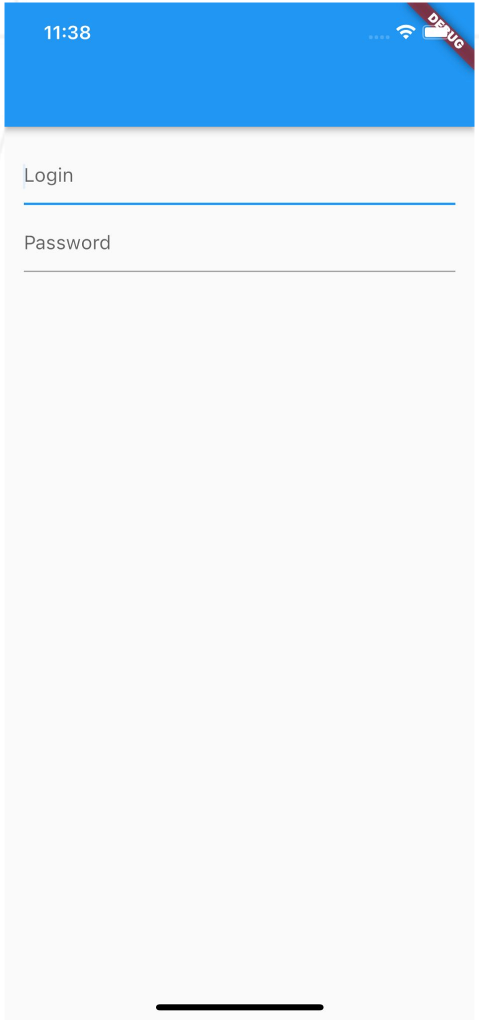
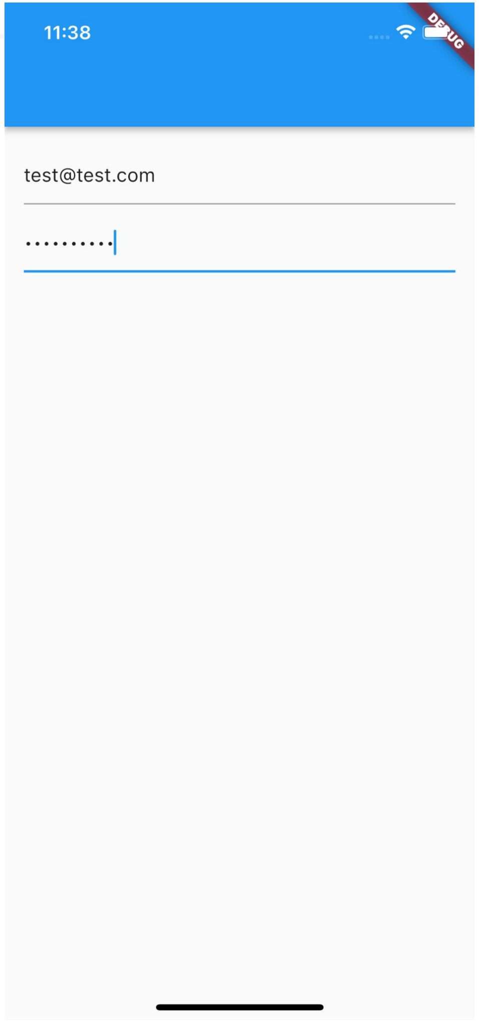
LeanCode

Programming Mobile Applications in Flutter

Forms

What is a Form?


11:38
DEBUG
Login
Password


11:38
DEBUG
test@test.com
••••••••••

TextField

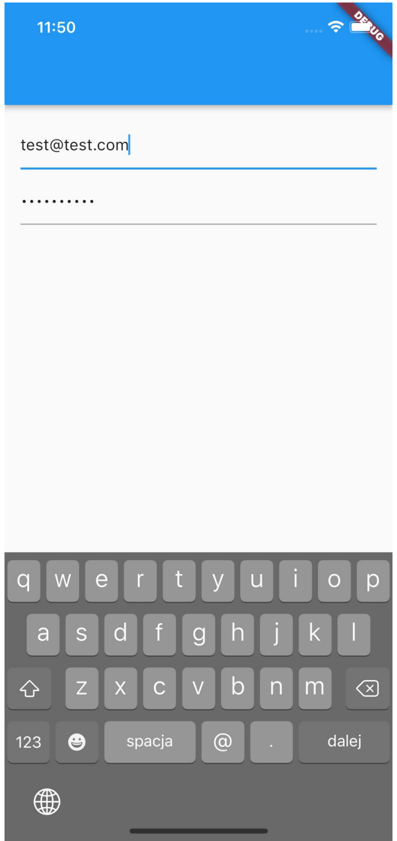
We build digital products.

