2018 －2019 学年第 二 学期

组号：G02



霸笔笔记APP开发

代码清单



实验课程名称 软件工程导论

专 业 班 级 软工1701 软工1702

学 号 31701401 31701257 31701371

学 生 姓 名 高兴欣 王晨旭 倪嘉玲

实验指导老师 杨枨

|  |  |  |
| --- | --- | --- |
| 文件状态： | 文件标识： | SE-2019-G02 |
| [ ]:草稿 | 当前版本： | 1.5 |
| [ ]:正式发布 | 作 者： | 高兴欣、王晨旭、倪嘉玲 |
| [ ]:正在修改 | 完成日期： | 2019年5月25日 |
| [ √ ]:正在修改 | 完成日期： | 2019年5月25日 |

# 版本历史

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **版本/状态** | **作者** | **参与者** | **起止日期** | **备注** |
| 1.0 | 高兴欣、王晨旭、倪嘉玲 | 高兴欣、王晨旭、倪嘉玲 | 2019-05-17  至  2019-05-25 | 初步完成代码清单 |
| 1.1 | 高兴欣、王晨旭、倪嘉玲 | 高兴欣、王晨旭、倪嘉玲 | 2019-05-31  至  2019-06-13 | 修订代码清单 |
| 1.2 | 高兴欣、王晨旭、倪嘉玲 | 高兴欣、王晨旭、倪嘉玲 | 2019-06-14  至  2019-06-21 | 修订代码清单 |

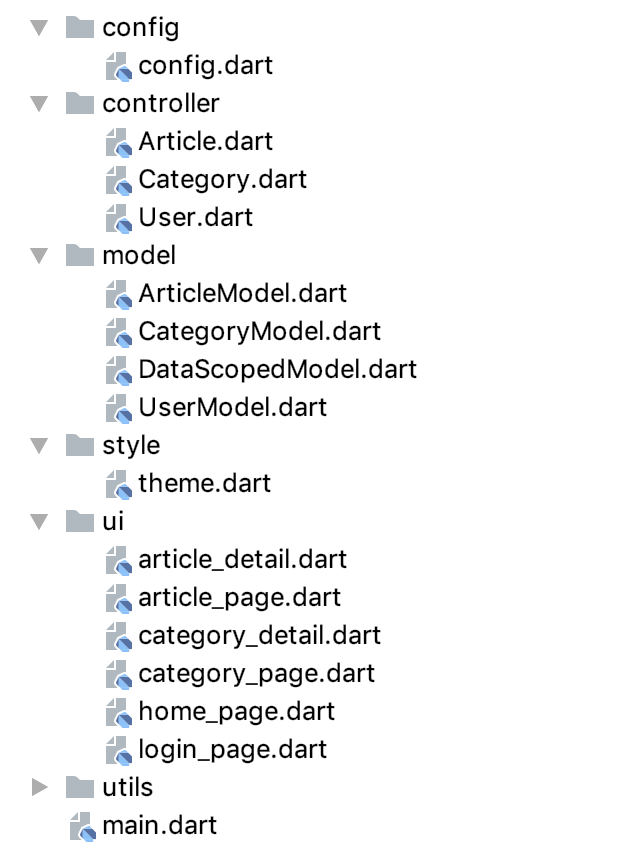
目录

[版本历史 2](#_Toc11421153)

[1. 服务器端代码 7](#_Toc11421154)

[2. 前端代码 36](#_Toc11421155)

