

1 AmazingRace Android Anwendung

Folgende Dokumentation dokumentiert die Entwicklung der mobilen Anwendung *AmazingRace*, die in *Java 7* für die Android-Plattform *API-Level 22*, *Android 5.1.1* entwickelt wurde.

1.1 Lösungsidee

Die Anwendung soll in auf drei Aktivitäten aufgeteilt werden, wobei die Login-Aktivität als Launcher-Aktivität definiert werden soll. Über diese Aktivität wird die Anwendung gestartet. Nach einem erfolgreichem Login soll diese Aktivität beendet werden, da es keinen Grund gibt warum ein Benutzer wieder auf diese Aktivität zurückkehren sollte.

Nach dem Login wird der Benutzer zu einer Übersicht aller Routen weitergeleitet, wo er eine Route auswählen kann. In dieser Ansicht sollen die einzelnen Routen mit einem Kontext-Menü versehen werden, die je nach Zustand der Route folgende Optionen bereitstellen soll.

1. *Play (Es gibt einen offenen Checkpoint)*
Wechselt zur Checkpoint-Aktivität.
2. *Open (Route bereits beendet)*
Wechselt zur Checkpoint-Aktivität
3. *Reset (Wenn zumindest ein Checkpoint besucht wurde)*
Setzt diese Route zurück und verbleibt auf dieser Aktivität

Ebenfalls soll eine Action-Bar implementiert werden, die folgende Optionen bereitstellen soll.

1. *Reset*
Setzt alle Routen zurück
2. *Reload*
Lädt alle Routen neu.
3. *Close*
Öffnet einen Dialog der im Falle einer Bestätigung die Anwendung beendet

Für alle Ladeoperationen soll ein *ProgressDialog* angezeigt werden, der eine dementsprechende Meldung beinhaltet (Z.b.: Load routes ...). Der verwendete Service soll über ein Interface abgebildet werden, somit beinhaltet die benutzenden Klasse keine Referenzen auf die konkrete Implementierung. Die Implementierung soll über eine *Factory* bereitgestellt werden, die die Service Instanz als *Singleton* verwaltet, da hier nur eine Instanz benötigt wird. Des Weiteren sollen *View-Models* implementiert werden, die die gesamte View-Logik beinhalten und keine Referenzen auf Android-Ressourcen beinhalten sollen.

Die Aktivitäten sollen die gesamte Interaktionslogik beinhalten und die möglichen *ServiceExceptions* der Serviceaufrufe, die über die View-Model erfolgen sollen und von diesen weitergereicht werden, behandeln, da hierbei Referenzen auf Android-Ressourcen benötigt werden. Diese *ServiceException* soll in einer abstrakten Basisklasse behandelt werden, von der alle Aktivitäten ableiten sollen. Somit ist diese Fehlerbehandlung (Anzeige für den Benutzer) in einer Klasse gekapselt und trotzdem können die Aktivitäten ihrerseits Fehlerbehandlungen durchführen.

Übung 3

1.1.1 Architektur

Folgend ist die Architektur der Anwendung *AmazingRace* dokumentiert.

Folgende Abbildung zeigt die Hierarchie der Klasse *AbstractActivity* die als Basisklasse aller Aktivitäten fungiert und gemeinsam genutzte Ressourcen und Konstanten bereitstellt.

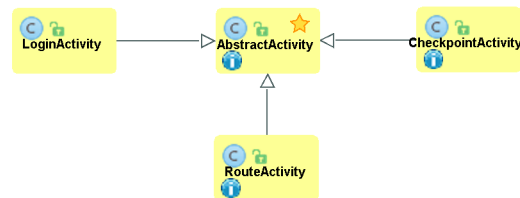


Abbildung 1: Klassenhierarchie der Klasse *AbstractActivity*

Folgende Abbildung zeigt die Paketstruktur der Anwendung *AmazingRace*. Als Wurzelnamensraum wurde *at.fh.ooe.moc5.amazingrace* gewählt unter dem sich alle Ressourcen befinden.

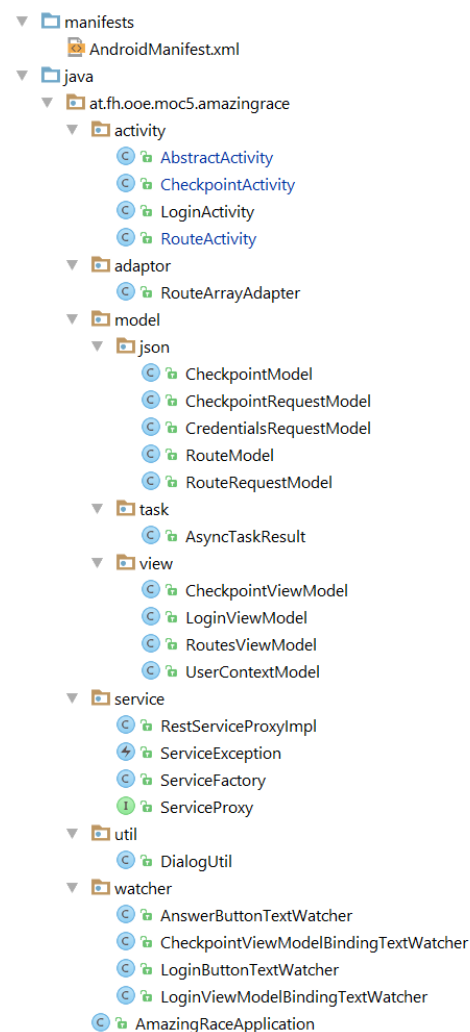


Abbildung 2: Paketstruktur der Anwendung

Übung 3

Folgende Abbildung zeigt die Ressourcenverzeichnisse in denen alle Ressourcen wie *Layout* usw. liegen. Es wurde hierbei darauf verzichtet, die Ressourcen für mehrere Auflösungen und Sprachen zu definieren.

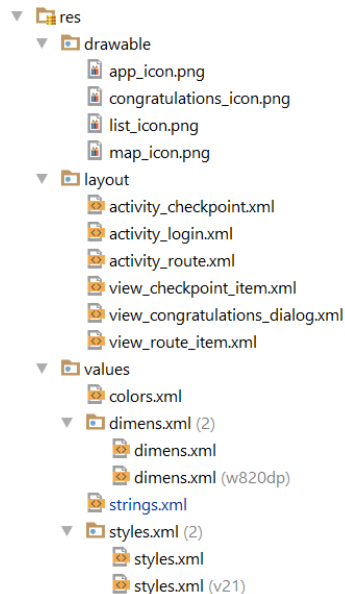


Abbildung 3: Ressourcen der Anwendung

Des Weiteren wurde eine Klasse *AmazingRaceApplication* eingeführt, die die nicht abgefangene Exceptions behandelt, den Benutzerkontext hält und gemeinsam verwendete Konstanten definiert.

1.1.2 Setup

Da ein Google Android API-Key verwendet wird wurde ein eigenes Zertifikat erstellt welches bei Google-Developer-Console registriert wurde. Dieses Zertifikat muss im Android-Studio bei der *Debug* und *Release* Konfiguration eingestellt werden, da ansonsten Google Maps nicht funktionieren wird. Es sollte zwar ein Zertifikat für die *Debug* und *Release* Konfiguration erstellt werden, aber es wurde hierbei darauf verzichtet und es wird nur ein Zertifikat für beide Konfigurationen verwendet.

1. *Alias*: amazingrace
2. *Key-Password*: amazingrace
3. *Keystore-Password*: amazingrace

Der Keystore befindet sich im Android-Studio-Projekt Wurzelverzeichnis und trägt den Name *amazingrace.jks*

Des Weiteren wurde folgender Emulator verwendet

1. *Device*: Nexus 5
2. *API-Level*: 22
3. *Architektur*: x86_64 (schneller und *ARM* hatte falsche Play-Service Version)

Die Applikation wurde ebenfalls auf eine *Samsung S6-Edge* getestet.

Übung 3

1.2 Source

Im folgenden ist der gesamte Source der Anwendung aufgeführt.

1.2.1 Manifest

Listing 1: AndroidManifest.xml

```

1  <?xml version="1.0" encoding="utf-8"?>
2  <manifest xmlns:android="http://schemas.android.com/apk/res/android"
3      package="at.fh.ooe.moc5.amazingrace">
4
5      <!-- Permissions -->
6      <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
7      <uses-permission android:name="android.permission.ACCESS_WIFI_STATE" />
8      <uses-permission android:name="android.permission.INTERNET" />
9      <uses-permission android:name="android.permission.LOCATION_HARDWARE" />
10     <uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION" />
11     <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
12
13     <application
14         android:name=".AmazingRaceApplication"
15         android:allowBackup="true"
16         android:icon="@drawable/app_icon"
17         android:label="@string/application_title"
18         android:supportsRtl="true"
19         android:theme="@style/AppTheme">
20
21         <!-- Main Activity -->
22         <activity
23             android:name=".activity.LoginActivity"
24             android:label="@string/application_title">
25             <intent-filter>
26                 <action android:name="android.intent.action.MAIN" />
27                 <category android:name="android.intent.category.LAUNCHER" />
28             </intent-filter>
29         </activity>
30
31         <!-- explicit accessible activities -->
32         <activity
33             android:name=".activity.RouteActivity"
34             android:label="@string/activity_title_route" />
35         <activity
36             android:name=".activity.CheckpointActivity"
37             android:label="@string/activity_title_checkpoint" />
38
39         <!-- Meta-Data for google maps -->
40         <meta-data
41             android:name="com.google.android.gms.version"
42             android:value="@integer/google_play_services_version" />
43         <meta-data
44             android:name="com.google.android.maps.v2.API_KEY"
45             android:value="@string/google_maps_key" />
46     </application>
47 </manifest>

```

Übung 3

1.2.2 Aktivitäten

Listing 2: AmazingRaceApplication.java

```

1 package at.fh.ooe.moc5.amazingrace;
2
3 import android.app.Application;
4 import android.util.Log;
5 import android.widget.Toast;
6
7 import at.fh.ooe.moc5.amazingrace.model.view.UserContextModel;
8
9 /**
10  * Created by Thomas on 12/24/2015.
11  * This is the custom application implementation which handles uncaught exceptions.
12  * The application will get killed on a an uncaught exception.
13  */
14 public class AmazingRaceApplication extends Application {
15
16     public UserContextModel loggedUser;
17
18     public static final String INTENT_EXTRA_ROUTE = "INTENT_EXTRA_ROUTE";
19
20     /**
21      * Registers an exception handler for uncaught exceptions.
22      */
23     public void onCreate() {
24         super.onCreate();
25         // Setup handler for uncaught exceptions.
26         Thread.setDefaultUncaughtExceptionHandler(new Thread.UncaughtExceptionHandler() {
27             @Override
28             public void uncaughtException(Thread thread, Throwable e) {
29                 handleUncaughtException(thread, e);
30             }
31         });
32     }
33
34     /**
35      * Handles uncaught exceptions which are considered critical, therefore the application will
36      * exist with an error.
37      * @param thread the current thread the exception occurred
38      * @param e the occurred exception
39      */
40     public void handleUncaughtException(Thread thread, Throwable e) {
41         e.printStackTrace();
42         Log.e("error", "Uncaught exception caught", e);
43         System.exit(1);
44     }
45
46     /**
47      * gets the logged user context
48      *
49      * @return the user context representing the logged user
50      */
51     public UserContextModel getLoggedUser() {
52         return loggedUser;
53     }
54
55     /**
56      * Sets the logged user which can be accessed by all implemented activities.
57      *
58      * @param loggedUser the logged user

```

Übung 3

```
59     */
60     public void setLoggedInUser(UserContextModel loggedInUser) {
61         this.loggedInUser = loggedInUser;
62     }
63 }
```

Listing 3: AbstractActivity.java

```

1 package at.fh.ooe.moc5.amazingrace.activity;
2
3 import android.app.ProgressDialog;
4 import android.content.Context;
5 import android.content.DialogInterface;
6 import android.content.Intent;
7 import android.net.ConnectivityManager;
8 import android.net.NetworkInfo;
9 import android.os.Bundle;
10 import android.support.v7.app.AppCompatActivity;
11 import android.util.Log;
12 import android.widget.Toast;
13
14 import java.io.Serializable;
15 import java.util.Objects;
16
17 import at.fh.ooe.moc5.amazingrace.AmazingRaceApplication;
18 import at.fh.ooe.moc5.amazingrace.R;
19 import at.fh.ooe.moc5.amazingrace.service.ServiceException;
20 import at.fh.ooe.moc5.amazingrace.util.DialogUtil;
21
22 /**
23  * Created by Thomas on 12/28/2015.
24  * <p/>
25  * This is the base class for all activities which forces the activity to define one view model it
26  * ↪ uses.
27  * This class provides common functionality usable by all activities.
28  */
29 public class AbstractActivity<T extends Serializable> extends AppCompatActivity {
30
31     protected T viewModel;
32     protected boolean restored = Boolean.FALSE;
33     protected AmazingRaceApplication application;
34     protected Boolean validViewModel = Boolean.TRUE;
35
36     private ProgressDialog progress;
37
38     private static final String VIEW_MODEL = "AbstractActivity#VIEW_MODEL";
39
40     // region Constants
41
42     /**
43      * Enumeration for specifying the available menu group ids
44      */
45     public static enum MenuGroup {
46         ROUTE_ITEM(0),
47         OPTIONS(1);
48
49         public final int value;
50
51         private MenuGroup(int value) {
52             this.value = value;
53         }
54
55         public static MenuGroup getMenuGroupForValue(int value) {
56             for (MenuGroup group : MenuGroup.values()) {
57                 if (group.value == value) {
58                     return group;
59                 }
60             }
61         }
62     }
63
64     // endregion
65 }

```

Übung 3

```

60
61         throw new IllegalArgumentException("MenuGroup with value '" + value + "' not found");
62     }
63 }
64
65 /**
66  * Enumeration for specifying the available menu ids
67  */
68 public static enum MenuId {
69     PLAY(0),
70     RESET(1),
71     RESET_ALL_ROUTES(2),
72     CLOSE(3),
73     RELOAD(4);
74
75     public final int value;
76
77     private MenuId(int value) {
78         this.value = value;
79     }
80
81     public static MenuId getMenuIdForValue(int value) {
82         for (MenuId id : MenuId.values()) {
83             if (id.value == value) {
84                 return id;
85             }
86         }
87
88         throw new IllegalArgumentException("MenuGroup with value '" + value + "' not found");
89     }
90 }
91 //endregion
92
93 //region Lifecycle Methods
94
95 /**
96  * Initializes the backed application so that it can be accessed by the concrete activity
97  ↪ application.
98  * It also tries to get the saved view model from the savedInstanceState which can be null
99  ↪ here if haven't been saved before.
100  *
101  * @param savedInstanceState
102  */
103 @Override
104 protected void onCreate(Bundle savedInstanceState) {
105     super.onCreate(savedInstanceState);
106     application = (AmazingRaceApplication) getApplication();
107     if (savedInstanceState != null) {
108         viewModel = (T) savedInstanceState.get(VIEW_MODEL);
109         restored = (viewModel != null);
110     }
111 }
112
113 /**
114  * Re-initializes the application on resume of the activity.
115  */
116 @Override
117 protected void onResume() {
118     super.onResume();
119     application = (AmazingRaceApplication) getApplication();
120     if (viewModel == null) {
121         validViewModel = Boolean.FALSE;
122         openInvalidUserAccountDialog();
123     }
124 }

```


Übung 3

```

121     }
122 }
123
124 /**
125  * Saves the backed view model.
126  *
127  * @param outState the out coming state
128  */
129 @Override
130 protected void onSaveInstanceState(Bundle outState) {
131     super.onSaveInstanceState(outState);
132     outState.putSerializable(VIEW_MODEL, viewModel);
133 }
134 //endregion
135
136 /**
137  * Handles a service exception by displaying a toast with a proper message
138  *
139  * @param exception the occurred ServiceException
140  * @return true if handled here, false otherwise, which indicates that no error code or an
141  ↪ unknown error occurred was set.
142  */
143 protected boolean handleServiceException(ServiceException exception) {
144     Objects.requireNonNull(exception);
145     boolean handled = Boolean.FALSE;
146     ServiceException.ServiceErrorCode errorCode = exception.getErrorCode();
147     if (errorCode != null) {
148         switch (errorCode) {
149             case INVALID_REQUEST:
150                 Toast.makeText(AbstractActivity.this, R.string.error_request_invalid,
151                 ↪ Toast.LENGTH_LONG).show();
152                 handled = Boolean.TRUE;
153                 break;
154             case TIMEOUT:
155                 Toast.makeText(AbstractActivity.this, R.string.error_request_timeout,
156                 ↪ Toast.LENGTH_LONG).show();
157                 handled = Boolean.TRUE;
158                 break;
159             case UNKNOWN:
160                 Toast.makeText(AbstractActivity.this, R.string.error_unknown,
161                 ↪ Toast.LENGTH_LONG).show();
162                 handled = Boolean.TRUE;
163                 break;
164             case INVALID_CREDENTIALS:
165                 openInvalidUserAccountDialog();
166                 handled = Boolean.TRUE;
167         }
168     } else {
169         Toast.makeText(AbstractActivity.this, R.string.error_unknown,
170         ↪ Toast.LENGTH_LONG).show();
171     }
172     Log.e(this.getClass().getSimpleName(), "ServiceException occurred", exception);
173
174     return handled;
175 }
176
177 /**
178  * Opens the invalid user dialog which indicates that the user has become invalid during
179  ↪ application invocation.
180  * After the user clicked ok the user will be logged out.
181  */
182 protected void openInvalidUserAccountDialog() {
183     DialogUtil.createErrorDialog(AbstractActivity.this, R.string.error_no_network, new
184     ↪ DialogInterface.OnClickListener() {

