ACTIVITY **3.1.2**

User Authentication

INTRODUCTION

In the previous activity, you focused on the UI and created one screen that will be used for both logging in and signing up for the TripTracker app. In this activity, you will implement this functionality using the Backendless API.

Materials

- Computer with Android™ Studio
- Android™ tablet and USB cable, or a device emulator
- Free Backendless account per student

RESOURCES

Lesson 3.1 Reference Card for Backendless
Resources available online

ASCII Table
Resources available online

Procedure

Part I: The Backendless User Table

1. Create a new Backendless application.
   a. Log in to Backendless.
   a. Click the **CREATE APP** button near the top right of the screen.
   b. Enter **TripTracker** for the application name.
   a. Click **Create**.
2. In the column on the left side of the page, click the **Data** icon. By default, the Data page shows your Users table.

3. Above the table data, you should see a menu bar with additional links beginning with “DATA BROWSER”. Click the **SCHEMA** link.

   The Users table is shown in schema format with each column of the table shown in its own row.

   a. What type of data are name, email, and password?

   b. In the CONSTRAINTS column, you will see icons that indicate different settings for each field. Hover your mouse pointer over the icons to answer the question, what are required fields?

   c. Again using the CONSTRAINTS icons, what field(s) must have a unique value in this table?

   The IDENTITY column in the Users table indicates which field Backendless uses to uniquely identify a user. In Backendless, you can choose which field will be the identifier; for TripTracker, you will use the default field.

   d. What is the default Identity field for the Users table?

Part II: Sign Me Up

Before attempting to register a user, your TripTracker app should check that the user has entered both an email and a password. The act of checking whether data is entered correctly is called **data validation**.

data validation
The process of ensuring that the data entered into a computer system is correct and meaningful. This usually involves a set of validation rules embedded in the computer application.

4. As you did for CollegeApp in *Activity 2.2.2 Remote Database*, prepare your TripTracker project to use Backendless. (If necessary, refer to the activity for details.)

   a. Add the Backendless library dependency to your app’s Project Structure (**File** > **Project Structure**).

   b. Add the `INTERNET` and `ACCESS_NETWORK_STATE` user-permission tags to your *`AndroidManifest.xml`* file.

   c. Instead of hard-coding the values in code, add your Backendless Application ID and Backendless Android API Key to the *`strings.xml`* file.

   d. Add `import com.backendless.Backendless;` to *`LoginActivity.java`*.

   e. In `LoginActivity onCreate`, use `Backendless.initApp(…)` to initialize Backendless. Your call to `initApp` should reference both strings you created in step c. For example, use something similar to the following:

```
Backendless.initApp( this,
    getString(R.string.be_app_id),
    getString(R.string.be_android_api_key));
```

5. Create two `EditText` variable references, one for `enter_email` and one for `enter_password`. Initialize them using `findViewById`.
6. In `LoginActivity onCreate`, create a listener for the “Sign Me Up” button. (Use your own reference variables in place of those shown, if necessary.)

```java
MySignMeOnClickListener signMeUpListener = new MySignMeUpOnClickListener();
mSignMeUpButton.setOnClickListener(signMeUpListener);
```

7. Review the `trim()` and `isEmpty()` methods of the `String` class in the **official Java documentation**. Briefly describe their purpose.
8. As shown below, implement the listener to retrieve the email, password, and name values from your `EditText` fields. (Use your own reference variables in place of those shown, if necessary.)

```java
private class MySignUpOnClickListener implements View.OnClickListener {
    @Override
    public void onClick(View v) {
        String userEmail = mEmailEditText.getText().toString();
        String password = mPasswordEditText.getText().toString();
        String name = mNameEditText.getText().toString();

        userEmail = userEmail.trim();
        password = password.trim();
        name = name.trim();

        if (!userEmail.isEmpty() &&!password.isEmpty() && !name.isEmpty()) {

              /* register the user in Backendless */

        }
        else {
             /* warn the user of the problem */

        }
    }
}
```

If any of the required fields are empty, you will need to warn the user. Android provides an `AlertDialog` class that can do just that. This is illustrated in the next two steps.

9 First create the following definitions in *strings.xml*.

```
<string name="authentication_error_title">User
Authentication Error!</string>
<string name="empty_field_signup_error">Be sure you have
entered a valid email address, password, and name.</string>
```

10 Back in the `MySignMeUpOnClickListener`, add the following code to the else clause that will run in case of any empty values.

```
AlertDialog.Builder builder = new AlertDialog.
Builder(LoginActivity.this);
builder.setMessage(R.string.empty_field_signup_error);
builder.setTitle(R.string.authentication_error_title);
builder.setPositiveButton(android.R.string.ok, null);
AlertDialog dialog = builder.create();
dialog.show();
```

11 Test your app and verify that the alert appears if any of the fields are empty.

12 Validate that all is well with the email, password, and name and register the user.

a. Create a BackendlessUser instance variable called `user`.

b. Call the register method in `Backendless.UserService`, overriding the `handleResponse` and the `handleFault` methods as shown.

```
/* register in Backendless */
user.setEmail(userEmail);
user.setPassword(password);
user.setProperty("name", name);

Backendless.UserService.register(user,
new AsyncCallback<BackendlessUser>() {
    @Override
    public void handleResponse( BackendlessUser
                                backendlessUser ) {
            Log.i(TAG, "Registration successful for " +
                backendlessUser.getEmail());
    }
    @Override
    public void handleFault( BackendlessFault fault ) {
        Log.i(TAG, "Registration failed: " + fault.
            getMessage());
    }
} );
```

13 For the Backendless `user` object, notice the methods that are used to set the `userEmail` and `password`, and compare them to the method that sets `name`.