**一、设计说明**

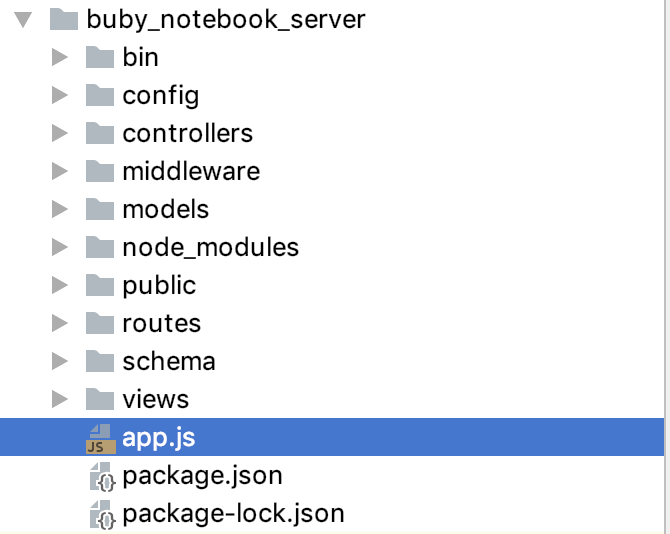


config里存放着各种配置字符串

controller里存放着各种操作文件

model里存放着数据的模型

style里存放着app的主题数据

ui里存放着ui设计文件

controller下存放着对数据的操作文件

model下存放着数据模型

routes下存放着接口数据

schema下存放着各种数据的存放模式

views存放着主文件

**二、源代码清单**

1. 服务器端代码

const sequelize = **new** Sequelize(**'buby\_notebook'**,**'root'**,**'zhzh.xyz'**,{  
 host:**'localhost'**,  
 dialect:**'mysql'**,  
 operatorsAliases:**false**,  
 dialectOptions:{  
 *//字符集* charset:**'utf8mb4'**,  
 supportBigNumbers: **true**,  
 bigNumberStrings: **true** },  
 pool:{  
 max: 5,  
 min: 0,  
 acquire: 30000,  
 idle: 10000  
 },  
 timezone: **'+08:00'** *//东八时区*});  
  
module.exports = {  
 sequelize  
};

const ArticleModel = require(**'../models/ArticleModel'**)  
const CategoryModel = require(**'../models/CategoryModel'**)  
  
class Article {  
 */\*\*  
 \* 创建笔记  
 \* @param ctx title 笔记标题  
 \* @param ctx categoryId 分类ID  
 \* @param ctx content 笔记内容  
 \*  
 \* @returns 成功创建笔记返回笔记详情数据，失败返回错误信息  
 \*/* static async create(ctx) {  
 *// 接收参数* let {title, categoryId, content} = ctx.request.body;  
 const user\_id = ctx.user.id;  
  
 let params = {  
 title,  
 user\_id,  
 categoryId,  
 content  
 }  
  
 *// 检测参数是否存在为空* let errors = [];  
 **for** (let item **in** params) {  
 **if** (params[item] === undefined) {  
 let index = errors.length + 1;  
 errors.push(**"错误"** + index + **": 参数: "** + item + **"不能为空"**)  
 }  
 }  
  
 **if** (errors.length > 0) {  
 ctx.response.status = 422;  
 ctx.body = {  
 code: 422,  
 message: errors  
 }  
  
 **return false**;  
 }  
  
 **try** {  
 *// 查询分类是否存在* let detail = await CategoryModel.detail(categoryId, user\_id);  
  
 **if** (!detail) {  
 ctx.response.status = 412;  
 ctx.body = {  
 code: 412,  
 message: `分类ID：${categoryId}不存在！`  
 }  
  
 **return false**;  
 }  
  
 *// 创建笔记* const {id} = await ArticleModel.create(params);  
 *// 查询笔记* const data = await ArticleModel.detail(id, user\_id);  
  
 ctx.response.status = 200;  
 ctx.body = {  
 code: 200,  
 message: `创建笔记成功`,  
 data  
 }  
 } **catch** (err) {  
 ctx.response.status = 500;  
 ctx.body = {  
 code: 500,  
 message: `创建笔记失败`,  
 data: err  
 }  
 }  
 }  
  
 */\*\*  
 \* 删除笔记数据  
 \*/* static async **delete**(ctx) {  
 let {id} = ctx.params;  
  
 *// 检测是否传入ID* **if** (!id) {  
 ctx.response.status = 412;  
 ctx.body = {  
 code: 412,  
 message: `ID不能为空`  
 }  
  
 **return false**;  
 }  
  
 **if** (isNaN(id)) {  
 ctx.response.status = 412;  
 ctx.body = {  
 code: 412,  
 message: `请传入正确的ID`  
 }  
  
 **return false**;  
 }  
  
 **try** {  
 const user\_id = ctx.user.id;  
 await ArticleModel.**delete**(id, user\_id);  
 ctx.response.status = 200;  
 ctx.body = {  
 code: 200,  
 message: `删除成功`  
 }  
 } **catch** (err) {  
 ctx.response.status = 500;  
 ctx.body = {  
 code: 500,  
 message: `删除失败`,  
 data: err  
 }  
 }  
 }  
  
