داکیومنت ساخت درخت با استفاده از qml و jquery

( )

فهرست

۱ تعریف مسئله

ساخت درخت با یک ui بسیار زیبا یکی از مسائل چالش‌برانگیز برای کیوت نیز می‌باشد این درخت باید ویژگی‌های ازجمله متن و عکس و نیز باشد و به‌صورت مرتب شد نیز کنار هم و خاصیت پدر فرزندی را به‌خوبی نیز اجرا کند . برای این کار را های مختلفی ازجمله ویجت نیز موردبررسی قرار گرفت و نتوانستیم همه ویژگی‌های یک درخت را نیز در ویجت پیاده‌سازی کنیم .راه‌حل دوم نیز با استفاده از qml و یک کتابخانه آماده[۱] این درخت ساخته شد اما کتابخانه با c++14 نوشته‌شده بود که مورداستفاده c++11 قرار نمی‌گرفت و برای downgrade کردن ان وقت زیاد و تا حدودی امکان‌پذیر نبود .

[۱] QUICKQanava

درنهایت به سرا jquery های موجود نیز رفتیم درخت‌های در jquery بسیار زیاد بودن اما یکی از آن‌ها تمام ویژگی‌های یک درخت و در وسعت بزرگ‌تر یک گراف را نیز داشت [۲] ما از این jquery نیز برای کار خود استفاده نموده‌ایم

در ادامه نحوه پیاده‌سازی و فایل موجود را توضیح می‌دهیم(کدها کامند گذاری شده‌اند)

۲ فایل‌های موجود

برای کار با این پروژه به دانش چند موضوع نیز احتیاج می‌باشد، javascript , html , qml این زبان‌ها اصول پروژه را تشکیل می‌دهند

۲-۱ فایل‌های c++

در این پروژه ۵ فایل c++ نیز وجود دارد

فایل main که شروع کنند برنامه نیز می‌باشد در این فایل به جر کدهای اولیه موجود کد دیگری نیز وجود ندارد و ما نیز از همان کد زمان ساخت پروژه نیز استفاده کرده‌ایم

فایل‌های mainwindow که برای نمایش صفحه ویجت ما نیز به‌کاربرده می‌شوند در این فایل کتابخانه‌های اضافه کرده و همین‌طور متغیرهای برای نمایش درخت ساخته‌ایم .

فایل datamanger که یک کلاس برای ارتباط متغیرهای c++ با qml نیز ساخت شده است که در این فایل فقط دو متغیر برای پاس دادن به qml استفاده می‌شود

۲-۲ فایل‌های javascript

سه فایل js. نیز در این پروژه مورداستفاده قرارگرفته است

فایل go.js کتابخانه اصلی ترسیم درخت این فایل می‌باشد و ما از این فایل برای ترسیم درخت نیز استفاده می‌کنیم

فایل qwebchanel.js که این فایل از داکیومنت های خود کیوت نیز می‌باشد برای ارتباط qml و فایل html نیز می‌باشد.

فایل jquery.js که این فایل برای خواندن فایل json که اطلاعات دران قرار دار مورداستفاده نیز قرار می‌گیرد

۲-۳ فایل qml

یک فایل main که برای استفاده ارتباطات و همچنین ایجاد یک بستر برای نمایش درخت نیز می‌باشد

[۲] Go.js

۲-۴ فایل html

درخت موجود در این فایل ساخته می‌شود و سپس با استفاده از qml در ویجت و نمایش داده می‌شود .

۳ پیاده‌سازی

۳-۱ main.qml

در فایل qml ابتدا یک صفحه وب می سازیم که یک url از کلاس datamanae نیز می گیرد .

```
Rectangle{
    anchors.fill: parent
    color: "black"
    WebEngineView{
        id:webEngine
        anchors.fill:parent
           url: datamanager.htmlURL()
        webChannel: webChanel

    }
}
```

یک فایل webchanel نیز داریم که با فایل qwebchanel.js در ارتباط نیز می باشد و متغیر هایی نیز برای این فایل تعریف میکنیم تا بتوانیم در فایل html از ان استفاده کنیم .

```
Component.onCompleted: {
    webChanel.registerObject("foo",myobject)
    webChanel.registerObject("bar",datamanager)
}
```

۳-۲ فایل mainwindow

ابتدا در ui این فایل یک horizontal layout اضافه میکنیم

با استفاده از سه کتابخانه qwidget ، Qqmlcontext و QqmlEngein نیز ارتباط بین ویجت و qml را نیز برقرا میکنیم به طوری که qml ما در ویجت نیز نمایش داده شود .

```
DataManager *d=new DataManager;
view= new QQuickView(QUrl(QLatin1String("qrc:/main.qml")));
view->engine()->rootContext()->setContextProperty("datamanager",d);
qmlwidget=QWidget::createWindowContainer(view);
ui->horizontalLayout->addWidget(qmlwidget);
```