```

Übung 3

```

178         @Override
179         public void onClick(DialogInterface dialog, int which) {
180             dialog.dismiss();
181             application.setLoggedUser(null);
182             finishAffinity();
183             startActivity(new Intent(AbstractActivity.this, LoginActivity.class));
184         }
185     }).show();
186 }
187
188 /**
189  * This method checks if an internet connection is available for the application.
190  * If a network connection is not available then a dialog is shown to notify the user.
191  *
192  * @return true if network is available false otherwise
193  * @see AbstractActivity#isOnline()
194  */
195 protected boolean checkAndDisplayAvailableNetwork() {
196     final boolean online = isOnline();
197     if (!online) {
198         DialogUtil.createErrorDialog(AbstractActivity.this, R.string.error_no_network, new
199             ↳ DialogInterface.OnClickListener() {
200                 @Override
201                 public void onClick(DialogInterface dialog, int which) {
202                     dialog.dismiss();
203                 }
204             }).show();
205     }
206     return online;
207 }
208
209 /**
210  * Opens an dialog to ask the user if he really wants to quit the application.
211  */
212 protected void openCloseApplicationDialog() {
213     DialogUtil.createYesNoAlertDialog(AbstractActivity.this, R.string.dialog_title_warning,
214         ↳ R.string.warning_want_quit, new DialogInterface.OnClickListener() {
215             @Override
216             public void onClick(DialogInterface dialog, int which) {
217                 dialog.dismiss();
218                 switch (which) {
219                     case DialogInterface.BUTTON_POSITIVE:
220                         application.setLoggedUser(null);
221                         finishAffinity();
222                         break;
223                 }
224             }
225         }).show();
226 }
227
228 /**
229  * Opens a progress dialog with an custom message.
230  *
231  * @param messageId the resource id of the message
232  */
233 protected void openProgressDialog(int messageId) {
234     progress = ProgressDialog.show(AbstractActivity.this, "", getString(messageId),
235         ↳ Boolean.TRUE);
236 }
237
238 /**
239  * Closes the progress dialog.
240  */

```

Übung 3

```
238     protected void closeProgressDialog() {
239         if (progress != null) {
240             progress.dismiss();
241             progress = null;
242         }
243     }
244
245     /**
246      * Helper method for checking if a network connection is available.
247      *
248      * @return true if a network connection is available, false otherwise
249      */
250     protected boolean isOnline() {
251         ConnectivityManager cm =
252             (ConnectivityManager) getSystemService(Context.CONNECTIVITY_SERVICE);
253         NetworkInfo netInfo = cm.getActiveNetworkInfo();
254         return netInfo != null && netInfo.isConnected();
255     }
256 }
```

Listing 4: LoginActivity.java

```

1 package at.fh.ooe.moc5.amazingrace.activity;
2
3 import android.content.Intent;
4 import android.os.*;
5 import android.view.View;
6 import android.widget.Button;
7 import android.widget.EditText;
8 import android.widget.Toast;
9
10 import at.fh.ooe.moc5.amazingrace.R;
11 import at.fh.ooe.moc5.amazingrace.model.task.AsyncTaskResult;
12 import at.fh.ooe.moc5.amazingrace.model.view.LoginViewModel;
13 import at.fh.ooe.moc5.amazingrace.model.view.UserContextModel;
14 import at.fh.ooe.moc5.amazingrace.service.ServiceException;
15 import at.fh.ooe.moc5.amazingrace.watcher.LoginButtonTextWatcher;
16 import at.fh.ooe.moc5.amazingrace.watcher.LoginViewModelBindingTextWatcher;
17
18 /**
19  * This class represents the login activity where an user need to login.
20  */
21 public class LoginActivity extends AbstractActivity<LoginViewModel> implements
22     ↳ View.OnClickListener {
23
24     // region Lifecycle Methods
25
26     /**
27      * Sets acitivites content layout and initializes the view model
28      *
29      * @param savedInstanceState
30      */
31     @Override
32     protected void onCreate(Bundle savedInstanceState) {
33         super.onCreate(savedInstanceState);
34         setContentView(R.layout.activity_login);
35         // Here we want always have an new view model
36         viewModel = new LoginViewModel();
37     }
38
39     /**
40      * Prepares the activity views if view model is valid.
41      */
42     @Override
43     protected void onResume() {
44         super.onResume();
45         // View model will always be valid here
46         prepareViews();
47     }
48
49     /**
50      * Open dialog when back button is pressed on this activity.
51      *
52      * @see AbstractActivity#openCloseApplicationDialog();
53      */
54     @Override
55     public void onBackPressed() {
56         openCloseApplicationDialog();
57     }
58     // endregion
59
60     //region Listeners

```

Übung 3

```

60
61  /**
62   * Onclick implementation which
63   *
64   * @param v
65   */
66  public void onClick(View v) {
67      switch (v.getId()) {
68          case R.id.loginBtn:
69              loginUser();
70              break;
71          case R.id.cancelBtn:
72              openCloseApplicationDialog();
73              break;
74      }
75  }
76  //endregion
77
78
79  //region Helper
80
81  /**
82   * Prepares the activity views.
83   */
84  private void prepareViews() {
85      // prepare username view
86      EditText username = (EditText) findViewById(R.id.usernameEdTxt);
87      username.addTextChangedListener(new LoginViewModelBindingTextWatcher(viewModel,
88          ↪ username));
89      username.addTextChangedListener(new LoginButtonTextWatcher(this, viewModel));
90      username.setText(viewModel.getPassword());
91
92      // prepare password view
93      EditText password = (EditText) findViewById(R.id.passwordEdTxt);
94      password.addTextChangedListener(new LoginViewModelBindingTextWatcher(viewModel,
95          ↪ password));
96      password.addTextChangedListener(new LoginButtonTextWatcher(this, viewModel));
97      username.setText(viewModel.getPassword());
98
99      // Prepare login button
100      Button loginButton = (Button) findViewById(R.id.loginBtn);
101      loginButton.setOnClickListener(this);
102
103      // prepare cancel button
104      Button cancelButton = (Button) findViewById(R.id.cancelBtn);
105      cancelButton.setOnClickListener(this);
106  }
107
108  /**
109   * Resets the views by setting string value to empty string.
110   */
111  private void resetView() {
112      EditText username = (EditText) findViewById(R.id.usernameEdTxt);
113      EditText password = (EditText) findViewById(R.id.passwordEdTxt);
114
115      // Could be called on non visible activity
116      if ((username != null) && (password != null)) {
117          username.setText("");
118          password.setText("");
119      }
120  }
121
122  /**

```

Übung 3

```

121     * Performs asynchronous login service call.
122     */
123     private void loginUser() {
124         if (checkAndDisplayAvailableNetwork()) {
125             new AsyncTask<Object, Object, AsyncTaskResult<UserContextModel>>() {
126                 @Override
127                 protected AsyncTaskResult<UserContextModel> doInBackground(Object... params) {
128                     UserContextModel model = null;
129                     Exception exception = null;
130                     try {
131                         model = viewModel.loginAction();
132                     } catch (Exception e) {
133                         exception = e;
134                     }
135
136                     return new AsyncTaskResult<UserContextModel>(model, exception);
137                 }
138
139                 @Override
140                 protected void onPreExecute() {
141                     super.onPreExecute();
142                     openProgressDialog(R.string.progress_login);
143                 }
144
145                 @Override
146                 protected void onPostExecute(AsyncTaskResult<UserContextModel> result) {
147                     super.onPostExecute(result);
148                     closeProgressDialog();
149                     // Exception occurred
150                     if (result.exception != null) {
151                         resetView();
152                         if (result.exception instanceof ServiceException) {
153                             handleServiceException(((ServiceException) result.exception));
154                         } else {
155                             Toast.makeText(LoginActivity.this, R.string.error_unknown,
156                                     ↪ Toast.LENGTH_SHORT).show();
157                         }
158                     }
159                     // Login failed due to invalid credentials
160                     else if (result.result == null) {
161                         resetView();
162                         Toast.makeText(LoginActivity.this, R.string.error_login_failed,
163                                 ↪ Toast.LENGTH_SHORT).show();
164                     }
165                     // Login ok
166                     else {
167                         application.setLoggedUser(result.result);
168                         final Intent intent = new Intent(LoginActivity.this, RouteActivity.class);
169                         startActivity(intent);
170                         // Finish this activity because no need to get back to this one
171                         finish();
172                     }
173                 }
174             }.execute();
175         }
176         // endregion

```

Listing 5: RouteActivity.java

```

1 package at.fh.ooe.moc5.amazingrace.activity;
2
3 import android.content.Intent;
4 import android.os.AsyncTask;
5 import android.os.Bundle;
6 import android.view.ContextMenu;
7 import android.view.Menu;
8 import android.view.MenuItem;
9 import android.view.View;
10 import android.widget.AdapterView;
11 import android.widget.ListView;
12 import android.widget.Toast;
13
14 import java.util.List;
15
16 import at.fh.ooe.moc5.amazingrace.AmazingRaceApplication;
17 import at.fh.ooe.moc5.amazingrace.R;
18 import at.fh.ooe.moc5.amazingrace.adapter.RouteArrayAdapter;
19 import at.fh.ooe.moc5.amazingrace.model.json.RouteModel;
20 import at.fh.ooe.moc5.amazingrace.model.task.AsyncTaskResult;
21 import at.fh.ooe.moc5.amazingrace.model.view.RoutesViewModel;
22 import at.fh.ooe.moc5.amazingrace.service.ServiceException;
23
24 /**
25  * This class represents the route activity which lists all available routes to the user.
26  */
27 public class RouteActivity extends AbstractActivity<RoutesViewModel> implements
↳ AdapterView.OnItemClickListener {
28
29     //region Lifecycle Methods
30
31     /**
32      * Initializes this activity.
33      *
34      * @param savedInstanceState
35      */
36     @Override
37     protected void onCreate(Bundle savedInstanceState) {
38         super.onCreate(savedInstanceState);
39         setContentView(R.layout.activity_route);
40         // Only if user is logged and view model hasn't been restored from bundle
41         if ((application.getLoggedUser() != null) && (!restored)) {
42             viewModel = new RoutesViewModel(application.getLoggedUser());
43         }
44         // Register route list for context menu
45         registerForContextMenu(findViewById(R.id.listRoute));
46     }
47
48     /**
49      * Prepares the activity views if view model is valid
50      */
51     @Override
52     protected void onResume() {
53         super.onResume();
54         if (validViewModel) {
55             prepareView();
56             loadRoutes();
57         }
58     }
59

```

Übung 3

```

60  /**
61   * Opens application close dialog if back button is pressed on this activity.
62   *
63   * @see AbstractActivity#openCloseApplicationDialog();
64   */
65  @Override
66  public void onBackPressed() {
67      super.onBackPressed();
68      openCloseApplicationDialog();
69  }
70
71  /**
72   * Creates the context menu items for the list view.
73   *
74   * @param menu    the menu to add options too.
75   * @param v        the view to add options for
76   * @param menuInfo the menuinfo related to the view
77   */
78  @Override
79  public void onCreateContextMenu(ContextMenu menu, View v, ContextMenu.ContextMenuInfo
    ↪ menuInfo) {
80      super.onCreateContextMenu(menu, v, menuInfo);
81      // Creates options for list view
82      if (R.id.listRoute == v.getId()) {
83          onCreateRouteListContextMenu(menu, (ListView) v, (AdapterView.AdapterContextMenuInfo)
    ↪ menuInfo);
84      }
85  }
86
87  /**
88   * Creates the menu items for the application action bar
89   *
90   * @param menu the menu to add options too
91   * @return true
92   */
93  @Override
94  public boolean onCreateOptionsMenu(Menu menu) {
95      menu.add(MenuGroup.OPTIONS.value, MenuId.RESET_ALL_ROUTES.value, 0,
    ↪ R.string.action_reset_all);
96      menu.add(MenuGroup.OPTIONS.value, MenuId.RELOAD.value, 0, R.string.action_reload);
97      menu.add(MenuGroup.OPTIONS.value, MenuId.CLOSE.value, 0, R.string.action_close);
98      return Boolean.TRUE;
99  }
100
101  /**
102   * Handles the menu options select and delegates to the proper method.
103   *
104   * @param item the menu item which has been selected
105   * @return
106   */
107  @Override
108  public boolean onOptionsItemSelected(MenuItem item) {
109      switch (MenuId.getMenuIdForValue(item.getItemId())) {
110          case RESET_ALL_ROUTES:
111              resetRoute(null);
112              return Boolean.TRUE;
113          case RELOAD:
114              loadRoutes();
115              return Boolean.TRUE;
116          case CLOSE:
117              openCloseApplicationDialog();
118              return Boolean.TRUE;
119          default:

```


Übung 3

```

120         return super.onOptionsItemSelected(item);
121     }
122 }
123
124 //endregion
125
126 //region Utilities
127
128 /**
129  * Creates the menu items for the list view
130  *
131  * @param menu    the context menu to add items too
132  * @param v        the list view to create items for
133  * @param menuInfo the list view related menu info
134  */
135 private void onCreateRouteListContextMenu(ContextMenu menu, ListView v,
136     AdapterView.AdapterContextMenuInfo menuInfo) {
137     RouteModel route = (RouteModel) v.getItemAtPosition(menuInfo.position);
138     if (route.getNextCheckpoint() != null) {
139         menu.add(MenuGroup.ROUTE_ITEM.value, MenuId.PLAY.value, 0, R.string.action_play);
140         if (!route.getVisitedCheckpoints().isEmpty()) {
141             menu.add(MenuGroup.ROUTE_ITEM.value, MenuId.RESET.value, 1,
142                 ↪ R.string.action_reset);
143         }
144     } else {
145         menu.add(MenuGroup.ROUTE_ITEM.value, MenuId.PLAY.value, 0, R.string.action_open);
146         menu.add(MenuGroup.ROUTE_ITEM.value, MenuId.RESET.value, 0, R.string.action_reset);
147     }
148 }
149
150 /**
151  * Handles the selected context item
152  *
153  * @param item the selected menu item
154  * @return true if item has been handled
155  */
156 @Override
157 public boolean onContextItemSelected(MenuItem item) {
158     switch (MenuGroup.getMenuGroupForValue(item.getGroupId())) {
159         case ROUTE_ITEM:
160             return handleRouteItemSelected(item);
161         default:
162             return super.onContextItemSelected(item);
163     }
164 }
165
166 /**
167  * Handles the selected route menu item.
168  *
169  * @param item the selected menu item
170  * @return true if item has been handled
171  */
172 private boolean handleRouteItemSelected(MenuItem item) {
173     final RouteModel model = (RouteModel) ((ListView)
174         ↪ findViewById(R.id.listRoute)).getItemAtPosition(((AdapterView.AdapterContextMenuInfo)
175         ↪ item.getMenuInfo()).position);
176     switch (MenuId.getMenuIdForValue(item.getItemId())) {
177         case PLAY:
178             goToCheckpointActivity(model);
179             return Boolean.TRUE;
180         case RESET:
181             resetRoute(model);
182             return Boolean.TRUE;
183     }
184 }

