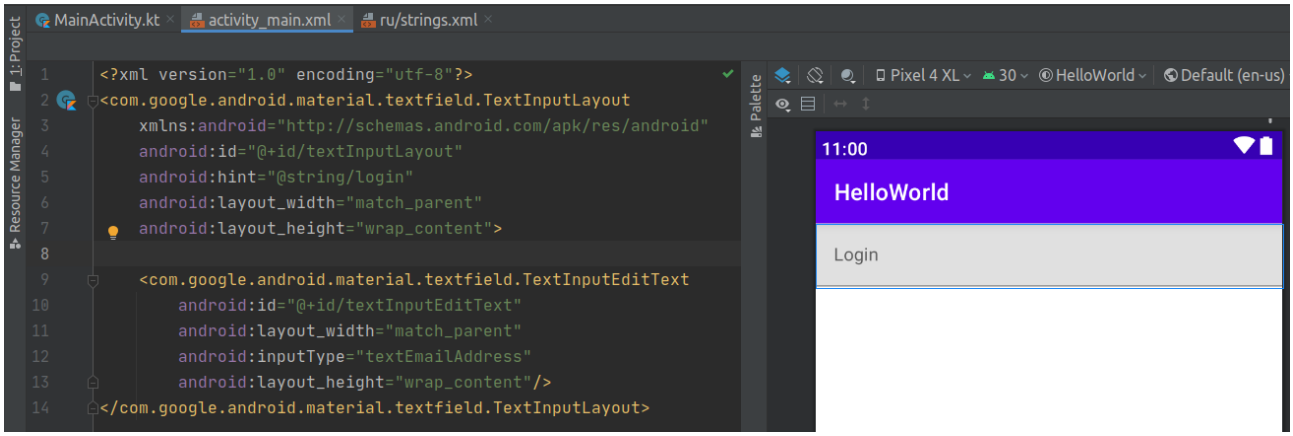
Форма ввода

Пишем стандартные инпуты

Содержание

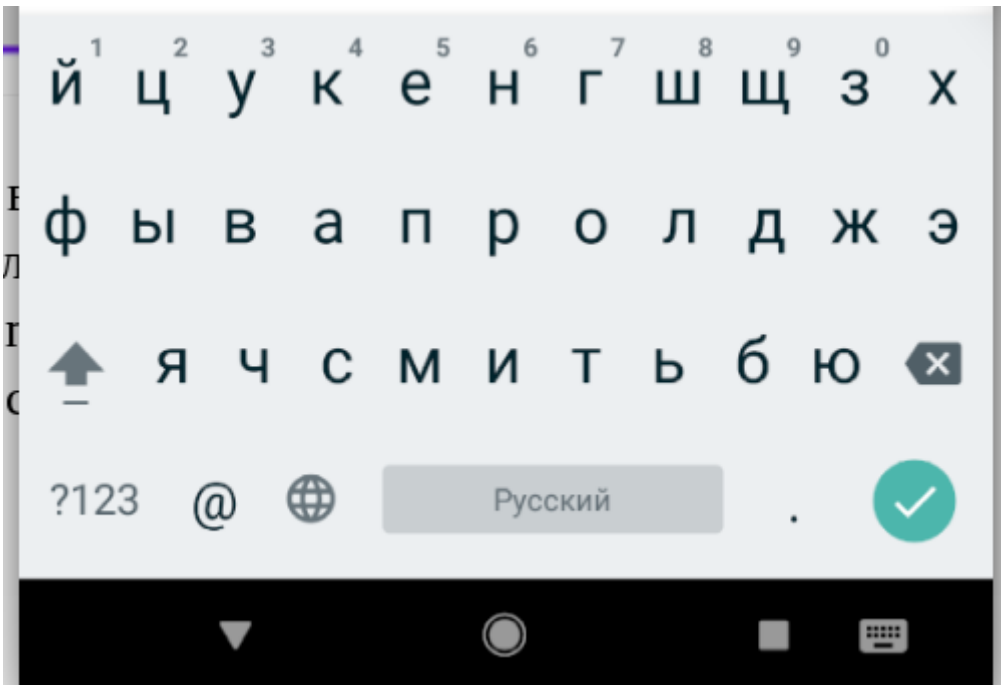
1. Поля ввода

До сих пор мы лишь отображали текст и изображения. Настало время рассмотреть как вводить с клавиатуры. Давайте для начала рассмотрим поле ввода логина типа почта