 */\*\*  
 \* 修改笔记  
 \*/* static async update(ctx) {  
 let {id} = ctx.params;  
  
 *// 检测是否传入ID* **if** (!id) {  
 ctx.response.status = 412;  
 ctx.body = {  
 code: 412,  
 message: `ID不能为空`  
 }  
  
 **return false**;  
 }  
  
 **if** (isNaN(id)) {  
 ctx.response.status = 412;  
 ctx.body = {  
 code: 412,  
 message: `请传入正确的ID`  
 }  
  
 **return false**;  
 }  
  
 *// 接收参数* let {title, categoryId, content} = ctx.request.body;  
  
 let params = {  
 title,  
 categoryId,  
 content  
 }  
  
 **try** {  
 const user\_id = ctx.user.id;  
 await ArticleModel.update(id, user\_id, params);  
 let data = await ArticleModel.detail(id, user\_id);  
 **if** (data) {  
 ctx.response.status = 200;  
 ctx.body = {  
 code: 200,  
 message: `更新笔记成功`,  
 data  
 }  
 } **else** {  
 ctx.response.status = 403;  
 ctx.body = {  
 code: 403,  
 message: `找不到笔记`,  
 data  
 }  
 }  
 } **catch** (err) {  
 ctx.response.status = 500;  
 ctx.body = {  
 code: 500,  
 message: `更新笔记失败`,  
 data: err  
 }  
 }  
 }  
  
 */\*\*  
 \* 获取笔记列表  
 \*/* static async list(ctx) {  
 const user\_id = ctx.user.id;  
  
 **try** {  
 const data = await ArticleModel.list(user\_id);  
 **if** (data) {  
 ctx.response.status = 200;  
 ctx.body = {  
 code: 200,  
 message: `获取笔记列表成功！`,  
 data  
 }  
 } **else** {  
 ctx.response.status = 403;  
 ctx.body = {  
 code: 403,  
 message: `查询为空`,  
 data  
 }  
 }  
 } **catch** (err) {  
 ctx.response.status = 500;  
 ctx.body = {  
 code: 500,  
 message: `获取笔记列表失败`,  
 data: err  
 }  
 }  
 }  
  
 */\*\*  
 \* 查询文章详情  
 \* @param ctx id 文章ID  
 \*  
 \* @returns 文章的详情  
 \*/* static async detail(ctx) {  
 *// 文章ID* let {id} = ctx.params;  
  
 *// 检测是否传入ID* **if** (!id) {  
 ctx.response.status = 412;  
 ctx.body = {  
 code: 412,  
 message: `文章ID为空，请传入查询的文章ID`  
 }  
  
 **return false**;  
 }  
  
 **if** (isNaN(id)) {  
 ctx.response.status = 412;  
 ctx.body = {  
 code: 412,  
 message: `请传入正确的文章ID`  
 }  
  
 **return false**;  
 }  
  
 **try** {  
 const user\_id = ctx.user.id;  
 let data = await ArticleModel.detail(id, user\_id);  
  
 **if** (data) {  
 ctx.response.status = 200;  
 ctx.body = {  
 code: 200,  
 message: `查询文章成功`,  
 data  
 }  
 } **else** {  
 ctx.response.status = 403;  
 ctx.body = {  
 code: 403,  
 message: `查询不到`  
 }  
 }  
 } **catch** (err) {  
 ctx.response.status = 500;  
 ctx.body = {  
 code: 500,  
 message: `查询文章失败`,  
 data: err  
 }  
 }  
 }  
}  
  
module.exports = Article

const CategoryModel = require(**'../models/CategoryModel'**)  
  
class Category {  
 */\*\*  
 \* 创建分类  
 \*/* static async create(ctx) {  
 let {name} = ctx.request.body;  
 const user\_id = ctx.user.id;  
  
 let params = {  
 name,  
 user\_id  
 }  
  
 *// 检测参数是否存在为空* let errors = [];  
 **for** (let item **in** params) {  
 **if** (params[item] === undefined) {  
 let index = errors.length + 1;  
 errors.push(**"错误"** + index + **": 参数: "** + item + **"不能为空"**)  
 }  
 }  
 **if** (errors.length > 0) {  
 ctx.response.status = 412;  
 ctx.body = {  
 code: 412,  
 message: errors  
 }  
 **return false**;  
 }  
  
