Présentation du Projet tutoré

Javora

Jérémy VARANGE - Yohan BORDES- Noah LACASTE • 23/03/2021

Plan

Partie 1 - Vue d'ensemble

Partie 2 - Fonctionnalités de base

- Thèmes
- Menu fichier
- Menu édition

Partie 3 - Snippets

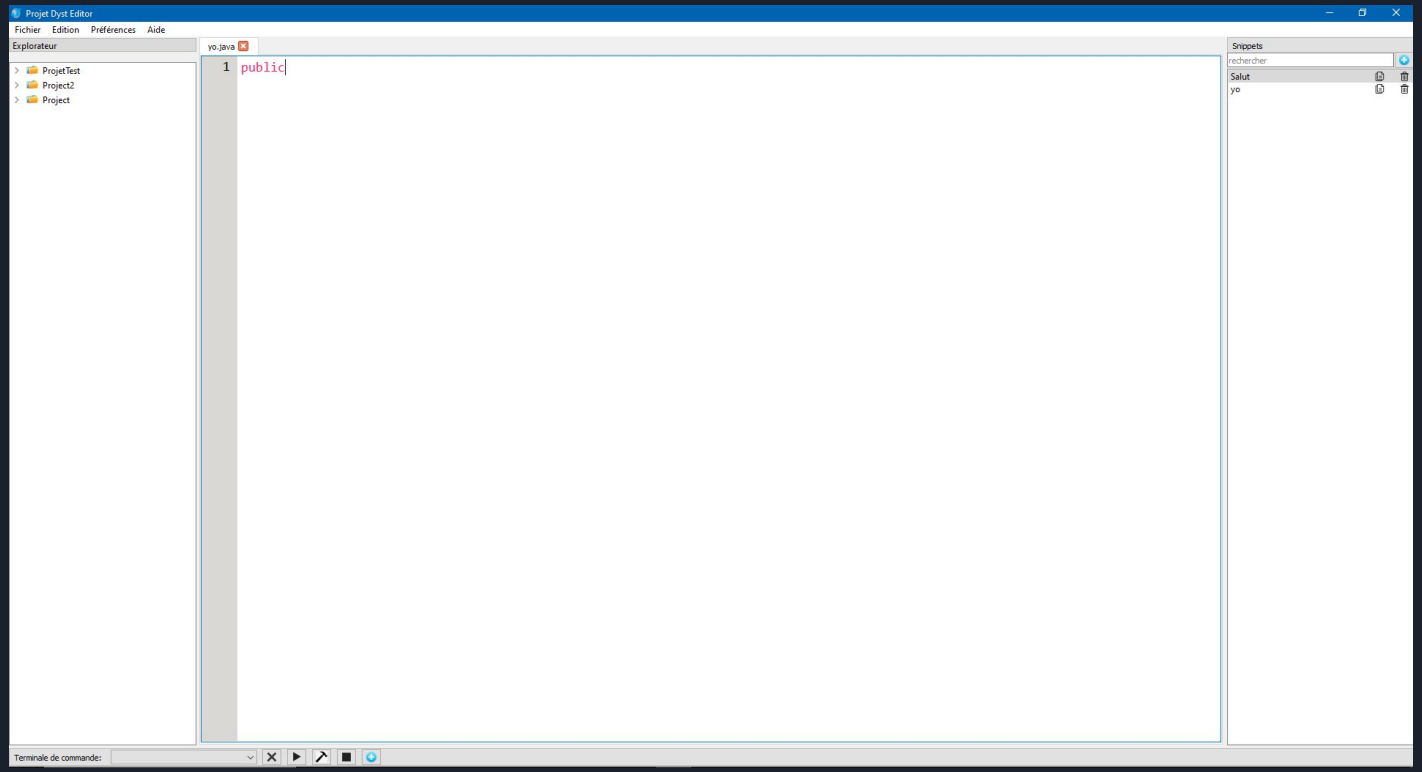