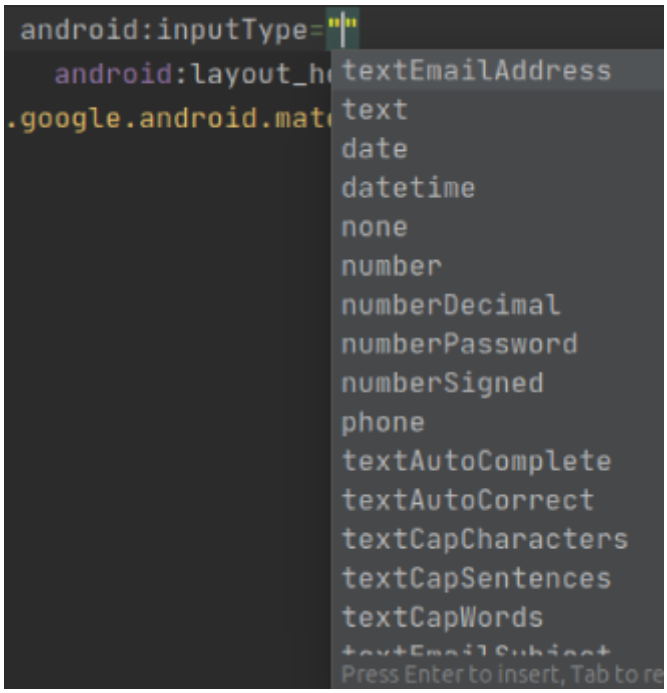


Как видите нам нужно 2 вью – TextInputLayout и внутри TextInputEditText. Почему? Для красоты и уже готовой анимации. Можете запустить и протестировать. Из-за обертки над полем ввода hint будет подниматься наверх при вводе. Можете посмотреть как

Если вы сделаете без обертки TextInputLayout то у вас hint будет в EditText и он исчезнет когда будете вводить текст. Что касается самого поля ввода то мы можем указывать какого типа текст ожидается в нем – в нашем случае textEmailAddress. Это не означает что юзер автоматически сможет ввести лишь валидные адреса эл.почты. Это лишь значит что будет видна нужная клавиатура. Смотрите



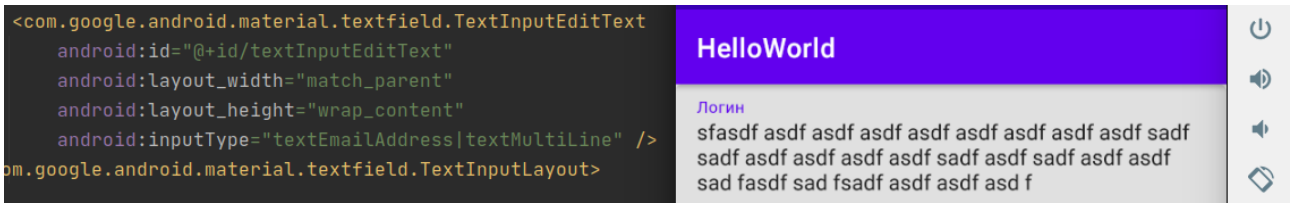
Видите? Собачка. Есть и другие варианты. Можете просмотреть все когда вводите



Как видите с помощью этого атрибута можно поднимать нужную клавиатуру. Например у вас может быть лишь ввод чисел или числового пароля. Далее мы добавим второе поле ввода с типом пароль. Вы увидете как введенный текст превращается автоматически в ****.

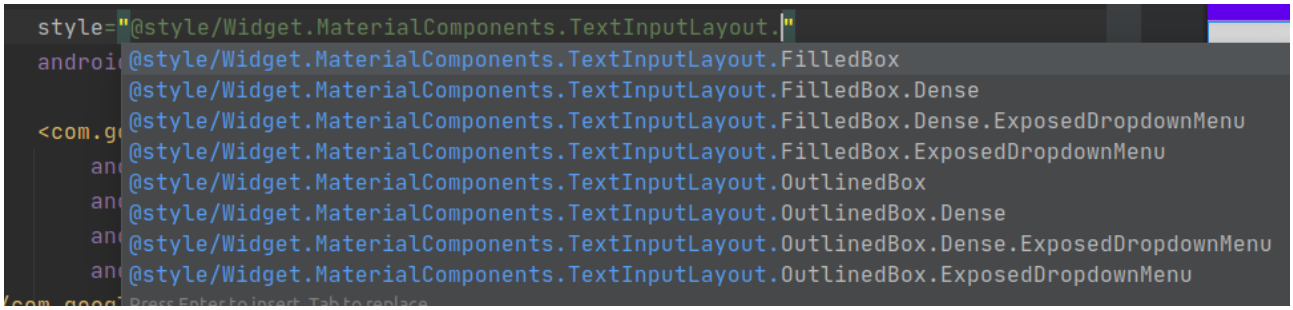
Кстати, поле ввода по умолчанию в 1 линию. Если вам нужно в несколько линий, то нужно выбрать android:inputType="textMultiLine". Но вы спросите, а что если мне надо ввести почту но чтобы в несколько линий, тогда как? Можно выбирать несколько значений посредством логического или бинарного. Например так. Так что вы можете комбинировать их.

Ладно, рассмотрим тогда такой вопрос – если inputType лишь поднимает нужный вид клавиатуры, то как нам проверять что юзер ввел действительно валидную почту? Через код конечно же, если помните мы писали валидаторы текста, они нам помогут сегодня.