 **try** {  
 const {id} = await CategoryModel.create(params);  
 const data = await CategoryModel.detail(id, user\_id);  
  
 ctx.response.status = 200;  
 ctx.body = {  
 code: 200,  
 message: `创建分类成功`,  
 data  
 }  
 } **catch** (err) {  
 ctx.response.status = 500;  
 ctx.body = {  
 code: 500,  
 message: `创建分类失败`,  
 data: err  
 }  
 }  
  
 }  
  
 */\*\*  
 \* 获取分类列表  
 \*/* static async list(ctx) {  
 **try** {  
 const user\_id = ctx.user.id;  
 let data = await CategoryModel.list(user\_id);  
  
 ctx.response.status = 200;  
 ctx.body = {  
 code: 200,  
 message: `获取分类列表成功！`,  
 data  
 }  
 } **catch** (err) {  
 ctx.response.status = 500;  
 ctx.body = {  
 code: 500,  
 message: `获取分类列表失败`,  
 data: err  
 }  
 }  
 }  
  
 */\*\*  
 \* 删除分类数据  
 \* @param ctx  
 \*  
 \* @returns 删除成功返回true，失败返回错误信息  
 \*/* static async **delete**(ctx) {  
 let {id} = ctx.params;  
  
 *// 检测是否传入ID* **if** (!id) {  
 ctx.response.status = 412;  
 ctx.body = {  
 code: 412,  
 message: `ID不能为空`  
 }  
  
 **return false**;  
 }  
  
 **if** (isNaN(id)) {  
 ctx.response.status = 412;  
 ctx.body = {  
 code: 412,  
 message: `请传入正确的ID`  
 }  
  
 **return false**;  
 }  
  
 **try** {  
 const user\_id = ctx.user.id;  
 let data = await CategoryModel.detail(id, user\_id);  
  
 **if** (!data) {  
 ctx.response.status = 412;  
 ctx.body = {  
 code: 412,  
 message: `请传入正确的ID`  
 }  
  
 **return false**;  
 } **else**{  
 *// 检测分类下是否有文章关联，如果有文章关联则报出不能删除错误* let hasArticle = await CategoryModel.article(id, user\_id);  
  
 **if** (hasArticle && hasArticle[0].articles.length > 0) {  
 ctx.response.status = 403;  
 ctx.body = {  
 code: 403,  
 message: `此分类下有关联文章，不能删除`  
 }  
  
 **return false**;  
 } **else** {  
 await CategoryModel.**delete**(id, user\_id);  
  
 ctx.response.status = 200;  
 ctx.body = {  
 code: 200,  
 message: `删除成功`  
 }  
 }  
 }  
 } **catch** (err) {  
 ctx.response.status = 500;  
 ctx.body = {  
 code: 500,  
 message: `删除失败`,  
 data: err  
 }  
 }  
  
 }  
  
 */\*\*  
 \* 更新分类数据  
 \* @param ctx id 分类ID  
 \* @param ctx name 分类名称  
 \*  
 \* @returns 更新成功返回更新后的数据，失败返回错误信息  
 \*/* static async update(ctx) {  
 let {id} = ctx.params;  
  
 *// 检测是否传入ID* **if** (!id) {  
 ctx.response.status = 412;  
 ctx.body = {  
 code: 412,  
 message: `ID不能为空`  
 }  
  
 **return false**;  
 }  
  
 **if** (isNaN(id)) {  
 ctx.response.status = 412;  
 ctx.body = {  
 code: 412,  
 message: `请传入正确的ID`  
 }  
  
 **return false**;  
 }  
  
 let {name} = ctx.request.body;  
  
 **if** (!name) {  
 ctx.response.status = 412;  
 ctx.body = {  
 code: 412,  
 message: `name不能为空`  
 }  
  
 **return false**;  
 }  
  
 let params = {  
 name,  
 }  
  
 **try** {  
 const user\_id = ctx.user.id;  
  
 await CategoryModel.update(id, user\_id, params);  
 let data = await CategoryModel.detail(id, user\_id);  
  
 ctx.response.status = 200;  
 ctx.body = {  
 code: 200,  
 message: `更新分类成功`,  
 data  
 }  
  
