

Sub-second integration tests for your React Native app and Bluetooth device

September, 2021

Lars Thorup
fullstackagile.eu

Agenda

- A story about test automation at SOUNDBOKS
- How can we test an app that controls a Bluetooth device?
 - End-to-end tests are slow and fragile
 - Unit tests with manual mocks lie to us
 - Mock recording gives us fast, robust integration testing
- Demo!

```
$ ./node_modules/.bin/jest app/redux/modules/bluetooth/connectToDevice.unit.test.ts
PASS app/redux/modules/bluetooth/connectToDevice.unit.test.ts (10.786s)
  redux/modules/bluetooth/connectToDevice
    ✓ should connect to device and verify resulting state (165ms)

Test Suites: 1 passed, 1 total
Tests:       1 passed, 1 total
Snapshots:  0 total
Time:        10.983s, estimated 13s
Ran all test suites matching /app\\redux\\modules\\bluetooth\\connectToDevice.unit.test.ts/i.
Done in 12.96s.
```

Test automation at SOUNDBOKS

- Bluetooth Performance Speaker
 - Equalizer
 - Lock
 - Join
 - LOUD!



- End-to-end testing
 - Appium
 - Jenkins on a Macbook Pro
 - Samsung, Huawei, iPhone



SOUNDBOKS end-to-end testing

- 15 scenarios, 1-5 minutes per scenario
- Total feedback time: 2:30 hours
 - 38 minutes on Samsung
 - 55 minutes on iPhone
- Improvement on manual testing
- False negatives >10% of the time

```
Running "mocha --spec "output/tsc/e2e-test/src/scenario/settings/ble/auto-reconnect.e2e.test.js""
```

```
auto-reconnect - samsung-sb3
  ✓ should login and control speaker (29354ms)
  ✓ should power speaker off (1019ms)
  ✓ should eventually show speaker as "connecting" (6137ms)
  ✓ should power speaker on (2021ms)
  ✓ should eventually show speaker as "connected" again (17987ms)
```

```
5 passing (1m)
```

When to use end-to-end testing vs unit testing?

- End-to-end testing

- Faster and cheaper than manual testing!
- Covers the entire system: device, phone, server
- Exposed to real-world timing & wireless noise
- Relentlessly uncovers hard-to-reproduce issues

- Unit testing

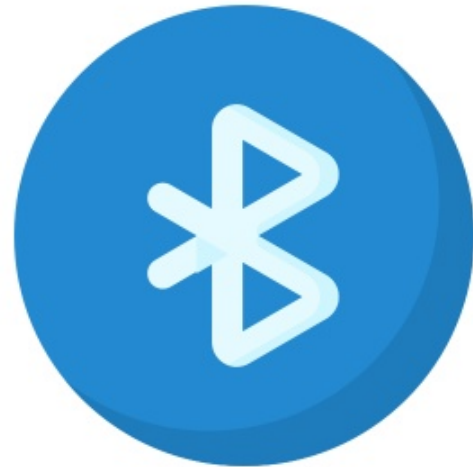
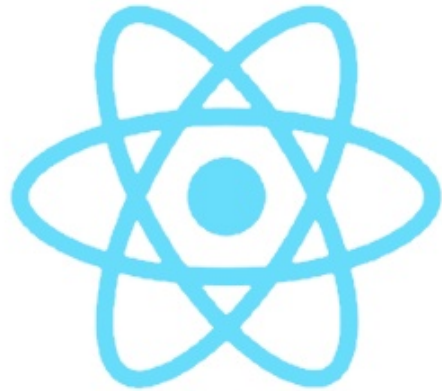
- No physical setup
- Much faster feedback - minutes instead of hours
- Much more robust - ~100% trustworthy feedback

→ Have a few of these

→ Use for most testing!

Unit testing a React Native / BLE app

- React Native
- react-native-ble-plx
- Jest



Manual mocks

- Simulated behavior of BLE device
- Hard coded BLE messages and traffic patterns

```
jest.spyOn(bleManager, 'startDeviceScan').mockImplementation(async (uuids, options, listener) {  
  let scanIndex = 0;  
  const devices = [  
    { name: '#212222', id: 'AA:AA:AA:AA' },  
    { name: '#212223', id: 'AA:AA:AA:AB' },  
    { name: '#212224', id: 'AA:AA:AA:AC' },  
  ];  
  setInterval(() => {  
    scanIndex = (scanIndex + 1) < devices.length ? scanIndex + 1 : 0;  
    listener(null, devices[scanIndex]);  
  }, 1);  
});
```

But... manual mocks lie to us!