Но перед этим давайте скажем пару слов о стиле. Если помните мы указывали стиль для текста – textAppearance. Для полей ввода тоже есть стили и это гуговые материал стили

По умолчанию у вас FilledBox. Но можно поменять вот так. Давайте выберем OutlineBox.Dense и посмотрим как изменится юай.



А еще я добавил layout_margin чтобы не было впритык к стенкам

Как видите стало гораздо лучше. Все стили можете посмотреть на material.io

Ладно, давайте вернемся к валидации поля ввода. Для начала нам нужно найти вьюшки.

```
val textInputLayout = findViewById<TextInputLayout>(R.id.textInputLayout)
val textInputEditText = textInputLayout.editText as TextInputEditText
```

Кстати, можно не ставить айди для поля ввода, мы можем его получить из textInputLayout.

Хорошо, пока что у нас нет кнопки, по нажатию которой бы мы смогли проверить валидность введенного текста. Как же нам это сделать? Можно в реальном времени! Вы наверно видели когда в поле ввода сразу обрабатывается текст и выдаются ошибки. Кстати, textInputLayout

так же нужен для красивого отображения ошибок ввода. Сейчас рассмотрим. Чтобы обрабатывать в реальном времени введенный текст нам нужно слушать изменения. Для этого есть такая штука как TextWatcher. Давайте посмотрим на него.

```
textInputEditText.addTextChangedListener(object : TextWatcher {
    override fun beforeTextChanged(s: CharSequence?, start: Int, count: Int, after: Int) {
        TODO( reason: "Not yet implemented")
    }

    override fun onTextChanged(s: CharSequence?, start: Int, before: Int, count: Int) {
        TODO( reason: "Not yet implemented")
    }

    override fun afterTextChanged(s: Editable?) {
        TODO( reason: "Not yet implemented")
    }
})
```

Как видите у него 3 метода – до того как текст изменился, во время и после. Вы можете выбрать любой метод для обработки. Зачастую выбирается 1 из 3, а 2 других остаются пустыми. Для этого можно сделать класс который будет иметь дефолтные пустые реализации

```
abstract class SimpleTextWatcher : TextWatcher {
    override fun beforeTextChanged(s: CharSequence?, start: Int, count: Int, after: Int) {
    }

    override fun onTextChanged(s: CharSequence?, start: Int, before: Int, count: Int) {
    }

    override fun afterTextChanged(s: Editable?) {
    }
}
```

И в нашем коде будет лишь 1 метод который нам нужен. Посмотрите как красиво

```
textInputEditText.addTextChangedListener(object : SimpleTextWatcher() {
    override fun afterTextChanged(s: Editable?) {

    }
})
```

И здесь можно написать валидацию. Кстати, вам не нужно придумывать ничего – в андроид уже есть патерн для проверки почты. Смотрите

```
textInputEditText.addTextChangedListener(object : SimpleTextWatcher() {
    override fun afterTextChanged(s: Editable?) {
        if (android.util.Patterns.EMAIL_ADDRESS.matcher(s.toString()).matches()) {
            //this is really email
        } else {
            // still not email
        }
    }
})
```

EMAIL_ADDRESS можно провалиться и посмотреть – регулярное выражение для проверки эл.почты. Вам не нужно писать свои правила. Проверять что строка содержит ‘@’, точку и минимум 3 символа. Все это уже есть. Ладно. Мы написали проверку, что мы должны делать? Давайте покажем ошибку если введена не почта, а когда она введена давайте уберем ошибку и покажем уведомление внутри экрана через Toast.

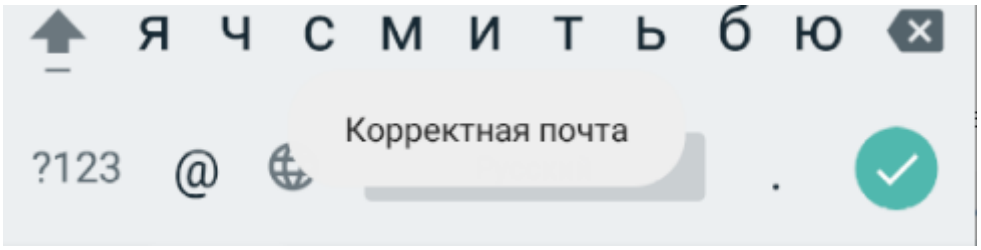
```
override fun afterTextChanged(s: Editable?) {
    val valid = android.util.Patterns.EMAIL_ADDRESS.matcher(s.toString()).matches()
    textInputLayout.isErrorEnabled = !valid
    val error = if (valid) "" else getString(R.string.invalid_email_message)
    textInputLayout.error = error
    if (valid) Toast.makeText(
        context: this@MainActivity,
        R.string.valid_email_message,
        Toast.LENGTH_LONG
    ).show()
}
```

Как видите сначала нужно включить ошибку и потом уже дать ей текст, без isErrorEnabled не работает, можете проверить. Кстати, я оставил Toast.makeText на той же линии что и if просто чтобы влезло в экран.