Partie 4 - Console

Partie 5 - Modèle

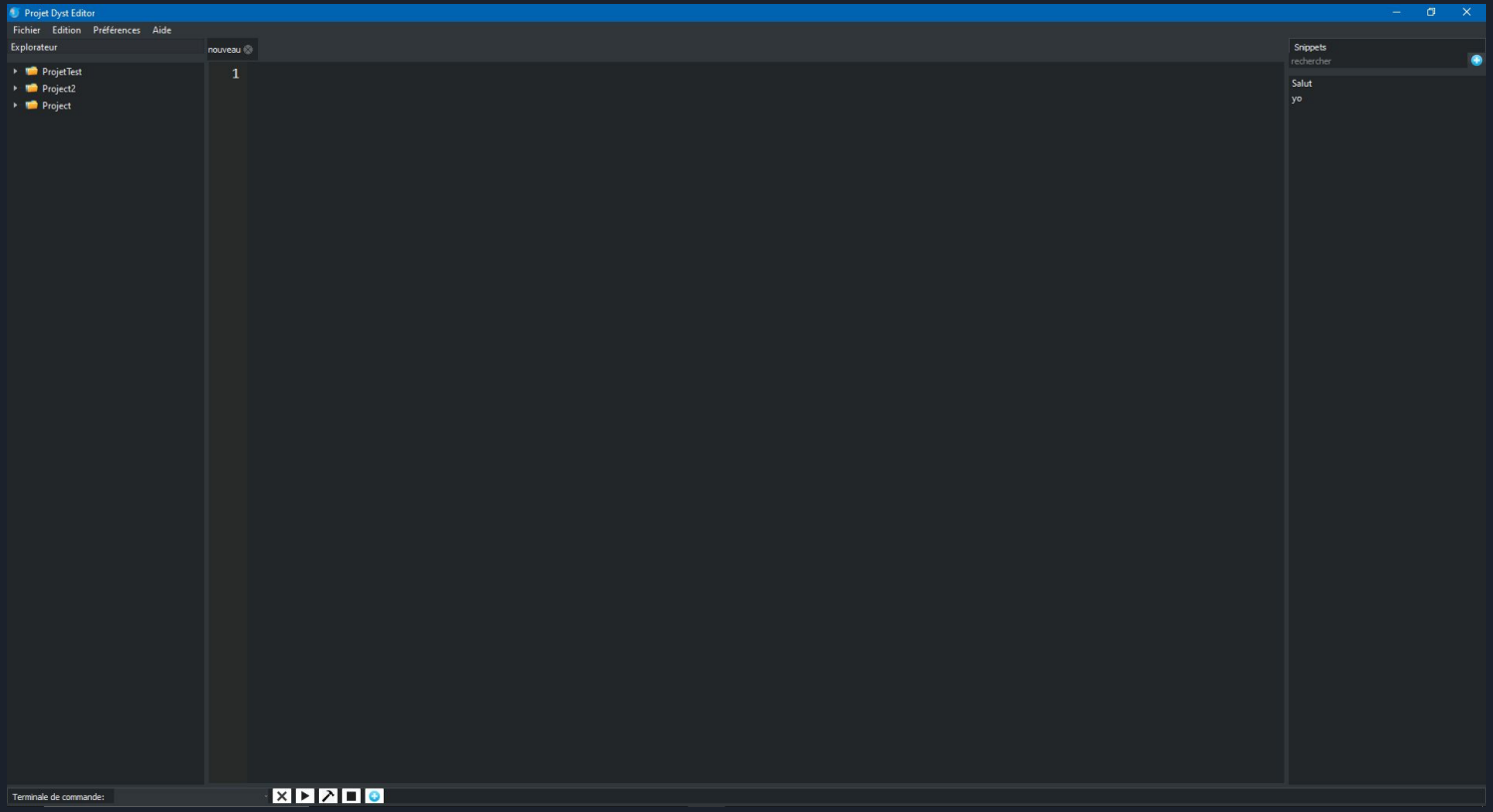
Partie 6 - Explorateur de fichier

Partie 7 - Editeur

Vue d'ensemble

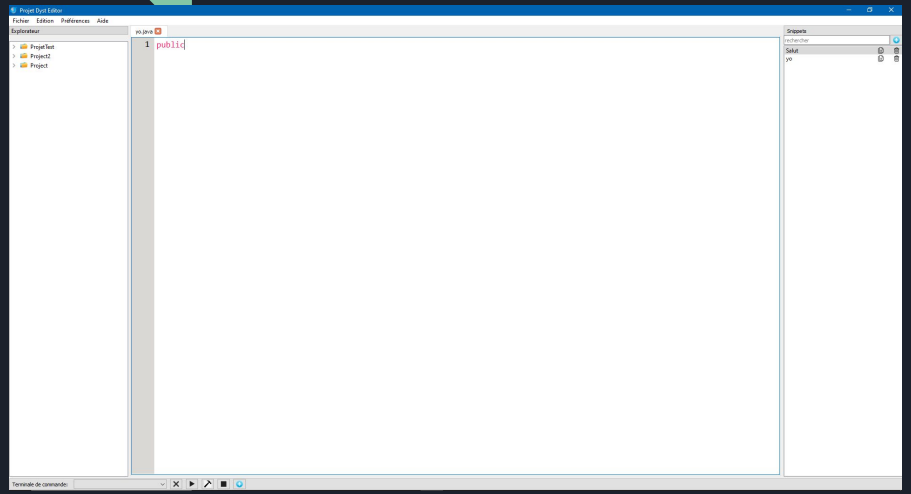


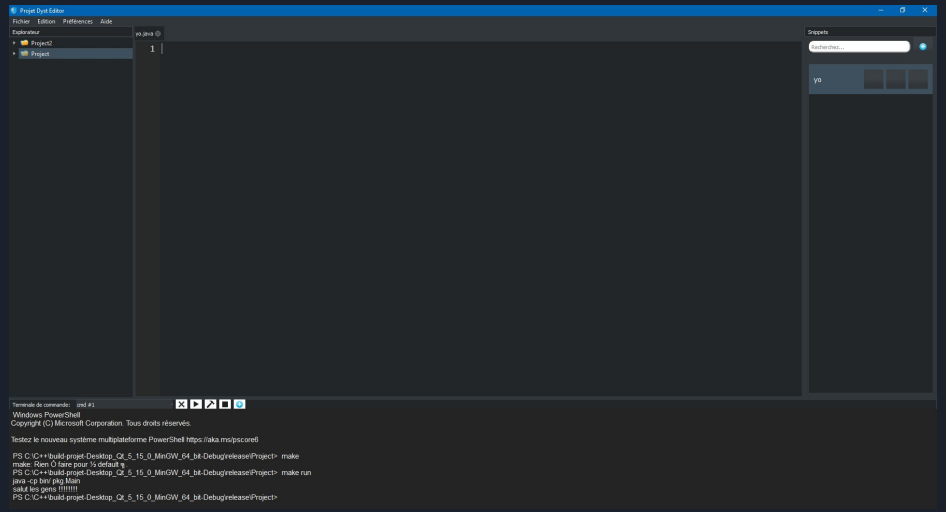
Vue d'ensemble



Partie 2 – Fonctionnalités de base

- Thèmes





Partie 2 – Fonctionnalités de base

- Thèmes

QSS thème clair

```
*[titlebar="true"]{
    background: rgb(225, 225, 225);
    border: 1px solid rgb(185, 185, 185);
    min-height: 20px;
}

QTreeView
{
    margin-top: 12px;
}
```