LeanCode

TextField

- Lets the user enter text, either with hardware keyboard or with an onscreen keyboard
- Controlling the text
  - *onChanged(), onSubmitted()*
  - *TextEditingController*
    - Initial value
    - Needs to be disposed!!
- Decoration
  - Hints
  - Style

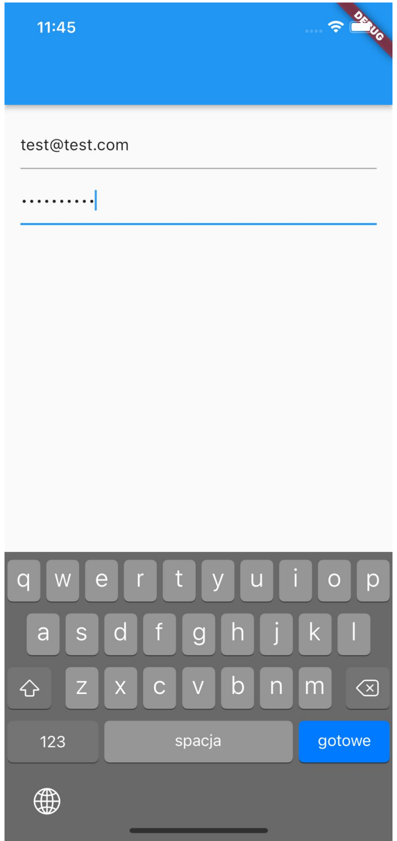
TextField - Keyboard


11:50

DEBUG

test@test.com

••••••••••


11:45

DEBUG

test@test.com

••••••••••

TextField - Keyboard

- *TextInputAction*
  - *next*
  - *done*
  - *search*
  - *…*
- *TextInputType*
  - *emailAddress*
  - *text*
  - *phone*
  - *…*
- Useful properties
  - *obscureText*
  - *enableSuggestions*
  - *autocorrect*

TextField - Focus

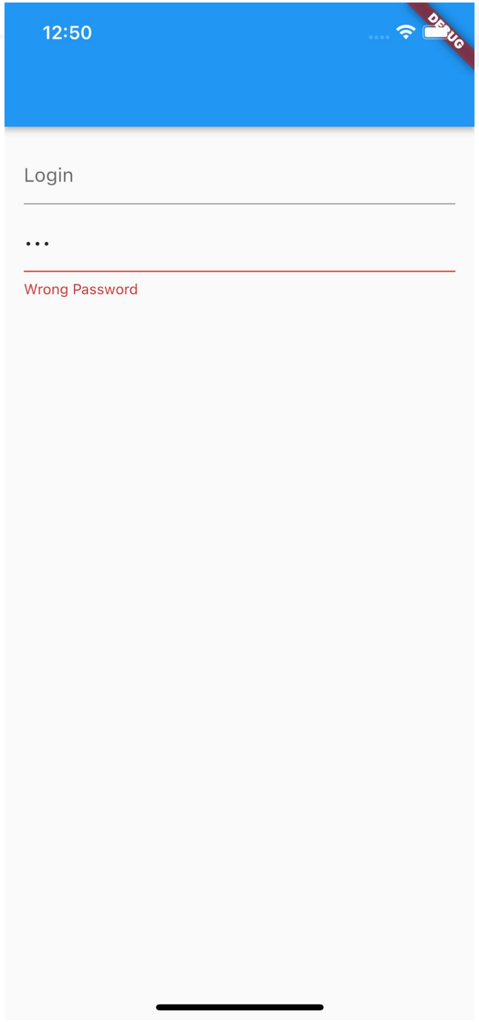
- Auto
  - *TextInputAction.next*
  - If inputs form a group
- *FocusNode*
  - *requestFocus*
  - Needs to be disposed!!