Как видите все автоматически подкрасилось в красный и видна иконка ошибки справа.

Когда введенный текст валиден, то ошибка уходит

Ну и Toast виден поверх всего на экране



Проблема в том, что уже когда вы ввели mail.co ваша почта валидна и скрывать клавиатуру не нужно. Если вы заметили, то в правом нижнем углу кнопка птичка – если нажать на нее, то клавиатура сама уйдет.

И давайте сделаем кое-что интересное, например упростим жизнь юзеру. Давайте подставлять gmail.com когда юзер после @ ввел g. Т.е. если почта кончается на @g то автоматом подставить @gmail.com. Как это сделать? Давайте для простоты уберем отображение ошибки чтобы меньше кода видеть на скрине

```
override fun afterTextChanged(s: Editable?) {
    val input = s.toString()
    if (input.endsWith( suffix: "@g")) {
        val fullMail = "${input}mail.com"
        textInputEditText.setText(fullMail)
    }
}
```

Как видите все работает, но каретка вернулась в начало. Как исправить? Нужно ее переместить руками в конец. Для этого можно написать экстеншн функцию


```
fun TextInputEditText.setTextCorrectly(text:CharSequence) {
    setText(text)
    setSelection(text.length)
}
```

Теперь каретка идет в конец когда мы программно устанавливаем текст. Но скажу я вам что у нас есть 1 проблема. Чтобы это понять нам нужно залогировать вызовы. Дело в том, что после того как мы программно установили текст – textWatcher срабатывает еще раз. Давайте поставим логи для этого. В андроид есть логер Log.

```
override fun afterTextChanged(s: Editable?) {
    Log.d( tag: "TextWatcherTag",  msg: "afterTextChanged $s")
    val input = s.toString()
    if (input.endsWith( suffix: "@g")) {
        Log.d( tag: "TextWatcherTag",  msg: "programmatically set text")
        val fullMail = "${input}mail.com"
        textInputEditText.setTextCorrectly(fullMail)
    }
}
```

Логи смотрим в LogCat на дне AC. Log.d означает Debug и можно фильтровать в логкате. Так же мы поставили тег и по нему тоже можно фильтровать (также есть Log.e - errors)

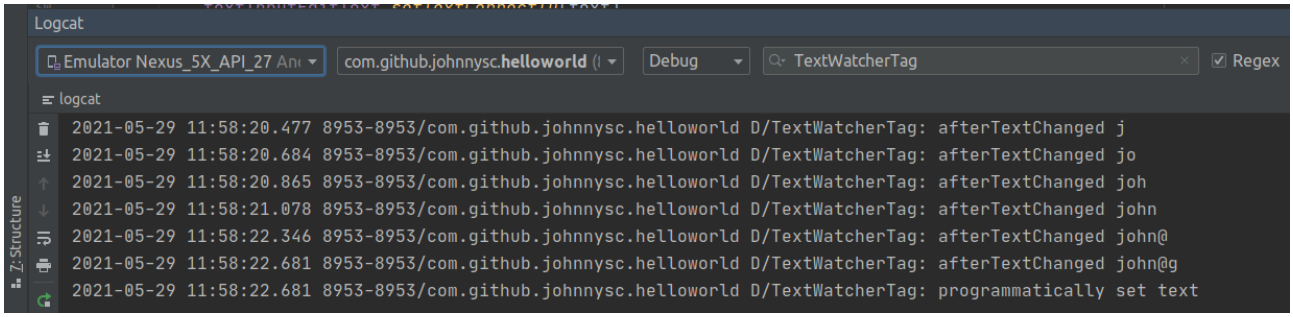
Как видите мы лишний раз вызываем метод проверки. Да, сейчас он простой и ничего критичного не происходит. Но в другом случае это может быть запрос в сеть и мы не хотим лишний раз этого делать. Как быть? Здесь популярное решение – перед програмной установкой текста убирать textWatcher и возвращать после. Для этого надо вынести наш watcher в переменную. Смотрите

```
private const val TAG = "TextWatcherTag"

class MainActivity : AppCompatActivity() {

    private lateinit var textInputEditText: TextInputEditText

    private val textWatcher: TextWatcher = object : SimpleTextWatcher() {
        override fun afterTextChanged(s: Editable?) {
            Log.d(TAG,  msg: "afterTextChanged $s")
            val input = s.toString()
            if (input.endsWith( suffix: "@g")) {
                Log.d(TAG,  msg: "programmatically set text")
                setText("${input}mail.com")
            }
        }
    }

    private fun setText(text: String) {
        textInputEditText.removeTextChangedListener(textWatcher)
        textInputEditText.setTextCorrectly(text)
        textInputEditText.addTextChangedListener(textWatcher)
    }
```

