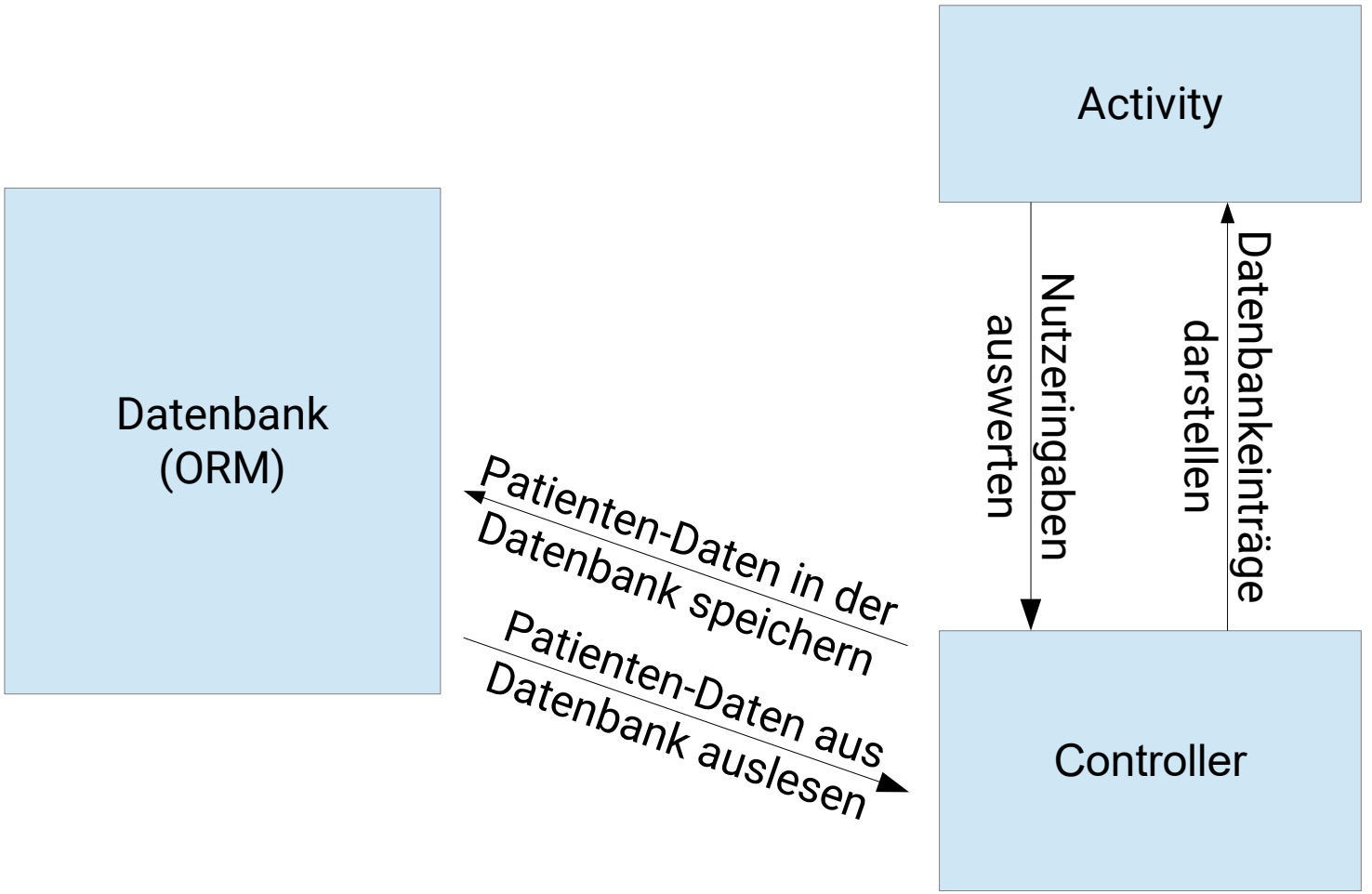
Dokumentation Pflegi3000

Fabian Kock, 27983
Rafael Hingerl, 27985

Struktur der App



Datenbank

```java
@DatabaseTable(tableName = "patient")
public class PatientEntity {
    @DatabaseField(columnName = "pId", generatedId = true) private int p_id;
    @DatabaseField(columnName = "FirstName") private String firstname;
    @DatabaseField(columnName = "LastName") private String lastname;
    @DatabaseField(columnName = "Gender") private char gender;
    @DatabaseField(columnName = "InsuranceNr") private int insuranceNumber;
    @DatabaseField(columnName = "InsuranceType", foreign = true) private InsuranceEntity insuranceEntity;
    @DatabaseField private int day;
    @DatabaseField private int month;
    @DatabaseField private int year;

    public PatientEntity() {/*ORMLite needs a default Constructor*/};
```

Die Datenbank wurde mithilfe von Object Relational Mapping (ORMLite) umgesetzt.
Die Tabellen werden in Entity-Klassen umgesetzt und auf die einzelnen Variablen wird mit getter- und setter-Methoden zugegriffen