Demo

Validation

We build digital products.

LeanCode


12:50

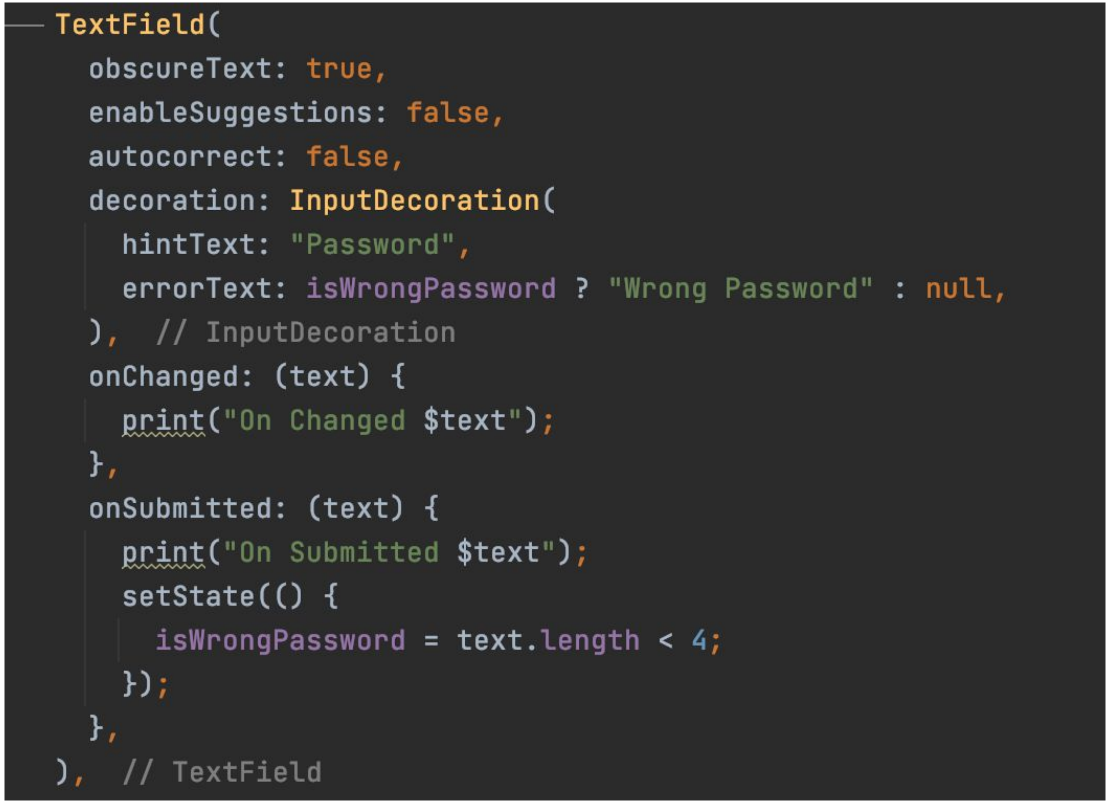
DEBUG

Login

•••

Wrong Password

Validation



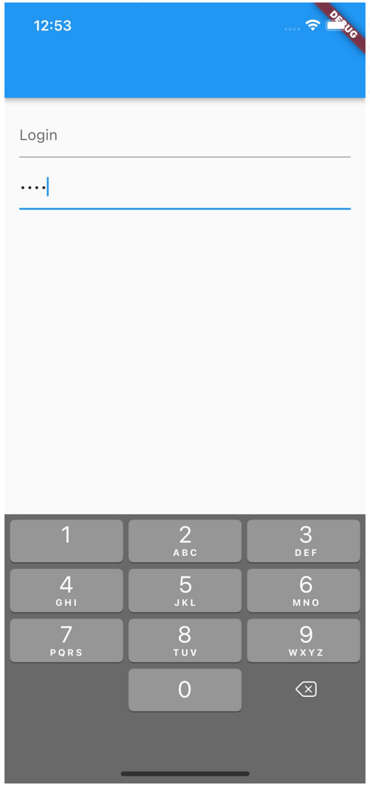
```
TextField(
  obscureText: true,
  enableSuggestions: false,
  autocorrect: false,
  decoration: InputDecoration(
    hintText: "Password",
    errorText: isWrongPassword ? "Wrong Password" : null,
  ), // InputDecoration
  onChanged: (text) {
    print("On Changed $text");
  },
  onSubmitted: (text) {
    print("On Submitted $text");
    setState(() {
      isWrongPassword = text.length < 4;
    });
  },
), // TextField
```