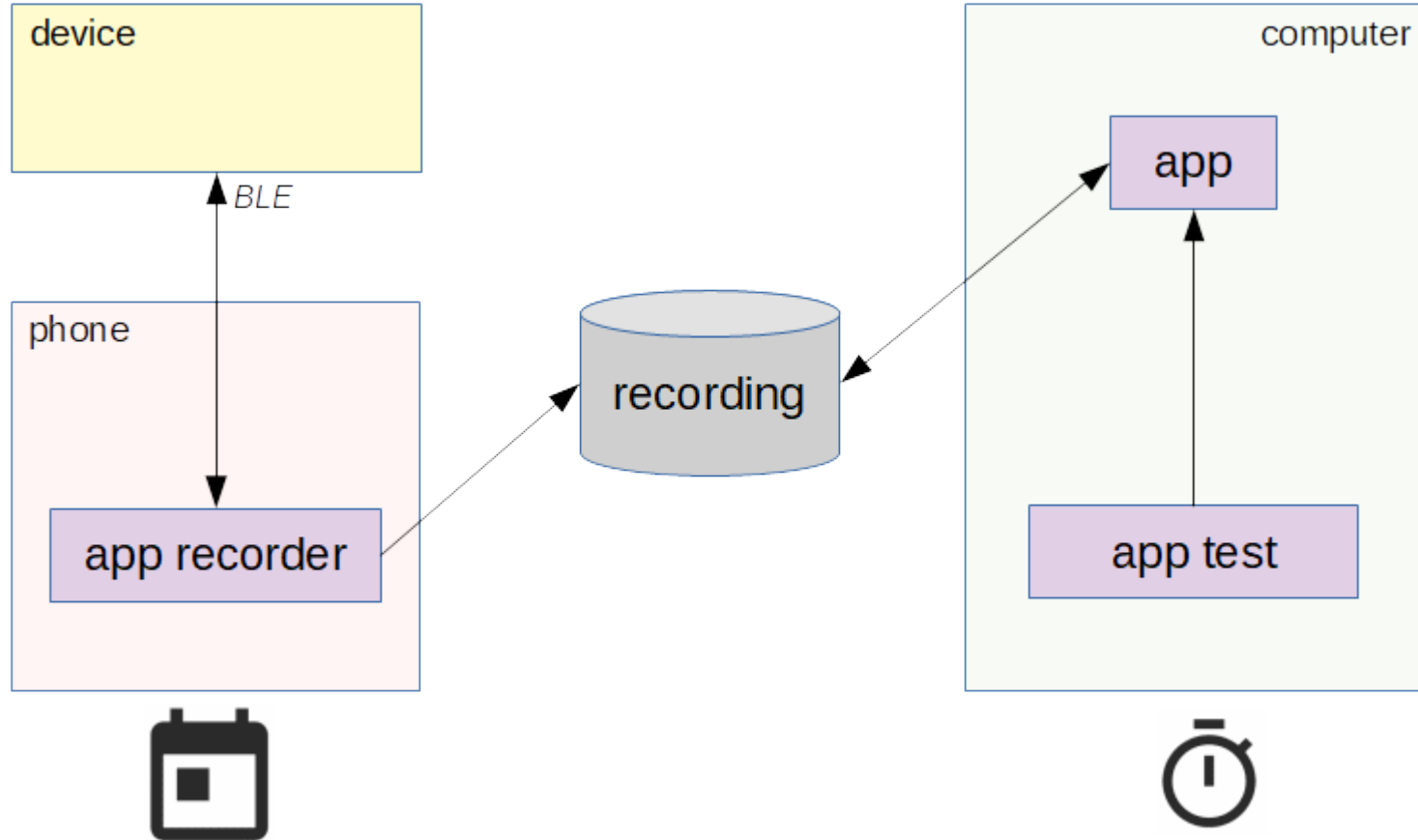
- To test app code in isolation...
...we manually mock device traffic
- When the protocol changes but the app code is not changed...
...the app will break
- But unit tests will still pass!?!
- So...
...manual mocks lie to us!



Can we get the best of both worlds?