Datenbank

Auf die einzelnen Entity-Objecte greifen wir mit Data Access Objects (DAO) zu.
Mithilfe der DaoFactory können wir die Dao's der einzelnen Entities anfordern
Oder neu erstellen, falls diese noch nicht implementiert sind

```
public Dao<PatientEntity, Integer> getPatientDAO() throws SQLException {

    if (patientDAO == null) {
        patientDAO= databaseHelper.getDao(PatientEntity.class);
    }
    return patientDAO;
}
public Dao<InsuranceEntity, Integer> getInsuranceDAO() throws SQLException {

    if(insuranceDAO == null){
        insuranceDAO = databaseHelper.getDao(InsuranceEntity.class);
    }
    return insuranceDAO;
}

public Dao<MedikamentEntity, Integer> getMedikamentDAO() throws SQLException {
```

Entity in die Datenbank speichern

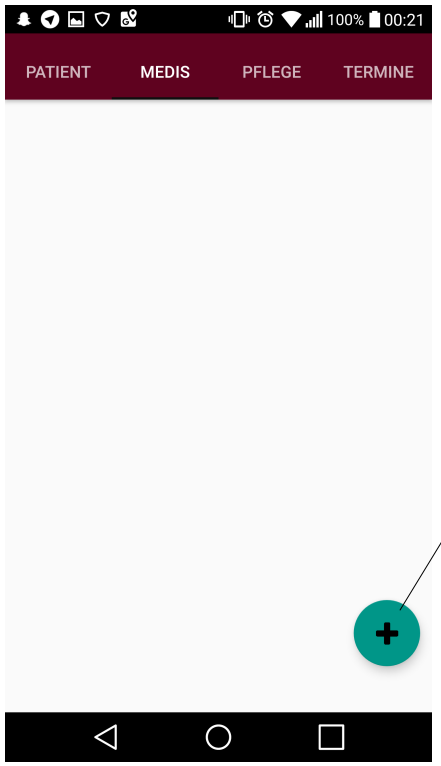
Auf diese Weise wird ein Medikament Entity-Objekt erzeugt und in die Datenbank gespeichert. Das Prinzip der „Patient hinzufügen“ ist identisch.
- neues Entity-Objekt erzeugen
- Werte zuweisen
- Mit DAO in die Datenbank speichern


```
try {

    Dao<MedikamentEntity, Integer> mDao = daofactory.getMedikamentDAO();
    Dao<PatientMedikamentConnection, Integer> pmDao = daofactory.getPatientMedikamentDAO();
    Dao<PatientEntity, Integer> pDao = daofactory.getPatientDAO();

    MedikamentEntity newMedikament = new MedikamentEntity(mName, mDosis, hour, minute);

    PatientMedikamentConnection newConnection = new PatientMedikamentConnection(pDao.queryForId(patientID), newMedikament);

    mDao.create(newMedikament);
    pmDao.create(newConnection);

} catch (SQLException e) {
    e.printStackTrace();
}
```

Pdao.create(Entity) zum neu hinzufügen
Pdao.update(Entity) zum aktualisieren von bereits bestehenden Entities

Sicht-Wechsel

In unseren einzelnen Fragmenten haben Wir meistens eine edit und eine show Ansicht. Durch das drücken eines Buttons wechselt die Sicht von show zu edit und umgekehrt.



In dem Medikament Fragment z.B. wird durch das drücken des FloatingActionButtons In die edit Sicht gewechselt. In dieser Sicht können änderungen vorgenommen werden

```
public void switchAddShow() {

    ConstraintLayout show = this.mainactivity.findViewById(R.id.show_medikament);
    ConstraintLayout add = this.mainactivity.findViewById(R.id.add_medikament);

    if (add.getVisibility() == View.INVISIBLE) {
        show.setVisibility(View.INVISIBLE);
        add.setVisibility(View.VISIBLE);
    } else {
        add.setVisibility(View.INVISIBLE);
        show.setVisibility(View.VISIBLE);
    }
}
```

Aufbau der Sichten

```
<android.support.constraint.ConstraintLayout
    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/show_medikament"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".activities.show_patient_activity.ShowPatientFragmente.MedikamentFragment">

    <ListView
        android:id="@+id/medikamente_list"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:layout_marginTop="20dp"/>

    <android.support.design.widget.FloatingActionButton
        android:id="@+id/fab"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:src="@drawable/plus"
        android:layout_marginBottom="39dp"
        android:layout_marginRight="10dp"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintRight_toRightOf="parent"/>


</android.support.constraint.ConstraintLayout>
```

Hier haben wir die show Sicht des Medikamenten Fragment. Die Sicht Ist innerhalt eines ConstraintLayouts Containers und so können einfach alle Elemente unsichtbar und die der anderen Sicht Sichtbar gemacht werden