We build digital products.

LeanCode

Validation - KeyboardType

- Validation can be simplified by setting proper Keyboard Type
- It can break *TextInputAction*



Regular Expression

Regular Expression

- Regular expressions are **Patterns**, and can as such be used to match strings or parts of strings
- **Dart** regular expressions have the same syntax and semantics as **JavaScript** regular expressions
-
```
RegExp exp = RegExp(r"(\w+)");
String str = "Parse my string";
Iterable<RegExpMatch> matches = exp.allMatches(str);
```

- Note the use of a *raw string* (a string prefixed with **r**) in the example above. Use a raw string to treat each character in a string as a literal character

Validation - InputFormatters

- You can control input using *InputFormatter*
- Max length - *LengthLimitingTextInputFormatter(10)*
- Filtering
  - Only digits - *FilteringTextInputFormatter.digitsOnly*
  - Allow list
    - *FilteringTextInputFormatter.allow(RegExp('[a-zA-Z]'))*
  - Deny list
    - *FilteringTextInputFormatter.deny(RegExp('[a]'))*
    - FilteringTextInputFormatter.deny(RegExp(r'\s'), replacementString: "space")

Demo

Form & TextFormField

Form

- An optional container for grouping together multiple form field widgets
- Each individual form field should be wrapped in a *FormField* widget, with the *Form* widget as a common ancestor of all of those
- Call methods on *FormState* to **save**, **reset**, or **validate** each *FormField* that is a descendant of this *Form*
- Key!

FormField

- A single form field
- This widget maintains the current state of the form field, so that updates and validation errors are visually reflected in the UI.
- ***TextFormField*** *-* A *FormField that contains a TextField*

Demo

Hooks

Flutter Hooks

- https://pub.dev/packages/flutter_hooks
- Hooks are a new kind of object that manages a *Widget* life-cycles
- **They exist for one reason: increase the code-sharing between widgets by removing duplicates**

Flutter Hooks

- The following defines a hook that prints the time a *State* has been alive

  Source: https://pub.dev/packages/flutter_hooks#how-to-use

```
class _TimeAlive extends Hook<void> {
  const _TimeAlive();

  @override
  _TimeAliveState createState() => _TimeAliveState();
}

class _TimeAliveState extends HookState<void, _TimeAlive> {
  DateTime start;

  @override
  void initHook() {
    super.initHook();
    start = DateTime.now();
  }

  @override
  void build(BuildContext context) {}

  @override
  void dispose() {
    print(DateTime.now().difference(start));
    super.dispose();
  }
}
```