در شکل بالا یک شی از کلاس datamanager نیز ساخته‌ایم و مام property های این کلاس را به qml داده‌ایم بعدازاین کار تمام متغیرها و کلاس‌های موجود در datamanager نیز در main.qml در دسترس نیز می‌باشد(این روش برای زمانی هست که پروژه اصلی ما ویحت نیز باشد)

۳-۳ فایل orgcharteditor

ساخت درخت و نمایش دخت در این فایل نیز وجود دارد ابتدا در این فایل تمام فایل‌های javascript قبلی را در بستر html نیز اضافه می‌کنیم و یک بستر برای نمایش می‌سازیم

```
<!DOCTYPE html>
<html lang="en">
<body bgcolor="#92a8d1">
<script type="text/javascript" src="qrc:/qwebchannel.js"></script>
<script src="qrc:/go.js"></script>
<script src="qrc:/jquery-3.6.0.min.js"></script>
<div id="allSampleContent" class="p-4 w-full">
```

سپس با استفاده از کتابخانه go.js نیز درخت خود و الزامات آن را پیاده‌سازی می‌کنیم.

```
function init() {
  // Since 2.2 you can also author concise templates with method chaining instead of GraphObject.make
  // For details, see https://gojs.net/latest/intro/buildingObjects.html
  const $ = go.GraphObject.make;  // for conciseness in defining templates

  myDiagram =
    $(go.Diagram, "myDiagramDiv", // must be the ID or reference to div
      {
        maxSelectionCount: 1, // users can select only one part at a time
        layout:
          $(go.TreeLayout,
            {
              treeStyle: go.TreeLayout.StyleLastParents,
              arrangement: go.TreeLayout.ArrangementHorizontal,
              // properties for most of the tree:
              angle: 90,
              layerSpacing: 35,
              // properties for the "last parents":
              alternateAngle: 90,
              alternateLayerSpacing: 35,
              alternateAlignment: go.TreeLayout.AlignmentBus,
              alternateNodeSpacing: 20
            }),
        "undoManager.isEnabled": true // enable undo & redo
      });
```

```
// manage boss info manually when a node or link is deleted from the diagram
myDiagram.addDiagramListener("SelectionDeleting", e => {
  var part = e.subject.first(); // e.subject is the myDiagram.selection collection,
  // so we'll get the first since we know we only have one selection
  myDiagram.startTransaction("clear boss");
  if (part instanceof go.Node) {
    var it = part.findTreeChildrenNodes(); // find all child nodes
    while (it.next()) { // now iterate through them and clear out the boss information
      var child = it.value;
      var bossText = child.findObject("boss"); // since the boss TextBlock is named, we can access it by name
      if (bossText === null) return;
      bossText.text = "";
    }
  } else if (part instanceof go.Link) {
    var child = part.toNode;
    var bossText = child.findObject("boss"); // since the boss TextBlock is named, we can access it by name
    if (bossText === null) return;
    bossText.text = "";
  }
  myDiagram.commitTransaction("clear boss");
});
```

بخشی از کدها در شکل بالا نیز می‌باشد برای استفاده بیشتر می‌توانید به کدها مراجعه کنید

برای این‌که بتوانیم فایل json موجود را که در فایل datamanager نیز قرار دارد از فایل jquery و qwebchanel نیز استفاده می‌کنیم و آدرس مدنظر را به آن می‌دهیم

