```

2. Onglet glissant

Dans un premier temps nous avons besoin de rajouter une autre dépendance (“compile ‘com.android.support:design:23.0.1’ “) dans le gradle :

```
dependencies {
    compile fileTree(dir: 'libs', include: ['*.jar'])
    compile 'com.android.support:appcompat-v7:23.0.0'
    compile 'com.android.support:recyclerview-v7:23.0.+'
    compile 'com.android.support:design:23.0.1'
}
```

Une fois écrit, reconstruisez votre projet. (“Rebuild Project”).

Nous allons maintenant construire une nouvelle activité qui aura 3 onglets glissant.

Créer une nouvelle activité qui aura pour nom : TabMovieDetailsActivity.

Ouvrez le fichier activity_tab_movie_details.xml et définissez les paramètres suivant :

```
<android.support.design.widget.CoordinatorLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <android.support.design.widget.AppBarLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:theme="@style/ThemeOverlay.AppCompat.Dark.ActionBar">

        <android.support.v7.widget.Toolbar
            android:id="@+id/toolbar"
            android:layout_width="match_parent"
            android:layout_height="?attr/actionBarSize"
            android:background="?attr/colorPrimary"
            app:layout_scrollFlags="scroll|enterAlways"
            app:popupTheme="@style/ThemeOverlay.AppCompat.Light" />

        <android.support.design.widget.TabLayout
            android:id="@+id/tabs"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            app:tabMode="fixed"
            app:tabGravity="fill"/>
    </android.support.design.widget.AppBarLayout>

    <android.support.v4.view.ViewPager
        android:id="@+id/viewpager"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        app:layout_behavior="android.support.design.widget.AppBarLayout$ScrollingVie..." />
</android.support.design.widget.CoordinatorLayout>
```

Puis ouvrez TabMovieDetailsActivity.java, le code permettra d’avoir 3 onglets coulissants avec un texte et une icône définissant chaque onglet.

```
public class TabMovieDetailsActivity extends AppCompatActivity {

    private Toolbar toolbar;
    private TabLayout tabLayout;
    private ViewPager viewPager;
    private ArrayList<Movie> listMovie;
    private int i;

    private int[] tabIcons = {
            R.mipmap.ic_info,
            R.mipmap.like,
            R.mipmap.heart_break
    };
    private ImageView playBtn;
    private ImageView movieImage;
```

Nous définissons nos variables.

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_tab_movie_details);
    Bundle extras = getIntent().getExtras();
    final int i = extras.getInt("intMovie");
    initializeListMovie();
    setTitle(listMovie.get(i).getName());

    toolbar = (Toolbar) findViewById(R.id.toolbar);
    setSupportActionBar(toolbar);
    getSupportActionBar().setDisplayHomeAsUpEnabled(true);
    viewPager = (ViewPager) findViewById(R.id.viewpager);
    setupViewPager(viewPager);

    movieImage = (ImageView) findViewById(R.id.movieImage);
    movieImage.setImageDrawable(listMovie.get(i).getHeader());

    playBtn = (ImageView) findViewById(R.id.playBtn);
    playBtn.setOnClickListener((v) -> {
            Intent browserIntent = new Intent(Intent.ACTION_VIEW, Uri.parse(listMovie.get(i).getUrl()));
            startActivity(browserIntent);
    });
    tabLayout = (TabLayout) findViewById(R.id.tabs);
    tabLayout.setupWithViewPager(viewPager);
    setupTabIcons();

}
```

Dans la fonction onCreate, nous récupérons les paramètres transmis par le MainActivity grâce au Bundle. Nous initialisons les variables que nous allons utiliser. getSupportActionBar().setDisplayHomeAsUpEnabled(true) permet d'afficher une flèche de retour sur l'écran défini dans l'AndroidManifest. Il faut rajouter la balise meta-data comme ci-dessous

```
<activity
    android:name=".TabMovieDetailsActivity"
    android:label="TabMovieDetailsActivity"
    android:parentActivityName=".MainActivity">
    <meta-data
        android:name="android.support.PARENT_ACTIVITY"
        android:value=".MainActivity" />
</activity>
```

Les images suivantes concernent toujours la classe TabMovieDetailsActivity.java

```
private void setupViewPager(ViewPager viewPager) {
    ViewPagerAdapter adapter = new ViewPagerAdapter(getSupportFragmentManager());
    adapter.addFragment(new MovieDetailsFragment(), " Details");
    adapter.addFragment(new MovieGoodCommentFragment(), " Like");
    adapter.addFragment(new MovieBadCommentFragment(), " Dislike");
    viewPager.setAdapter(adapter);
}

private void setupTabIcons() {
    tabLayout.getTabAt(0).setIcon(tabIcons[0]);
    tabLayout.getTabAt(1).setIcon(tabIcons[1]);
    tabLayout.getTabAt(2).setIcon(tabIcons[2]);
}
```

La méthode setupViewPager() permet de lier les onglet à un fragment. C'est dans cette fonction qu'il faudra rajouter des fragments si vous souhaitez avoir plus d'onglet.