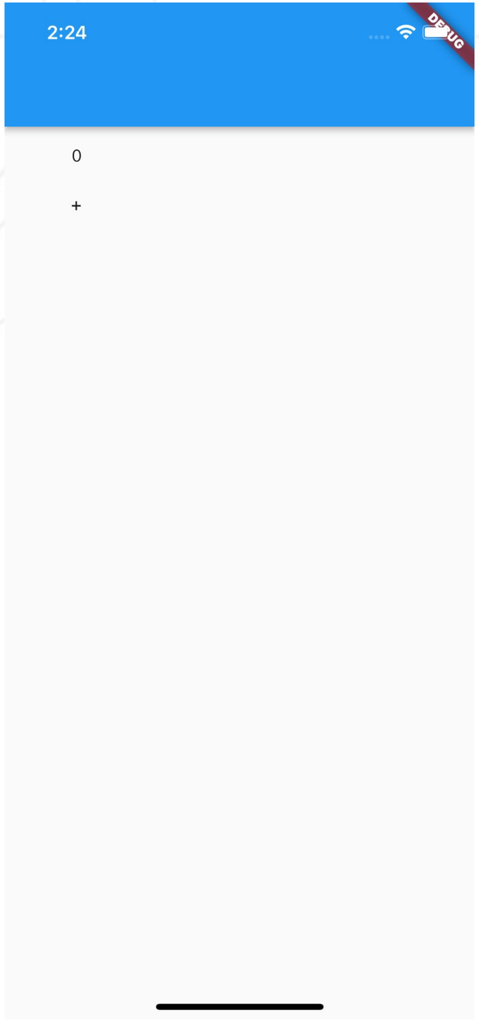
Flutter Hooks Rules

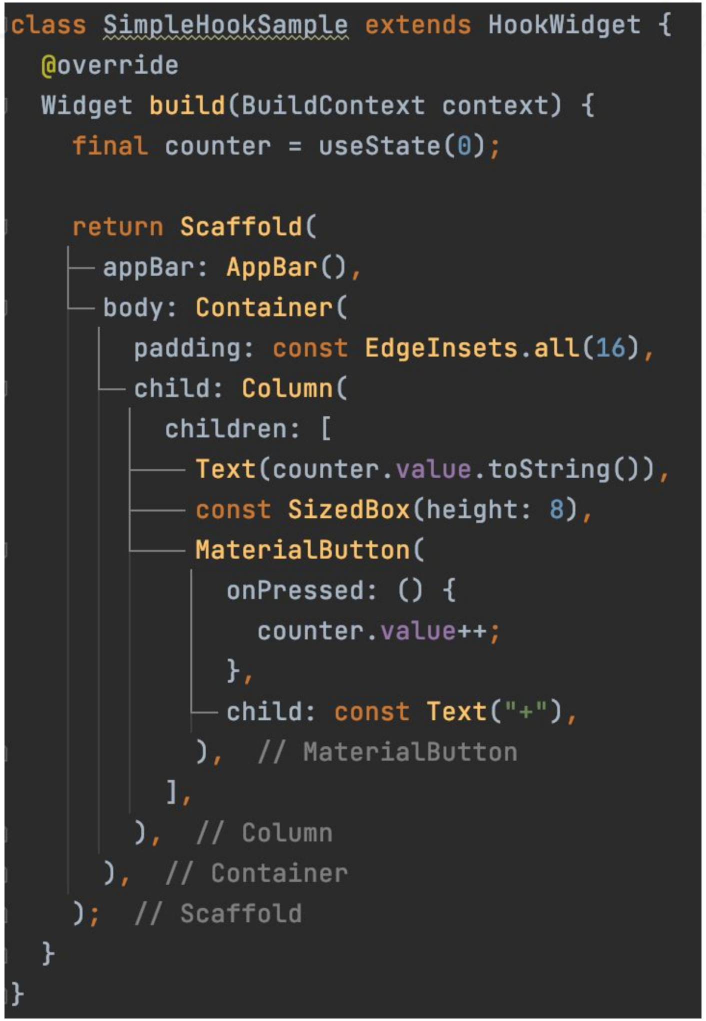
Due to hooks being obtained from their index, some rules must be respected:

- **DO** always prefix your hooks with use
- **DO** call hooks unconditionally
- **DON'T** wrap *use* into a condition

Existing Flutter Hooks

- **useEffect** - Useful for side-effects and optionally canceling them
- **useState** - Create variable and subscribes to it
- **useMemoized** - Cache the instance of a complex object
- **useTextEditingController** - Create a *TextEditingController*
- **useFocusNode** - Create a *FocusNode*
- *…*





```
class SimpleHookSample extends HookWidget {
  @override
  Widget build(BuildContext context) {
    final counter = useState(0);

    return Scaffold(
      appBar: AppBar(),
      body: Container(
        padding: const EdgeInsets.all(16),
        child: Column(
          children: [
            Text(counter.value.toString()),
            const SizedBox(height: 8),
            MaterialButton(
              onPressed: () {
                counter.value++;
              },
              child: const Text("+"),
            ), // MaterialButton
          ],
        ), // Column
      ), // Container
    ); // Scaffold
  }
}
```

Demo

Should I validate data locally?