```

Übung 3

```

180         default:
181             return Boolean.FALSE;
182     }
183 }
184
185 /**
186  * Prepares this activity views.
187  */
188 private void prepareView() {
189     // prepare route list view
190     ListView listView = (ListView) findViewById(R.id.listRoute);
191     listView.removeAllViewsInLayout();
192     if (listView.getAdapter() == null) {
193         final RouteArrayAdapter adapter = new RouteArrayAdapter(RouteActivity.this);
194         listView.setAdapter(adapter);
195         listView.setOnItemClickListener(this);
196     }
197 }
198
199 /**
200  * Loads the routes into the adapter
201  */
202 private void loadRoutes() {
203     if (checkAndDisplayAvailableNetwork()) {
204         new AsyncTask<Object, Object, AsyncTaskResult<List<RouteModel>>>() {
205             @Override
206             protected AsyncTaskResult<List<RouteModel>> doInBackground(Object... params) {
207                 List<RouteModel> routes = null;
208                 Exception exception = null;
209                 try {
210                     routes = viewModel.loadRoutes();
211                 } catch (Exception e) {
212                     exception = e;
213                 }
214
215                 return new AsyncTaskResult<List<RouteModel>>(routes, exception);
216             }
217
218             @Override
219             protected void onPreExecute() {
220                 super.onPreExecute();
221                 openProgressDialog(R.string.progress_loading_routes);
222             }
223
224             @Override
225             protected void onPostExecute(AsyncTaskResult<List<RouteModel>> result) {
226                 super.onPostExecute(result);
227                 closeProgressDialog();
228                 // Error occurred
229                 if (result.exception != null) {
230                     // ServiceException occurred
231                     if (result.exception instanceof ServiceException) {
232                         handleServiceException(((ServiceException) result.exception));
233                     } else {
234                         Toast.makeText(RouteActivity.this, R.string.error_unknown,
235                                     ↳ Toast.LENGTH_LONG).show();
236                     }
237                 }
238                 // Lists where loaded
239                 else {
240                     RouteArrayAdapter adapter = (RouteArrayAdapter) ((ListView)
241                                     ↳ findViewById(R.id.listRoute)).getAdapter();
242                     adapter.clear();
243                 }
244             }
245         }.execute();
246     }
247 }
248
249 // Lists where loaded
250 else {
251     RouteArrayAdapter adapter = (RouteArrayAdapter) ((ListView)
252         ↳ findViewById(R.id.listRoute)).getAdapter();
253     adapter.clear();
254 }
255 }
256 }
257 }
258 }
259 }
260 }
261 }
262 }
263 }
264 }
265 }
266 }
267 }
268 }
269 }
270 }
271 }
272 }
273 }
274 }
275 }
276 }
277 }
278 }
279 }
280 }
281 }
282 }
283 }
284 }
285 }
286 }
287 }
288 }
289 }
290 }
291 }
292 }
293 }
294 }
295 }
296 }
297 }
298 }
299 }
300 }
301 }
302 }
303 }
304 }
305 }
306 }
307 }
308 }
309 }
310 }
311 }
312 }
313 }
314 }
315 }
316 }
317 }
318 }
319 }
320 }
321 }
322 }
323 }
324 }
325 }
326 }
327 }
328 }
329 }
330 }
331 }
332 }
333 }
334 }
335 }
336 }
337 }
338 }
339 }
340 }
341 }
342 }
343 }
344 }
345 }
346 }
347 }
348 }
349 }
350 }
351 }
352 }
353 }
354 }
355 }
356 }
357 }
358 }
359 }
360 }
361 }
362 }
363 }
364 }
365 }
366 }
367 }
368 }
369 }
370 }
371 }
372 }
373 }
374 }
375 }
376 }
377 }
378 }
379 }
380 }
381 }
382 }
383 }
384 }
385 }
386 }
387 }
388 }
389 }
390 }
391 }
392 }
393 }
394 }
395 }
396 }
397 }
398 }
399 }
400 }
401 }
402 }
403 }
404 }
405 }
406 }
407 }
408 }
409 }
410 }
411 }
412 }
413 }
414 }
415 }
416 }
417 }
418 }
419 }
420 }
421 }
422 }
423 }
424 }
425 }
426 }
427 }
428 }
429 }
430 }
431 }
432 }
433 }
434 }
435 }
436 }
437 }
438 }
439 }
440 }
441 }
442 }
443 }
444 }
445 }
446 }
447 }
448 }
449 }
450 }
451 }
452 }
453 }
454 }
455 }
456 }
457 }
458 }
459 }
460 }
461 }
462 }
463 }
464 }
465 }
466 }
467 }
468 }
469 }
470 }
471 }
472 }
473 }
474 }
475 }
476 }
477 }
478 }
479 }
480 }
481 }
482 }
483 }
484 }
485 }
486 }
487 }
488 }
489 }
490 }
491 }
492 }
493 }
494 }
495 }
496 }
497 }
498 }
499 }
500 }
501 }
502 }
503 }
504 }
505 }
506 }
507 }
508 }
509 }
510 }
511 }
512 }
513 }
514 }
515 }
516 }
517 }
518 }
519 }
520 }
521 }
522 }
523 }
524 }
525 }
526 }
527 }
528 }
529 }
530 }
531 }
532 }
533 }
534 }
535 }
536 }
537 }
538 }
539 }
540 }
541 }
542 }
543 }
544 }
545 }
546 }
547 }
548 }
549 }
550 }
551 }
552 }
553 }
554 }
555 }
556 }
557 }
558 }
559 }
560 }
561 }
562 }
563 }
564 }
565 }
566 }
567 }
568 }
569 }
570 }
571 }
572 }
573 }
574 }
575 }
576 }
577 }
578 }
579 }
580 }
581 }
582 }
583 }
584 }
585 }
586 }
587 }
588 }
589 }
590 }
591 }
592 }
593 }
594 }
595 }
596 }
597 }
598 }
599 }
600 }
601 }
602 }
603 }
604 }
605 }
606 }
607 }
608 }
609 }
610 }
611 }
612 }
613 }
614 }
615 }
616 }
617 }
618 }
619 }
620 }
621 }
622 }
623 }
624 }
625 }
626 }
627 }
628 }
629 }
630 }
631 }
632 }
633 }
634 }
635 }
636 }
637 }
638 }
639 }
640 }
641 }
642 }
643 }
644 }
645 }
646 }
647 }
648 }
649 }
650 }
651 }
652 }
653 }
654 }
655 }
656 }
657 }
658 }
659 }
660 }
661 }
662 }
663 }
664 }
665 }
666 }
667 }
668 }
669 }
670 }
671 }
672 }
673 }
674 }
675 }
676 }
677 }
678 }
679 }
680 }
681 }
682 }
683 }
684 }
685 }
686 }
687 }
688 }
689 }
690 }
691 }
692 }
693 }
694 }
695 }
696 }
697 }
698 }
699 }
700 }
701 }
702 }
703 }
704 }
705 }
706 }
707 }
708 }
709 }
710 }
711 }
712 }
713 }
714 }
715 }
716 }
717 }
718 }
719 }
720 }
721 }
722 }
723 }
724 }
725 }
726 }
727 }
728 }
729 }
730 }
731 }
732 }
733 }
734 }
735 }
736 }
737 }
738 }
739 }
740 }
741 }
742 }
743 }
744 }
745 }
746 }
747 }
748 }
749 }
750 }
751 }
752 }
753 }
754 }
755 }
756 }
757 }
758 }
759 }
760 }
761 }
762 }
763 }
764 }
765 }
766 }
767 }
768 }
769 }
770 }
771 }
772 }
773 }
774 }
775 }
776 }
777 }
778 }
779 }
780 }
781 }
782 }
783 }
784 }
785 }
786 }
787 }
788 }
789 }
790 }
791 }
792 }
793 }
794 }
795 }
796 }
797 }
798 }
799 }
800 }
801 }
802 }
803 }
804 }
805 }
806 }
807 }
808 }
809 }
810 }
811 }
812 }
813 }
814 }
815 }
816 }
817 }
818 }
819 }
820 }
821 }
822 }
823 }
824 }
825 }
826 }
827 }
828 }
829 }
830 }
831 }
832 }
833 }
834 }
835 }
836 }
837 }
838 }
839 }
840 }
841 }
842 }
843 }
844 }
845 }
846 }
847 }
848 }
849 }
850 }
851 }
852 }
853 }
854 }
855 }
856 }
857 }
858 }
859 }
860 }
861 }
862 }
863 }
864 }
865 }
866 }
867 }
868 }
869 }
870 }
871 }
872 }
873 }
874 }
875 }
876 }
877 }
878 }
879 }
880 }
881 }
882 }
883 }
884 }
885 }
886 }
887 }
888 }
889 }
890 }
891 }
892 }
893 }
894 }
895 }
896 }
897 }
898 }
899 }
900 }
901 }
902 }
903 }
904 }
905 }
906 }
907 }
908 }
909 }
910 }
911 }
912 }
913 }
914 }
915 }
916 }
917 }
918 }
919 }
920 }
921 }
922 }
923 }
924 }
925 }
926 }
927 }
928 }
929 }
930 }
931 }
932 }
933 }
934 }
935 }
936 }
937 }
938 }
939 }
940 }
941 }
942 }
943 }
944 }
945 }
946 }
947 }
948 }
949 }
950 }
951 }
952 }
953 }
954 }
955 }
956 }
957 }
958 }
959 }
960 }
961 }
962 }
963 }
964 }
965 }
966 }
967 }
968 }
969 }
970 }
971 }
972 }
973 }
974 }
975 }
976 }
977 }
978 }
979 }
980 }
981 }
982 }
983 }
984 }
985 }
986 }
987 }
988 }
989 }
990 }
991 }
992 }
993 }
994 }
995 }
996 }
997 }
998 }
999 }
1000 }

```

Übung 3

```

241         adapter.addAll(result.result);
242         invalidateOptionsMenu();
243     }
244 }
245 }.execute();
246 }
247 }
248
249 /**
250  * Resets the given route.
251  *
252  * @param model the route to reset
253  */
254 private void resetRoute(final RouteModel model) {
255     if (checkAndDisplayAvailableNetwork()) {
256         new AsyncTask<Object, Object, AsyncTaskResult<Boolean>>() {
257             @Override
258             protected AsyncTaskResult<Boolean> doInBackground(Object... params) {
259                 Boolean result = null;
260                 Exception exception = null;
261                 try {
262                     if (model != null) {
263                         result = viewModel.resetRoute(model);
264                     } else {
265                         result = viewModel.resetAllRoutes();
266                     }
267                 } catch (Exception e) {
268                     exception = e;
269                 }
270
271                 return new AsyncTaskResult<Boolean>(result, exception);
272             }
273
274             @Override
275             protected void onPreExecute() {
276                 super.onPreExecute();
277                 openProgressDialog(R.string.progress_resetting_routes);
278             }
279
280             @Override
281             protected void onPostExecute(AsyncTaskResult<Boolean> result) {
282                 super.onPostExecute(result);
283                 closeProgressDialog();
284                 // Error occurred
285                 if (result.exception != null) {
286                     // ServiceException occurred
287                     if (result.exception instanceof ServiceException) {
288                         handleServiceException(((ServiceException) result.exception));
289                     } else {
290                         Toast.makeText(RouteActivity.this, R.string.error_unknown,
291                                     ↪ Toast.LENGTH_LONG).show();
292                     }
293                 }
294                 // Reset failed on rest method
295                 else if (!result.result) {
296                     Toast.makeText(RouteActivity.this, R.string.error_route_reset_failed,
297                                     ↪ Toast.LENGTH_LONG).show();
298                 }
299                 // Reset ok
300                 else {
301                     loadRoutes();
302                 }
303             }
304         }
305     }
306 }

```

Übung 3

```

302         }.execute();
303     }
304 }
305
306 /**
307  * Fires an intent which redirects teh user to the checkpoint activity.
308  *
309  * @param model the route to serialize to the checkpoint activity
310  */
311 private void goToCheckpointActivity(RouteModel model) {
312     Intent intent = new Intent(RouteActivity.this, CheckpointActivity.class);
313     intent.putExtra(AmazingRaceApplication.INTENT_EXTRA_ROUTE, model);
314     startActivity(intent);
315 }
316 //endregion
317
318 //region Listener
319
320 /**
321  * Clcik listener which redirects to the checkpoint activity
322  *
323  * @param parent the adapter
324  * @param view the view
325  * @param position the item position
326  * @param id the view id
327  */
328 @Override
329 public void onItemClick(AdapterView<?> parent, View view, int position, long id) {
330     goToCheckpointActivity(((RouteModel) parent.getItemAtPosition(position)));
331 }
332 //endregion
333 }

```

Übung 3

Listing 6: CheckpointActivity.java

```

1 package at.fh.ooe.moc5.amazingrace.activity;
2
3 import android.content.DialogInterface;
4 import android.content.Intent;
5 import android.graphics.Color;
6 import android.os.AsyncTask;
7 import android.os.Bundle;
8 import android.view.Menu;
9 import android.view.MenuItem;
10 import android.view.View;
11 import android.widget.Button;
12 import android.widget.EditText;
13 import android.widget.ImageView;
14 import android.widget.LinearLayout;
15 import android.widget.TextView;
16 import android.widget.Toast;
17
18 import com.google.android.gms.maps.CameraUpdateFactory;
19 import com.google.android.gms.maps.GoogleMap;
20 import com.google.android.gms.maps.MapFragment;
21 import com.google.android.gms.maps.OnMapReadyCallback;
22 import com.google.android.gms.maps.model.BitmapDescriptorFactory;
23 import com.google.android.gms.maps.model.LatLng;
24 import com.google.android.gms.maps.model.MarkerOptions;
25 import com.google.android.gms.maps.model.PolylineOptions;
26
27 import at.fh.ooe.moc5.amazingrace.AmazingRaceApplication;
28 import at.fh.ooe.moc5.amazingrace.R;
29 import at.fh.ooe.moc5.amazingrace.model.json.CheckpointModel;
30 import at.fh.ooe.moc5.amazingrace.model.json.RouteModel;
31 import at.fh.ooe.moc5.amazingrace.model.task.AsyncTaskResult;
32 import at.fh.ooe.moc5.amazingrace.model.view.CheckpointViewModel;
33 import at.fh.ooe.moc5.amazingrace.service.ServiceException;
34 import at.fh.ooe.moc5.amazingrace.util.DialogUtil;
35 import at.fh.ooe.moc5.amazingrace.watcher.AnswerButtonTextWatcher;
36 import at.fh.ooe.moc5.amazingrace.watcher.CheckpointViewModelBindingTextWatcher;
37
38 public class CheckpointActivity extends AbstractActivity<CheckpointViewModel> implements
↵ View.OnClickListener, DialogInterface.OnClickListener {
39
40     //region Lifecycle Methods
41
42     /**
43      * Prepares the view model depending on the intent serialized route model
44      *
45      * @param savedInstanceState the saved bundle instance
46      */
47     @Override
48     protected void onCreate(Bundle savedInstanceState) {
49         super.onCreate(savedInstanceState);
50         setContentView(R.layout.activity_checkpoint);
51         if (application.getLoggedUser() != null) {
52             final Intent intent = getIntent();
53             if (intent.getExtras().containsKey(AmazingRaceApplication.INTENT_EXTRA_ROUTE)) {
54                 RouteModel route = (RouteModel)
↵                 intent.getExtras().get(AmazingRaceApplication.INTENT_EXTRA_ROUTE);
55                 viewModel = new CheckpointViewModel(application.getLoggedUser(), route);
56             }
57         }
58     }

```

Übung 3

```

59
60 /**
61  * Prepares this activity views if view mdoel is valid.
62  */
63 @Override
64 protected void onResume() {
65     super.onResume();
66     if (validViewModel) {
67         prepareView();
68     }
69 }
70
71 /**
72  * Creates the options menu items
73  *
74  * @param menu the menu to add items too
75  * @return true
76  */
77 @Override
78 public boolean onCreateOptionsMenu(Menu menu) {
79     if (!viewModel.getVisitedCheckpoints().isEmpty()) {
80         menu.add(MenuGroup.OPTIONS.value, MenuId.RESET.value, 0, R.string.action_reset);
81     }
82     menu.add(MenuGroup.OPTIONS.value, MenuId.CLOSE.value, 0, R.string.action_close);
83     return Boolean.TRUE;
84 }
85
86 /**
87  * Handles an options menu item select
88  *
89  * @param item the selected menu item
90  * @return true if items has been handled
91  */
92 @Override
93 public boolean onOptionsItemSelected(MenuItem item) {
94     switch (MenuId.getIdForValue(item.getItemId())) {
95         case RESET:
96             resetRoute();
97             return Boolean.TRUE;
98         case CLOSE:
99             openCloseApplicationDialog();
100             return Boolean.TRUE;
101         default:
102             return super.onOptionsItemSelected(item);
103     }
104 }
105 //endregion
106
107 //region Helper
108
109 /**
110  * Prepares this acitivity views.
111  */
112 public void prepareView() {
113     // set default container visibility
114     findViewById(R.id.routeCheckpointContainer).setVisibility(View.VISIBLE);
115     findViewById(R.id.routeCheckpointMapContainer).setVisibility(View.VISIBLE);
116     findViewById(R.id.routeCheckpointListContainer).setVisibility(View.GONE);
117
118     // prepare list view of visited checkpoints
119     // ListView is not used because of outer wrapping ScrollView
120     LinearLayout listViewReplacement = (LinearLayout)
121         ↪ findViewById(R.id.routeCheckpointListView);