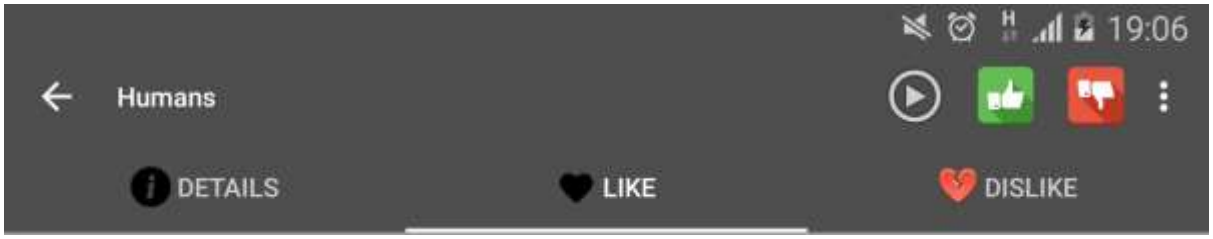
La méthode setupTabIcons() permet d'ajouter une icone au titre de l'onglet.

```
class ViewPagerAdapter extends FragmentPagerAdapter {
    private final List<Fragment> mFragmentList = new ArrayList<>();
    private final List<String> mFragmentTitleList = new ArrayList<>();

    public ViewPagerAdapter(FragmentManager manager) {
        super(manager);
    }

    @Override
    public Fragment getItem(int position) {
        return mFragmentList.get(position);
    }

    @Override
    public int getCount() {
        return mFragmentList.size();
    }

    public void addFragment(Fragment fragment, String title) {
        mFragmentList.add(fragment);
        mFragmentTitleList.add(title);
    }

    @Override
    public CharSequence getPageTitle(int position) {
        return mFragmentTitleList.get(position);
    }
}
```

Les fragments appelés dans cette classe sont du même modèle que celui que nous avons créé dans la partie “Sliding menu”

La classe FragmentPagerAdapter permet de garder les fragments appelés sont active tant que l’activité est utilisé.

Vous devriez avoir :



3. Structure des interfaces

Pour structurer nos affichages nous avons opté pour l'utilisation de LinearLayout et de Layout_weight. Cela permet de définir l'espace que prendra chaque élément dans l'affichage que ce soit en hauteur ou en largeur.

Exemple avec notre home_fragment.xml :


```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:background="@color/colorPrimary"
    android:id="@+id/fragment_home"
    android:gravity="center"
    android:orientation="vertical"
    android:fillViewport="true"
    android:padding="10dp" >

    <LinearLayout
        android:layout_width="fill_parent"
        android:layout_height="fill_parent"
        android:layout_alignParentBottom="true"
        android:layout_gravity="center"
        android:orientation="vertical"
        android:paddingLeft="20dp"
        android:paddingRight="20dp"
        android:paddingTop="20dp" >

        <ImageView
            android:layout_width="match_parent"
            android:layout_weight="6"
            android:layout_height="0px"
            android:id="@+id/movieImage"
            android:src="@mipmap/hqdefault"
            android:layout_centerInParent="true"
            android:layout_marginTop="20dp"
            android:layout_marginBottom="20dp"
            android:onClick="posterEvent"
            android:layout_gravity="center" />

        <LinearLayout
            android:layout_width="match_parent"
            android:layout_height="0px"
            android:layout_weight="2"
            android:orientation="horizontal" >
            <ImageView
                android:layout_width="0px"
                android:layout_weight="1"
                android:layout_height="wrap_content"
                android:id="@+id/rightArrow"
                android:src="@mipmap/thumb_up_min"
                android:layout_centerInParent="true"

                android:layout_gravity="center" />

            <ImageView
                android:layout_width="0px"
                android:layout_weight="1"
                android:layout_height="wrap_content"
                android:id="@+id/dejaVuIcon"
                android:onClick="dejaVuEvent"
                android:src="@mipmap/eyes"
                android:layout_centerInParent="true"
                android:layout_gravity="center" />

            <ImageView
                android:layout_width="0px"
                android:layout_weight="1"
                android:layout_height="wrap_content"
                android:id="@+id/leftArrow"
                android:src="@mipmap/thumb_down"

                android:layout_centerInParent="true"
                android:layout_gravity="center" />
        </LinearLayout>
```

Comme décris dans le code nous avons l'image qui prend un poids de 6 pour la hauteur de l'écran disponible et le LinearLayout contenant les 3 boutons prend un poids de 2 de la hauteur. Nous avons un poids total de 8 donc l'image prend 6/8 de la place et le LinearLayout 2/8

Dans la partie des 3 boutons nous mettons un poids de 1 à chaque bouton ce qui divise l'espace en 3 partie.

Pour réarranger les boutons lorsque l'utilisateur change de format d'écran, un écran plus grand ou l'orientation, nous avons utilisé un dossier que nommons layout-land et layout-sw600dp.

En effet le code ci-dessus permet aux composants d'occuper l'espace que nous lui fournissons mais ne change pas de position. Donc pour changer de position nous devons créer un nouveau dossier et nous recréons un fichier du même nom que celui que nous avons besoin d'adapter.