Local validation

- Pros
  - Saving network resources
  - Making backend happy
  - The user receives feedback immediately
- Cons
  - **You'll need to do force update if validation is wrong or specification changes!!!**
  - **Backend might need to support different apps version!!**
  - Not really saving network & backend resources if validation is wrong

It depends

When to validate?

- Simple data that should never change
  - Empty fields
  - Required fields
  - Minimum password length
- You are 100% sure specification won’t change

When not?

- Complex data
  - Exact password specification
  - Email*
- Things that might change in the future

Why?

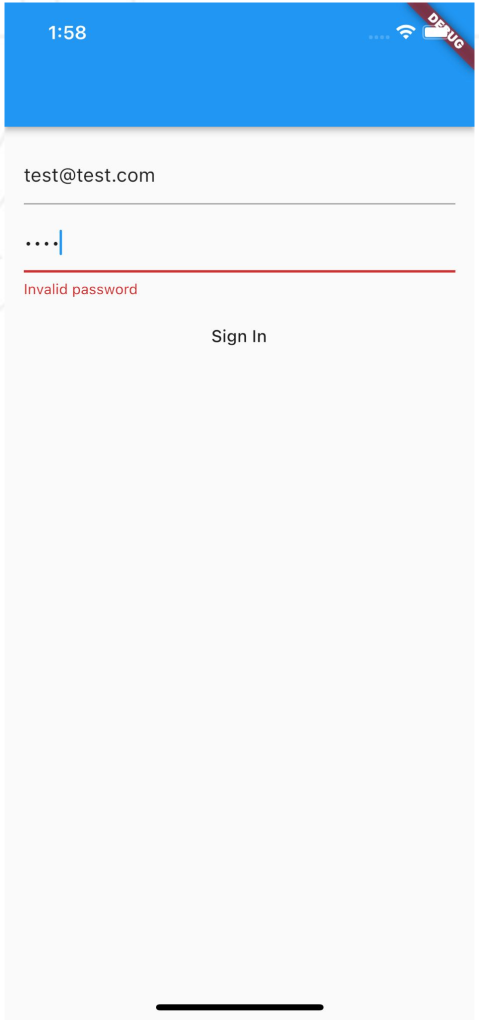
- Regular expressions might be tricky
- Email Address Regular Expression
  - **There is no perfect email regex!**
  - General Email Regex (RFC 5322 Official Standard) - **99.99%**
  - (?:[a-z0-9!#$%&'*+/=?^_`{|}~-]+(?:\.[a-z0-9!#$%&'*+/=?^_`{|}~-]+)*|"(?:[\x01-\x08\x0b\x0c\x0e-\x1f\x21\x23-\x5b\x5d-\x7f]|\\[\x01-\x09\x0b\x0c\x0e-\x7f])*")@(?:(?:[a-z0-9](?:[a-z0-9-]*[a-z0-9])?\.)+[a-z0-9](?:[a-z0-9-]*[a-z0-9])?|\[(?:(?:25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)\.){3}(?:25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?|[a-z0-9-]*[a-z0-9]:(?:[\x01-\x08\x0b\x0c\x0e-\x1f\x21-\x5a\x53-\x7f]|\\[\x01-\x09\x0b\x0c\x0e-\x7f])+)\])

Can validation lead to a data breach?

We build digital products.

LeanCode

YES!