 } **catch** (err) {  
 ctx.response.status = 500;  
 ctx.body = {  
 code: 500,  
 message: `更新失败`,  
 data: err  
 }  
 }  
 }  
  
 */\*\*  
 \* 查询ID分类下的所有文章  
 \*/* static async article(ctx) {  
 let {id} = ctx.params;  
  
 *// 检测是否传入ID* **if** (!id) {  
 ctx.response.status = 412;  
 ctx.body = {  
 code: 412,  
 message: `分类ID不能为空`  
 }  
  
 **return false**;  
 }  
  
 **if** (isNaN(id)) {  
 ctx.response.status = 412;  
 ctx.body = {  
 code: 412,  
 message: `请传入正确的分类ID`  
 }  
  
 **return false**;  
 }  
  
 **try** {  
 const user\_id = ctx.user.id;   
 const data = await CategoryModel.article(id, user\_id);  
  
 **if** (data) {  
 ctx.response.status = 200;  
 ctx.body = data  
 **return true**;  
 } **else** {  
 ctx.response.status = 403;  
 ctx.body = {  
 code: 403,  
 message: `没有获取到数据`,  
 data  
 }  
 **return false**;  
 }  
 } **catch** (err) {  
 ctx.response.status = 500;  
 ctx.body = {  
 code: 500,  
 message: `获取文章失败`,  
 data: err  
 }  
 }  
 }  
}  
  
module.exports = Category

const UserModel = require(**'../models/UserModel'**);  
const jwt = require(**'jsonwebtoken'**);  
const secret = require(**'../config/secret.json'**);  
const bcrypt = require(**'bcryptjs'**);  
const util = require(**'util'**);  
const verify = util.promisify(jwt.verify);  
  
class User {  
 static async create(ctx) {  
 let {email, username, password} = ctx.request.body;  
  
 let params = {  
 email,  
 username,  
 password  
 }  
  
 let errors = [];  
 **for** (let item **in** params) {  
 **if** (params[item] === undefined) {  
 let index = errors.length + 1;  
 errors.push(**"错误"** + index + **": 参数: "** + item + **"不能为空"**)  
 }  
 }  
  
 **if** (errors.length > 0) {  
 ctx.response.status = 412;  
 ctx.body = {  
 code: 412,  
 message: errors  
 }  
 **return false**;  
 }  
  
 *// 查询用户名是否重复* const existUser = await UserModel.email(params.email)  
  
 **if** (existUser) {  
 ctx.response.status = 403;  
 ctx.body = {  
 code: 403,  
 message: **"用户已经存在"** }  
  
 } **else** {  
  
 **try** {  
 *// 加密密码* const salt = bcrypt.genSaltSync();  
 const hash = bcrypt.hashSync(params.password, salt);  
 params.password = hash;  
  
 *// 创建用户* await UserModel.create(params);  
 const newUser = await UserModel.email(params.email)  
  
 *// 签发token* const userToken = {  
 username: newUser.username,  
 email: newUser.email,  
 id: newUser.id  
 }  
  
 *// 储存token失效有效期1小时* const token = jwt.sign(userToken, secret.sign, {expiresIn: **'10h'**});  
  
 ctx.response.status = 200;  
 ctx.body = {  
 code: 200,  
 message: `创建用户成功`,  
 data: token  
 }  
 } **catch** (err) {  
 ctx.response.status = 500;  
 ctx.body = {  
 code: 500,  
 message: err  
 }  
 }  
 }  
  
 }  
  
 */\*\*  
 \* 查询用户信息  
 \*/* static async info(ctx) {  
 *// 获取jwt* const token = ctx.header.authorization;  
  
 **if** (!token) {  
 ctx.response.status = 403;  
 ctx.body = {  
 code: 403,  
 message: **"Headers Token不能为空"** }  
 }  
  
 let payload  
 **try** {  
 *// 解密payload，获取用户名和ID* payload = await verify(token.split(**' '**)[1], secret.sign)  
 const user = {  
 id: payload.id,  
 email: payload.email,  
 username: payload.username  
 }  
  
 ctx.response.status = 200;  
 ctx.body = {  
 code: 200,  
 message: **'查询成功！'**,  
 data: user  
 }  
  