Par exemple créer un fichier fragment_home.xml dans le dossier layout-land.

Layout-land correspond au fichier xml que nous utiliserons lorsque l'orientation est en paysage.

sw600dp correspond au fichier xml que nous utiliserons lorsque le matériel a un écran d'au moins 600dp.

Si vous souhaitez faire apparaître des composants lors d'une action, comme un clique, vous avez un paramètre "visibility" qui prend 3 paramètres :

-*Gone* signifie que l'élément ne prend pas de place et disparaît de l'écran.

-*Invisible* signifie que l'objet garde la place qui l'occupe bien qu'il soit plus visible.

-Visible, est l'état par défaut de chaque élément, nous pouvons voir l'élément..

Par exemple, dans notre code pour afficher et enlever nos commentaire nous avons mis un "listener" sur un objet de la liste et changé la visibilité :

```
convertView.setOnClickListener(new View.OnClickListener(){
    @Override
    public void onClick(View v) {
        viewHolder.linearLayoutComment.setVisibility(
                viewHolder.linearLayoutComment.getVisibility()==View.GONE ? View.VISIBLE:View.GONE);
    }
});
```

2) Les outils de développement et de tests utilisés

Nous avons utilisé Android Studio pour développer l'application Android. Il faut au préalable avoir java installé.
Les tests ont été effectué sur un Galaxy S4, S6 et un Acer A1.

3) Déploiement

Le déploiement peut se faire sous différente forme :

- Par le Play Store
- Via un server web, en proposant l'apk en téléchargement
- Depuis le PC via ADB (Android Debug Bridge)
- Copier l'application sur l'appareil mobile

4) Tests des capacités d'adaptation

Pour effectuer les test d'adaptation, nous avons dans un premier temps utilisé l'emulateur d'Android puis nous avons lancé l'application dans les différents environnement mobile et étudié son comportement en paysage et en portrait.

III-Développement avec React

La seconde technologie que nous avons voulu utiliser s'appelle React. React est une librairie JavaScript destinée au développement des interfaces.

Nous avons travaillé avec un starter kit qui s'appuie sur le langage ES6 (ECMA Script Edition 6). Notre starter kit assure le live reloading (rechargement direct).

1) Installation React starter kit

Avant de commencer l'installation, veuillez vérifier que vous avez node et python installés sur votre machine.

Si ce n'est pas le cas, veuillez suivre les instructions suivantes.

A - Node

Pour installer Node.js, il vous suffit de télécharger l'installeur à l'adresse suivante et ensuite de l'exécuter.

https://nodejs.org/en/download/

B - Python

Comme pour Node, téléchargez et exécutez l'installeur.

https://www.python.org/downloads/

Note: Si vous rencontrez des problèmes après l'installation de python pendant le lancement du starter kit, pensez à ajouter le chemin de votre répertoire python dans la variable d'environnement PATH.

C - React Starter Kit

Pour installer ce starter kit, vous pouvez suivre les étapes suivantes:

```
1) git clone –o react-starter-kit -b master --single-branch
\https://github.com/kriasoft/react-starter-kit.git MyApp
2) cd MyApp
3) npm install
4) npm start
```

Note: Il est possible que l'absence de certaines librairies empêche le serveur de démarrer. Dans le message d'erreur vous pourrez lire qu'une librairie manque ou est impossible à trouver. Dans ce cas là, vous pouvez installer manuellement la librairie manquante avec la commande suivante.

```
npm install NameOfMissingLibrary
```