```

Übung 3

```

121 listViewReplacement.removeAllViews();
122 listViewReplacement.setVisibility(View.VISIBLE);
123 for (int i = 0; i < viewModel.getVisitedCheckpoints().size(); i++) {
124     CheckpointModel model = viewModel.getRoute().getVisitedCheckpoints().get(i);
125     View view = View.inflate(CheckpointActivity.this, R.layout.view_checkpoint_item,
126         ↪ null);
127     ((TextView) view.findViewById(R.id.checkpointItemText)).setText(model.getName());
128     ((ImageView) view.findViewById(R.id.checkpointItemImage))
129         .setImageResource(R.drawable.maps_marker_green_icon);
130     listViewReplacement.addView(view);
131 }
132 // Add next checkpoint to list with open marker
133 if (viewModel.getNextCheckpoint() != null) {
134     final CheckpointModel model = viewModel.getNextCheckpoint();
135     View view = View.inflate(CheckpointActivity.this, R.layout.view_checkpoint_item,
136         ↪ null);
137     ((TextView) view.findViewById(R.id.checkpointItemText)).setText(
138         model.getName()
139     );
140     ((ImageView) view.findViewById(R.id.checkpointItemImage))
141         .setImageResource(R.drawable.maps_marker_red_icon);
142     listViewReplacement.addView(view);
143 }
144 // route already finished
145 if (viewModel.getNextCheckpoint() == null) {
146     findViewById(R.id.routeCheckpointContainer).setVisibility(View.GONE);
147     findViewById(R.id.routeCheckpointListContainer).setVisibility(View.VISIBLE);
148 }
149 // checkpoints still open
150 else {
151     Button answerButton = (Button) findViewById(R.id.answerBtn);
152     answerButton.setEnabled(Boolean.FALSE);
153     answerButton.setOnClickListener(this);
154
155     ((TextView) findViewById(R.id.nextCheckpointTxt))
156         .setText(viewModel.getNextCheckpoint().getName());
157     ((TextView) findViewById(R.id.nextCheckpointHintTxt))
158         .setText(viewModel.getNextCheckpoint().getHint());
159
160     EditText answer = (EditText) findViewById(R.id.hintAnswerEdTxt);
161     answer.setText("");
162     answer.addTextChangedListener(new CheckpointViewModelBindingTextWatcher(viewModel));
163     answer.addTextChangedListener(new AnswerButtonTextWatcher(this, viewModel));
164 }
165
166 // Prepare maps
167 MapFragment mapFragment = (MapFragment)
168     ↪ getFragmentManager().findFragmentById(R.id.routeCheckpointMap);
169 mapFragment.getMapAsync(new OnMapReadyCallback() {
170     @Override
171     public void onMapReady(GoogleMap map) {
172         // clear former set map properties
173         map.clear();
174         // enable ui functions
175         map.getUiSettings().setZoomControlsEnabled(Boolean.TRUE);
176         map.getUiSettings().setAllGesturesEnabled(Boolean.TRUE);
177         map.getUiSettings().setCompassEnabled(Boolean.TRUE);
178         LatLng zoomLocation = null;
179         PolylineOptions lineOptions = null;
180         // Get visited points
181         if (!viewModel.getRoute().getVisitedCheckpoints().isEmpty()) {
182             lineOptions = new PolylineOptions();

```

Übung 3

```

181         for (int i = 0; i < viewModel.getRoute().getVisitedCheckpoints().size(); i++)
182             ↪ {
183                 CheckpointModel model =
184                     ↪ viewModel.getRoute().getVisitedCheckpoints().get(i);
185                 LatLng location = new LatLng(model.getLatitude(), model.getLongitude());
186                 map.addMarker(new MarkerOptions().position(location)
187                     .title((i + 1) + ". " + model.getName())
188                     .icon(
189                         BitmapDescriptorFactory.defaultMarker(
190                             BitmapDescriptorFactory.HUE_GREEN
191                         )
192                     )
193                     .showInfoWindow());
194                 lineOptions.add(location);
195                 // Remember last for focusing on it
196                 if (i == (viewModel.getVisitedCheckpoints().size() - 1)) {
197                     zoomLocation = location;
198                 }
199             }
200         map.addPolyline(lineOptions.color(Color.GREEN));
201     }
202     // Add next checkpoint which is not part of list and gets red line from it to the
203     ↪ former marker
204     if (viewModel.getRoute().getNextCheckpoint() != null) {
205         zoomLocation = new
206             ↪ LatLng(viewModel.getRoute().getNextCheckpoint().getLatitude(),
207             ↪ viewModel.getRoute().getNextCheckpoint().getLongitude());
208         map.addMarker(
209             new MarkerOptions()
210                 .position(zoomLocation)
211                 .title(viewModel.getNextCheckpoint().getName())
212                 .icon(BitmapDescriptorFactory
213                     .defaultMarker(BitmapDescriptorFactory.HUE_RED))
214             ).showInfoWindow();
215         // If visited checkpoints exist
216         if (!viewModel.getRoute().getVisitedCheckpoints().isEmpty()) {
217             map.addPolyline(new PolylineOptions().add(
218                 lineOptions.getPoints().get(lineOptions.getPoints().size()
219                     ↪ - 1), zoomLocation).color(Color.RED)
220             );
221         }
222     }
223     // move camera to last or next checkpoint
224     map.moveCamera(CameraUpdateFactory.newLatLngZoom(zoomLocation, 16));
225 }
226 });
227 }
228 //endregion
229
230 //region Listeners
231
232 /**
233  * Toggles the visibility if one container header gets clicked
234  *
235  * @param view the clicked view
236  */
237 public void toggleVisibility(View view) {
238     View checkpointListContainer = findViewById(R.id.routeCheckpointListContainer);
239     View checkpointMapContainer = findViewById(R.id.routeCheckpointMapContainer);
240
241     // Always references to parent LinearLayout so not all clickable children need to be
242     ↪ handled here
243     switch (((View) view.getParent()).getId()) {

```


Übung 3

```

237         case R.id.routeCheckpointListContainerLabel:
238             checkpointListContainer.setVisibility((View.VISIBLE ==
                ↪ checkpointListContainer.getVisibility()) ? View.GONE : View.VISIBLE);
239             break;
240         case R.id.routeCheckpointMapContainerLabel:
241             checkpointMapContainer.setVisibility((View.VISIBLE ==
                ↪ checkpointMapContainer.getVisibility()) ? View.GONE : View.VISIBLE);
242             break;
243     }
244 }
245
246 /**
247  * Handles the button clicks by redirecting to proper method.
248  *
249  * @param v the clicked view
250  */
251 @Override
252 public void onClick(View v) {
253     switch (v.getId()) {
254         case R.id.answerBtn:
255             onAnswerButtonClick((Button) v);
256             break;
257     }
258 }
259
260 @Override
261 public void onClick(DialogInterface dialog, int which) {
262     dialog.dismiss();
263 }
264 //endregion
265
266 /**
267  * Handles the answer button click
268  *
269  * @param button the clicked answer button
270  */
271 private void onAnswerButtonClick(Button button) {
272     if (checkAndDisplayAvailableNetwork()) {
273         new AsyncTask<Object, Object, AsyncTaskResult<Boolean>>() {
274             @Override
275             protected AsyncTaskResult<Boolean> doInBackground(Object... params) {
276                 Boolean result = null;
277                 Exception exception = null;
278                 try {
279                     result = viewModel.validateSecret();
280                 } catch (Exception e) {
281                     exception = e;
282                 }
283
284                 return new AsyncTaskResult<Boolean>(result, exception);
285             }
286
287             @Override
288             protected void onPreExecute() {
289                 super.onPreExecute();
290                 openProgressDialog(R.string.progress_validating_secret);
291             }
292
293             @Override
294             protected void onPostExecute(AsyncTaskResult<Boolean> result) {
295                 super.onPostExecute(result);
296                 closeProgressDialog();
297                 if (result.exception != null) {

```

Übung 3

```

298         // ServiceException occurred
299         if (result.exception instanceof ServiceException) {
300             handleServiceException(((ServiceException) result.exception));
301         } else {
302             Toast.makeText(CheckpointActivity.this, R.string.error_unknown,
303                 ↳ Toast.LENGTH_LONG).show();
304         }
305     } else if (result.result) {
306         reloadRoute(Boolean.TRUE);
307     } else {
308         Toast.makeText(CheckpointActivity.this, R.string.info_secret_wrong,
309             ↳ Toast.LENGTH_LONG).show();
310         ((TextView) CheckpointActivity.this
311             .findViewById(R.id.nextCheckpointTxt)).setText("");
312     }
313 }
314 }.execute();
315 }
316
317 /**
318  * Reloads the route
319  */
320 private void reloadRoute(final boolean checkIfFinished) {
321     if (checkAndDisplayAvailableNetwork()) {
322         new AsyncTask<Object, Object, AsyncTaskResult<Boolean>>() {
323             @Override
324             protected AsyncTaskResult<Boolean> doInBackground(Object... params) {
325                 Boolean found = null;
326                 Exception exception = null;
327                 try {
328                     found = viewModel.reloadRoute();
329                 } catch (Exception e) {
330                     exception = e;
331                 }
332                 return new AsyncTaskResult<Boolean>(found, exception);
333             }
334
335             @Override
336             protected void onPostExecute(AsyncTaskResult<Boolean> result) {
337                 super.onPostExecute(result);
338                 closeProgressDialog();
339                 // Exception occurred
340                 if (result.exception != null) {
341                     if (result.exception instanceof ServiceException) {
342                         handleServiceException(((ServiceException) result.exception));
343                     } else {
344                         Toast.makeText(CheckpointActivity.this, R.string.error_unknown,
345                             ↳ Toast.LENGTH_SHORT).show();
346                     }
347                 }
348                 // Route found
349                 else if (result.result) {
350                     prepareView();
351                     invalidateOptionsMenu();
352                     if (checkIfFinished) {
353                         if (viewModel.getNextCheckpoint() != null) {
354                             Toast.makeText(CheckpointActivity.this, R.string.info_secret_ok,
355                                 ↳ Toast.LENGTH_LONG).show();
356                         } else {
357                             DialogUtil.createCustomContentDialog(CheckpointActivity.this,
358                                 ↳ R.layout.view_congratulations_dialog,
359                                 ↳ R.string.dialog_title_info, CheckpointActivity.this).show();
360                         }
361                     }
362                 }
363             }
364         }
365     }
366 }

```

Übung 3

```

355         }
356     }
357 }
358 // Route not found, go back to routes activity
359 else {
360     startActivity(new Intent(CheckpointActivity.this, RouteActivity.class));
361     finish();
362 }
363 }
364
365 @Override
366 protected void onPreExecute() {
367     super.onPreExecute();
368     openProgressDialog(R.string.progress_updating_route);
369 }
370 }.execute();
371 }
372 }
373
374 /**
375  * Resets the current route.
376  */
377 private void resetRoute() {
378     if (checkAndDisplayAvailableNetwork()) {
379         new AsyncTask<Object, Object, AsyncTaskResult<Boolean>>() {
380             @Override
381             protected AsyncTaskResult<Boolean> doInBackground(Object... params) {
382                 Boolean result = null;
383                 Exception exception = null;
384                 try {
385                     result = viewModel.resetRoute();
386                 } catch (Exception e) {
387                     exception = e;
388                 }
389
390                 return new AsyncTaskResult<Boolean>(result, exception);
391             }
392
393             @Override
394             protected void onPreExecute() {
395                 super.onPreExecute();
396                 openProgressDialog(R.string.progress_resetting_routes);
397             }
398
399             @Override
400             protected void onPostExecute(AsyncTaskResult<Boolean> result) {
401                 super.onPostExecute(result);
402                 closeProgressDialog();
403                 // Error occurred
404                 if (result.exception != null) {
405                     // ServiceException occurred
406                     if (result.exception instanceof ServiceException) {
407                         handleServiceException(((ServiceException) result.exception));
408                     } else {
409                         Toast.makeText(CheckpointActivity.this, R.string.error_unknown,
410                                     ↵ Toast.LENGTH_LONG).show();
411                     }
412                 }
413                 // Reset failed on rest method
414                 else if (!result.result) {
415                     Toast.makeText(CheckpointActivity.this, R.string.error_route_reset_failed,
416                                 ↵ Toast.LENGTH_LONG).show();
417                 }
418             }
419         }
420     }
421 }

```

Übung 3

```
416         // Reset ok
417         else {
418             reloadRoute(Boolean.FALSE);
419         }
420     }
421     }.execute();
422 }
423 }
424 //endregion
425 }
```

Übung 3

1.2.3 Adapter

Listing 7: RouteArrayAdapter.java

```

1  package at.fh.ooe.moc5.amazingrace.adapter;
2
3  import android.content.Context;
4  import android.view.View;
5  import android.view.ViewGroup;
6  import android.widget.AdapterView;
7  import android.widget.ImageView;
8  import android.widget.TextView;
9
10 import at.fh.ooe.moc5.amazingrace.R;
11 import at.fh.ooe.moc5.amazingrace.model.json.RouteModel;
12
13 /**
14  * Created by Thomas on 12/25/2015.
15  * The adapter for the route items
16  */
17 public class RouteArrayAdapter extends ArrayAdapter<RouteModel> {
18
19     public RouteArrayAdapter(Context ctx) {
20         super(ctx, 0);
21     }
22
23     @Override
24     public View getView(int position, View convertView, ViewGroup parent) {
25         if (convertView == null) {
26             convertView = View.inflate(getContext(), R.layout.view_route_item, null);
27         }
28
29         RouteModel route = getItem(position);
30         ImageView icon = ((ImageView) convertView.findViewById(R.id.routeItemImage));
31         if (route.getNextCheckpoint() == null) {
32             icon.setImageResource(R.drawable.done_icon);
33         } else {
34             icon.setImageResource(R.drawable.app_icon);
35         }
36         ((TextView) convertView.findViewById(R.id.routeItemText))
37             .setText(route.toItemString(getContext().getString(R.string.done)));
38
39         return convertView;
40     }
41 }

```

Übung 3

1.2.4 JSON-Models

Listing 8: CheckpointModel.java

```

1  package at.fh.ooe.moc5.amazingrace.model.json;
2
3  import com.google.gson.annotations.SerializedName;
4
5  import java.io.Serializable;
6  import java.util.Objects;
7
8  /**
9   * Created by Thomas on 12/25/2015.
10  */
11  public class CheckpointModel implements Serializable {
12
13      @SerializedName("Id")
14      private String id;
15      @SerializedName("Number")
16      private int number;
17      @SerializedName("Name")
18      private String name;
19      @SerializedName("Hint")
20      private String hint;
21      @SerializedName("Latitude")
22      private double latitude;
23      @SerializedName("Longitude")
24      private double longitude;
25      private transient boolean unvisited = Boolean.FALSE;
26
27      public CheckpointModel() {
28      }
29
30      //region Getter and Setter
31      public String getId() {
32          return id;
33      }
34
35      public void setId(String id) {
36          this.id = id;
37      }
38
39      public int getNumber() {
40          return number;
41      }
42
43      public void setNumber(int number) {
44          this.number = number;
45      }
46
47      public String getName() {
48          return name;
49      }
50
51      public void setName(String name) {
52          this.name = name;
53      }
54
55      public String getHint() {
56          return hint;
57      }
58
59      public void setHint(String hint) {

```

Übung 3

```

60         this.hint = hint;
61     }
62
63     public double getLatitude() {
64         return latitude;
65     }
66
67     public void setLatitude(double latitude) {
68         this.latitude = latitude;
69     }
70
71     public double getLongitude() {
72         return longitude;
73     }
74
75     public void setLongitude(double longitude) {
76         this.longitude = longitude;
77     }
78
79     public boolean isUnvisited() {
80         return unvisited;
81     }
82
83     public void setUnvisited(boolean unvisited) {
84         this.unvisited = unvisited;
85     }
86     //endregion
87
88     @Override
89     public boolean equals(Object o) {
90         if (this == o) return true;
91         if (o == null || getClass() != o.getClass()) return false;
92         CheckpointModel that = (CheckpointModel) o;
93         return Objects.equals(id, that.id);
94     }
95
96     @Override
97     public int hashCode() {
98         return Objects.hash(id);
99     }
100 }

```

Listing 9: CheckpointRequestModel.java

```

1  package at.fh.ooe.moc5.amazingrace.model.json;
2
3  import com.google.gson.annotations.SerializedName;
4
5  import java.io.Serializable;
6  import java.util.Objects;
7
8  /**
9   * Created by Thomas on 12/25/2015.
10  */
11  public class CheckpointRequestModel extends CredentialsRequestModel implements Serializable {
12
13      @SerializedName("CheckpointId")
14      public final String checkpointId;
15      @SerializedName("Secret")
16      public final String secret;
17
18      public CheckpointRequestModel(String username, String password, String checkpointId, String
19          ↪ secret) {
20          super(username, password);
21          this.checkpointId = checkpointId;
22          this.secret = secret;
23      }
24
25      @Override
26      public boolean equals(Object o) {
27          if (this == o) return true;
28          if (o == null || getClass() != o.getClass()) return false;
29          if (!super.equals(o)) return false;
30          CheckpointRequestModel that = (CheckpointRequestModel) o;
31          return Objects.equals(checkpointId, that.checkpointId);
32      }
33
34      @Override
35      public int hashCode() {
36          return Objects.hash(super.hashCode(), checkpointId);
37      }
38  }

```


Listing 10: CredentialsRequestModel.java

```

1 package at.fh.ooe.moc5.amazingrace.model.json;
2
3 import com.google.gson.annotations.SerializedName;
4
5 import java.io.Serializable;
6 import java.util.Objects;
7
8 /**
9  * Created by Thomas on 12/25/2015.
10 */
11 public class CredentialsRequestModel implements Serializable {
12
13     @SerializedName("UserName")
14     public final String userName;
15     @SerializedName("Password")
16     public final String password;
17
18     public CredentialsRequestModel(String userName, String password) {
19         this.userName = userName;
20         this.password = password;
21     }
22
23     public String toQueryString() {
24         return String.format("userName=%s&password=%s", userName, password);
25     }
26
27     @Override
28     public boolean equals(Object o) {
29         if (this == o) return true;
30         if (o == null || getClass() != o.getClass()) return false;
31         CredentialsRequestModel that = (CredentialsRequestModel) o;
32         return Objects.equals(userName, that.userName);
33     }
34
35     @Override
36     public int hashCode() {
37         return Objects.hash(userName);
38     }
39 }