 } **catch** (err) {  
 ctx.response.status = 500;  
 ctx.body = {  
 code: 500,  
 message: err  
 }  
 }  
 }  
  
 */\*\*  
 \* 登录  
 \*/* static async login(ctx) {  
 const {email, password} = ctx.request.body  
 *// 查询用户* const userDetail = await UserModel.email(email)  
  
 **if** (!userDetail) {  
 ctx.response.status = 403;  
 ctx.body = {  
 code: 403,  
 message: **"用户不存在"** }  
  
 **return false**;  
 }  
  
  
 *// 判断前端传递的用户密码是否与数据库密码一致* **if** (bcrypt.compareSync(password, userDetail.password)) {  
 *// 用户token* const userToken = {  
 username: userDetail.username,  
 id: userDetail.id,  
 email: userDetail.email,  
 }  
 *// 签发token* const token = jwt.sign(userToken, secret.sign, {expiresIn: **'10h'**});  
  
 ctx.response.status = 200;  
 ctx.body = {  
 code: 200,  
 message: **"登录成功"**,  
 data: {  
 id: userDetail.id,  
 username: userDetail.username,  
 email: userDetail.email,  
 token: token  
 }  
 }  
  
 } **else** {  
 ctx.response.status = 401;  
 ctx.body = {  
 code: 401,  
 message: **"用户名或密码错误"** }  
 }  
 }  
}  
  
module.exports = User

module.exports = [  
 *// 文章列表* **"/api/v1/user/register"**,  
 **"/api/v1/user/login"**]

const jwt = require(**'jsonwebtoken'**)  
const secret = require(**'../config/secret'**)  
const util = require(**'util'**)  
const verify = util.promisify(jwt.verify)  
const JWTPath = require(**'./JWTPath'**)  
  
  
*/\*\*  
 \* 判断token是否可用  
 \*/*module.exports = **function** () {  
 **return** async **function** (ctx, next) {  
  
 *// 检测过滤的路由就不做解析JWT了* **if** (JWTPath.find(item => item === ctx.request.url)) {  
 await next()  
  
 **return false**;  
 }  
  
 **try** {  
 *// 获取jwt* const token = ctx.header.authorization  
 **if** (token) {  
 let payload  
 **try** {  
 *// 解密payload，获取用户名和ID* payload = await verify(token.split(**' '**)[1], secret.sign)  
 ctx.user = {  
 email: payload.email,  
 id: payload.id  
 }  
  
 } **catch** (err) {  
 ctx.status = 401;  
 ctx.body = {  
 code: 401,  
 message: **"Token身份无效!"** }  
 }  
 }  
 await next()  
  
 } **catch** (err) {  
 **if** (err.status === 401) {  
 ctx.status = 401;  
 ctx.body = {  
 code: 401,  
 err  
 }  
 } **else** {  
 ctx.status = 500;  
 ctx.body = {  
 code: 500,  
 err  
 }  
 }  
  
 }  
 }  
}

const db = require(**'../config/db'**);  
const Sequelize = db.sequelize;  
const Op = Sequelize.Op;  
const Article = Sequelize.import(**'../schema/article'**);  
const Category = Sequelize.import(**'../schema/category'**);  
  
Category.hasMany(Article); *// 将会添加 categoryId 到 ArticleModel 模型*Article.belongsTo(Category, {foreignKey: **'categoryId'**});  
  
Article.sync({force: **false**});  
  
class ArticleModel {  
 */\*\*  
 \* 创建笔记  
 \* @param data 创建笔记的参数  
 \* @returns {Promise<void>}  
 \*/* static async create(data) {  
 **return** await Article.create(data);  
 }  
  
 */\*\*  
 \* 获取笔记详情数据  
 \*/* static async detail(id, user\_id) {  
 **return** await Article.findOne({  
 where: {  
 id,  
 user\_id  
 },  
 include: [{  
 model: Category,  
 where: {categoryId: Sequelize.col(**'article.categoryId'**)}  
 }],  
 attributes: {exclude: [**'user\_id'**, **'created\_at'**, **'updated\_at'**]}  
 })  
 }  
  