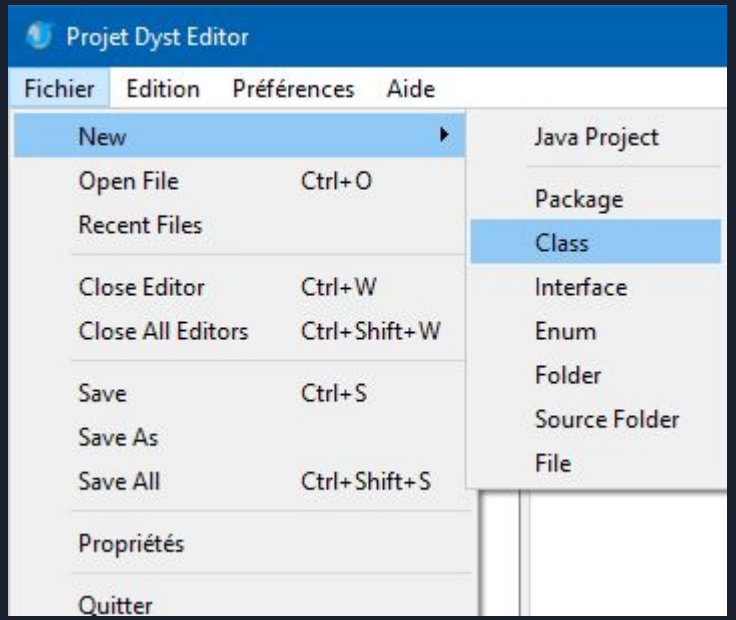
QSS thème sombre

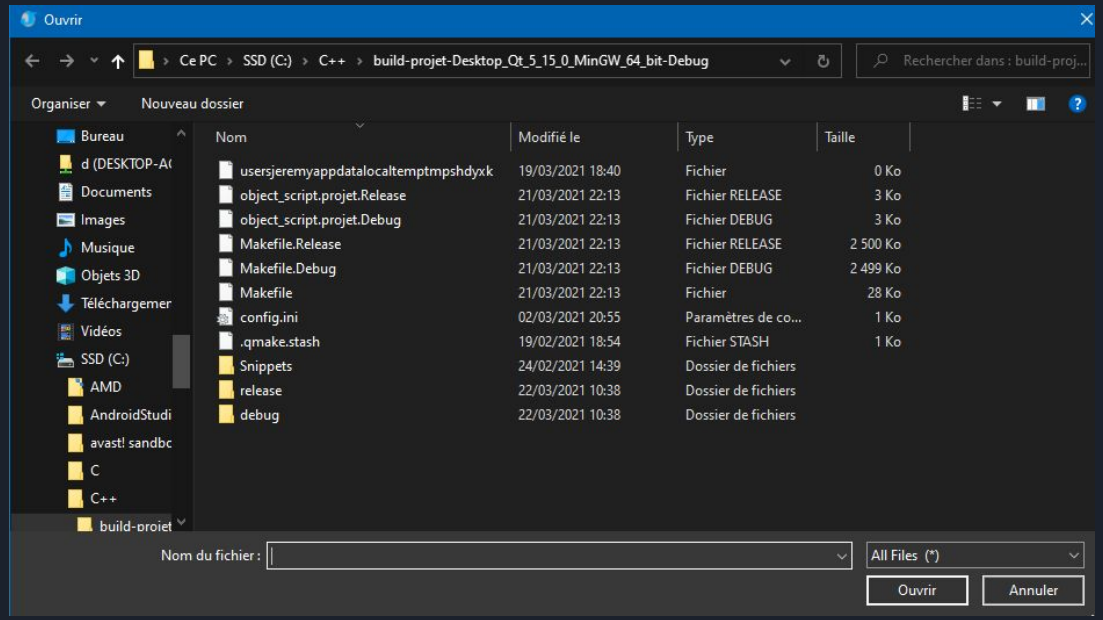
```
QAbstractSpinBox::up-arrow:hover
{
    border-image: url(:/dark/up_arrow.svg);
    width: 0.9ex;
    height: 0.6ex;
}

QAbstractSpinBox::down-arrow,
QAbstractSpinBox::down-arrow:disabled,
QAbstractSpinBox::down-arrow:off
{
    border-image: url(:/dark/down_arrow_disabled.svg);
    width: 0.9ex;
    height: 0.6ex;
}

QAbstractSpinBox::down-arrow:hover
{
    border-image: url(:/dark/down_arrow.svg);
    width: 0.9ex;
    height: 0.6ex;
}
*[nameSnippet="true"]{
    background: transparent;
    color: white;
    border: 0ex solid black;
}

*[titlebar="true"] QPushButton{
    background: white;
    border-top: #454b52;
    border-bottom: #454b52;
}

*[titlebar="true"]{
    background: #232629;
    border: 1px solid #454b52;
    min-height: 20px;
}

QLabel
{
    background: transparent;
    border: 0ex solid black;
}
```