```

Listing 11: RouteModel.java

```

1  package at.fh.ooe.moc5.amazingrace.model.json;
2
3  import com.google.gson.annotations.SerializedName;
4
5  import java.io.Serializable;
6  import java.util.Comparator;
7  import java.util.List;
8  import java.util.Objects;
9
10 /**
11  * Created by Thomas on 12/25/2015.
12  */
13 public class RouteModel implements Serializable {
14
15     @SerializedName("Id")
16     private String id;
17     @SerializedName("Name")
18     private String name;
19     @SerializedName("VisitedCheckpoints")
20     private List<CheckpointModel> visitedCheckpoints;
21     @SerializedName("NextCheckpoint")
22     private CheckpointModel nextCheckpoint;
23
24     public RouteModel() {
25     }
26
27     //region Getter and Setter
28     public String getId() {
29         return id;
30     }
31
32     public void setId(String id) {
33         this.id = id;
34     }
35
36     public String getName() {
37         return name;
38     }
39
40     public void setName(String name) {
41         this.name = name;
42     }
43
44     public List<CheckpointModel> getVisitedCheckpoints() {
45         return visitedCheckpoints;
46     }
47
48     public void setVisitedCheckpoints(List<CheckpointModel> visitedCheckpoints) {
49         this.visitedCheckpoints = visitedCheckpoints;
50     }
51
52     public CheckpointModel getNextCheckpoint() {
53         return nextCheckpoint;
54     }
55
56     public void setNextCheckpoint(CheckpointModel nextCheckpoint) {
57         this.nextCheckpoint = nextCheckpoint;
58     }
59     //endregion
60

```

Übung 3

```

61     //region Utilities
62     public boolean isDone() {
63         return nextCheckpoint == null;
64     }
65     //endregion
66
67     public String toItemString(final String doneString) {
68         final StringBuilder sb = new StringBuilder(name);
69         if (!isDone()) {
70             Objects.requireNonNull(doneString);
71             sb.append(" (");
72             sb.append(visitedCheckpoints.size()).append("x").append(doneString);
73             sb.append(")");
74         }
75         return sb.toString();
76     }
77
78     @Override
79     public boolean equals(Object o) {
80         if (this == o) return true;
81         if (o == null || getClass() != o.getClass()) return false;
82         RouteModel that = (RouteModel) o;
83         return Objects.equals(id, that.id);
84     }
85
86     @Override
87     public int hashCode() {
88         return Objects.hash(id);
89     }
90 }

```

Listing 12: RouteRequestModel.java

```

1 package at.fh.ooe.moc5.amazingrace.model.json;
2
3 import com.google.gson.annotations.SerializedName;
4
5 import java.io.Serializable;
6 import java.util.Objects;
7
8 /**
9  * Created by Thomas on 12/25/2015.
10 */
11 public class RouteRequestModel extends CredentialsRequestModel {
12
13     @SerializedName("RouteId")
14     public final String routeId;
15
16     public RouteRequestModel(CredentialsRequestModel model, String routeId) {
17         this(model.userName, model.password, routeId);
18     }
19
20     public RouteRequestModel(String username, String password, String routeId) {
21         super(username, password);
22         this.routeId = routeId;
23     }
24
25     @Override
26     public boolean equals(Object o) {
27         if (this == o) return true;
28         if (o == null || getClass() != o.getClass()) return false;
29         if (!super.equals(o)) return false;
30         RouteRequestModel that = (RouteRequestModel) o;
31         return Objects.equals(routeId, that.routeId);
32     }
33
34     @Override
35     public int hashCode() {
36         return Objects.hash(super.hashCode(), routeId);
37     }
38 }

```

1.2.5 Task-Models

Listing 13: AsyncTaskResult.java

```
1 package at.fh.ooe.moc5.amazingrace.model.task;
2
3 /**
4  * Created by Thomas on 12/25/2015.
5  * This class represents an async task result which allows handling of any occurred exception
6  * in the postExecute method instead of the doInBackground where we are not in the UI Thread.
7  */
8 public class AsyncTaskResult<T> {
9
10     public final T result;
11     public final Exception exception;
12
13     public AsyncTaskResult(T result, Exception exception) {
14         this.result = result;
15         this.exception = exception;
16     }
17 }
```

Übung 3

1.2.6 View-Models

Listing 14: LoginViewModel.java

```

1 package at.fh.ooe.moc5.amazingrace.model.view;
2
3 import org.apache.commons.lang3.StringUtils;
4
5 import java.io.Serializable;
6
7 import at.fh.ooe.moc5.amazingrace.model.json.CredentialsRequestModel;
8 import at.fh.ooe.moc5.amazingrace.service.ServiceProxy;
9 import at.fh.ooe.moc5.amazingrace.service.ServiceException;
10 import at.fh.ooe.moc5.amazingrace.service.ServiceProxyFactory;
11
12 /**
13  * Created by Thomas on 12/24/2015.
14  */
15 public class LoginViewModel implements Serializable {
16
17     private String username;
18     private String password;
19
20     private final ServiceProxy restProxy;
21
22     public LoginViewModel() {
23         restProxy = ServiceProxyFactory.createServiceProxy();
24     }
25
26     /**
27      * Answers the question if this model is valid whic its is if a username and password are set.
28      *
29      * @return true if valid, false otherwise
30      */
31     public boolean isValid() {
32         return (((username != null) && (!StringUtils.isEmpty(username))) && ((password != null) &&
33             ↳ (!StringUtils.isEmpty(password))));
34     }
35
36     //region Actions
37
38     /**
39      * Logs the user in by validating the user credentials via the proxy service.
40      *
41      * @return true if username password are valid, false otherwise
42      * @throws ServiceException if an rproxy service exception occurred
43      */
44     public UserContextModel loginAction() throws ServiceException {
45         final CredentialsRequestModel model = new CredentialsRequestModel(username, password);
46         final boolean isValidCredentials = restProxy.validateCredentials(model);
47         if (isValidCredentials) {
48             return new UserContextModel(model);
49         }
50         return null;
51     }
52
53     /**
54      * Resets this model by resetting the username and password
55      */
56     public void reset() {
57         username = null;
58         password = null;

```

Übung 3

```
59     }
60
61     //endregion
62     //region Getter and Setter
63     public void setUsername(String username) {
64         this.username = username;
65     }
66
67     public String getUsername() {
68         return username;
69     }
70
71     public void setPassword(String password) {
72         this.password = password;
73     }
74
75     public String getPassword() {
76         return password;
77     }
78     //endregion
79 }
```

Listing 15: RoutesViewModel.java

```

1 package at.fh.ooe.moc5.amazingrace.model.view;
2
3 import java.io.Serializable;
4 import java.util.Collections;
5 import java.util.Comparator;
6 import java.util.List;
7 import java.util.Objects;
8
9 import at.fh.ooe.moc5.amazingrace.model.json.RouteModel;
10 import at.fh.ooe.moc5.amazingrace.model.json.RouteRequestModel;
11 import at.fh.ooe.moc5.amazingrace.service.ServiceProxy;
12 import at.fh.ooe.moc5.amazingrace.service.ServiceProxyFactory;
13 import at.fh.ooe.moc5.amazingrace.service.ServiceException;
14
15 /**
16  * Created by Thomas on 12/25/2015.
17  */
18 public class RoutesViewModel implements Serializable {
19
20     private UserContextModel userContext;
21     private RouteModel selectedRoute;
22     private List<RouteModel> routes;
23
24     private ServiceProxy proxy;
25
26     public RoutesViewModel(UserContextModel userContext) {
27         Objects.requireNonNull(userContext, "UserContext must not be null");
28         this.userContext = userContext;
29         proxy = ServiceProxyFactory.createServiceProxy();
30     }
31
32     //region Actions
33
34     /**
35      * Loads the routes
36      *
37      * @return the found routes
38      * @throws ServiceException if the proxy service throw an exception
39      */
40     public List<RouteModel> loadRoutes() throws ServiceException {
41         routes = proxy.loadRoutes(userContext.getCredentialsModel());
42         Collections.sort(routes, new Comparator<RouteModel>() {
43             @Override
44             public int compare(RouteModel lhs, RouteModel rhs) {
45                 if (lhs.isDone() == rhs.isDone()) {
46                     if (lhs.getVisitedCheckpoints().size() == rhs.getVisitedCheckpoints().size())
47                         ↪ {
48                             return lhs.getName().toUpperCase().compareTo(rhs.getName().toUpperCase());
49                         } else {
50                             return Integer.valueOf(rhs.getVisitedCheckpoints().size())
51                                 .compareTo(lhs.getVisitedCheckpoints().size());
52                         }
53                 } else {
54                     return Boolean.valueOf(lhs.isDone()).compareTo(rhs.isDone());
55                 }
56             }
57         });
58         return routes;
59     }

```


Übung 3

```

60  /**
61   * Resets the given route
62   *
63   * @param model the route to reset
64   * @return true if reset, false otherwise
65   * @throws ServiceException if the proxy service throw an exception
66   */
67  public boolean resetRoute(RouteModel model) throws ServiceException {
68      return proxy.resetRoute(new RouteRequestModel(userContext.getCredentialsModel(),
69          ↪ model.getId()));
69  }
70
71  /**
72   * Resets all routes
73   *
74   * @return true if reset, false otherwise
75   * @throws ServiceException if the proxy service throw an exception
76   */
77  public boolean resetAllRoutes() throws ServiceException {
78      return proxy.resetAllRoutes(userContext.getCredentialsModel());
79  }
80  //endregion
81
82  //region Getter and Setter
83  public RouteModel getSelectedRoute() {
84      return selectedRoute;
85  }
86
87  public void setSelectedRoute(RouteModel selectedRoute) {
88      this.selectedRoute = selectedRoute;
89  }
90
91  public List<RouteModel> getRoutes() {
92      return routes;
93  }
94  //endregion
95  }

```

Listing 16: UserContextModel.java

```

1 package at.fh.ooe.moc5.amazingrace.model.view;
2
3 import java.io.Serializable;
4 import java.util.Calendar;
5 import java.util.Objects;
6
7 import at.fh.ooe.moc5.amazingrace.model.json.CredentialsRequestModel;
8
9 /**
10  * Created by Thomas on 12/24/2015.
11  * This model represents an logged user for the application.
12  */
13 public class UserContextModel implements Serializable {
14
15     private final CredentialsRequestModel model;
16     private final Calendar loginDate;
17
18     public UserContextModel(CredentialsRequestModel model) {
19         Objects.requireNonNull(model);
20         this.model = model;
21         this.loginDate = Calendar.getInstance();
22     }
23
24     //region Getter and Setter
25     public CredentialsRequestModel getCredentialsModel() {
26         return model;
27     }
28
29     public Calendar getLoginDate() {
30         return loginDate;
31     }
32     //endregion
33 }

```

Listing 17: CheckpointViewModel.java

```

1 package at.fh.ooe.moc5.amazingrace.model.view;
2
3 import java.io.Serializable;
4 import java.util.List;
5 import java.util.Objects;
6
7 import at.fh.ooe.moc5.amazingrace.model.json.CheckpointModel;
8 import at.fh.ooe.moc5.amazingrace.model.json.RouteModel;
9 import at.fh.ooe.moc5.amazingrace.model.json.RouteRequestModel;
10 import at.fh.ooe.moc5.amazingrace.model.json.CheckpointRequestModel;
11 import at.fh.ooe.moc5.amazingrace.service.ServiceProxy;
12 import at.fh.ooe.moc5.amazingrace.service.ServiceException;
13 import at.fh.ooe.moc5.amazingrace.service.ServiceProxyFactory;
14
15 /**
16  * Created by Thomas on 12/25/2015.
17  */
18 public class CheckpointViewModel implements Serializable {
19
20     private String answer;
21     private final UserContextModel userContext;
22     private RouteModel route;
23
24     private ServiceProxy proxy;
25
26     public CheckpointViewModel(UserContextModel userContext, RouteModel route) {
27         Objects.requireNonNull(userContext, "View model needs user context set");
28         Objects.requireNonNull(route, "View model needs route set");
29
30         this.userContext = userContext;
31         this.route = route;
32         proxy = ServiceProxyFactory.createServiceProxy();
33     }
34
35     //region Actions
36     public boolean validateSecret() throws ServiceException {
37         if (!isValid()) {
38             return Boolean.FALSE;
39         }
40
41         return proxy.validateCheckpointSecret(new
42             ↪ CheckpointRequestModel(userContext.getCredentialsModel().userName,
43             ↪ userContext.getCredentialsModel().password, getNextCheckpoint().getId(), answer));
44     }
45
46     public boolean reloadRoute() throws ServiceException {
47         final List<RouteModel> routes = proxy.loadRoutes(userContext.getCredentialsModel());
48         int idx = -1;
49         if ((idx = routes.indexOf(route)) != -1) {
50             route = routes.get(idx);
51         }
52
53         return (route != null);
54     }
55
56     public boolean resetRoute() throws ServiceException {
57         return proxy.resetRoute(new RouteRequestModel(userContext.getCredentialsModel(),
58             ↪ route.getId()));
59     }
60
61 }

```

Übung 3

```
58     public boolean isValid() {
59         return (answer != null) && (!answer.trim().isEmpty());
60     }
61     //endregion
62
63     //region Getter and Setter
64     public String getAnswer() {
65         return answer;
66     }
67
68     public void setAnswer(String answer) {
69         this.answer = answer;
70     }
71
72     public UserContextModel getUserContext() {
73         return userContext;
74     }
75
76     public RouteModel getRoute() {
77         return route;
78     }
79
80     public CheckpointModel getNextCheckpoint() {
81         return route.getNextCheckpoint();
82     }
83
84     public List<CheckpointModel> getVisitedCheckpoints() {
85         return route.getVisitedCheckpoints();
86     }
87     //endregion
88 }
```

Übung 3

1.2.7 Service

Listing 18: ServiceProxy.java

```

1 package at.fh.ooe.moc5.amazingrace.service;
2
3 import java.io.Serializable;
4 import java.util.List;
5
6 import at.fh.ooe.moc5.amazingrace.model.json.CredentialsRequestModel;
7 import at.fh.ooe.moc5.amazingrace.model.json.RouteModel;
8 import at.fh.ooe.moc5.amazingrace.model.json.RouteRequestModel;
9 import at.fh.ooe.moc5.amazingrace.model.json.CheckpointRequestModel;
10
11 /**
12  * Created by Thomas on 12/25/2015.
13  * The specification of the service proxy
14  */
15 public interface ServiceProxy extends Serializable {
16
17     /**
18      * Validates the given user credentials.
19      *
20      * @param model the model containing authentication data
21      * @return true if the credentials are valid, false otherwise
22      * @throws ServiceException if an service error occurs. See contained error code for details
23      */
24     boolean validateCredentials(CredentialsRequestModel model) throws ServiceException;
25
26     /**
27      * Validates the checkpoint secret.
28      *
29      * @param model the model containing the checkpoint data and authentication data
30      * @return true if valid, false otherwise
31      * @throws ServiceException if an service error occurs. See contained error code for details
32      */
33     boolean validateCheckpointSecret(CheckpointRequestModel model) throws ServiceException;
34
35     /**
36      * Loads all routes.
37      *
38      * @param model the credentials model for authentication
39      * @return the loaded routes
40      * @throws ServiceException if an service error occurs. See contained error code for details
41      */
42     List<RouteModel> loadRoutes(CredentialsRequestModel model) throws ServiceException;
43
44     /**
45      * Resets the given route.
46      *
47      * @param model the model containing the route data and authentication data
48      * @return true if reset, false otherwise
49      * @throws ServiceException if an service error occurs. See contained error code for details
50      */
51     boolean resetRoute(RouteRequestModel model) throws ServiceException;
52
53     /**
54      * Resets all routes.
55      *
56      * @param model the model containing the authentication data
57      * @return true if reset, false otherwise
58      * @throws ServiceException if an service error occurs. See contained error code for details
59      */

```

Übung 3

```
60 |         boolean resetAllRoutes(CredentialsRequestModel model) throws ServiceException;  
61 |     }
```