What are the methods and why do think they differ?

14 Based on your previous `AlertDialog`, create another one for when registration fails. In place of `builder.setMessage(R.string.empty_field_signup_error)`, you can display the error message from Backendless with `builder.setMessage(fault.getMessage())`.

15 Save and test your app, registering with your Backendless account.

After signing up, nothing on the screen will change (you will fix this next), but the Android Studio logcat should show that the user was registered successfully.

16 Perform more tests to make sure you cover all data validation scenarios you have programmed so far. Here are some suggested cases to test:

- Leave email empty.
- Leave password empty.
- Enter blank spaces for email.
- Enter blank spaces for password.
- Enter a value for both email and password.
- Register another user.
- Attempt to sign up two users with the same email. This should fail.

Part III: Log In

Now that you have the registration process complete, provide the log-in functionality.

17 Similar to `MySignMeUpOnClickListener`, create a new inner class called `MyLoginOnClickListener` that implements `View.OnClickListener` and overrides the `onClick` method.

The log-in process is nearly identical to the registration process. The differences are:

- A name is not required to log in.
- In place of the `.register(…)` method, the `Backendless.UserService` object calls the `.login(…)` method as shown.

```
Backendless.UserService.login(userEmail, password,
    new AsyncCallback<BackendlessUser>()
```

18 Use your code from `MySignMeUpOnClickListener` to implement `MyLoginOnClickListener`.

a. Use the Backendless `login` method described above in place of `register`.

b. Modify the message in the alert dialog to reflect that an email address and password are required.

c. Create the new strings in *`string.xml`* for the log-in error messages.

19 Test that a user can log in and also test for a failed attempt. As with the sign-up process, your screen will not change.

Part IV: Secure Passwords

In TripTracker, you use an email address and a password to secure access to your app. In real-world systems, weak or easy-to-guess passwords represent one of the biggest security risks facing our cyber world today. Weak passwords allow criminals to access sensitive information, such as banking information, health and family records, and secret government projects. Computer algorithms can guess passwords using dictionaries from around the world and information from social media sites. Your pet's name is one of the easiest-to-guess passwords.

But passwords don't have to be such a problem. Strong, difficult-to-guess passwords or *passphrases* are easy to create and can provide great security for your online accounts and activity. You should change your passwords frequently and use different passphrases for different sites and accounts.

Features of strong, secure passwords:

- Use both upper- and lowercase letters
- Contain numbers
- Contain symbols such as _ & # @ !
- Omit commonly spelled words
- Include multiple words or phrases

For example, a good passphrase would be `JL_a5unnyDay!` (just like a sunny day). Notice JL is a *mnemonic* that can help you memorize your passwords.

20 Create a new, very safe, but easy-to-remember password and write it down here (you will not want to actually use this one, since other people may have access to this document).

21 (Optional) Create a new email account to use when you fill out web forms and use it when you shop, make travel reservations, register for a club or sports team, for example.

Part V: Validate Registration Data

In this part of the activity, you will further validate data the user has entered when signing up for an account. First, you will use the `AlertDialogs` to warn the user about errors in their data. Creating the dialogs will require the same code just with a different message. This is a great place for a new method. In creating the new method for the `AlertDialog`, you will temporarily create a bug in the next few steps and then learn how to fix it.

22 Revisit the presentation *3.1.2 Java String Review* to review some of the `String` methods you will use to validate data.

23 Create a new method to warn the user (e.g., `warnUser()`) that they have entered something incorrectly. The method should accept a `String` as a parameter, show an `AlertDialog`, and display the value of the `String` parameter as the dialog's message.

24 Use your new `warnUser` method in place of the two existing `AlertDialogs` (one dialog for handling a fault and the other for empty data fields).

You introduced a bug when you called your `warnUser` method with a variation of `R.string.value`. In other uses, `R.string.values` are automatically cast to a `String` by the Java Virtual Machine, but because the JVM checks the data type of the parameters in a method call, you will need to explicitly cast them with the `getString` method.

25 Fix the bug using the `getString(R.string.value)` method.

Use the Java `String` methods you already know to do some more *basic* data validation during the sign-up process. (Full validation is an optional step you may choose to do at the end of this activity.) If all authentication scenarios are met, register the user; otherwise, warn the user and do not register them. Remember to define your messages in ***`strings.xml`***.

26 Validate the user data with these scenarios:

- Their email address must contain @ and . characters.
- Their password must be at least six characters long.
- Their password may not be the same as their email address.