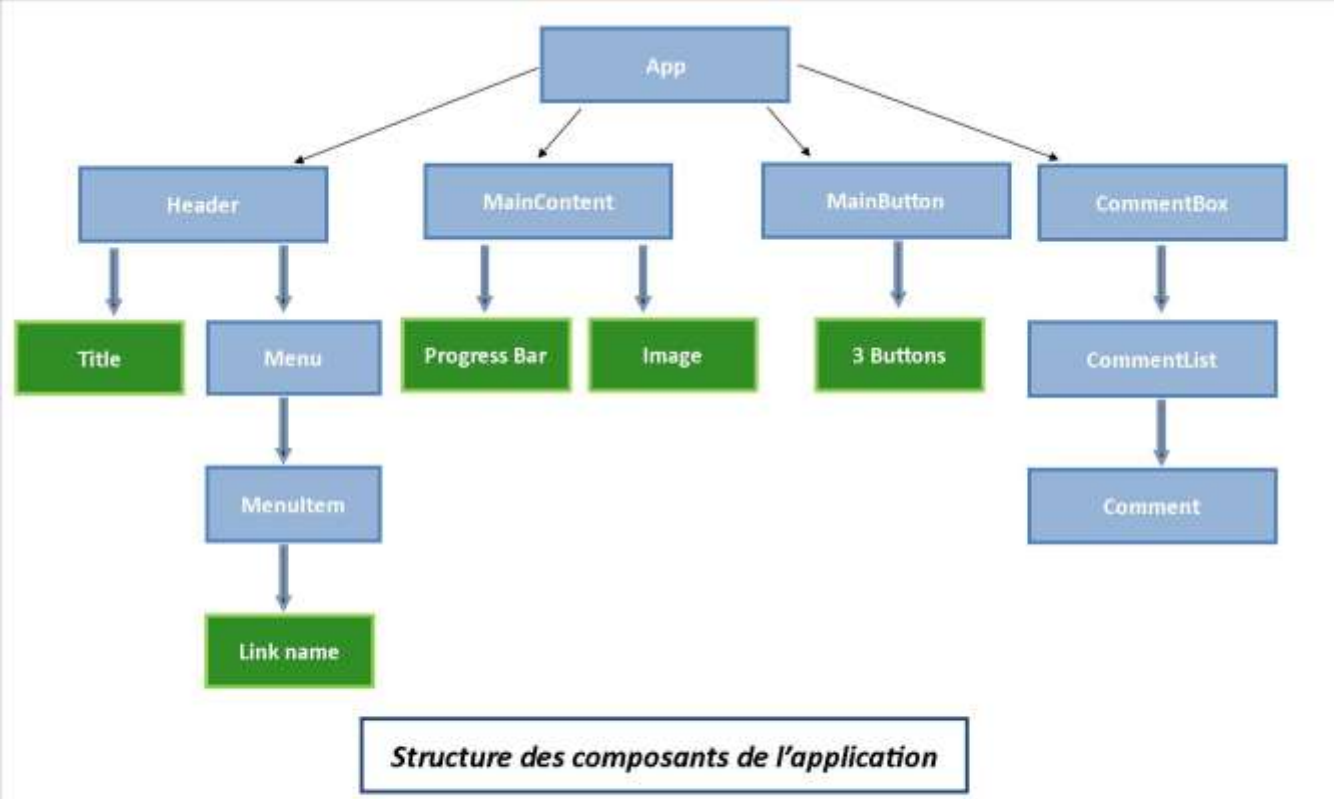
D - Apprendre à programmer avec React

Nous avons utilisé un tutoriel afin de se familiariser avec React. Il nous a permis de comprendre les bases et le fonctionnement de React. Vous trouverez le lien vers le tutoriel dans la Bibliographie.

2) Comment l'application a-t-elle été réalisée ?

A - Architecture des classes

L'un des principal avantage de React est son système de Component. Dans notre projet, nous avons créé un objet (une classe) pour chaque élément graphique. L'image suivante illustre la structure de notre projet.



Structure des composants de l'application

Légende:
Bleu: Nos composants
Vert: Les autres éléments graphiques

B - Composant: App

App est la classe principale de notre projet. Elle fait office de racine pour tout notre projet. Pas à pas, nous allons ajouté les composants graphiques à l'intérieur de cette classe.

Nous la déclarons comme une classe fille de Component. Cela va nous permettre d'utiliser les propriétés de la classe héritée.

Les importations

Deux types d'importations sont nécessaires. Tout d'abord les importations pour utiliser React et intégrer le style.

```
import React, { PropTypes, Component } from 'react';
import styles from './App.css';
import withContext from '../../decorators/withContext';
import withStyles from '../../decorators/withStyles';
```

La librairie React se trouve dans le module node React. App.css se situe dans le même repertoire que App.js. WithContext et withStyles sont deux fichiers javascript qui se trouvent dans le repertoire src/decorators.

Ensuite nous importons les composants React que nous souhaitons utiliser. Comme nos composants sont tous créés dans le même répertoire src/components, toutes les importations se font de la même façon avec la structure suivante.

Note: Pour le reste du tutoriel, nous ne décrirons pas les importations pour les autres composants afin d'alléger leur présentation. Néanmoins, vous trouverez en Annexe 1 le tableau des importations par fichier.

Le coeur de la classe App

```
class App extends Component {

  constructor(){
    super();
  }

  static propTypes = {
    children: PropTypes.element.isRequired,
    error: PropTypes.object
  };



  render() {
    return !this.props.error ? (
      <div>
        <Header />
        <MainContent/>
        <CommentBox/>
        <BadCommentBox/>
        <Feedback />
        <Footer />
      </div>
    ) : this.props.children;
  }

}

export default App;
```

Constructor

Le constructeur de la classe App n'est pas nécessaire dans ce cas là. En revanche, si l'on souhaite ajouter des méthodes utilisant les propriétés (ou état) de la classe Component, il est nécessaire d'initialiser ces paramètres. Nous verrons un exemple d'initialisation dans la classe Header.

PropTypes

Dans cette méthode on déclare qu'une propriété est spécifique dans notre classe. Dans notre cas, nous utilisons les deux attributs error et children dans la fonction render pour traiter un cas d'erreur venant d'un objet.

Render

La fonction render est nécessaire à toute classe lorsque l'on développe avec React puisqu'elle constitue la base de chaque composant. Elle retourne un arbre de composants React. Sans elle, la construction de l'arbre ne peut se faire. Cette fonction examine this.props, les propriétés du composant, et this.state, son état, et retourne un seul élément utilisable sous forme de la balise <App>.