Listing 19: RestServiceProxyImpl.java

```

1 package at.fh.ooe.moc5.amazingrace.service;
2
3 import com.google.gson.GsonBuilder;
4 import com.google.gson.JsonSyntaxException;
5 import com.google.gson.reflect.TypeToken;
6
7 import java.io.BufferedReader;
8 import java.io.BufferedWriter;
9 import java.io.InputStreamReader;
10 import java.io.OutputStreamWriter;
11 import java.net.SocketTimeoutException;
12 import java.net.URL;
13 import java.util.List;
14 import java.util.Objects;
15
16 import javax.net.ssl.HttpURLConnection;
17
18 import at.fh.ooe.moc5.amazingrace.model.json.CredentialsRequestModel;
19 import at.fh.ooe.moc5.amazingrace.model.json.RouteModel;
20 import at.fh.ooe.moc5.amazingrace.model.json.RouteRequestModel;
21 import at.fh.ooe.moc5.amazingrace.model.json.CheckpointRequestModel;
22 import at.fh.ooe.moc5.amazingrace.service.ServiceException.ServiceErrorCode;
23
24 /**
25  * Created by Thomas on 12/24/2015.
26  * <p/>
27  * Class for communicating with the rest backend.
28  */
29 public class RestServiceProxyImpl implements ServiceProxy {
30
31     public static final String REST_URL =
32         ↪ "https://demo.nexperts.com/MOC5/AmazingRaceService/AmazingRaceService.svc";
33     public static final String CHECK_CREDENTIALS = "/CheckCredentials";
34     public static final String GET_ROUTES_METHOD = "/GetRoutes";
35     public static final String METHOD_VISIT_CHECKPOINT = "/InformAboutVisitedCheckpoint";
36     public static final String METHOD_RESET_ALL_ROUTES = "/ResetAllRoutes";
37     public static final String METHOD_RESET_ROUTE = "/ResetRoute";
38
39     public static final int DEFAULT_TIME_OUT = 3000;
40
41     @Override
42     public boolean validateCredentials(CredentialsRequestModel model) throws ServiceException {
43         Objects.requireNonNull(model, "Cannot check credentials for null model");
44
45         try {
46             URL url = new URL(new StringBuilder(REST_URL)
47                 .append(CHECK_CREDENTIALS)
48                 .append("?")
49                 .append(model.toQueryString()).toString());
50             HttpURLConnection connection = (HttpURLConnection) url.openConnection();
51             connection.setRequestMethod("GET");
52             connection.setConnectTimeout(DEFAULT_TIME_OUT);
53             connection.setUseCaches(Boolean.FALSE);
54
55             return invokeBooleanResultMethod(connection, null, Boolean.FALSE);
56         } catch (ServiceException e) {
57             throw e;
58         } catch (Exception e) {
59             throw new ServiceException(ServiceErrorCode.UNKNOWN, e);
60         }
61     }
62 }

```

Übung 3

```

60     }
61
62     @Override
63     public boolean validateCheckpointSecret(CheckpointRequestModel model) throws ServiceException
64     ↪ {
65         Objects.requireNonNull(model, "Cannot check checkpoint for null model");
66
67         try {
68             URL url = new URL(new
69                 ↪ StringBuilder(REST_URL).append(METHOD_VISIT_CHECKPOINT).toString());
70             HttpURLConnection connection = (HttpURLConnection) url.openConnection();
71             connection.setRequestMethod("POST");
72             connection.setRequestProperty("Content-Type", "application/json");
73             connection.setConnectTimeout(DEFAULT_TIME_OUT);
74             connection.setUseCaches(Boolean.FALSE);
75             connection.setDoInput(Boolean.TRUE);
76             connection.setDoOutput(Boolean.TRUE);
77
78             return invokeBooleanResultMethod(connection, model, Boolean.TRUE);
79         } catch (ServiceException e) {
80             throw e;
81         } catch (Exception e) {
82             throw new ServiceException(ServiceErrorCode.UNKNOWN, e);
83         }
84     }
85
86     @Override
87     public List<RouteModel> loadRoutes(CredentialsRequestModel model) throws ServiceException {
88         Objects.requireNonNull(model, "Cannot get routes with missing credentials");
89
90         if (!validateCredentials(model)) {
91             throw new ServiceException(ServiceErrorCode.INVALID_CREDENTIALS);
92         }
93
94         try {
95             URL url = new URL(new StringBuilder(REST_URL)
96                 .append(GET_ROUTES_METHOD)
97                 .append("?")
98                 .append(model.toQueryString()).toString());
99             HttpURLConnection connection = (HttpURLConnection) url.openConnection();
100             connection.setRequestMethod("GET");
101             connection.setConnectTimeout(DEFAULT_TIME_OUT);
102             connection.setUseCaches(Boolean.FALSE);
103
104             final String response = readResponse(connection);
105             GsonBuilder builder = new GsonBuilder();
106             return builder.create().fromJson(response, new TypeToken<List<RouteModel>>() {
107                 ↪ .getType();
108             } catch (JsonSyntaxException e) {
109                 throw new ServiceException(ServiceErrorCode.INVALID_REQUEST, e);
110             } catch (Exception e) {
111                 throw new ServiceException(ServiceErrorCode.UNKNOWN, e);
112             }
113         }
114     }
115
116     @Override
117     public boolean resetRoute(RouteRequestModel model) throws ServiceException {
118         Objects.requireNonNull(model, "Cannot reset route for null model");
119
120         try {
121             URL url = new URL(new StringBuilder(REST_URL).append(METHOD_RESET_ROUTE).toString());
122             HttpURLConnection connection = (HttpURLConnection) url.openConnection();
123             connection.setRequestMethod("POST");

```


Übung 3

```

121         connection.setRequestProperty("Content-Type", "application/json");
122         connection.setConnectTimeout(DEFAULT_TIMEOUT);
123         connection.setUseCaches(Boolean.FALSE);
124         connection.setDoInput(Boolean.TRUE);
125         connection.setDoOutput(Boolean.TRUE);
126
127         return invokeBooleanRequestMethod(connection, model, Boolean.TRUE);
128     } catch (ServiceException e) {
129         throw e;
130     } catch (Exception e) {
131         throw new ServiceException(ServiceErrorCode.UNKNOWN, e);
132     }
133 }
134
135 @Override
136 public boolean resetAllRoutes(CredentialsRequestModel model) throws ServiceException {
137     Objects.requireNonNull(model, "Cannot reset all routes for null model");
138
139     try {
140         URL url = new URL(new
141             ↳ StringBuilder(REST_URL).append(METHOD_RESET_ALL_ROUTES).toString());
142         HttpsURLConnection connection = (HttpsURLConnection) url.openConnection();
143         connection.setRequestMethod("POST");
144         connection.setRequestProperty("Content-Type", "application/json");
145         connection.setConnectTimeout(DEFAULT_TIMEOUT);
146         connection.setUseCaches(Boolean.FALSE);
147         connection.setDoInput(Boolean.TRUE);
148         connection.setDoOutput(Boolean.TRUE);
149
150         return invokeBooleanRequestMethod(connection, model, Boolean.TRUE);
151     } catch (ServiceException e) {
152         throw e;
153     } catch (Exception e) {
154         throw new ServiceException(ServiceErrorCode.UNKNOWN, e);
155     }
156 }
157
158 // region Helper
159 /**
160  * Invokes a remote service method with an boolean result.
161  *
162  * @param connection the connection to the service method
163  * @param jsonModel the jsonModel to write into the connection, maybe null.
164  * @param checkCredentials true if the credentials shall be checked before calling the actual
165  ↳ service
166  * @param <T> the type of the json model which must be at least the
167  ↳ CredentialsModel
168  * @return true or false
169  * @throws ServiceException if the request failed for any reason, see contained error code for
170  ↳ details
171  */
172 private <T extends CredentialsRequestModel> boolean
173     ↳ invokeBooleanRequestMethod(HttpsURLConnection connection, T jsonModel, boolean
174     ↳ checkCredentials) throws ServiceException {
175     Objects.requireNonNull(connection, "Cannot invoke method on null connection");
176     if (checkCredentials) {
177         Objects.requireNonNull(jsonModel, "Cannot check credentials for null model");
178     }
179     if ((checkCredentials) && (!validateCredentials(new
180         ↳ CredentialsRequestModel(jsonModel.userName, jsonModel.password)))) {
181         throw new ServiceException(ServiceErrorCode.INVALID_CREDENTIALS);
182     }
183 }

```

Übung 3

```

177
178     try {
179         // Write Data for Post requests
180         if ((connection.getRequestMethod().equals("POST")) && (jsonModel != null)) {
181             writeData(connection, jsonModel);
182         }
183         // Read response
184         final String response = readResponse(connection);
185         if ((!"true".equals(response)) && (!"false".equals(response))) {
186             throw new ServiceException(ServiceErrorCode.INVALID_REQUEST);
187         }
188         return Boolean.parseBoolean(response);
189     } catch (ServiceException e) {
190         throw e;
191     } catch (Exception e) {
192         throw new ServiceException(ServiceErrorCode.UNKNOWN, e);
193     }
194 }
195
196 /**
197  * Reads the response from the connection.
198  *
199  * @param connection the connection to read response from
200  * @return the response string
201  * @throws ServiceException if an error occurs
202  * @see ServiceErrorCode for the ServiceException contained error code
203  */
204 private String readResponse(URLConnection connection) throws ServiceException {
205     Objects.requireNonNull(connection, "Cannot read from null connection");
206     try {
207         if (connection.getResponseCode() != 200) {
208             throw new ServiceException(ServiceErrorCode.REQUEST_NOT_OK);
209         }
210         try (final BufferedReader reader = new BufferedReader(new
211             ↳ InputStreamReader(connection.getInputStream()))) {
212             final StringBuilder builder = new StringBuilder();
213             String line;
214             while ((line = reader.readLine()) != null) {
215                 builder.append(line);
216             }
217             return builder.toString();
218         } catch (SocketTimeoutException ste) {
219             throw new ServiceException(ServiceErrorCode.TIMEOUT, ste);
220         } catch (Exception e) {
221             throw new ServiceException(ServiceErrorCode.UNKNOWN, e);
222         }
223     }
224
225 /**
226  * Writes data to the connection.
227  *
228  * @param connection the connection to write to
229  * @param jsonModelInstance the json model instance
230  * @throws ServiceException if the write fails, see contained error code for details
231  */
232 private void writeData(URLConnection connection, Object jsonModelInstance) throws
233     ↳ ServiceException {
234     Objects.requireNonNull(connection, "Cannot write to null connection");
235     try {
236         try (final BufferedWriter writer = new BufferedWriter(new
237             ↳ OutputStreamWriter(connection.getOutputStream()))) {
238             final String data = new GsonBuilder().create().toJson(jsonModelInstance);

```

Übung 3

```
237         writer.write(data, 0, data.length());
238         writer.flush();
239     }
240     } catch (JsonSyntaxException jse) {
241         throw new ServiceException(ServiceErrorCode.INVALID_REQUEST, jse);
242     } catch (SocketTimeoutException ste) {
243         throw new ServiceException(ServiceErrorCode.TIMEOUT, ste);
244     } catch (Exception e) {
245         throw new ServiceException(ServiceErrorCode.UNKNOWN, e);
246     }
247 }
248 // endregion
249 }
```

Listing 20: ServiceException.java

```

1 package at.fh.ooe.moc5.amazingrace.service;
2
3 /**
4  * Created by Thomas on 12/25/2015.
5  */
6 public class ServiceException extends Exception {
7     /**
8      * Enumeration which defines the common service errors
9      */
10    public static enum ServiceErrorCode {
11        INVALID_REQUEST,
12        TIMEOUT,
13        UNKNOWN,
14        REQUEST_NOT_OK,
15        INVALID_CREDENTIALS
16    }
17
18    private ServiceErrorCode errorCode;
19
20    public ServiceException() {
21        this(null, null, null);
22    }
23
24    public ServiceException(String detailMessage) {
25        this(null, detailMessage, null);
26    }
27
28    public ServiceException(String detailMessage, Throwable throwable) {
29        this(null, detailMessage, throwable);
30    }
31
32    public ServiceException(Throwable throwable) {
33        this(null, null, throwable);
34    }
35
36    public ServiceException(ServiceErrorCode errorCode) {
37        this(errorCode, null, null);
38    }
39
40    public ServiceException(ServiceErrorCode errorCode, String detailMessage) {
41        this(errorCode, detailMessage, null);
42    }
43
44
45    public ServiceException(ServiceErrorCode errorCode, Throwable throwable) {
46        this(errorCode, null, throwable);
47    }
48
49    public ServiceException(ServiceErrorCode errorCode, String detailMessage, Throwable throwable)
50    ↪ {
51        super(detailMessage, throwable);
52        this.errorCode = errorCode;
53    }
54
55    public ServiceErrorCode getErrorCode() {
56        return errorCode;
57    }
58 }

```

Listing 21: ServiceProxyFactory.java

```
1 package at.fh.ooe.moc5.amazingrace.service;
2
3 /**
4  * Created by Thomas on 12/24/2015.
5  */
6 public class ServiceProxyFactory {
7
8     private static ServiceProxy proxy;
9
10    private ServiceProxyFactory() {
11    }
12
13    public static ServiceProxy createServiceProxy() {
14        if (proxy == null) {
15            proxy = new RestServiceProxyImpl();
16        }
17        return proxy;
18    }
19 }
```

Übung 3

1.2.8 Utilities

Listing 22: DialogUtil.java

```

1  package at.fh.ooe.moc5.amazingrace.util;
2
3  import android.app.AlertDialog;
4  import android.content.Context;
5  import android.content.DialogInterface;
6
7  import java.util.Objects;
8
9  import at.fh.ooe.moc5.amazingrace.R;
10
11  /**
12   * Created by Thomas on 12/24/2015.
13   * Utility for creating dialogs.
14   */
15  public class DialogUtil {
16
17      /**
18       * Creates a error alert dialog.
19       *
20       * @param ctx      the context to create the alert dialog for
21       * @param message   the resource id of the alert dialog error message
22       * @param listener  the button listener
23       * @return the created alert dialog
24       */
25  public static AlertDialog createErrorDialog(final Context ctx, final int message, final
26      ↪ DialogInterface.OnClickListener listener) {
27      Objects.requireNonNull(ctx, "Cannot create dialog for null context");
28      Objects.requireNonNull(listener, "Listener for buttons mut be given");
29
30      return new AlertDialog.Builder(ctx)
31          .setTitle(R.string.dialog_title_error)
32          .setMessage(message)
33          .setPositiveButton(R.string.action_ok, listener)
34          .create();
35  }
36
37  /**
38   * Creates a yes, no alert dialog
39   *
40   * @param ctx      the context to create the alert dialog for
41   * @param title     the resource id of the alert dialog title
42   * @param message   the resource id of the alert dialog message
43   * @param listener  the yes, no listener
44   * @return the created alert dialog
45   */
46  public static AlertDialog createYesNoAlertDialog(final Context ctx, final int title, final int
47      ↪ message, final DialogInterface.OnClickListener listener) {
48      Objects.requireNonNull(ctx, "Cannot create dialog for null context");
49      Objects.requireNonNull(listener, "Listener for buttons mut be given");
50
51      return new AlertDialog.Builder(ctx)
52          .setTitle(title)
53          .setMessage(message)
54          .setPositiveButton(R.string.action_yes, listener)
55          .setNegativeButton(R.string.action_no, listener)
56          .create();
57  }
58
59  /**

```

Übung 3

```
58     * Creates a alert dialog with an custom view as content.
59     *
60     * @param ctx      the context to create dialog for
61     * @param viewId    the resource id of the view
62     * @param title     the resource id of the alert dialog title
63     * @param listener  the button listener
64     * @return the created alert dialog
65     */
66     public static AlertDialog createCustomContentDialog(final Context ctx, final int viewId, final
↪     int title, final DialogInterface.OnClickListener listener) {
67         Objects.requireNonNull(ctx, "Cannot create dialog for null context");
68         Objects.requireNonNull(listener, "Listener for buttons mut be given");
69
70         return new AlertDialog.Builder(ctx)
71             .setTitle(title)
72             .setView(viewId)
73             .setPositiveButton(R.string.action_ok, listener)
74             .create();
75     }
76 }
```

Übung 3

1.2.9 Watcher

Listing 23: AnswerButtonTextWatcher.java

```

1  package at.fh.ooe.moc5.amazingrace.watcher;
2
3  import android.text.Editable;
4  import android.text.TextWatcher;
5  import android.widget.Button;
6
7  import at.fh.ooe.moc5.amazingrace.R;
8  import at.fh.ooe.moc5.amazingrace.activity.CheckpointActivity;
9  import at.fh.ooe.moc5.amazingrace.model.view.CheckpointViewModel;
10
11  /**
12   * Created by Thomas on 12/24/2015.
13   * A watcher which enables the answer button if the backed view model is valid, disables it
14   * ↪ otherwise
15   */
16  public class AnswerButtonTextWatcher implements TextWatcher {
17
18      private final CheckpointViewModel viewModel;
19      private final CheckpointActivity activity;
20
21      public AnswerButtonTextWatcher(CheckpointActivity activity, CheckpointViewModel viewModel) {
22          this.activity = activity;
23          this.viewModel = viewModel;
24      }
25
26      @Override
27      public void beforeTextChanged(CharSequence s, int start, int count, int after) {
28      }
29
30      @Override
31      public void onTextChanged(CharSequence s, int start, int before, int count) {
32      }
33
34      @Override
35      public void afterTextChanged(Editable s) {
36          ((Button) activity.findViewById(R.id.answerBtn)).setEnabled(viewModel.isValid());
37      }
38  }