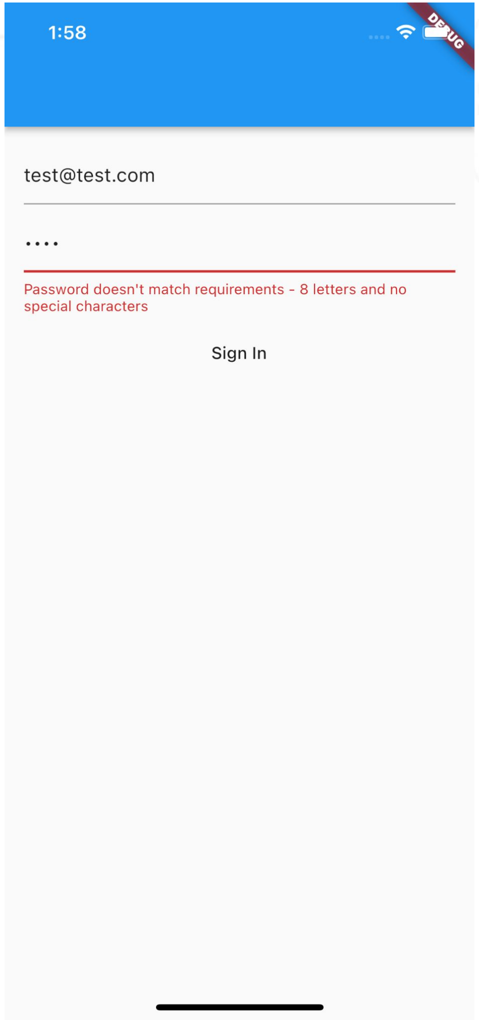
1:58

test@test.com

••••

Invalid password

Sign In



1:58

test@test.com

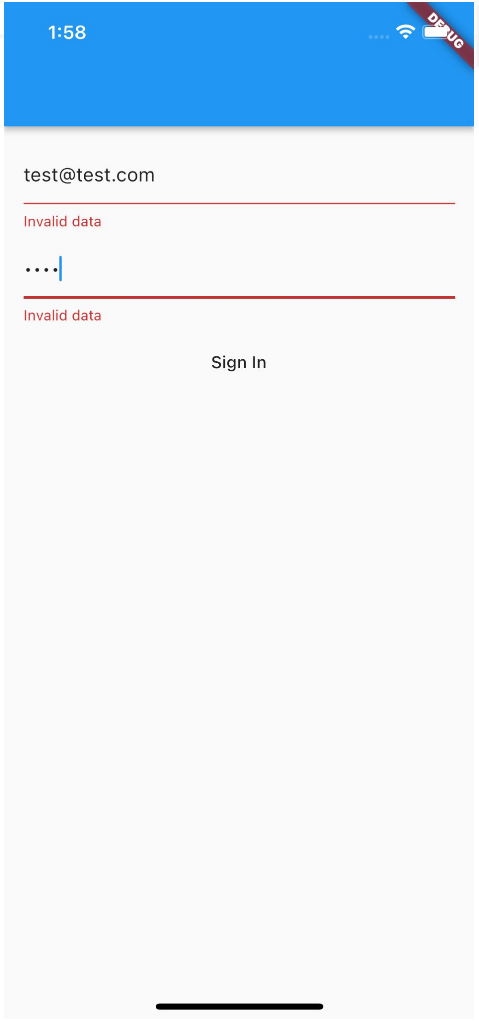
••••

Password doesn't match requirements - 8 letters and no special characters

Sign In

We build digital products.