Comme précisé dans le schéma "La structure des composants de l'application", notre classe contient tous les composants responsables de la structure de notre page. On déclare donc nos composants que l'on souhaite utiliser.

- <Header />: Notre header (haut de page) qui contient le menu et le nom de l'application
- <MainContent />: Contient l'image du film, le titre et la jauge de progression
- <CommentBox />: La boite qui contient les commentaires positifs
- <BadCommentBox />: La boite qui contient les commentaires
- <MainContent />: Contient les informations sur le film
- <Feedback />: Contient les trois boutons qui serviront à donner son avis sur le film
- <Footer />: Le footer de la page (pied de page).

Note: La modélisation des commentaires pourrait être modifiée. En effet, nous pourrions regrouper CommentBox et BadCommentBox en CommentBox. Par la suite, il suffirait d'appeler deux fois CommentBox avec un paramètre nous permettant de distinguer le cas des bons et mauvais commentaires.

```
<CommentBox data={'goodMovie'}/>
<CommentBox data={'badMovie'}/>
```

[Suite note] Ensuite il suffit de récupérer les données à l'intérieur de CommentBox par le biais this.props.data. Je n'ai pas fait ce choix car au début je n'étais pas certain de réussir à développer ces deux fonctionnalités. J'ai donc décidé de développer dans un premier temps les commentaires positifs et dans un second temps les négatifs. Fusionner ces deux parties serait assez rapide maintenant qu'elles sont créées.

Export default App

Cette ligne est importante puisqu'elle permet de d'exporter notre classe. C'est grâce à elle que l'on peut effectuer l'importation sous la forme

C- MainContent

```
import React, { Component } from 'react';
import styles from './MainContent.css';
import withStyles from '../../decorators/withStyles';
import Link from '../Link';

@withStyles(styles)
class MainContent extends Component {

  render() {
    return (
      <div className="MainContent">
        <div className="MainContent-container">
          <div className="MainContent-banner">
            <h3 className="MainContent-bannerTitle">Human the movie</h3><br/>
            <img className="Component-brandImg" src={require('./human_affiche_cinema.jpg')}  width="500" height="700" alt="React" />
            <div id="page-wrap">
              <p><progress className="Progress" id="progress" value="25" max="100"></progress> <span id="pourcentage"></span></p>
            </div>

          </div>
        </div>
      </div>
    );
  }

}

export default MainContent;
```

La classe MainContent est très simple. Elle contient le titre, l'image du site et la jauge de progression. Dans l'état actuel de l'application, ces valeurs sont "hard-codées" (écrites directement dans la méthode render). Pour améliorer l'utilisation de la classe, il serait possible de charger les noms des films et les titres des images depuis un fichier json ou une base de données. En modifiant l'état de la classe (this.state) on serait en mesure de modifier le contenu de la méthode render. Dans la suite de ce tutoriel, nous allons voir comment modifier l'état d'une classe.

D - CommentBox

La classe CommentBox contient une liste de commentaires positifs. Cette seule fonctionnalité ne justifie pas la création de cette classe. Nous avons fait ce choix à la conception car nous avions plus d'ambition pour cette classe.
Voici l'état actuel de cette classe:

```
class CommentBox extends Component {

  render() {
    return (
      <div className="CommentBox">
        <CommentList/>
      </div>
    );
  };
}


export default CommentBox;
```

Nous avons simplement défini la fonction render qui appelle simplement le composant CommentList. On définit un className pour pouvoir appliqué un css particulier à ce composant.

Comment pourrait-on améliorer cette classe ?

Avant de travailler avec ce starter kit, nous avons implémenté une zone de commentaire avec enregistrement/chargement des données depuis un fichier json. La mise à jour automatique du seul composant CommentBox s'effectuais suivant un intervalle que l'on définissait. Dans ce cas particulier on peut observer une des grandes forces de React: sa décomposition en Component. Il est possible de recharger un seul composant d'une page sans influer sur les autres entités de la page. Chaque composant est paramétrable indépendamment des uns et des autres.

La partie de chargement des données et de paramétrage se trouvait justement dans la classe CommentBox. Vous trouverez dans notre rendu le code pour cette zone de commentaire.

E - CommentList

Comme pour la classe précédente, GoodCommentList est relativement simple. Nous avons uniquement redéfinit la fonction render avec un titre l'appel à deux commentaires.

```
@withStyles(styles)
class CommentList extends Component{

  render() {
    return (
      <div className="CommentList">
        <h2> LIKE </h2>
        <Comment />
        <Comment />
      </div>
    );
  };
}

export default CommentList;
```

L'objectif était, là encore, de se servir du fichier json pour charger en paramètre de la classe un identifiant correspondant à un commentaire précis. La classe CommentList traite si la liste de commentaires que l'on souhaite afficher correspond aux commentaires positifs ou négatifs. Ensuite, on effectue une recherche pour récupérer les identifiants de tous les commentaires et on les utilise comme paramètre de la classe Comment.