```


Listing 24: CheckpointViewModelBindingTextWatcher.java

```

1 package at.fh.ooe.moc5.amazingrace.watcher;
2
3 import android.text.Editable;
4 import android.text.TextWatcher;
5
6 import java.util.Objects;
7
8 import at.fh.ooe.moc5.amazingrace.model.view.CheckpointViewModel;
9
10 /**
11  * Created by Thomas on 12/24/2015.
12  * Performs a binding to the given view model
13  */
14 public class CheckpointViewModelBindingTextWatcher implements TextWatcher {
15
16     private final CheckpointViewModel view;
17
18     public CheckpointViewModelBindingTextWatcher(CheckpointViewModel view) {
19         Objects.requireNonNull(view);
20         this.view = view;
21     }
22
23     @Override
24     public void beforeTextChanged(CharSequence s, int start, int count, int after) {
25     }
26
27     @Override
28     public void onTextChanged(CharSequence s, int start, int before, int count) {
29     }
30
31     @Override
32     public void afterTextChanged(Editable s) {
33         view.setAnswer(s.toString());
34     }
35 }

```

Listing 25: LoginButtonTextWatcher.java

```
1 package at.fh.ooe.moc5.amazingrace.watcher;
2
3 import android.text.Editable;
4 import android.text.TextWatcher;
5 import android.widget.Button;
6
7 import at.fh.ooe.moc5.amazingrace.R;
8 import at.fh.ooe.moc5.amazingrace.activity.LoginActivity;
9 import at.fh.ooe.moc5.amazingrace.model.view.LoginViewModel;
10
11 /**
12  * Created by Thomas on 12/24/2015.
13  * A watcher which enables the login button if the backed view model is valid, disables it
14  * ↪ otherwise
15  */
16 public class LoginButtonTextWatcher implements TextWatcher {
17     private LoginViewModel viewModel;
18     private final LoginActivity activity;
19
20     public LoginButtonTextWatcher(LoginActivity activity, LoginViewModel viewModel) {
21         this.activity = activity;
22         this.viewModel = viewModel;
23     }
24
25     @Override
26     public void beforeTextChanged(CharSequence s, int start, int count, int after) {
27     }
28
29     @Override
30     public void onTextChanged(CharSequence s, int start, int before, int count) {
31     }
32
33     @Override
34     public void afterTextChanged(Editable s) {
35         Button loginButton = (Button) activity.findViewById(R.id.loginBtn);
36         loginButton.setEnabled(viewModel.isValid());
37     }
38 }
```

Listing 26: LoginViewModelBindingTextWatcher.java

```

1  package at.fh.ooe.moc5.amazingrace.watcher;
2
3  import android.text.Editable;
4  import android.text.TextWatcher;
5  import android.widget.EditText;
6
7  import java.util.Objects;
8
9  import at.fh.ooe.moc5.amazingrace.R;
10 import at.fh.ooe.moc5.amazingrace.model.view.LoginViewModel;
11
12 /**
13  * Created by Thomas on 12/24/2015.
14  * Performs a binding to the given view model
15  */
16 public class LoginViewModelBindingTextWatcher implements TextWatcher {
17
18     private final LoginViewModel view;
19     private final EditText text;
20
21     public LoginViewModelBindingTextWatcher(LoginViewModel view, EditText text) {
22         Objects.requireNonNull(view);
23         Objects.requireNonNull(text);
24         this.view = view;
25         this.text = text;
26     }
27
28     @Override
29     public void beforeTextChanged(CharSequence s, int start, int count, int after) {
30     }
31
32     @Override
33     public void onTextChanged(CharSequence s, int start, int before, int count) {
34     }
35
36     @Override
37     public void afterTextChanged(Editable s) {
38         switch (text.getId()) {
39             case R.id.usernameEdTxt:
40                 view.setUsername(s.toString());
41                 break;
42             case R.id.passwordEdTxt:
43                 view.setPassword(s.toString());
44                 break;
45         }
46     }
47 }

```

Übung 3

1.3 Layout

Listing 27: activity_login.xml

```

1  <?xml version="1.0" encoding="utf-8"?>
2  <RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
3      xmlns:tools="http://schemas.android.com/tools"
4      android:layout_width="match_parent"
5      android:layout_height="match_parent"
6      android:paddingBottom="@dimen/activity_vertical_margin"
7      android:paddingLeft="@dimen/activity_horizontal_margin"
8      android:paddingRight="@dimen/activity_horizontal_margin"
9      android:paddingTop="@dimen/activity_vertical_margin"
10     tools:context="at.fh.ooe.moc5.amazingrace.activity.LoginActivity">
11
12
13     <LinearLayout
14         android:layout_width="match_parent"
15         android:layout_height="wrap_content"
16         android:layout_centerVertical="true"
17         android:orientation="vertical"
18         android:weightSum="1">
19
20         <LinearLayout
21             android:layout_width="match_parent"
22             android:layout_height="wrap_content"
23             android:layout_gravity="center_horizontal"
24             android:orientation="horizontal">
25
26             <TextView
27                 android:id="@+id/usernameLabel"
28                 android:layout_width="wrap_content"
29                 android:layout_height="match_parent"
30                 android:layout_gravity="center"
31                 android:layout_weight="1"
32                 android:gravity="left|center"
33                 android:text="@string/label_username"
34                 android:textAppearance="?android:attr/textAppearanceMedium" />
35
36             <EditText
37                 android:id="@+id/usernameEdTxt"
38                 android:layout_width="wrap_content"
39                 android:layout_height="wrap_content"
40                 android:layout_gravity="center|center"
41                 android:layout_weight="8" />
42         </LinearLayout>
43
44         <LinearLayout
45             android:layout_width="match_parent"
46             android:layout_height="wrap_content"
47             android:layout_gravity="center_horizontal"
48             android:orientation="horizontal">
49
50             <TextView
51                 android:id="@+id/passwordLabel"
52                 android:layout_width="wrap_content"
53                 android:layout_height="match_parent"
54                 android:layout_gravity="left|center"
55                 android:layout_weight="1"
56                 android:gravity="left|center"
57                 android:text="@string/label_password"
58                 android:textAppearance="?android:attr/textAppearanceMedium" />
59

```

Übung 3

```

60         <EditText
61             android:id="@+id/passwordEdTxt"
62             android:layout_width="wrap_content"
63             android:layout_height="wrap_content"
64             android:layout_gravity="center|center"
65             android:layout_weight="8"
66             android:inputType="textPassword" />
67     </LinearLayout>
68
69     <RelativeLayout
70         android:layout_width="match_parent"
71         android:layout_height="wrap_content"
72         android:layout_weight="0.07"
73         android:orientation="horizontal"
74         android:weightSum="1">
75
76         <LinearLayout
77             android:layout_width="wrap_content"
78             android:layout_height="wrap_content"
79             android:layout_alignParentLeft="false"
80             android:layout_alignParentTop="false"
81             android:layout_centerHorizontal="true"
82             android:layout_centerInParent="true"
83             android:layout_centerVertical="true"
84             android:orientation="horizontal">
85
86             <Button
87                 android:id="@+id/loginBtn"
88                 android:layout_width="wrap_content"
89                 android:layout_height="wrap_content"
90                 android:layout_margin="5pt"
91                 android:gravity="center|center"
92                 android:text="@string/action_login" />
93
94             <Button
95                 android:id="@+id/cancelBtn"
96                 android:layout_width="wrap_content"
97                 android:layout_height="wrap_content"
98                 android:layout_margin="5pt"
99                 android:gravity="center|center"
100                 android:text="@string/action_cancel" />
101
102         </LinearLayout>
103
104     </RelativeLayout>
105 </LinearLayout>
106 </RelativeLayout>

```

Listing 28: activity_route.xml

```
1 <?xml version="1.0" encoding="utf-8"?>
2 <RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
3     xmlns:tools="http://schemas.android.com/tools"
4     android:layout_width="match_parent"
5     android:layout_height="match_parent"
6     android:paddingBottom="@dimen/activity_vertical_margin"
7     android:paddingLeft="@dimen/activity_horizontal_margin"
8     android:paddingRight="@dimen/activity_horizontal_margin"
9     android:paddingTop="@dimen/activity_vertical_margin"
10    tools:context="at.fh.ooe.moc5.amazingrace.activity.RouteActivity">
11
12    <LinearLayout
13        android:layout_width="match_parent"
14        android:layout_height="match_parent"
15        android:orientation="vertical"
16        android:weightSum="1">
17
18        <ListView
19            android:id="@+id/listRoute"
20            android:layout_width="match_parent"
21            android:layout_height="match_parent"/>
22    </LinearLayout>
23 </RelativeLayout>
```

Listing 29: activity_checkpoint.xml

```

1  <?xml version="1.0" encoding="utf-8"?>
2  <RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
3      xmlns:tools="http://schemas.android.com/tools"
4      android:layout_width="match_parent"
5      android:layout_height="wrap_content"
6      android:paddingBottom="@dimen/activity_vertical_margin"
7      android:paddingLeft="@dimen/activity_horizontal_margin"
8      android:paddingRight="@dimen/activity_horizontal_margin"
9      android:paddingTop="@dimen/activity_vertical_margin"
10     tools:context=".activity.CheckpointActivity">
11
12     <ScrollView
13         android:layout_width="match_parent"
14         android:layout_height="match_parent"
15         android:fillViewport="true">
16
17         <LinearLayout
18             android:layout_width="match_parent"
19             android:layout_height="match_parent"
20             android:layout_alignParentStart="true"
21             android:layout_alignParentTop="true"
22             android:orientation="vertical">
23
24             <LinearLayout
25                 android:id="@+id/routeCheckpointContainer"
26                 android:layout_width="match_parent"
27                 android:layout_height="wrap_content"
28                 android:layout_alignParentStart="true"
29                 android:layout_alignParentTop="true"
30                 android:layout_margin="5pt"
31                 android:orientation="vertical">
32
33                 <TextView
34                     android:id="@+id/nextCheckpointTxt"
35                     android:layout_width="match_parent"
36                     android:layout_height="wrap_content"
37                     android:layout_marginBottom="5pt"
38                     android:textAppearance="?android:attr/textAppearanceMedium"
39                     android:textStyle="bold"/>
40
41                 <TextView
42                     android:id="@+id/nextCheckpointHintTxt"
43                     android:layout_width="match_parent"
44                     android:layout_height="wrap_content"
45                     android:layout_marginBottom="5pt"
46                     android:textAppearance="?android:attr/textAppearanceMedium" />
47
48                 <EditText
49                     android:id="@+id/hintAnswerEdTxt"
50                     android:layout_width="match_parent"
51                     android:layout_height="wrap_content"
52                     android:layout_marginBottom="5pt" />
53
54                 <Button
55                     android:id="@+id/answerBtn"
56                     android:layout_width="wrap_content"
57                     android:layout_height="wrap_content"
58                     android:gravity="center|center"
59                     android:text="@string/action_answer" />
60             </LinearLayout>

```

Übung 3

```

61
62 <LinearLayout
63     android:id="@+id/routeCheckpointMapContainerLabel"
64     android:layout_width="match_parent"
65     android:layout_height="wrap_content"
66     android:layout_marginBottom="5pt"
67     android:orientation="horizontal">
68
69     <ImageView
70         android:layout_width="20pt"
71         android:layout_height="20pt"
72         android:clickable="true"
73         android:onClick="toggleVisibility"
74         android:src="@drawable/app_icon" />
75
76     <TextView
77         android:layout_width="match_parent"
78         android:layout_height="match_parent"
79         android:layout_marginLeft="5pt"
80         android:clickable="true"
81         android:gravity="left|center"
82         android:onClick="toggleVisibility"
83         android:text="@string/checkpoint_group_map"
84         android:textAppearance="?android:attr/textAppearanceLarge"
85         android:textStyle="bold" />
86 </LinearLayout>
87
88 <LinearLayout
89     android:id="@+id/routeCheckpointMapContainer"
90     android:layout_width="match_parent"
91     android:layout_height="200dp"
92     android:layout_marginBottom="10pt"
93     android:layout_alignParentStart="true"
94     android:layout_alignParentTop="true"
95     android:orientation="vertical">
96
97     <fragment
98         android:id="@+id/routeCheckpointMap"
99         android:name="com.google.android.gms.maps.MapFragment"
100         android:layout_width="match_parent"
101         android:layout_height="match_parent" />
102 </LinearLayout>
103
104 <LinearLayout
105     android:id="@+id/routeCheckpointListContainerLabel"
106     android:layout_width="match_parent"
107     android:layout_height="wrap_content"
108     android:layout_marginBottom="5pt"
109     android:orientation="horizontal">
110
111     <ImageView
112         android:layout_width="20pt"
113         android:layout_height="20pt"
114         android:clickable="true"
115         android:onClick="toggleVisibility"
116         android:src="@drawable/list_icon" />
117
118     <TextView
119         android:layout_width="match_parent"
120         android:layout_height="match_parent"
121         android:layout_marginLeft="5pt"
122         android:clickable="true"
123         android:gravity="left|center"

```


Übung 3

```

124         android:onClick="toggleVisibility"
125         android:text="@string/checkpoint_group_list"
126         android:textAppearance="?android:attr/textAppearanceLarge"
127         android:textStyle="bold" />
128     </LinearLayout>
129
130     <LinearLayout
131         android:id="@+id/routeCheckpointListContainer"
132         android:layout_width="match_parent"
133         android:layout_height="wrap_content"
134         android:layout_alignParentStart="true"
135         android:layout_alignParentTop="true"
136         android:orientation="vertical">
137
138         <LinearLayout
139             android:id="@+id/routeCheckpointListView"
140             android:layout_width="match_parent"
141             android:layout_height="wrap_content"
142             android:layout_below="@+id/routeListTitleTxt"
143             android:orientation="vertical" />
144
145     </LinearLayout>
146 </ScrollView>
147 </RelativeLayout>

```

Listing 30: view_checkpoint_item.xml

```

1  <?xml version="1.0" encoding="utf-8"?>
2  <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
3      android:layout_width="match_parent"
4      android:layout_height="wrap_content"
5      android:layout_centerVertical="true"
6      android:orientation="horizontal">
7
8      <ImageView
9          android:id="@+id/checkpointItemImage"
10         android:layout_width="15pt"
11         android:layout_height="15pt"
12         android:layout_margin="3pt" />
13
14     <TextView
15         android:id="@+id/checkpointItemText"
16         android:layout_width="match_parent"
17         android:layout_height="match_parent"
18         android:layout_margin="3pt"
19         android:gravity="left|center" />
20 </LinearLayout>

```

Übung 3

Listing 31: view_route_item.xml

```

1  <?xml version="1.0" encoding="utf-8"?>
2  <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
3      android:layout_width="match_parent"
4      android:layout_height="wrap_content"
5      android:layout_centerVertical="true"
6      android:orientation="horizontal">
7
8      <ImageView
9          android:id="@+id/routeItemImage"
10         android:layout_width="15pt"
11         android:layout_height="15pt"
12         android:layout_margin="3pt" />
13
14     <TextView
15         android:id="@+id/routeItemText"
16         android:layout_width="match_parent"
17         android:layout_height="match_parent"
18         android:layout_margin="3pt"
19         android:gravity="left|center" />
20 </LinearLayout>

```

Listing 32: view_congratulations_dialog.xml

```

1  <?xml version="1.0" encoding="utf-8"?>
2  <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
3      android:layout_width="match_parent"
4      android:layout_height="wrap_content"
5      android:layout_centerVertical="true"
6      android:orientation="vertical"
7      android:weightSum="1">
8
9      <ImageView
10         android:layout_width="50pt"
11         android:layout_height="50pt"
12         android:src="@drawable/congratulations_icon"
13         android:layout_gravity="center|center"/>
14
15     <TextView
16         android:layout_width="match_parent"
17         android:layout_height="match_parent"
18         android:layout_margin="5pt"
19         android:layout_weight="1"
20         android:gravity="center|center"
21         android:text="@string/info_congratulations" />
22 </LinearLayout>

```

Übung 3

1.4 Values

Listing 33: strings.xml

```

1 <resources>
2   <string name="application_title">AmazingRace</string>
3   <string name="activity_title_login">Login AmazingRace</string>
4   <string name="activity_title_route">AmazingRace Routes</string>
5   <string name="activity_title_checkpoint">Route checkpoints</string>
6   <string name="action_answer">Answer</string>
7   <string name="action_cancel">Cancel</string>
8   <string name="action_ok">Ok</string>
9   <string name="action_yes">Yes</string>
10  <string name="action_login">Login</string>
11  <string name="action_settings">Settings</string>
12  <string name="action_no">No</string>
13  <string name="action_play">Play</string>
14  <string name="action_reset">Reset</string>
15  <string name="action_reset_all">Reset All</string>
16  <string name="action_close">Close</string>
17  <string name="action_open">Open</string>
18  <string name="action_reload">Reload</string>
19  <string name="label_username">Username</string>
20  <string name="label_password">Password</string>
21  <string name="progress_login">Logging in ...</string>
22  <string name="progress_loading_routes">Loading routes ...</string>
23  <string name="progress_updating_route">Updating route ...</string>
24  <string name="progress_validating_secret">Validating secret</string>
25  <string name="progress_title">Processing</string>
26  <string name="progress_resetting_routes">Resetting route(s) ...</string>
27  <string name="dialog_title_error">Error</string>
28  <string name="dialog_title_warning">Warning</string>
29  <string name="dialog_title_info">Info</string>
30  <string name="error_request_invalid">Sorry. Request failed because it was invalid</string>
31  <string name="error_request_timeout">Sorry. Request failed because of an timeout</string>
32  <string name="error_unknown">Sorry. An unknown error occurred</string>
33  <string name="error_login_failed">Username and/or password not correct</string>
34  <string name="error_user_became_invalid">Your user is not valid anymore. You will be logged
    ↪ out.</string>
35  <string name="error_user_not_logged">Your are no longer logged in. You will be returned to the
    ↪ login page</string>
36  <string name="error_route_reset_failed">The route(s) could not be reset. Please try
    ↪ again</string>
37  <string name="error_no_network">This application needs an active internet connection to
    ↪ work</string>
38  <string name="warning_want_quit">Do you really want to quit ?</string>
39  <string name="info_secret_ok">Secret ok. Next checkpoint has been enabled</string>
40  <string name="info_secret_wrong">Sorry wrong secret. Guess again</string>
41  <string name="info_congratulations">Congratulations, you have finished this route</string>
42  <string name="finished">finished</string>
43  <string name="done">done</string>
44  <string name="open">open</string>
45  <string name="checkpoint_group_secret">Checkpoint Secret</string>
46  <string name="checkpoint_group_list">Checkpoint List</string>
47  <string name="checkpoint_group_map">Checkpoint Map</string>
48  <string name="google_maps_key">AIzaSyCNV6I75G-wopgeM3st0HeUiTzgWcINjy8</string>
49 </resources>

```

Übung 3

1.5 Tests

Folgend sind die Tests der Anwendung *AmazingRace* angeführt.