Ничего страшного если сразу ничего не понятно, со второй попытки легче

```
val textInputLayout = findViewById<TextInputLayout>(R.id.textInputLayout)
textInputEditText = textInputLayout.editText as TextInputEditText

textInputEditText.addTextChangedListener(textWatcher)
```

Мы вынесли textWatcher в переменную и поле ввода тоже. Теперь мы убираем textWatcher перед установкой значения и ставим обратно. Давайте запустим код и проверим логи



Кажется что все прекрасно. Но мы создали баг! Попробуйте удалять с конца. Когда вы дойдете до момента, когда у вас текст кончается на @g он автоматически будет дополняться. И вы никогда не сотрете ввод. Хотя можно переместить каретку или выбрать все и удалить.

Вообще так не надо делать. Никогда не делайте за юзера что-либо. Вы можете предложить! Пусть после того как юзер ввел @ у вас выплывают популярные домены типа @gmail.com @mail.ru и так далее. Но это мы рассмотрим в другой лекции. А пока давайте посмотрим на то, что мы сделали – юзер начинает вводить почту и сразу получает ошибку. Это тоже плохо. Лучше чтобы юзер нажимал на некую кнопку когда закончит ввод и тогда уже узнает что ввел некорректные данные. Но перед тем как мы перейдем к теме кнопок, давайте я вам покажу изящное решение для textWatcher в случае если вы не ставите програмно текст.

```
        textInputEditText.listenChanges { text -> Log.d(TAG, text) }
    }
}

fun TextInputEditText.listenChanges(block: (text: String) -> Unit) {
    addTextChangedListener(object : SimpleTextWatcher() {
        override fun afterTextChanged(s: Editable?) {
            block.invoke(s.toString())
        }
    })
}
```

Я написал экстеншн функцию с лямбдой и теперь место вызова в 1 линию. Красота же ну.

2. Кнопки

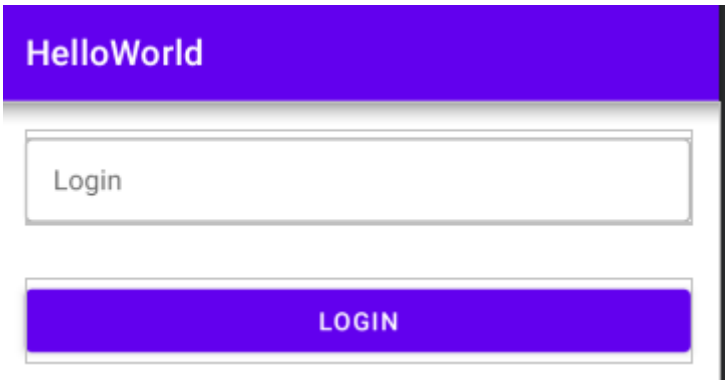
Я не хотел рассматривать такую простую штуки как кнопка в Андроид в отрыве от других вьюх. Потому давайте рассмотрим такой кейс – у нас на экране будет 1 поле логина и кнопка на дне экрана. И здесь самое время вспомнить про контейнер вьюх который был в самом начале создания проекта. Но я все равно не хочу использовать сейчас ConstraintsLayout ведь у нас всего 2 вью – поле ввода и кнопка на дне. Под эту задачу идеально подойдет LinearLayout. Для этого давайте поменяем наш xml файл. Вы всегда можете использовать pallete слева от отображения во вкладке design но истинные разработчики пишут как код так и разметку руками.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    xmlns:android="http://schemas.android.com/apk/res/android">

    <com.google.android.material.textfield.TextInputLayout...>

</LinearLayout>
```

Как вы могли догадаться по названию – это контейнер для вью который располагает их в линию. Если задали orientation vertical то вертикально, можно и горизонтально. Сейчас у нас 1 вью внутри – поле ввода, надо добавить еще кнопку. Давайте добавим.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    xmlns:android="http://schemas.android.com/apk/res/android">

    <com.google.android.material.textfield.TextInputLayout...>

    <Button
        android:id="@+id/loginButton"
        android:layout_margin="@dimen/padding_small"
        android:text="@string/login"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"/>
</LinearLayout>
```