F - Comment

La classe Comment contient le contenu d'un commentaire ainsi que l'animation d'ouverture et de fermeture d'un commentaire.

Render

Commençons par la fonction Render. Un commentaire est structuré de la façon suivante.

Nous avons deux cas pour l'affichage du commentaire, lorsque le commentaire est ouvert et lorsqu'il est fermé.

Comment développer cette fonctionnalité avec React ?

Dans un développement javascript normal, nous pourrions utiliser la fonction onClick sur le titre du commentaire et jouer sur l'apparition et la disparition du contenu du commentaire et de son auteur. Vous trouverez en Annexe 2 un exemple de fonction javascript que nous pourrions développer dans notre cas.

Avec React, la démarche est différente. Nous devons utiliser l'état (this.state) de la classe afin d'apporter des modifications dynamiques à notre composant.

```
render() {
  return (
    <div className="Comment">
      {this.state.showResults ?
        <li class="Item1">
          <a href="#" onClick={this.onClick.bind(this)}> Awesome movie !!</a>
          <ul>
            <li class="subitem1"> <a>I really enjoy that movie </a></li>
            <li class="subitem2"> <a>Rémi Pourtier  </a></li>
          </ul>
        </li>
        :<li class="Item1">
        <a href="#" onClick={this.onClick.bind(this)}> Awesome movie !! </a>
      </li>
        }
    </div>
  );
};
```

Nous allons créer une variable showResults. Nous allons tester cette variable dans le render et modifier l'affichage en fonction de sa valeur. Quand showResults est à vrai, nous afficherons le commentaire complet avec son contenu et son auteur. Sinon, nous afficherons uniquement le titre du commentaire.

Initalisation

Il est impératif d'initialiser cette variable afin de pouvoir l'utiliser. Nous allons mettre par défaut la variable this.state à false afin d'avoir une liste de commentaire fermée lors du chargement de la page.

Constructor

Dans le constructeur de notre classe nous allons initialiser la valeur de l'état de la fonction (this.state) par notre variable showResults elle même initialisée à false.

```
constructor(){
  super();
  this.state={showResults : false};
}
```

Note: Cette ligne est propre aux classes ES6. Si l'on utilisait les composants de React classiques, nous n'aurions pas cette ligne (nous n'aurions d'ailleurs pas de constructeur) mais la fonction getInitialState. Cette dernière est appelée avant que le composant soit construit. La valeur retournée est utilisée comme valeur initiale à this.state. Cette fonction serait alors écrite de la façon suivante.

```
getInitialState(){
  return {
    showResults: false
  };
};
```

onClick

OnClick est la fonction que nous allons appeler lorsque l'utilisateur clique sur un commentaire pour mettre à jour l'état de notre classe. Simplement, nous allons tester la valeur de this.state.showResults. Si sa valeur est à false, on l'a passe à true (et inversement).
Il faut savoir que pour effectuer ce changement de valeur, la ligne suivante ne fonctionne pas.

```
this.state.showResults = false;
```

ShowResults n'est pas une variable comme les autres puisqu'elle est associée à l'état du composant. Je pense que ce choix a été fait pour des raisons de sécurité car quand l'état d'un composant est mis à jour, ce dernier est automatiquement 're-render'. C'est à dire, le composant est rechargé dans l'arbre grâce au renvoi automatique de la méthode render. Un changement d'état peut donc avoir des conséquences importantes sur un composant.

Mais alors, comment faire ?

Pour remplacer la valeur, nous devons utiliser une méthode spécifique appelée setState. Cette méthode peut mettre à jour une ou plusieurs variables d'état (this.state.anyVariable). Nous pouvons également implémenter une fonction en paramètre dont la/les valeurs retournées vont mettre à jour l'état du composant.

```
onClick(){
  if(this.state.showResults==true){
    this.setState({showResults: false});
  }
  else{
    this.setState({showResults: true});
  }
};
```

G - Header

Ce composant est l'entête de la page. Il contient le titre du site ainsi que le menu. Nous allons tout d'abord vous présenter les éléments qui constituent le menu avant de vous expliquer le composant Header. Nous faisons ce choix afin de mieux comprendre le fonctionnement de la classe Header.

Pour le menu nous avons fait le choix d'un menu glissant. Voyons ensemble comment construire ce menu.

Composant: Menu

```
class Menu extends Component{

  constructor(){
    super();
    this.state={visible : false};
  }
```

Dans la classe Menu a deux buts: elle retourne le menu actuel et active la visibilité du menu en s'appuyant sur les méthodes show et hide.
Avant de continuer, il est impératif d'initialiser l'attribut visible que nous allons utiliser dans nos méthodes. Cette initialisation se fait dans le constructeur et on lui donne la valeur false.

Show

```
show() {
  this.setState({ visible: true });
  document.addEventListener("click", this.hide.bind(this));
};
```

Show va modifier la visibilité du menu. En utilisant la méthode setState, on passe la visibilité à true. Dans le même temps, elle ajoute un listener pour fermer le menu lorsque l'utilisateur clique en dehors du menu.