1.5.1 Login Aktivität

Folgend sind die Tests der Login Aktivität angeführt.

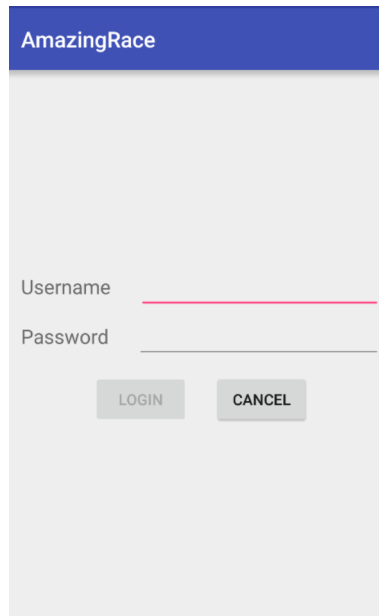


Abbildung 4: Login Aktivität

Der Button *Login* ist hierbei solange deaktiviert solange eine der beiden Eingabefelder keinen Text beinhaltet. Dies bedeutet aber auch, das leere Passwörter nicht unterstützt werden.

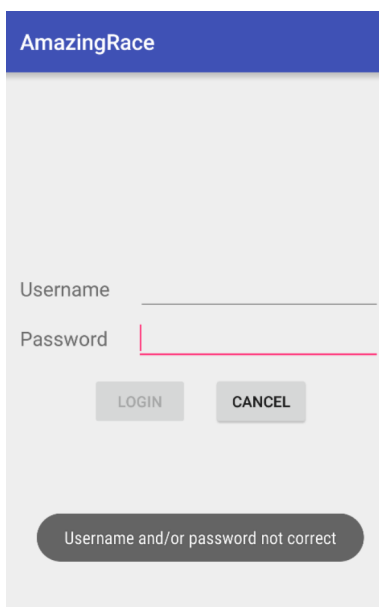


Abbildung 5: Login fehlgeschlagen

Sollte der Login fehlschlagen so werden beide Eingabefelder zurückgesetzt und eine Fehlermeldung in Form eines *Toasts* angezeigt.

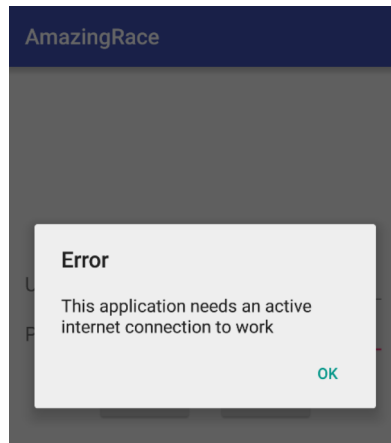


Abbildung 6: Keine Internetverbindung

Sollte keine Netzwerkverbindung vorhanden sein so wird ein entsprechender *Dialog* angezeigt.

1.5.2 Routen Aktivität

Folgend sind die Tests für die Routen Aktivität angeführt.

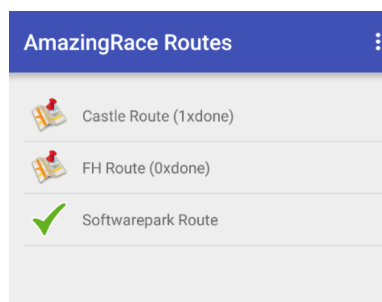


Abbildung 7: Routen Aktivität

Nach einem erfolgreichen Login oder durch Drücken des *Back Button* in der Checkpoint Aktivität gelangt man zur Übersicht der verfügbaren Routen. Hierbei werden die Routen wie folgt sortiert.

1. Offene Routen
 - (a) Anzahl der besuchten Checkpoints absteigend
 - (b) Routenname aufsteigend
2. Abgeschlossene Routen
 - (a) Routenname aufsteigend

Ist eine Route bereits abgeschlossen so wird eine entsprechender Icon angezeigt.

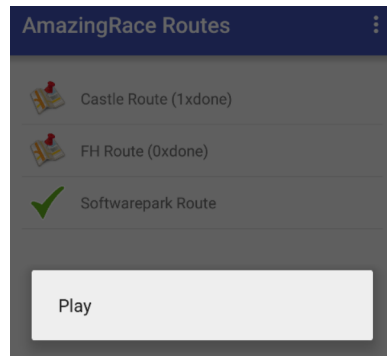


Abbildung 8: Kontextmenü einer offenen Route

Ist eine Route offen so wird beim Kontextmenü lediglich eine Option *Play* angezeigt.

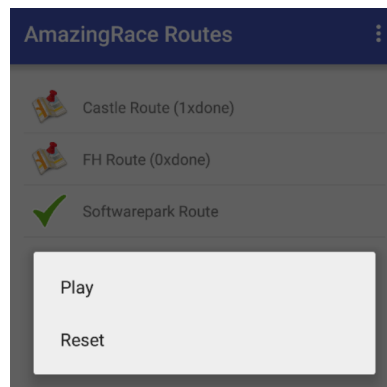


Abbildung 9: Kontextmenü einer gestarteten Route

Wurde eine Route bereits gestartet so werden beim Kontextmenü zwei Optionen *Play*, *Reset* angezeigt.

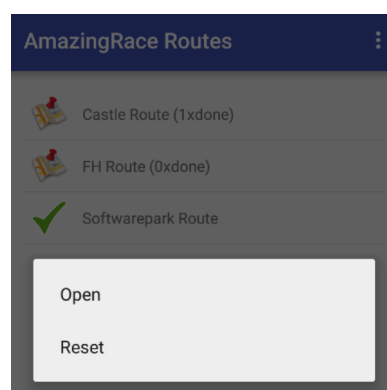


Abbildung 10: Kontextmenü einer abgeschlossenen Route

Ist eine Route bereits abgeschlossen so werden beim Kontextmenü zwei Optionen *Open*, *Reset* angezeigt.

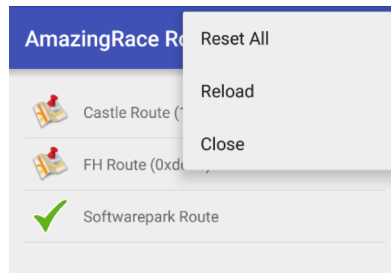


Abbildung 11: Routen Aktivität Menü

Die Aktivität stellt drei Optionen *Reset All*, *Reload*, *Close* über ein Menü zur Verfügung.

1. *Reset All*
Setzt alle zur Verfügung stehenden Routen zurück
2. *Reload*
Lädt alle zur Verfügung stehenden Routen erneut
3. *Close*
Öffnet einen Dialog, über den im Falle einer Bestätigung die Anwendung geschlossen wird.

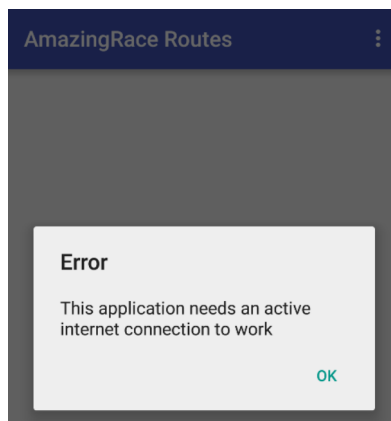


Abbildung 12: Keine Netzwerkverbindung

Sollte keine Netzwerkverbindung vorhanden sein so wird ein entsprechender Dialog angezeigt.

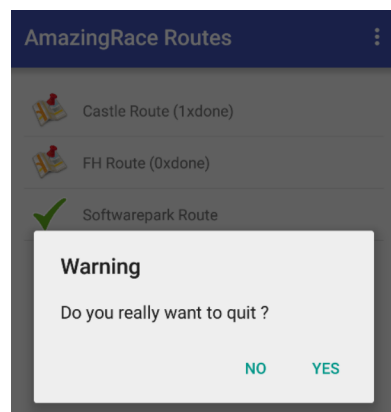


Abbildung 13: Anwendung schließen Dialog

Wird *Yes* gedrückt wird die Anwendung geschlossen, im Falle von *No* wird lediglich der *Dialog* geschlossen.

1.5.3 Checkpoint Aktivität

Folgend sind die Tests für die Checkpoint Aktivität angeführt.

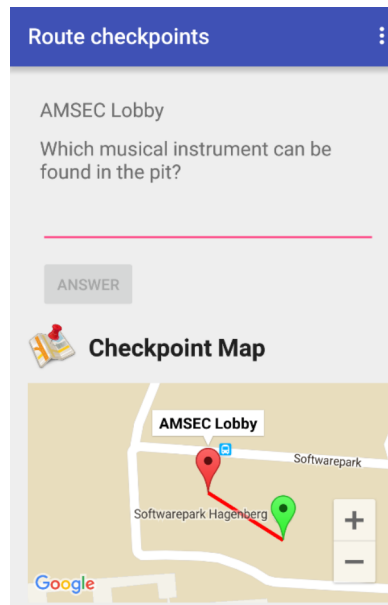


Abbildung 14: Offene Route Teil 1

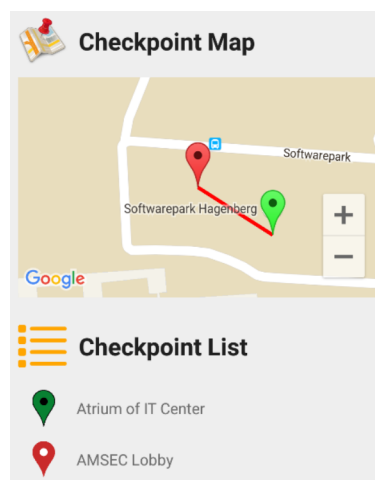


Abbildung 15: Offene Route Teil 2

Nachdem eine Route ausgewählt wurde wird auf die Checkpoint Aktivität gewechselt. Es gibt zwei Möglichkeiten die Checkpoints anzuzeigen.

1. *Google Map*
Zeigt alle Checkpoints an wobei ein noch nicht besuchter Checkpoint rot markiert wird
2. *Checkpoint Liste*
Zeigt alle besuchten Checkpoints in Listenform an wobei ein nicht besuchter Checkpoint ebenfalls rot markiert wird.

In diesem Fall ist die Route noch nicht abgeschlossen.

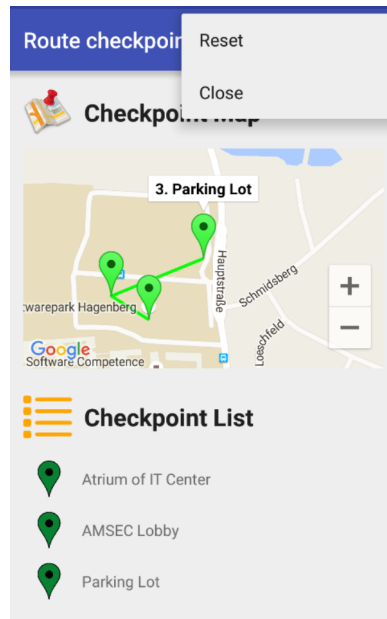


Abbildung 16: Optionen einer abgeschlossene oder gestarteten Route

Bei einer abgeschlossenen oder bereits gestarteten Route stehen die zwei Optionen *Reset*, *Close* zur Verfügung.

1. *Reset*
Setzt die aktuelle Route zurück
2. *Close*
Öffnet einen Dialog wobei bei dessen Bestätigung die Anwendung geschlossen wird.

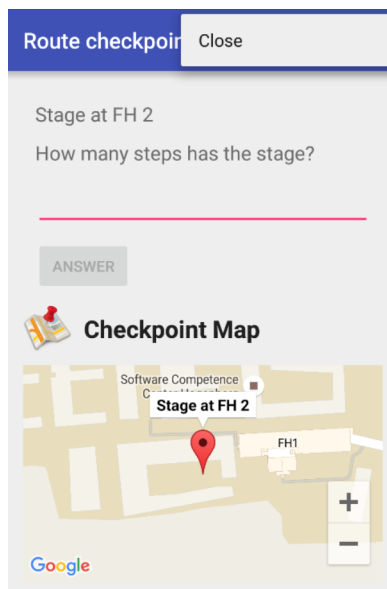


Abbildung 17: Optionen einer offenen Route

Bei einer offene Route steht nur die eine Option *Close* zur Verfügung.

Übung 3

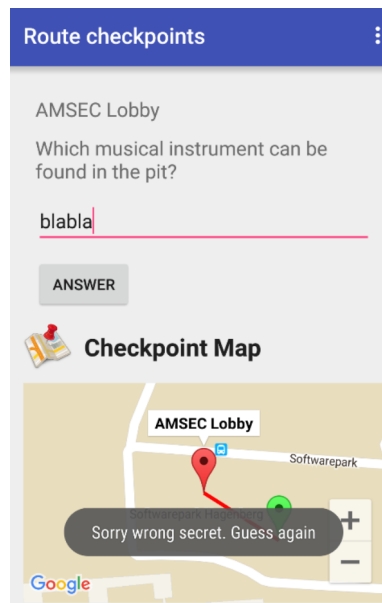


Abbildung 18: Frage falsch beantwortet

Sollte eine Frage falsch beantwortet worden sein so wird eine Meldung in Form eines Toast ausgegeben.

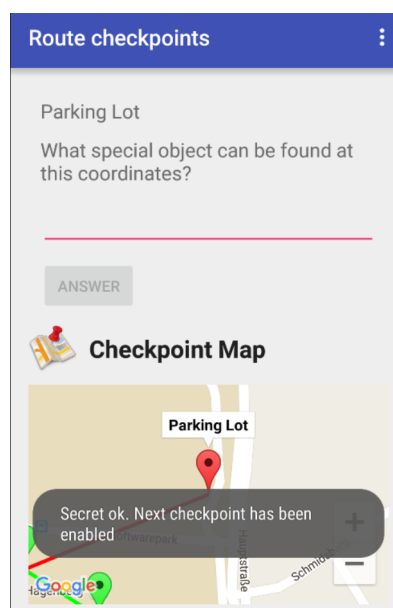


Abbildung 19: Frage korrekt beantwortet

Sollte eine Frage korrekt beantwortet worden sein so wird eine Meldung in Form eines *Toast* ausgegeben.

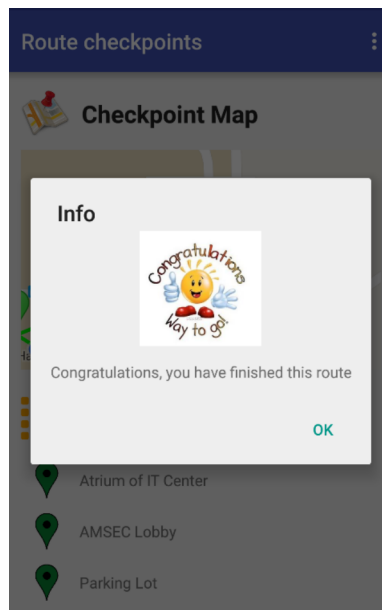


Abbildung 20: Route abgeschlossen

Nachdem die letzte Frage korrekt beantwortet wurde wird ein *Dialog* mit einer Erfolgsmeldung ausgegeben. Es wird auf der Checkpoint Aktivität verblieben wobei die Ansicht sich wie bei einer abgeschlossenen Route ändert.

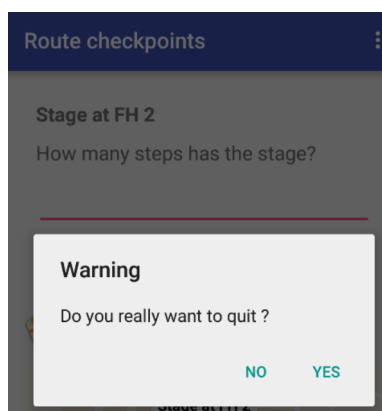


Abbildung 21: Anwendung schließen Dialog

Wird *Yes* gedrückt wird die Anwendung geschlossen, im Falle von *No* wird lediglich der *Dialog* geschlossen.