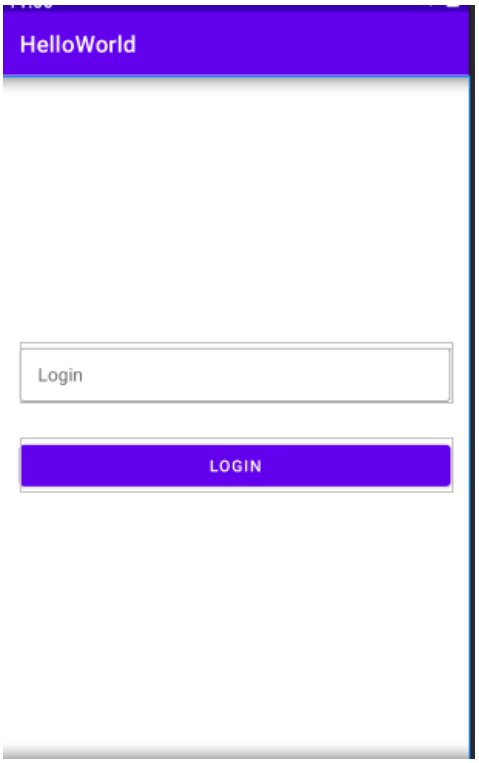
Но как видите наша кнопка идет сразу после поля ввода. Это не хорошо. Мы хотели чтобы она была на дне. Как же нам это сделать? Но давайте я вам покажу сначала нечто иное. Предположим мы хотим чтобы все вью были сгруппированы в центре. Это делается легко.



Просто добавим центрирование в контейнер

```
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:gravity="center"
    android:orientation="vertical">
```

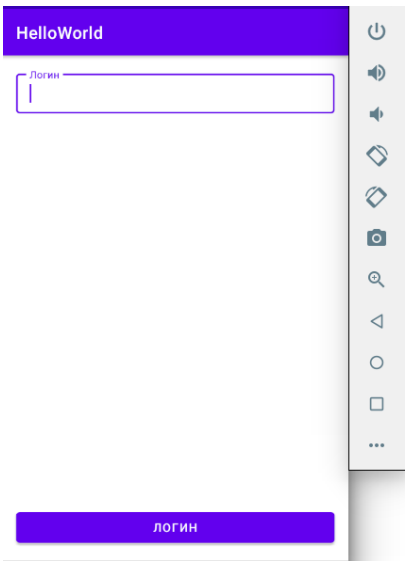
И вуаля – поле ввода вместе с кнопкой оказались по центру экрана.

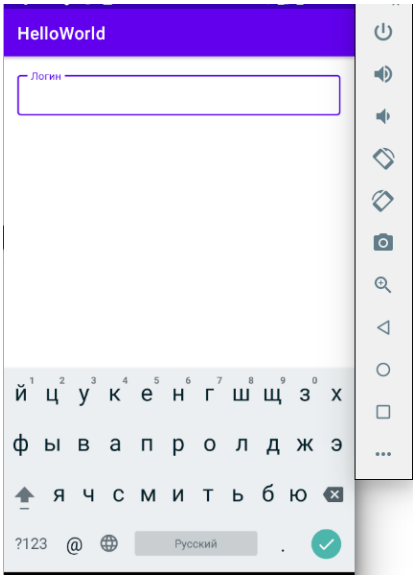


Хорошо. Давайте теперь вернемся к задаче. Нам нужно чтобы поле ввода было наверху, а кнопка внизу, а между ними? Пустота? Именно. Для этого в Андроид есть Space, но какой высоты? Нам нужно вычислить высоту экрана, высоту поля ввода и высоту кнопки и потом отнять от первого второе и третье чтобы поставить высоту пустоте? Нет конечно же. В LinearLayout есть самая крутая штука – вес. Помните пропорции из школы? Стороны треугольника соотносятся между собой как 1:2? Когда одна сторона условно в 2 раза больше другой. То же самое и здесь. Но в каких пропорциях поле ввода и кнопка с пустотой? Неважно. Можно вместо высоты пустоте задать вес в единицу и тогда оно подстроится под экран. Т.е. у вас будет – поле ввода, пустота и кнопка. У поля ввода и у кнопки есть своя высота, значит все остальное пространство займет пустота. Посмотрите на это.

```
    <com.google.android.material.textfield.TextInputLayout...>

    <Space
        android:layout_width="match_parent"
        android:layout_weight="1"
        android:layout_height="0dp"/>
    <Button...>
</LinearLayout>
```

И да,не забудьте убрать gravity у контейнера. Как видите вместо высоты у нас 0, значит компилятор будет смотреть на вес – он 1 т.е. будет занимать все пространство. Смотрим.

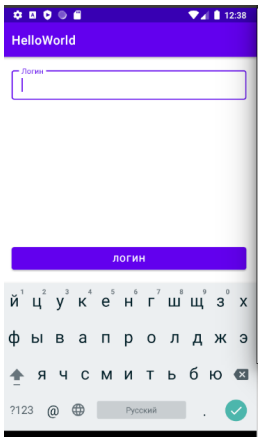




Все правильно! Но есть одно но. Начните вводить в поле почту. Упс. Клавиатура перекрывает кнопку. Теперь вам нужно ввести почту и нажать сначала на птичку или назад чтобы клавиатура ушла. И только потом нажать на кнопку логина. Не хорошо вышло. Но есть решение. Когда появляется клавиатура наш экран может меняться. Помните файл AndroidManifest? В нем можно прописать стратегию изменения. Откройте его через 2 Shift.

```
<activity
    android:name=".MainActivity"
    android:windowSoftInputMode="adjustResize">
    <intent-filter>
```