Hide

```
hide() {
  document.removeEventListener("click", this.hide.bind(this));
  this.setState({ visible: false });
};
```

La fonction hide quant à elle va cacher le menu et enlever le listener précédemment ajouté.

Render

```
render (){
  return <div className="menu">
    <div className={(this.state.visible ? "visible " : "") + this.props.alignment}>{this.props.children}</div>
  </div>;
}
```

Le render est divisé en deux parties. La première va tester la valeur de notre attribut visible et renvoyer “visible” ou ‘’’’ (champ vide). C’est le lien qui est fait entre notre composant et le CSS. En fonction, de la valeur de this.state.visible, on décide d’afficher ou pas le composant (la propriété visibility en CSS a pour valeur par défault visible). La seconde partie va servir a connaître l’alignement dans lequel nous sommes. This.props.alignment peut prendre comme valeur “left” ou “right”.

Le contenu du Menu est retourné à travers this.props.children. C’est une propriété spéciale qui représente le contenu du tag HTML dans le parent. Dans notre cas, c’est l’ensemble de nos MenuItem.

Composant: MenuItem

```
class MenuItem extends Component{

  constructor(){
    super();
  }
  navigate(hash) {
    window.location.hash = hash;
  };

  render() {
    return <div className="menu-item" onClick={this.navigate.bind(this, this.props.hash)}>{this.props.children}</div>;
  }
};

export default MenuItem;
```

La classe MenuItem a deux responsabilités principales: rendre l’Item lui même et s’occuper des actions de l’utilisateur.
On crée tout d’abord une div menu-item utilisée par la feuille de style. Ensuite, on rajoute une action lorsque l’utilisateur clique qui fait appelle à la fonction navigate.
Le texte du menu item est donné à travers la propriété children.

Explications Header

```
@withStyles(styles)
class Header extends Component {

  constructor(){
    super();
    this.showLeft=this.showLeft.bind(this);
    this.showRight=this.showRight.bind(this);
  }
  showLeft() {
    this.refs.left.show();
  };

  showRight() {
    this.refs.right.show();
  };

  render() {
    return (
      <div className="Header">
        <div className="Header-container">
          <a className="Header-brand" href="/" onClick={Link.handleClick}>
            <span className="Header-brandTxt"></span>
          </a>
          <button onClick={this.showLeft}>Show Left Menu!</button>
          <button onClick={this.showRight}>Show Right Menu!</button>

          <Menu ref="left" alignment="left">
            <MenuItem hash="first-page">First Page</MenuItem>
            <MenuItem hash="second-page">Second Page</MenuItem>
            <MenuItem hash="third-page">Third Page</MenuItem>
          </Menu>

          <Menu ref="right" alignment="right">
            <MenuItem hash="first-page">First Page</MenuItem>
            <MenuItem hash="second-page">Second Page</MenuItem>
            <MenuItem hash="third-page">Third Page</MenuItem>
          </Menu>
          <Navigation className="Header-nav" />
          <div className="Header-banner">
            <h1 className="Header-bannerTitle">Fimo</h1>
            <p className="Header-bannerDesc">Find my movie</p>

          </div>
        </div>
      </div>
    );
  }

}
```

Maintenant que nous avons nos composants du menu, nous allons les intégrer. Nous vous présentons ici un menu glissant à droite et à gauche afin de vous laisser le choix de la position de votre menu.

Render

Dans la méthode render nous avons les divs qui déclarent notre header. Au sein de ces balises nous avons notre menu de gauche et celui de droite. On ajoute à la déclaration du menu deux boutons pour activer l'affichage du menu. Ces boutons sont reliés aux methodes showLeft et showRight.

Constructor

Comme pour les méthodes de la classe Comment, on initialise nos fonctions dans le constructeurs.

ShowLeft

Cette méthode appelle la fonction show de la classe Menu. Pour cela, elle passe par la référence de notre menu déclaré dans le render.

```
<Menu ref="left" alignment="left">
```

La ligne this.refs.left.show signifie que l'on récupère la fonction show du composant dont la référence est left.

ShowRight

On applique le même raisonnement pour la fonction showRight sauf que cette fois on appelle la fonction show du composant dont la référence est right.

```
<Menu ref="right" alignment="right">
```

H - FeedBack

```
class FeedBack extends Component {

  render() {
    return (
      <div className="Feedback">
        <div className="Feedback-container">

          <a className="Feedback-link" href="#"><img className="Icon" src={require('./dislike_icon.png')} width="100px" height="100px" alt="React" />
          </a>

          <span className="Feedback-spacer">|</span>
          <a className="Feedback-link"  href="#"><img className="Icon" src={require('./view_icon.png')} width="100px" height="100px" alt="React" /></a>
          <span className="Feedback-spacer">|</span>
          <a className="Feedback-link" href="#"><img className="Icon" src={require('./like_icon.png')} width="100px" height="100px" alt="React" /></a>
        </div>
      </div>
    );
  }
}

export default FeedBack;
```

Cette classe a été construite pour accueillir nos trois boutons: j'aime, je n'aime pas et j'ai déjà vu. Ces boutons sont déclarés dans la méthode render.