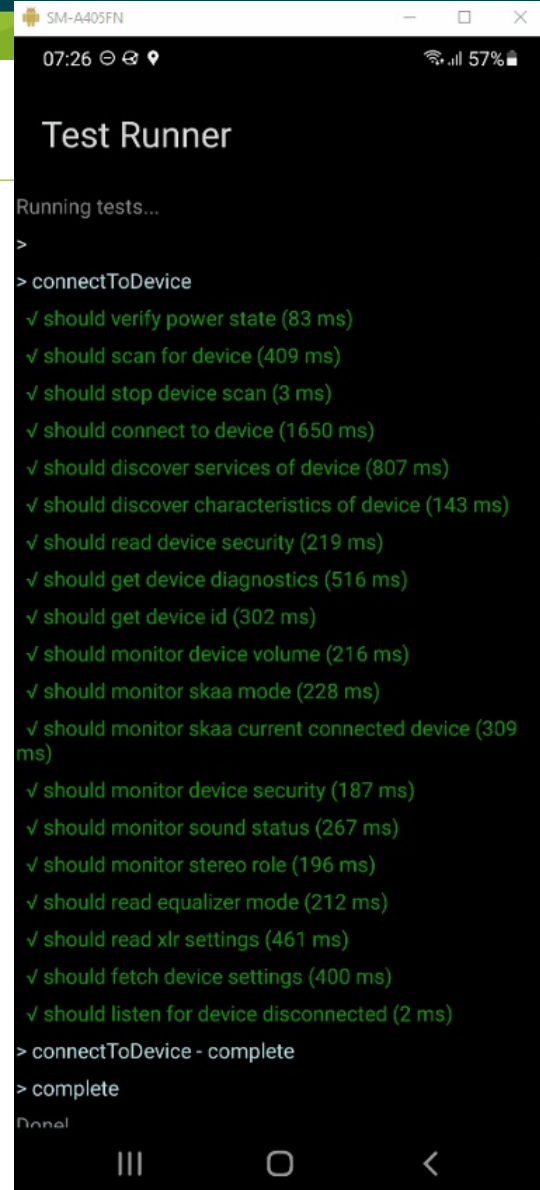
- Use **mock recording** when unit testing
- Provides true integration testing
- Gives speed and robustness of unit testing
- Not just for HTTP - also for Bluetooth BLE traffic

Mock recording for true integration testing



Record traffic

- Run occasionally
 - E.g. weekly
 - When making protocol changes
- Against real device
- Capture real traffic
- Verify expectations



```
SM-A405FN
07:26 57%
Test Runner
Running tests...
>
> connectToDevice
  ✓ should verify power state (83 ms)
  ✓ should scan for device (409 ms)
  ✓ should stop device scan (3 ms)
  ✓ should connect to device (1650 ms)
  ✓ should discover services of device (807 ms)
  ✓ should discover characteristics of device (143 ms)
  ✓ should read device security (219 ms)
  ✓ should get device diagnostics (516 ms)
  ✓ should get device id (302 ms)
  ✓ should monitor device volume (216 ms)
  ✓ should monitor skaa mode (228 ms)
  ✓ should monitor skaa current connected device (309 ms)
  ✓ should monitor device security (187 ms)
  ✓ should monitor sound status (267 ms)
  ✓ should monitor stereo role (196 ms)
  ✓ should read equalizer mode (212 ms)
  ✓ should read xlr settings (461 ms)
  ✓ should fetch device settings (400 ms)
  ✓ should listen for device disconnected (2 ms)
> connectToDevice - complete
> complete
Done!
```

Test the app

- Use recording as BLE mock
- Tests are fast and robust
 - 200ms per test
 - 50 BLE messages
- You can run tests as often as you want
- True integration testing

```
$ ./node_modules/.bin/jest app/redux/modules/bluetooth/connectToDevice.unit.test.ts
iteration 0
iteration 1
iteration 2
iteration 3
iteration 4
iteration 5
iteration 6
iteration 7
iteration 8
iteration 9
PASS app/redux/modules/bluetooth/connectToDevice.unit.test.ts (12.708s)
  redux/modules/bluetooth/connectToDevice
    ✓ should connect to device and verify resulting state (213ms)
```

Sample recording

```
{
  "records": [
    {
      "type": "command",
      "command": "state",
      "request": {},
      "response": "PoweredOn"
    },
    {
      "type": "command",
      "command": "startDeviceScan",
      "request": {
        "uuidList": [
          "F5C26570-64EC-4906-B998-6A7302879A2B"
        ],
        "scanOptions": {
          "allowDuplicates": true
        }
      }
    },
    {
      "type": "event",
      "event": "deviceScan",
      "args": {
        "device": {
          "id": "12-34-56-78-9A-BC",
          "localName": "#999001",
          "manufacturerData": "WAgAAQUAAen8F8KQP5qxI11txA==",
          "name": "#999001",
          "rssi": null
        },
        "error": null
      }
    }
  ]
}
```

Set up react-native-ble-plx mock with Jest

- `__mocks__/react-native-ble-plx.js`

```
import { State } from 'react-native-ble-plx';
import { BleManagerMock as BleManager } from 'react-native-ble-plx-mock-recorder';

export {
  State,
  BleManager,
};
```

