QML Guidelines:

* Keep binding expressions as simple as possible. Avoid updating bound property when it's not necessary (triggering re-evaluation of binding expressions). See [Property bindings](https://doc.qt.io/qt-5.11/qtquick-performance.html#property-bindings) for more details.
* Prefer using concrete types instead of var. See [Using var vs concrete types](https://doc.qt.io/qt-5.11/qtquick-bestpractices.html#type-safety) for details.
* Avoid using massive JavaScript functions in qml. See [JavaScript code](https://doc.qt.io/qt-5.11/qtquick-performance.html#javascript-code) fro details.
* Prefer using simple Row, Column types instead if RowLayout, ColumLayout. If usage of Layout type is unavoidable read [Layouts](https://doc.qt.io/qt-5.11/qtquick-bestpractices.html#using-qt-quick-layouts) and [Qt Quick Layouts overview](https://doc.qt.io/qt-5.11/qtquicklayouts-overview.html)
* Use Loader for lazy instantiation. See [Loader](https://doc.qt.io/qt-5.11/qtquick-performance.html#lazy-initialization) for details.
* Almost all qml views should have data models. See [Data Models](https://doc.qt.io/qt-5.11/qtquick-performance.html#models-and-views), [QAbstractItemModel subclass](https://doc.qt.io/qt-5.11/qtquick-modelviewsdata-cppmodels.html" \l "qabstractitemmodel-subclass) and [qml ListModel](https://doc.qt.io/qt-5.11/qtquick-modelviewsdata-modelview.html" \l "models)
* Don't use [Qt.createComponent()](https://doc.qt.io/qt-5.11/qml-qtqml-qt.html" \l "createComponent-method) and [Qt.createQmlObject()](https://doc.qt.io/qt-5.11/qml-qtqml-qt.html" \l "createQmlObject-method). Everything that is described here: [Dynamic object creation](https://doc.qt.io/qt-5.11/qtqml-javascript-dynamicobjectcreation.html) should be avoided in Prima.
* Keep QML object attributes in the same order: <https://doc.qt.io/qt-5.11/qml-codingconventions.html#qml-object-declarations>
* Qt qml documentation to read:
  + [Qt Quick Best Practices](https://doc.qt.io/qt-5.11/qtquick-bestpractices.html)
  + [Performance Considerations And Suggestions](https://doc.qt.io/qt-5.11/qtquick-performance.html)
  + [Integration QML and C++](https://doc.qt.io/qt-5.11/qtqml-cppintegration-topic.html)
  + [Overview - QML and C++ Integration](https://doc.qt.io/qt-5.11/qtqml-cppintegration-overview.html#choosing-the-correct-integration-method-between-c-and-qml)
  + [Exposing Attributes of C++ Types to QML](https://doc.qt.io/qt-5.11/qtqml-cppintegration-exposecppattributes.html)
  + [Writing QML Extensions with C++](https://doc.qt.io/qt-5.11/qtqml-tutorials-extending-qml-example.html)
  + [Data Type Conversion Between QML and C++](https://doc.qt.io/qt-5.11/qtqml-cppintegration-data.html#data-ownershipand%20C++)
* Open question 1: qml [Instantiator](http://doc.qt.io/qt-5/qml-qtqml-instantiator.html) vs Repeater vs Loader ???
  + Instantiator is a combination of Repeater and Loader. It's useful when you have a model that you use to create many objects dynamically. Also supports asynchronous loading.