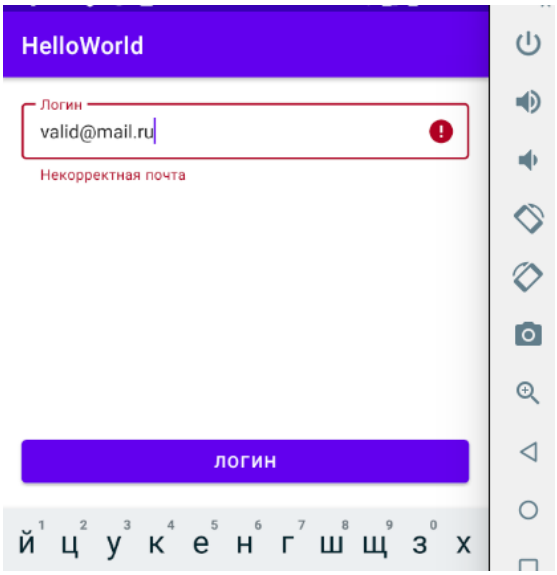
Просто добавьте 1 линию и перезапустите проект. Как видите теперь все корректно.



Хорошо. Мы вроде как решили же что будет проверять корректность введенной почты по нажатию на кнопку. Но сейчас наша кнопка при нажатии ничего не делает. Как исправить? Правильный ответ : в коде. Давайте посмотрим

```
val loginButton = findViewById<Button>(R.id.loginButton)
loginButton.setOnClickListener { it: View!
    if (EMAIL_ADDRESS.matcher(textInputEditText.text.toString()).matches()) {
        Snackbar.make(loginButton, text: "Go to postLogin", Snackbar.LENGTH_LONG).show()
    } else {
        textInputLayout.isErrorEnabled = true
        textInputLayout.error = getString(R.string.invalid_email_message)
    }
}
```

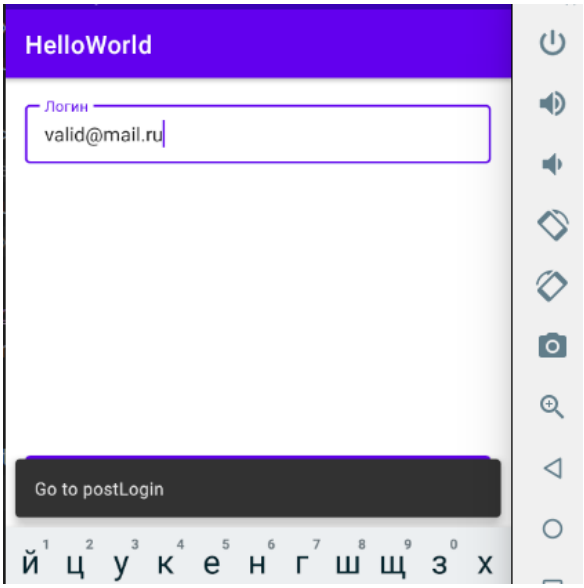
Мы устанавливаем обработчик нажатия простым методом – setOnClickListener. Его так же можно убирать если вам нужно (далее я расскажу когда и зачем). Опять проверяем на валидность текста и если там ошибка показываем ее. И мы опять создали баг. Как? А теперь поле ввода всегда будет иметь ошибку даже если мы после первой попытки будем вводить корректную почту. Смотрите. Сначала введем некорректную и нажмем на кнопку и потом введем корректную. Но ошибка осталась!



Кстати, не удивляйтесь что у меня русская клавиатура, а я пишу на английском. Ведь на эмуляторе можно вводить с клавиатуры ноутбука. Ок, что же делать? Использовать textWatcher. Когда юзер вводит что-либо – убрать ошибку и все.

```
textInputEditText.listenChanges { textInputLayout.isErrorEnabled = false }
```

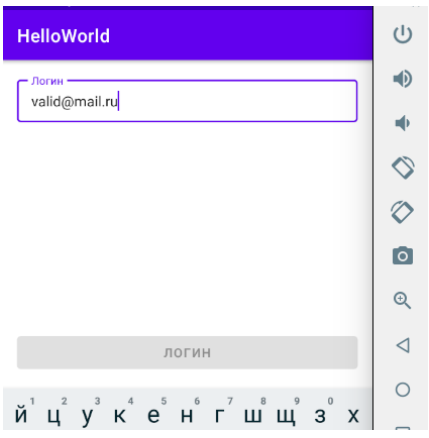
Теперь вводим некорректную почту, жмем кнопку, далее у нас ошибка в поле ввода, начинаем вводить иной текст и сразу же пропадает ошибка. Вуаля! Но мы теперь породили иную багу (да сколько можно!). Да, нашу кнопку можно нажимать по 100 раз если в поле ввода корркетные данные. Какой правильный путь? Не дать юзеру нажимать ее. Мы можем убрать clickListener просто передав null (смотрим исходники – Ctrl + клик) и потом вернуть обработчик если нужно. Но юзер тогда будет кликать по кнопке и ничего не будет происходить. Хочется действительно запретить клики. Как сделать? Можно поменять доступность кнопки к нажатию. Смотрите!