Write app tests using Jest

```
describe('DeviceList', () => {  
  it('should load and show device info', async () => {  
    const recording = JSON.parse(fs.readFileSync('../deviceList.recording.json'));  
    const { blePlayer } = getBleManager();  
    blePlayer.mockWith(recording);  
  
    // when: render the app  
    render(withStore(<DeviceListScreen />, configureStore()));  
  
    // when: simulating BLE scan response  
    act(() => {  
      blePlayer.playUntil('scanned'); // Note: causes re-render,  
    });  
  
    // when: clicking a device  
    fireEvent.press(screen.getByA11yLabel('Connect to "The Speaker"'));  
  
    // then: eventually battery level is shown  
    expect(screen.findByA11yLabel('"The Speaker" battery level')).toHaveTextContent('🔋 42%');  
  });  
});
```

load recording to simulate BLE traffic

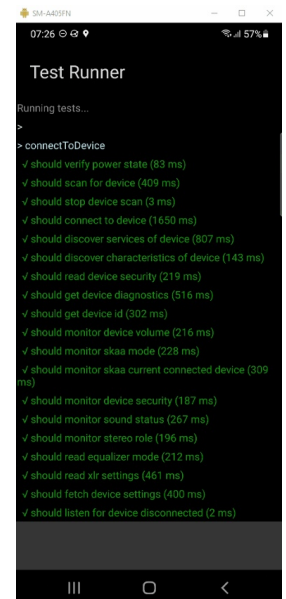
render app

click buttons

Recorder app

- A lightweight React Native app
- Dedicated to recording traffic for the app tests
- Documents what traffic is necessary for given scenarios
- Evolves over time with new or updated scenarios
- Has to be created in addition to the app tests
 - Jest does not run on the phone
 - react-native-ble-plx does not run on a laptop

```
npx react-native init BleAppRecorder --template react-native-ble-plx-mock-recorder-mocha-template
```



Write the recorder app

```
describe(recordingName, () => {
  let bleManager;
  let bleRecorder;
  let device;

  before(() => {
    bleRecorder = new BleRecorder({ bleManager: new BleManager() });
    bleManager = bleRecorder.bleManagerSpy;
  });

  it('should receive scan result', async () => {
    device = await new Promise((resolve, reject) => {
      bleManager.onStateChange((powerState) => {
        if (powerState === BleState.PoweredOn) {
          bleManager.startDeviceScan(null, null, (error, d) => {
            if (!error && bleRecorder.isExpected(d)) {
              resolve(d);
            } else if (error) {
              reject(error);
            }
          });
        }
      });
    }, true);
  });
  bleRecorder.label('scanned');
});

after(() => {
  bleRecorder.close();
});
});
```