Partie 2 – Fonctionnalités de base

- Menu fichier

Aperçu du menu fichier





Partie 2 – Fonctionnalités de base

- Menu fichier

Ouverture d’un fichier

```
void CodeEditorController::open(){
    QFile file(m_path);
    if(!file.open(QIODevice::ReadOnly | QIODevice::Text)){
        QMessageBox::warning(m_view,"impossible d'ouvrir le fichier","Impossible d'ouvrir le fichier "+m_path);
        return;
    }
    m_view->setUndoRedoEnabled(false);
    QTextStream in(&file);
    while(!in.atEnd()){
        m_view->appendPlainText(in.readLine());
    }
    file.close();
    m_view->setUndoRedoEnabled(true);
    m_view->document()->setModified(false);
    if(m_item != nullptr){
        connect(m_item,SIGNAL(rename(QString)),this,SLOT(rename(QString)));
        connect(m_item,SIGNAL(suppr()),this,SLOT(fileSuppr()));
    }
    QTextCursor cursor = m_view->textCursor();
    cursor.movePosition(QTextCursor::Start);
    m_view->setTextCursor(cursor);
```

Partie 2 - Fonctionnalités de base

- Menu édition

```
if(m_me->radio2->isChecked()){
    if(m_me->casse->isChecked()){
        while(editor->find(QRegExp(mot), QTextDocument::FindCaseSensitively)){
            m_me->count++;
            QTextEdit::ExtraSelection extra;
            extra.format.setBackground(color);
            extra.cursor = editor->textCursor();
            m_me->extraSelections.append(extra);
        }
```

Fonction de recherche dans l'éditeur de texte

```
        while(editor->find(m_me->mot->text())){
            QTextEdit::ExtraSelection extra;
            extra.format.setBackground(color);
            extra.cursor = editor->textCursor();
            extraSelections.append(extra);
            QTextCursor qc = editor->textCursor();
            if(qc.hasSelection()){
                isExist = true;
                qc.insertText(m_me->newMot->text());
            } else
                break;
        }
    }
    if(!isExist){
        QMessageBox msgBox(QMessageBox::Warning, "Mot inexistant", "Le mot n'existe pas.");
        msgBox.exec();
        s_Replace();
        return;
    }
    editor->setExtraSelections(extraSelections);
}
```

Fonction de remplacement dans l'éditeur de texte

Partie 3 - Snippets

Aperçu des snippets

Partie 3 - Snippets

Controller des snippet

```
void SnippetController::addSnippet(){
    SnippetDialog sd(m_sni->m_model);
    if(sd.exec() == QDialog::Accepted){
        qDebug() << "création du snippet";
        m_sni->m_model->addSnippet(sd.getSnippetName());
    }
}

void SnippetController::modifyFile(const QModelIndex &index){
    m_sni->m_fen->getController()->openEditor(QCoreApplication::applicationDirPath()+"/Snippets/" + m_sni->m_model->data(index,Qt::DisplayRole).toString() + ".java");
}
```