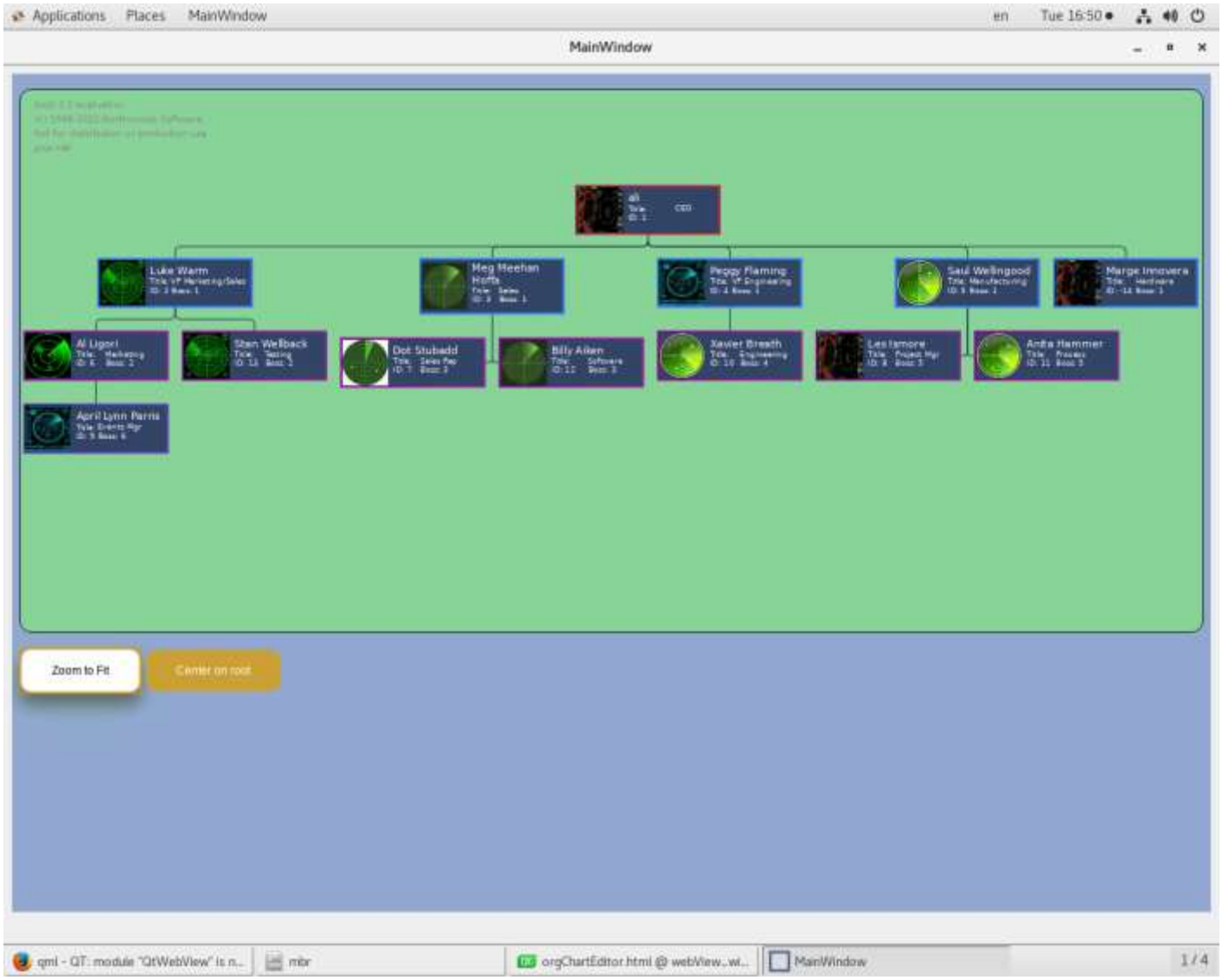
```
new QWebChannel(qt.webChannelTransport, function (channel) {
    // all published objects are available in channel.objects under
    // the identifier set in their attached WebChannel.id property
     var foo =channel.objects.foo;
    var dManager = channel.objects.bar;
    jQuery.getJSON(foo.hello,function(data){
    myDiagram.model =new go.TreeModel(data)});
    });
```

در پایان برای راحتی نمایش درخت دو button طراحی می‌کنیم و برای زیبایی از css نیز استفاده می‌کنیم

```
<style>
    .btton {
        background-color:#CBA135;
        border:none;
        color:white;
        padding:15px 32px;
        text-align:Center;
        text-decoration:none;
        display:inline-block;
        font-Size: 12px;
        border-radius:12px;
        transition-duration:1s
    }
    .btton:hover{
        background-color:#FFFFFF;
        color:black;
        border:2px solid #CBA135;
        box-shadow: 0 12px 16px 0 rgba(84,92,82,0.80),0 17px 50px 0 rgba(76,175,80,0.19)

    }
</style>
<p id="nodeDataArray"></p>
<div id="sample">
  <div id="myDiagramDiv" style="background-color: rgb(136, 212, 152); border: 1px solid black; border-radius:12px;heigh
  <p><button id="zoomToFit" class=btton>Zoom to Fit</button> <button id="centerRoot" class=btton>Center on root</button
  </div>
```

در پایان نتیجه کار به‌صورت زیر نیز می‌باشد



۴ نتیجه

درخت موجود یک نمونه نیز می‌باشد و می‌توان با استفاده از داکیومنت های خود go.js نمونه‌های زیباتری و با ویژگی‌های بیشتری نیز اضافه نمود.