Bsp. Appointment Fragment (AppointmentFragment.java)

```
@SuppressLint("ValidFragment")
public AppointmentFragment(int p_id, ControllerAppointmentFragment c) {
    // Required empty public constructor
    this.patient_id = p_id;
    this.controller = c;
}
```

Fragment und Controller wird in ShowPatient Activity erstellt und übergeben. Außerdem wird die ID des Patienten in der Datenbank übergeben

```
this.add_button.setOnClickListener(this.controller.getBtnListener());
this.save_button.setOnClickListener(this.controller.getBtnListener());
```

Button Listener wird im Controller erstellt und über getter geholt und den Buttons übergeben

Bsp. Appointment Fragment (AppointmentFragment.java)

```
//Calendar
this.caldroidFragment = new CaldroidFragment();
Bundle args = new Bundle();
Calendar cal = Calendar.getInstance();
args.putInt(CaldroidFragment.MONTH, cal.get(Calendar.MONTH) + 1);
args.putInt(CaldroidFragment.YEAR, cal.get(Calendar.YEAR));
caldroidFragment.setArguments(args);

List<AppointmentEntity> aEntities = this.controller.getAllAppointments();
this.eventMap = new HashMap<>();
ColorDrawable red = new ColorDrawable(getResources().getColor(R.color.red));
Date tempDate;

for(int i = 0; i < aEntities.size(); i++){

    Log.i( tag: "appointments",  msg: aEntities.get(i).getTName() + " "
                               + aEntities.get(i).getTimestamp() + " "
                               + aEntities.get(i).getTDescription() + " "
                               + aEntities.get(i).getTAddress());
    tempDate = new Date(aEntities.get(i).getTimestamp());
    caldroidFragment.setTextColorForDate(R.color.red, tempDate);
    //this.eventMap.put(tempDate, red);
}
//caldroidFragment.setBackgroundDrawableForDates(this.eventMap);
caldroidFragment.setCaldroidListener(this.controller.getCalendarListener());

android.support.v4.app.FragmentTransaction t = getActivity().getSupportFragmentManager().beginTransaction();
t.replace(R.id.calendar_view, caldroidFragment);
t.commit();

return view;
}
```

Holen aller Appointmenteinträge aus der Datenbank über den Controller (dort über DAOFactory)

Von allen Entities die zeitlichen Daten lesen und die entsprechenden Daten im Kalender farblich markieren

Listener für Interaktion mit dem Kalender setzen

Bsp. AppointmentFragment.java (AppointmentFragmentButtonListener.java)

```java
@Override
public void onClick(View view) {

    switch(view.getId()){

        case R.id.add_appointment_button:
            Log.i( tag: "appointments",  msg: "sel date: " + this.controller.getSelectedDate());
            if(this.controller.getSelectedDate() != null) {
                if (this.controller != null)
                    this.controller.changeVisibility();
            }
            else{
                Toast t = Toast.makeText(view.getContext(),  text: "Bitte ein Datum auswählen", Toast.LENGTH_SHORT);
                t.show();
            }
            break;

        case R.id.save_appointment_button:

            if(!this.controller.processInput()){
                Toast t = Toast.makeText(view.getContext(),  text: "Bitte alle Felder ausfüllen", Toast.LENGTH_SHORT);
                t.show();
            }
            else{
                if(this.controller != null)
                    this.controller.changeVisibility();
                this.controller.repaintCalendar();
            }
            break;

        default:
            break;
    }
}
```

Button Klick Funktionen implementieren. View wird nur gewechselt wenn ein gültiges Datum ausgewählt wurde.

View wird nur gewechselt falls etwas in die Datenbank geschrieben wurde (processInput() = true)

Bsp. Appointment Fragment (ControllerAppointmentFragment.java)

```java
public boolean processInput(){

    AppointmentFragment fragment = this.activity.getfAppointment();

    try{

        Dao<AppointmentEntity, Integer> aDao = this.daoFactory.getAppointmentDAO();
        Dao<PatientEntity, Integer> pDao = this.daoFactory.getPatientDAO();

        if(!fragment.getDescription().matches( regex: "") &&
            !fragment.getName().matches( regex: "") &&
            this.selectedDate.getTime() > 0) {

            PatientEntity patient = pDao.queryForId(this.activity.getPatient_id());

            AppointmentEntity appointmentEntity = new AppointmentEntity(fragment.getName(), this.selectedDate.getTime(), fragment.getDescription(), fragment.getAddress(), patient);
            aDao.create(appointmentEntity);
            Log.i( tag: "appointments",  msg: "Appointment created");

            return true;
        }
        else{
            return false;
        }

    }catch(SQLException e){
        e.printStackTrace();

        return false;
    }
}
```

Überprüfen ob Input Felder leer sind

Wenn nein dann hole aus der datenbank zuerst den Patienten mit der jeweiligen ID und schreibe einen neuen Appointment Datensatz in die Datenbank mit dem Patienten als Fremdschlüssel