Partie 4 - Console

Aperçu de la console

```
Terminale de commande:  cmd #1
PS C:\C++\build-projet-Desktop_Qt_5_15_0_MinGW_64_bit-Debug\release\Project> make run
java -cp bin/ pkg.Main
salut les gens !!!!!!!!
PS C:\C++\build-projet-Desktop_Qt_5_15_0_MinGW_64_bit-Debug\release\Project>  ls


    Répertoire : C:\C++\build-projet-Desktop_Qt_5_15_0_MinGW_64_bit-Debug\release\Project


Mode                LastWriteTime         Length Name
----                -------------         ------ ----
d-----        19/03/2021     22:06                bin
d-----        19/03/2021     18:45                res
d-----        19/03/2021     18:46                src
-a----        20/03/2021     16:02            516 .javora.jpml
-a----        20/03/2021     17:34            424 Makefile
-a----        19/03/2021     23:37           1253 Project.jar


PS C:\C++\build-projet-Desktop_Qt_5_15_0_MinGW_64_bit-Debug\release\Project>
```

Partie 4 - Console

Historique des commandes écrites

```
if(e->key() == Qt::Key_Up){
    QTextCursor cursor = this->getCurrentCmd();
    if(histoIndex == this->historique.size()-1){
            currentCmd = cursor.selectedText();
    }
    cursor.removeSelectedText();
    if(this->historique.size() > 0)
        this->insertPlainText(this->historique[histoIndex]);
    if(histoIndex>0)
            histoIndex--;
    return;
```

Mise en place d'un processus

```
void Console::build(){
    cmd.write("make\n");
    execute();
}

void Console::execute(){
    cmd.write("make run\n");
}

Console::~Console(){
    cmd.close();
}
```