Здесь конечно же у вас будет переход на другой экран и не нужно менять доступность кнопки, но у вас может быть другой в принципе экран, а не логин. Так что рассмотрим кейс.

```
loginButton.setOnClickListener {  it: View!
    if (EMAIL_ADDRESS.matcher(textInputEditText.text.toString()).matches()) {
        loginButton.isEnabled = false
        Snackbar.make(loginButton,  text: "Go to postLogin", Snackbar.LENGTH_LONG).show()
```

И вуаля – кнопка стала серой!



Исходя из вашей логики можно вернуть флаг в true когда нужно. В основном это когда от сервера пришел ответ. Но мы не рассмотрели еще один вопрос – клавиатура осталась открытой! Нам нужно все равно нажимать на 2 кнопки – сначала галка на дне и потом логин. Как же мы можем сами убрать клавиатуру при нажатии на кнопку? Просто погуглите.

```
loginButton.setOnClickListener { it: View!
    if (EMAIL_ADDRESS.matcher(textInputEditText.text.toString()).matches()) {
        val imm = this.getSystemService(INPUT_METHOD_SERVICE) as InputMethodManager
        imm.hideSoftInputFromWindow(textInputEditText.windowToken, flags: 0)
        loginButton.isEnabled = false
        Snackbar.make(loginButton, text: "Go to postLogin", Snackbar.LENGTH_LONG).show()
```

Можете вынести в экстеншн функцию для удобства и все будет короче.

```
fun AppCompatActivity.hideKeyboard(view: View) {
    val imm = this.getSystemService(AppCompatActivity.INPUT_METHOD_SERVICE) as InputMethodManager
    imm.hideSoftInputFromWindow(view.windowToken, flags: 0)
}
```

И в 1 линию в активити вызов

```
if (EMAIL_ADDRESS.matcher(textInputEditText.text.toString()).
    hideKeyboard(textInputEditText)
```

Это очень часто используется, так что напишите 1 раз метод и будет проще.

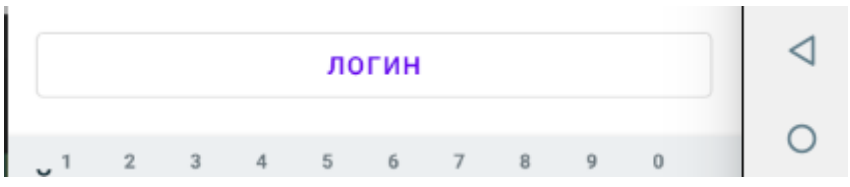
И на последок расскажу вам одну вещь – на самом деле кликать можно не только по куску текста и не только на кнопки – а на что угодно и даже на весь экран. Для этого вам нужно чтобы контейнер был кликабельным (все кроме FrameLayout, для него напишем в разметке 2 – clickable = true). Так что там где мы инициализаировали нашу кнопку можно было написать

```
val loginButton = findViewById<View>(R.id.loginButton)
loginButton.setOnClickListener { it: View!
```

Потому что все вью кликабельны. Можно сетить лиснеры всем и обработать клик на всех.

И на последок: у кнопок тоже могут быть стили! Посмотрите на готовые от гугла

```
<Button
    style="@style/Widget.MaterialComponents.Button.OutlinedButton"
    android:id="@+id/loginButton"
```



Для закрепления добавьте еще одно поле ввода для пароля и попробуйте написать валидацию с помощью патерна который мы прошли в разделе джавы.

p.s. про поля ввода есть еще несколько штук которые нужно было вам рассказать, но я оставляю вам на самостоятельное изучение. Погуглите про digits в разметке – можно не давать юзеру вводить что-либо кроме того что вы указали – но это плохой тон. Юзер будет вводить текст но ничего не будет видно. Лучше дайте юзеру ввести что он хочет и после обработайте. Например поле где нужно ввести лишь кириллицу – пусть вводит латиницу и выведите ему сразу что нужна кириллица. Так же есть атрибуты у поля ввода максимальных значений – погуглите сами ну и счетчик введенных символов. Это тоже сами. Плюс ко всему есть еще InputFilter – можно например не давать вводить эмоджи. Но я все же рекомендую давать юзеру свободу действий и постфактум обработать данные. И да, если поле ввода для номера телефона погуглите еще RedMadRobot: InputMask – оно автоматом превращает просто цифры в +7 (999) 111-22-33. Также попробуйте в домашнем задании перейти от первого поля ввода ко второму по нажатию на галку – вроде должно атоматически перейти, но вдруг нет – погуглите и решите задачу.