Note: Dans ce tutoriel, nous avons fait le choix de ne pas parler du developpement css. Nous orientons ainsi notre tutoriel uniquement sur React. Les feuilles de styles développées durant ce projet sont disponibles dans le code source.

3) Les tests utilisés

Pour nous tests, nous avons utiliser la fonction de google chrome nous permettant de tester l'adaptation de notre application sur différents appareils.

La version actuelle du système n'est pas satisfaisante en terme d'adaptation. En effet, nous avons rencontré de nombreux problèmes liés au starter kit. Il semblerait que ce kit a déjà une version de "responsivité". En conséquence, les outils que nous avons souhaité utiliser pour organiser l'adaptation de l'application ne fonctionnaient pas. Nous avons testé bootstrap, react-boostrap, react grid layout et même de simples media queries.
Il est évident que nous avons manqué une information puisque ce kit est utilisé par d'autres développeurs qui arrivent à construire des applications responsives.

Bibliographie

Android

http://www.androidhive.info/2015/04/android-getting-started-with-material-design/
http://www.androidhive.info/2015/09/android-material-design-working-with-tabs/

React

Installation:

Node: https://nodejs.org/en/download/

Python: https://www.python.org/downloads/

React Tutorial:
- Tutorial: https://facebook.github.io/react/docs/tutorial.html
- Sources: https://github.com/reactjs/react-tutorial

React Starter Kit:
- Yoeman generator: http://yeoman.io/generators/
- Sources: https://github.com/kriasoft/react-starter-kit

React-bootstrap: https://react-bootstrap.github.io/

Bootstrap: http://getbootstrap.com/

React grid layout: https://github.com/STRML/react-grid-layout

Media queries:
- https://openclassrooms.com/courses/apprenez-a-creer-votre-site-web-avec-html5-et-css3/mise-en-page-adaptative-avec-les-media-queries

Annexe

Annexe 1: React: Tableau des importations

App	import React, { PropTypes, Component } from 'react'; import styles from './App.css'; import withContext from '../../decorators/withContext'; import withStyles from '../../decorators/withStyles'; import Header from '../Header'; import Feedback from '../Feedback'; import Footer from '../Footer'; import MainContent from '../MainContent'; import CommentBox from '../CommentBox'; import BadCommentBox from '../BadCommentBox';
Header	import React, { Component } from 'react'; import styles from './Header.css'; import withStyles from '../../decorators/withStyles'; import Link from '../Link'; import Navigation from '../Navigation'; import Menu from '../Menu'; import MenuItem from '../MenuItem';
Menu	import React, { PropTypes, Component } from 'react'; import styles from './Menu.css'; import withContext from '../../decorators/withContext'; import withStyles from '../../decorators/withStyles';
MenuItem	import React, { PropTypes, Component } from 'react'; import styles from './MenuItem.css'; import withContext from '../../decorators/withContext'; import withStyles from '../../decorators/withStyles';
MainContent	import React, { Component } from 'react'; import styles from './MainContent.css'; import withStyles from '../../decorators/withStyles'; import Link from '../Link';
MainButton	import React, { Component } from 'react'; import styles from './MainButton.css'; import withStyles from '../../decorators/withStyles';
CommentBox	import React, { Component } from 'react'; import styles from './CommentBox.css'; import withStyles from '../../decorators/withStyles'; import Link from '../Link'; import CommentList from '../CommentList';
CommentList	import React, { Component } from 'react'; import styles from './CommentList.css'; import withStyles from '../../decorators/withStyles'; import Comment from '../Comment';
Comment	import React, { Component } from 'react'; import styles from './Comment.css'; import withStyles from '../../decorators/withStyles';
BadCommentBox BadCommentList BadComment	Ces classes ont la même structure d'import que les classes CommentBox, CommentList et Comment
Footer	import React, { PropTypes, Component } from 'react'; import styles from './Footer.css'; import withViewport from '../../decorators/withViewport'; import withStyles from '../../decorators/withStyles'; import Link from '../Link';

Annexe 2: Script Javascript

```
<script type="text/javascript">
$(function() {

    var comment_ul = $('.comment > li > ul'),
        comment_a  = $('.comment > li > a');

    comment_ul.hide();

    comment_a.click(function(e) {
        e.preventDefault();
        if(!$(this).hasClass('active')) {
            menu_a.removeClass('active');
            menu_ul.filter(':visible').slideUp('normal');
            $(this).addClass('active').next().stop(true,true).slideDown('normal');
        } else {
            $(this).removeClass('active');
            $(this).next().stop(true,true).slideUp('normal');
        }
    });

});
</script>
```

Nous avons vu dans la classe Comment une méthode pour ouvrir/fermer un commentaire. Nous avons développé une version Javascript classique afin de pouvoir comparer les deux versions.
D'un simple coup d'oeil, on se rend compte que React permet de programmer plus proprement. De plus, les sécurités apportées par les méthodes définies par le langage (comme setState) nous permettent de nous sentir plus confiant dans le programme que nous avons développé.