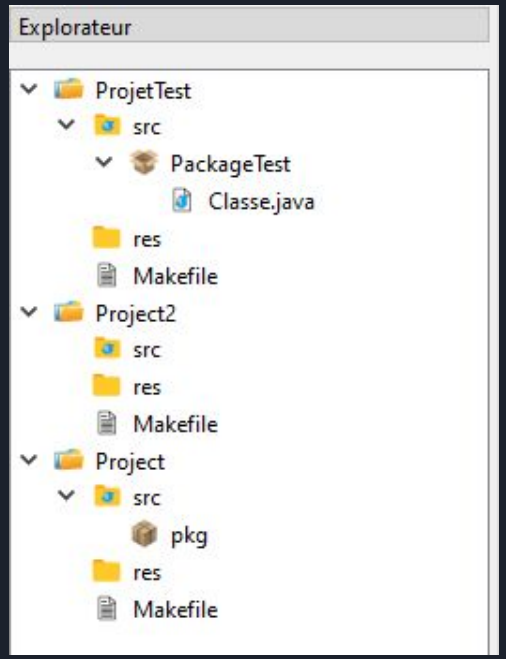
Partie 5 - Modèle

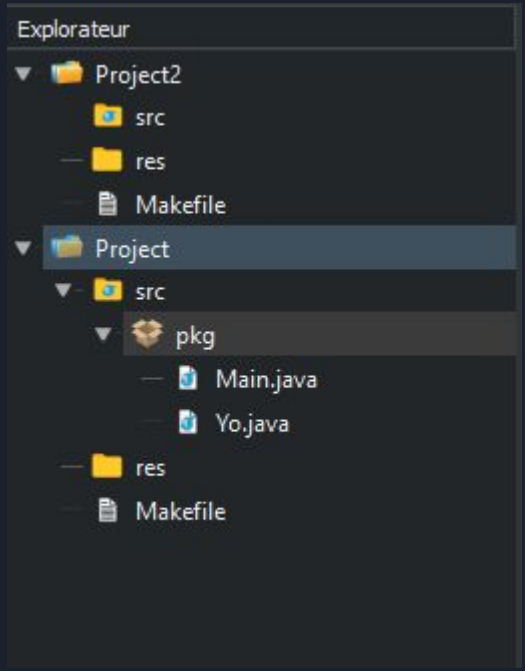
Récupération chemin des projets et chargement du modèle

```
Model::Model(): QAbstractItemModel(), root("root"), settings(QCoreApplication::applicationDirPath()+"/config.ini",QSettings::IniFormat), console(&settings), snippet(){
        settings.beginGroup("Projects");
        QStringList keys = settings.allKeys();
        for(int i = 0; i < keys.size(); i++){
                QVariant path = settings.value(keys.at(i),QVariant());
                if(!path.isNull() && path.isValid()){
                        root.appendChild(new DProject(path.toString()));
                }
    }
        settings.endGroup();
        printf("Constructeur\n");
}
```

Partie 6 - Explorateur de fichier

Aperçu de l'explorateur





Partie 6 - Explorateur de fichier

- Renommage

Contrôle du nom donné par l'utilisateur

```
void ExplorerDelegate::setModelData(QWidget *editor, QAbstractItemModel *model, const QModelIndex &index) const{
    TreeItem *item = (TreeItem*)index.internalPointer();
    QLineEdit* edit = (QLineEdit*)editor;
    if(edit->text().size() <= 0 || edit->text() == item->label()){
        return;
    }
    if(typeid(*item) == typeid(DProject)){
       QDir dir(item->getPath());
       dir.cd("..");
        if(dir.exists(edit->text())){
            QMessageBox::information(nullptr,"erreur de renommage", "Un dossier portant ce nom existe déjà à cet emplacement");
            return;
        }
        QStyledItemDelegate::setModelData(editor,model,index);
        return;
    }

    if(item->parent()->exist(edit->text())){
        QMessageBox::information(nullptr,"erreur de renommage", "Un autre élément du même dossier porte déjà ce nom.");
        return;
    }
```

Partie 6 - Explorateur de fichier

- Renommage

Contrôle du nom donné par l'utilisateur

```
bool Model::setData(const QModelIndex &index, const QVariant &value, int role){
    if(!index.isValid()){
        return false;
    }

    if(role == Qt::EditRole){
        TreeItem* item = ((TreeItem*)index.internalPointer());
        if(!item->setLabel(value.toString())){
            return false;
        }
        emit dataChanged(index,index);
        item->save();
        return true;
    }

    return false;
```

Partie 6 - Explorateur de fichier

- Renommage

```
bool TreeItem::setLabel(QString label){
    QFileInfo fi(getPath());
    if(!QFile::rename(fi.path()+"/"+m_label,fi.path()+"/"+label)){
        QMessageBox::warning(nullptr,"erreur renommage","impossible de renommer le fichier "+fi.path()+"/"+label);
        return false;
    }
    this->m_label = label;
    emit rename(fi.path()+"/"+label);
    propagRename();
    return true;
}
```

Partie 7 - Editeur

- Coloration syntaxique

```
1  public class BinaryTree {
2      Node root;
3      public void addNode(int key, String name) {
4          Node newNode = new Node(key, name);
5          this.value = 10;
6          int[] tab = new int[10];
7          String str = "chaine de caractère";
8          /*
9          ceci est un commentaire
10         multiligne /*
11         //commentaire
12         */
13         if (root == null) {
14             root = newNode;
15         } else {
16             Node focusNode = root;
17             Node parent;
18             while (true) {
19                 parent = focusNode;
20                 if (key < focusNode.key) {
21                     focusNode = focusNode.leftChild;
22                     if (focusNode == null) {
23                         parent.leftChild = new Node;
24                         return;
25                     }
26                 } else {
27                     focusNode = focusNode.rightChild;
28                     if (focusNode == null) {
29                         parent.rightChild = new Node;
30                         return;
31                     }
32                 }
33             }
34         }
35     }
```