Part VI: A Progress Dialog

27 Last but not least, it may take a while for the Backendless service to complete a login or registration request, so display a "please wait" or "busy" message. In both of the listener callback methods, add a `ProgressDialog` similar to the one below just before the Backendless method calls.

```
final ProgressDialog pDialog = ProgressDialog.
show(LoginActivity.this,
  "Please Wait!",
 "Creating a new account...",
 true);
```

Remember to use *`strings.xml`* and `getString()` in place of the literal the values.

28 Test your app to see the new dialog. When an error occurs in either the login or registration process, modify your code to dismiss the dialog with `pDialog.dismiss()`.

NOTE

When an error does *not* occur, the dialog will stay “forever” because you have not dismissed it and because you have not directed the user to a new Android activity. You will fix that in the next activity of this lesson.

Part VII: (Optional) Full Password Validation

29 Extend your password validation algorithm to include the first three criteria discussed in Part IV.

- Temporarily comment out the progress dialogs so you can test.
- Use String method `charAt` and get ASCII values of the characters in the password. For example, the ASCII value 65 begins the section of uppercase letters. See **3.1.2 ASCIITable**.
- Determine whether numbers, symbols, and both lowercase and uppercase characters are used.

Words are an extra challenge that require an open-source dictionary loaded into a database (and therefore, not recommended).

CONCLUSION

1. As defined in Backendless, name is an optional field in the User table. How would you make it required for your TripTracker app?
2. Consider the basic data validation you performed in this app. How could you improve it?

ASCII Characters and Binary Representation

RESOURCES

ASCII Table
Resources available online

DEC	BIN	Symbol	Description
32	00100000		Space
33	00100001	!	Exclamation mark
34	00100010	"	Double quotes
35	00100011	#	Number
36	00100100	$	Dollar sign
37	00100101	%	Percent
38	00100110	&	Ampersand
39	00100111	’	Single quote
40	00101000	(	Open parenthesis
41	00101001	)	Close parenthesis
42	00101010	*	Asterisk
43	00101011	+	Plus
44	00101100	,	Comma
45	00101101	-	Hyphen
46	00101110	.	Period
47	00101111	/	Slash or divide
48	00110000	0	Zero
49	00110001	1	One
50	00110010	2	Two
51	00110011	3	Three
52	00110100	4	Four
53	00110101	5	Five
54	00110110	6	Six
55	00110111	7	Seven
56	00111000	8	Eight
57	00111001	9	Nine

DEC	BIN	Symbol	Description
58	00111010	:	Colon
59	00111011	;	Semicolon
60	00111100	<	Less than
61	00111101	=	Equals
62	00111110	>	Greater than
63	00111111	?	Question mark
64	01000000	@	At symbol
65	01000001	A	Uppercase A
66	01000010	B	Uppercase B
67	01000011	C	Uppercase C
68	01000100	D	Uppercase D
69	01000101	E	Uppercase E
70	01000110	F	Uppercase F
71	01000111	G	Uppercase G
72	01001000	H	Uppercase H
73	01001001	I	Uppercase I
74	01001010	J	Uppercase J
75	01001011	K	Uppercase K
76	01001100	L	Uppercase L
77	01001101	M	Uppercase M
78	01001110	N	Uppercase N
79	01001111	O	Uppercase O
80	01010000	P	Uppercase P
81	01010001	Q	Uppercase Q
82	01010010	R	Uppercase R
83	01010011	S	Uppercase S
84	01010100	T	Uppercase T
85	01010101	U	Uppercase U
86	01010110	V	Uppercase V
87	01010111	W	Uppercase W
88	01011000	X	Uppercase X
89	01011001	Y	Uppercase Y
90	01011010	Z	Uppercase Z
91	01011011	[	Opening bracket
92	01011100	\	Backslash
93	01011101	]	Closing bracket

DEC	BIN	Symbol	Description
94	01011110	^	Caret
95	01011111	_	Underscore
96	01100000	`	Grave accent
97	01100001	a	Lowercase a
98	01100010	b	Lowercase b
99	01100011	c	Lowercase c
100	01100100	d	Lowercase d
101	01100101	e	Lowercase e
102	01100110	f	Lowercase f
103	01100111	g	Lowercase g
104	01101000	h	Lowercase h
105	01101001	i	Lowercase i
106	01101010	j	Lowercase j
107	01101011	k	Lowercase k
108	01101100	l	Lowercase l
109	01101101	m	Lowercase m
110	01101110	n	Lowercase n
111	01101111	o	Lowercase o
112	01110000	p	Lowercase p
113	01110001	q	Lowercase q
114	01110010	r	Lowercase r
115	01110011	s	Lowercase s
116	01110100	t	Lowercase t
117	01110101	u	Lowercase u
118	01110110	v	Lowercase v
119	01110111	w	Lowercase w
120	01111000	x	Lowercase x
121	01111001	y	Lowercase y
122	01111010	z	Lowercase z
123	01111011	{	Opening brace
124	01111100	|	Vertical bar
125	01111101	}	Closing brace
126	01111110	~	Equivalency sign - tilde
127	01111111		Delete