LeanCode

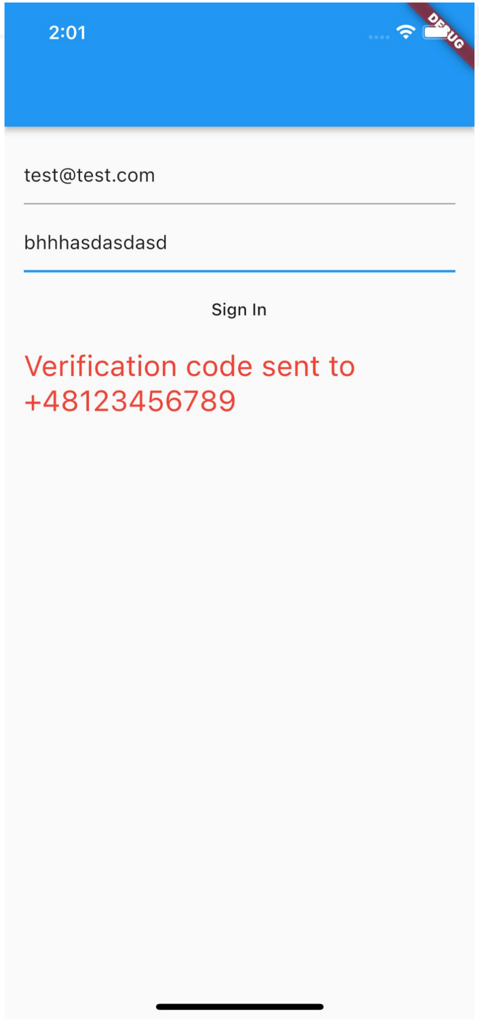


test@test.com

Invalid data

••••

Invalid data


2:01

DEBUG

test@test.com

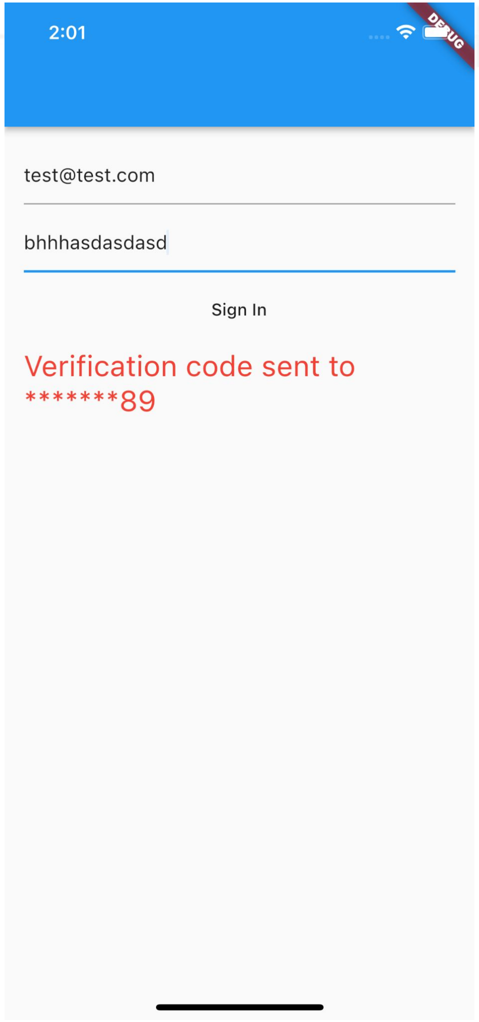
bhhhasdasdasd

Sign In

Verification code sent to +48123456789

We build digital products.

LeanCode

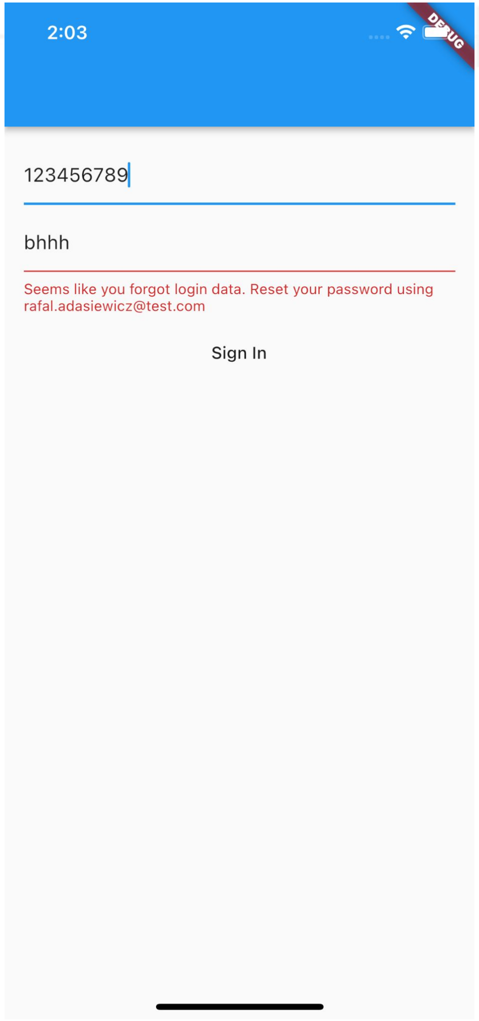


test@test.com

bhhhasdasdasd

Sign In

Verification code sent to *******89



123456789

bhhh

Seems like you forgot login data. Reset your password using rafal.adasiewicz@test.com

Questions?