Partie 7 - Editeur
- Coloration syntaxique

Coloration des mots clés

```
keywordFormat.setFontItalic(true);
const QString keywordPatterns[] = {
    QStringLiteral("\\bchar\\b"), QStringLiteral("\\bclass\\b"), QStringLiteral("\\bconst\\b"),
    QStringLiteral("\\bdouble\\b"), QStringLiteral("\\benum\\b"), QStringLiteral("\\bexplicit\\b"),
    QStringLiteral("\\bString\\b"), QStringLiteral("\\bint\\b"),
    QStringLiteral("\\blong\\b"), QStringLiteral("\\bnamespace\\b"), QStringLiteral("\\boperator\\b"),
    QStringLiteral("\\bshort\\b"), QStringLiteral("\\bsignals\\b"), QStringLiteral("\\bsigned\\b"),
    QStringLiteral("\\bvoid\\b"), QStringLiteral("\\bvolatile\\b"), QStringLiteral("\\bboolean\\b")
};

for (const QString &pattern : keywordPatterns) {
    rule.pattern = QRegularExpression(pattern);
    rule.format = keywordFormat;
    highlightingRules.append(rule);
}
```

Partie 7 - L'éditeur
- Coloration syntaxique

Fonctionnement des commentaires

```
for(int i = startIndex; i < text.length();i++){
        if(text[i] == "/"){
            if(i+1 >= text.length()) continue;
            if(text[i+1] == "/"){
                setFormat(i,text.length(),commentFormat);
                return;
            }else if(text[i+1] == "*"){
                int start = i;
                bool isClose = false;
                for(i=i+2;i < text.length();i++){
                    if(text[i] == "*" && i+1 < text.length() && text[i+1]=="/"){
                        i++;
                        setFormat(start, i-start+1,commentFormat);
                        isClose = true;
                        break;
                    }
                }
                if(!isClose){
                    setFormat(start, text.length(),commentFormat);
                    setCurrentBlockState(1);
                    return;
                }
```

Partie 7 - L’éditeur

Auto indentation

```
if(event->key() == Qt::Key_Tab){
        this->insertPlainText(textTab);
        textTab = "";
    }
```

Auto complétion d’une parenthèse

```
if(event->key() == Qt::Key_ParenLeft){
        this->insertPlainText(")");
        this->moveCursor(QTextCursor::PreviousCharacter);
    }
```

Partie 7 - L’éditeur

Recherche des accolades/guillemets/crochets/parenthèses correspondants

```
QTextBlock next = this->textCursor().block();
    int count = 0, index = -1;
    while(next != this->document()->end()){
        QString str = next.text();
        for(; pos < str.length(); pos++){
            if(str[pos] == left)
                count++;
            else if(str[pos] == right)
                count--;
            if(count == 0){
                index = next.position() + pos;
                break;
            }
        }
        if(count == 0) break;
        next = next.next();
        pos = 0;
    }
```

Partie 7 - L’éditeur
- Controller de l’éditeur

Appel à chaque ouverture d’éditeur

```
void Controller::openEditor(const QModelIndex &index){
    TreeItem *item = (TreeItem*)index.internalPointer();
    if(typeid(*item) == typeid(TreeItem) || typeid(*item) == typeid(DJavaFile) ){
        QTabWidget *tab = this->fen->getCentral();
        for(int i = 0; i < tab->count();i++){
            if(((DCodeEditor*)tab->widget(i))->getController().isItem(item)){
                tab->setCurrentWidget(tab->widget(i));
                return;
            }
        }
        DCodeEditor *edit = new DCodeEditor(item,tab);
        tab->addTab(edit,item->label());
        tab->setCurrentWidget(edit);
    }
}
```

Merci de nous avoir écouté