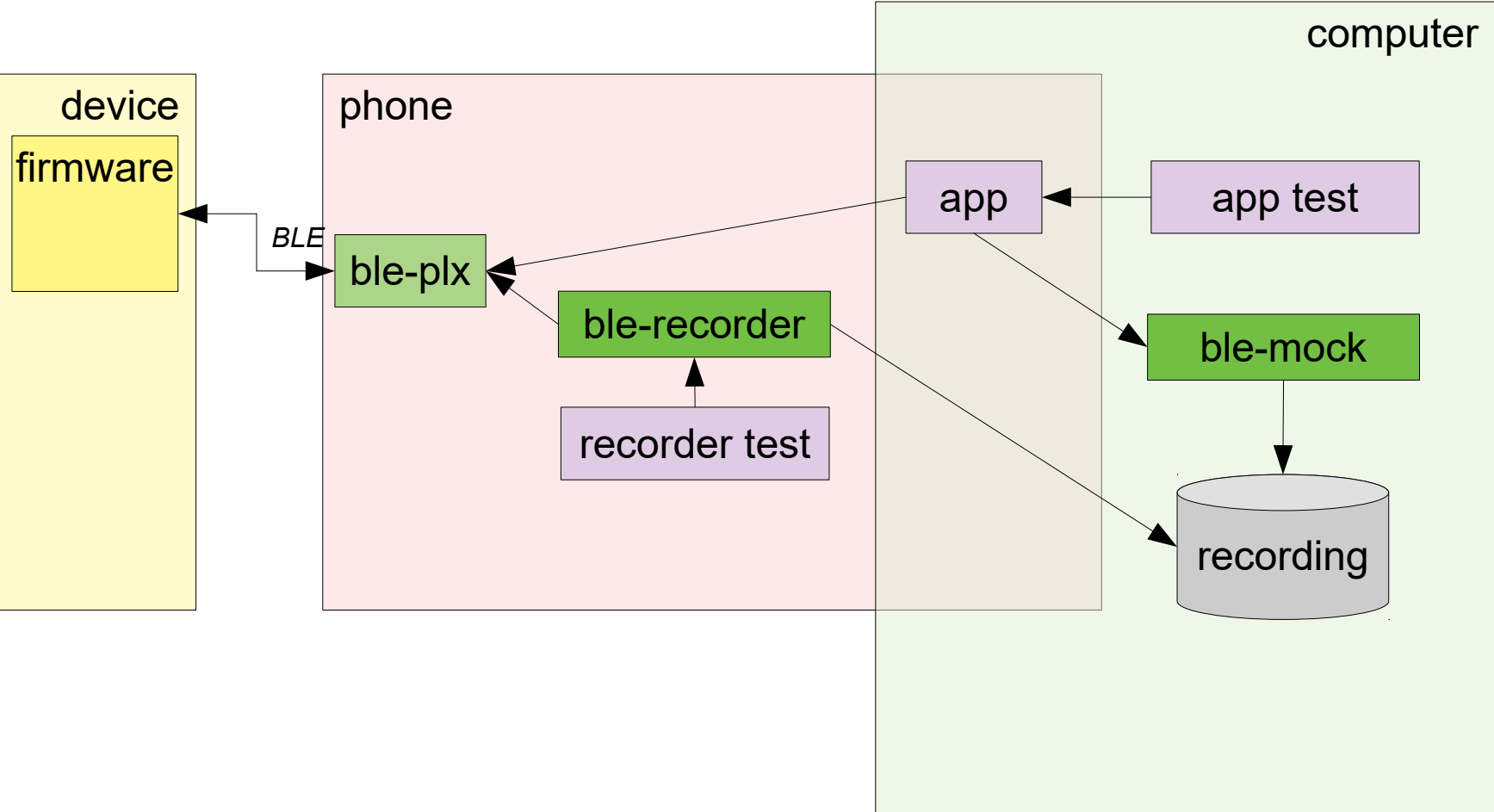
start recorder

generate traffic

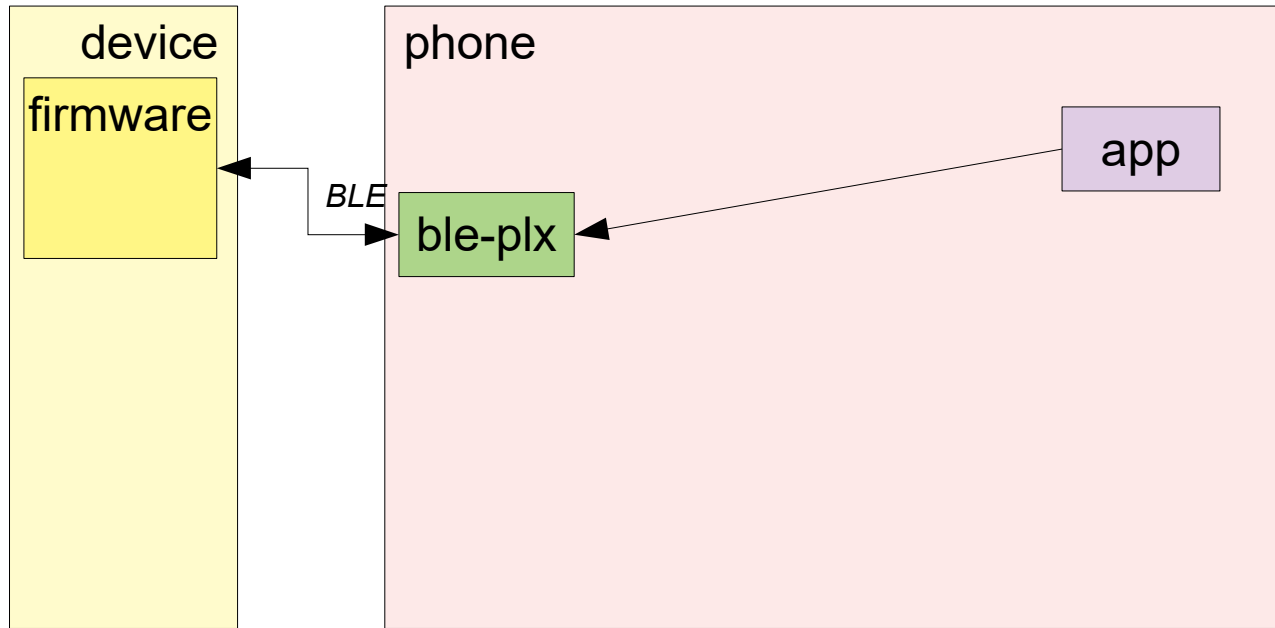
add label

save recording

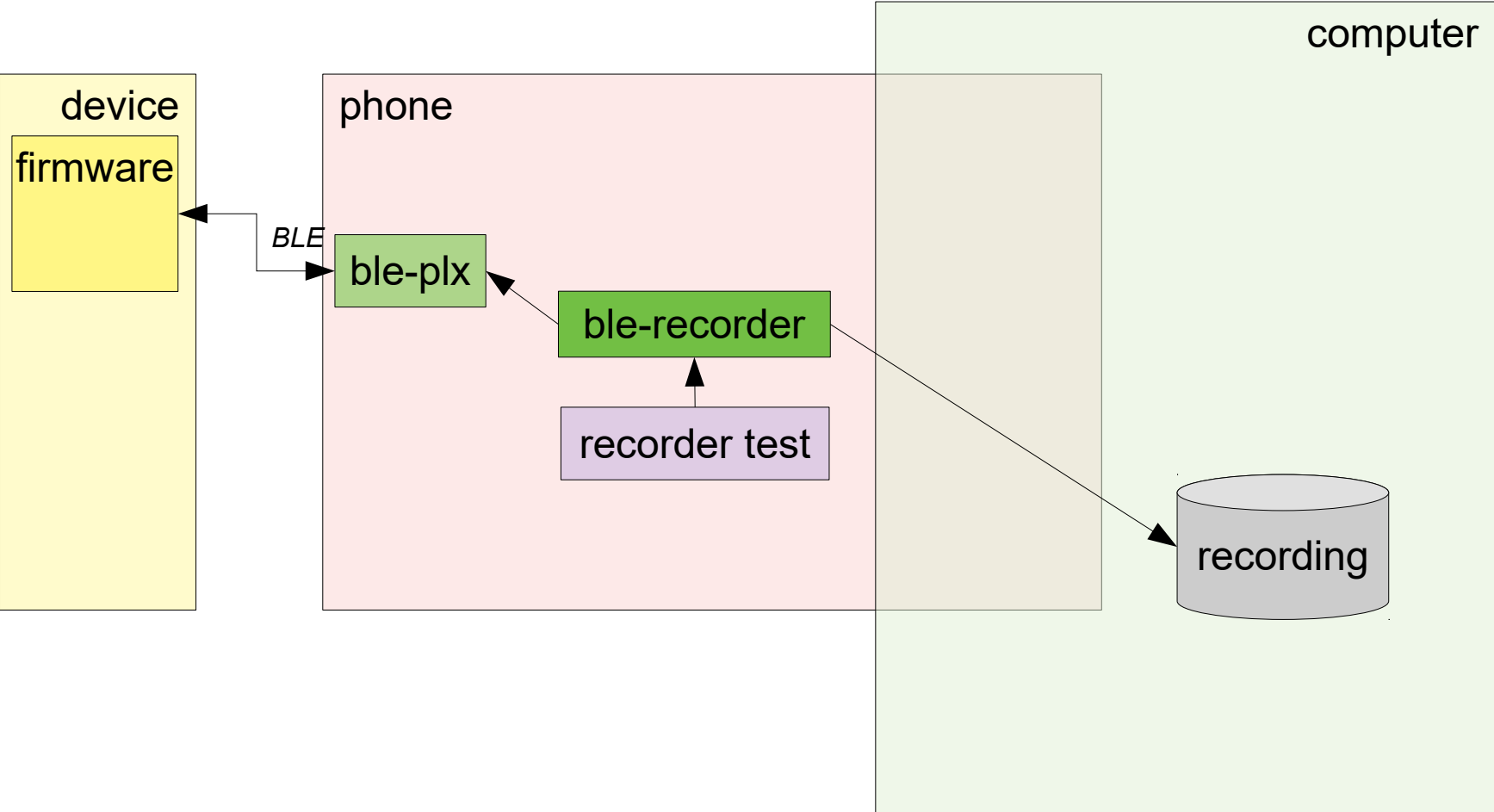
Architecture: overview



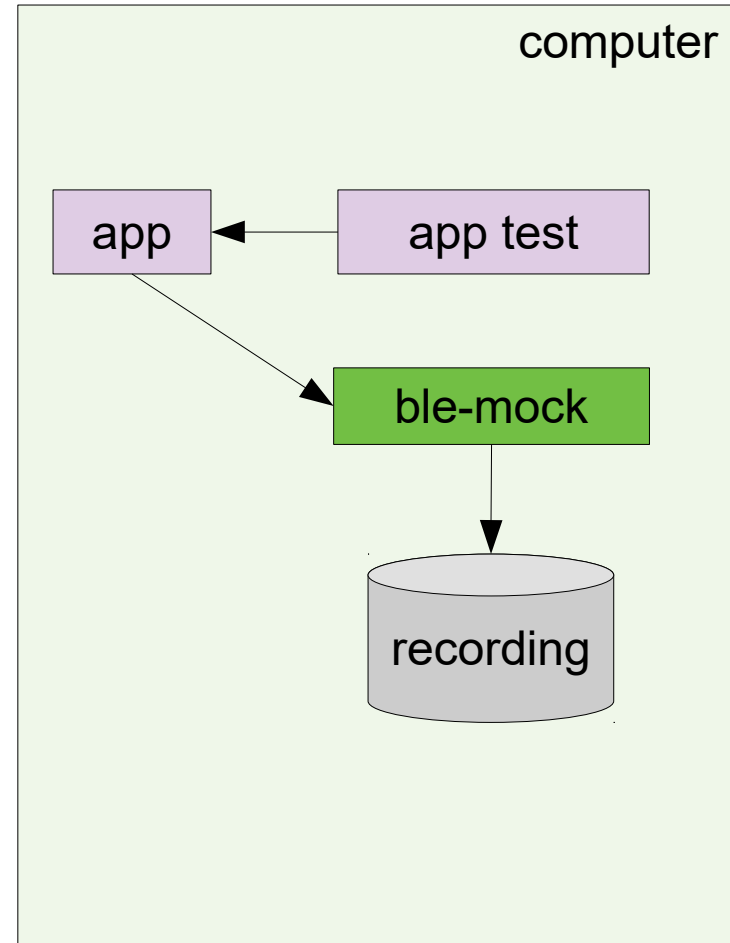
Architecture: running the app



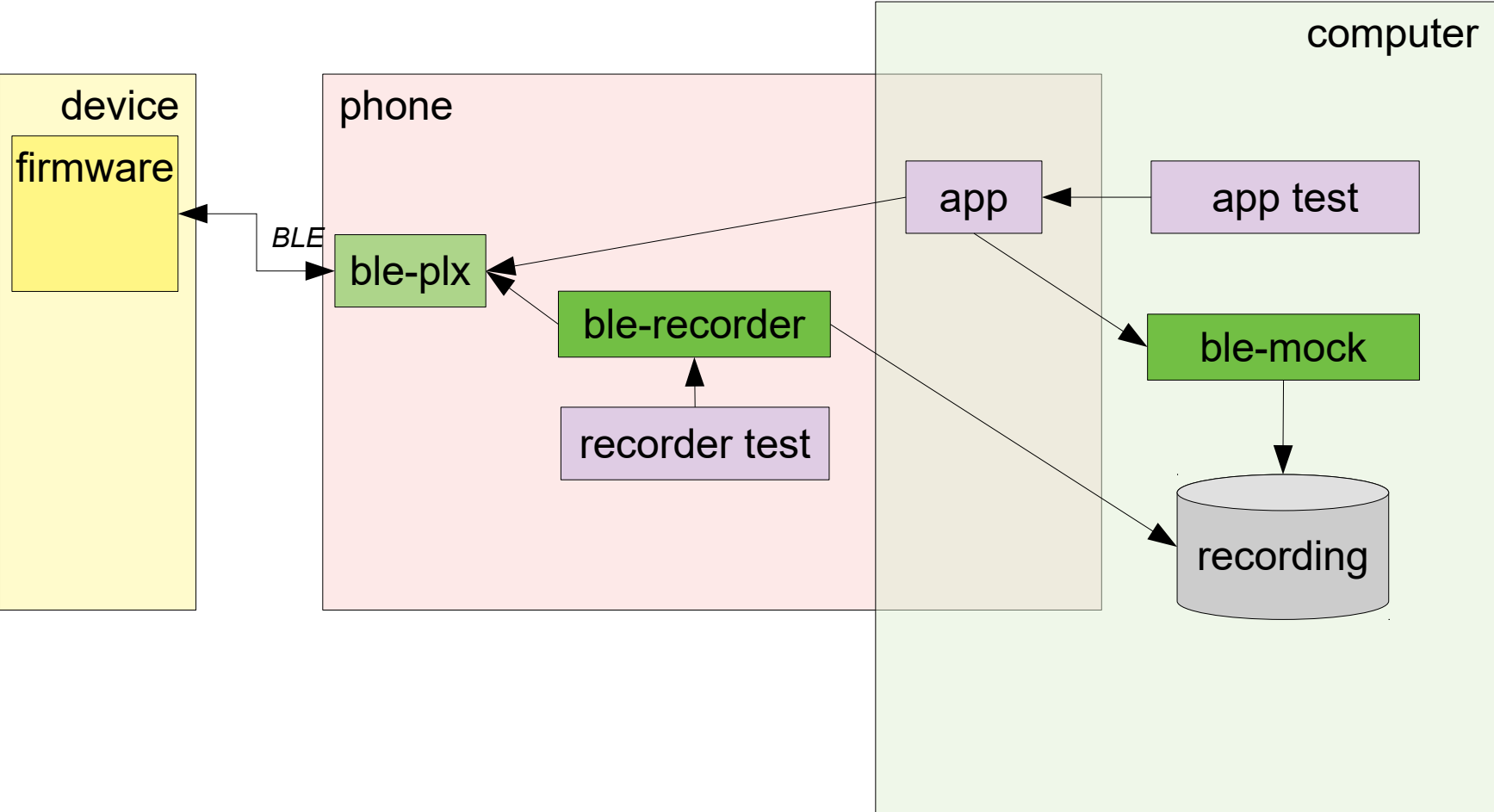
Architecture: recording traffic



Architecture: testing the app



Architecture: overview



Time for questions!



How to handle different developer devices?

- Run recordings locally as well as on build server
- Developers have different local devices with different ids
- Shared recordings should use canonical device id
- So: you can specify this mapping in the recorder app

```
const deviceMap = {
  expected: {
    '00:12:6F:BA:A7:74': {
      name: 'BeoPlay A1',
      recordId: '12-34-56-78-9A-BC',
    },
    '34:81:64:68:F7:E1': {
      name: 'BeoPlay A1',
      recordId: '12-34-56-78-9A-BC',
    },
  },
  record: {
    '12-34-56-78-9A-BC': {
      name: 'The Speaker',
    },
  },
};
```


How to handle varying values?

- Devices have different values of characteristics
- E.g. RSSI, battery level, volume
- Recording should use canonical values
- So: you can specify the recorded value and verify the actual value in the recorder app

```
it('should read battery level', async () => {
  const { id } = device;
  bleRecorder.queueRecordValue(base64FromUint8(42));

  const { value } = await bleManager.readCharacteristicForDevice(
    id,
    service.battery.uuid,
    characteristic.batteryLevel.uuid
  );

  const batteryLevel = uint8FromBase64(value);
  console.log(`(actual batteryLevel = ${batteryLevel})`);
  expect(batteryLevel).to.be.at.least(0);
  expect(batteryLevel).to.be.at.most(100);
});
```

Can we still do manual mocking?

- Have at least one *integration* test per BLE “message”
- Additional tests can be unit tests with manual mocks
 - parameterized tests, boundary testing, combinatorial testing
- So: you can manually mock BLE traffic in your app tests

```
const { blePlayer } = bleManagerMock;
blePlayer.mockWith({
  records: [
    { command: 'startDeviceScan', request: { uuidList: defaultUUIDs, scanOptions: {} }, type: 'command' },
    { event: 'deviceScan', args: { device }, autoPlay: false, type: 'event' },
    { command: 'stopDeviceScan', request: {}, type: 'command' },
    { label: 'scanned', type: 'label' },
  ]
});
```

How to debug recordings?

- BLE traffic
 - refers to services and characteristics by UUID
 - values are base64 encoded
 - not very readable!
- So: the recording file includes debugging information for your convenience

```
{
  "type": "command",
  "command": "readCharacteristicForDevice",
  "request": {
    "characteristicUUID": "00002a19-0000-1000-8000-00805f9b34fb",
    "id": "12-34-56-78-9A-BC",
    "serviceUUID": "0000180f-0000-1000-8000-00805f9b34fb"
  },
  "response": {
    "serviceUUID": "0000180f-0000-1000-8000-00805f9b34fb",
    "uuid": "00002a19-0000-1000-8000-00805f9b34fb",
    "value": "Kg=="
  },
  "debug": {
    "serviceUUID": "Battery Service",
    "characteristicUUID": "Battery Level",
    "value": "<Buffer 2a> '*'"
  }
},
```

Read more!

- Blog post
 - fullstackagile.eu/2021/06/24/bluetooth-ble-mock-recorder
- Repository
 - github.com/larsthorup/react-native-ble-plx-mock-recorder
- Package
 - npmjs.com/package/react-native-ble-plx-mock-recorder
- Contributions welcome!



Lars Thorup
fullstackagile.eu
twitter.com/larsthorup