 */\*\*  
 \* 更新笔记数据  
 \*/* static async update(id, user\_id, data) {  
 **return** await Article.update(data, {  
 where: {  
 id,  
 user\_id  
 },  
 fields: [**'title'**, **'categoryId'**, **'content'**]  
 });  
 }  
  
 */\*\*  
 \* 删除笔记数据  
 \*/* static async **delete**(id, user\_id) {  
 await Article.destroy({  
 where: {  
 id,  
 user\_id  
 }  
 })  
 **return true**;  
 }  
  
 */\*\*  
 \* 获取笔记列表  
 \* @returns {Promise<\*>}  
 \*/* static async list(user\_id) {  
 **return** await Article.findAll({  
 where: {  
 user\_id  
 },  
 attributes: [**'id'**, **'title'**, **'content'**, **'categoryId'**, **'updated\_at'**],  
 });  
 }  
}  
  
module.exports = ArticleModel

const db = require(**'../config/db'**);  
const Sequelize = db.sequelize;  
const Category = Sequelize.import(**'../schema/category'**);  
const Article = Sequelize.import(**'../schema/article'**);  
  
Category.sync({force: **false**});  
  
class CategoryModel {  
 */\*\*  
 \* 创建分类  
 \*/* static async create(data) {  
 **return** await Category.create(data)  
 }  
  
 */\*\*  
 \* 更新分类数据  
 \*/* static async update(id, user\_id, data) {  
 await Category.update(data, {  
 where: {  
 id,  
 user\_id  
 },  
 fields: [**'name'**]  
 });  
 **return true** }  
  
 */\*\*  
 \* 获取分类列表  
 \*/* static async list(user\_id) {  
 **return** await Category.findAll({  
 where: {  
 user\_id  
 },  
 attributes: [**'id'**, **'name'**],  
 })  
 }  
  
 */\*\*  
 \* 删除分类  
 \* @param id  
 \* @returns {Promise.<boolean>}  
 \*/* static async **delete**(id,user\_id) {  
 await Category.destroy({  
 where: {  
 id,  
 user\_id  
 }  
 })  
 **return true** }  
  
 */\*\*  
 \* 获取分类详情数据  
 \*/* static async detail(id, user\_id) {  
 **return** await Category.findOne({  
 where: {  
 id,  
 user\_id  
 },  
 })  
 }  
  
 */\*\*  
 \* 获取当前分类下所有文章  
 \*/* static async article(id, user\_id) {  
 **return** await Category.findAll({  
 where: {  
 id,  
 user\_id  
 },  
 include: [{  
 model: Article,  
 attributes: {exclude: [**'createdAt'**, **'updatedAt'**, **'user\_id'**, **'categoryId'**]}  
 }],  
 attributes: {exclude: [**'createdAt'**, **'updatedAt'**, **'user\_id'**]}  
 })  
 }  
}  
  
module.exports = CategoryModel

const db = require(**'../config/db'**)  
const Sequelize = db.sequelize  
const User = Sequelize.import(**'../schema/user.js'**)  
  
User.sync({force: **false**});  
  
class UserModel {  
 static async create(user) {  
 let {email, username, password} = user;  
  
 await User.create({  
 email,  
 username,  
 password  
 })  
  
 **return true** }  
  
 static async email(email) {  
 **return** await User.findOne({  
 where: {  
 email  
 }  
 })  
 }  
}  
  
module.exports = UserModel

const Router = require(**'koa-router'**)  
const User = require(**'../controllers/User'**)  
const Category = require(**'../controllers/Category'**)  
const Article = require(**'../controllers/Article'**)  
  
const Routers = **new** Router({  
 prefix: **'/api'**})  
  
*/\*\*  
 \* 用户接口  
 \*/  
// 用户注册*Routers.post(**'/user/register'**, User.create);  
*// 用户登录*Routers.post(**'/user/login'**, User.login);  
*// 获取用户信息*Routers.get(**'/user/info'**, User.info);  
  
*/\*\*  
 \* 分类接口  
 \*/  
// 创建分类*Routers.post(**'/category/create'**, Category.create);  
*// 删除分类*Routers.**delete**(**'/category/delete/:id'**, Category.**delete**);  
*// 更改分类*Routers.put(**'/category/update/:id'**, Category.update);  
*// 获取分类列表*Routers.get(**'/category/list'**, Category.list);  
*// 查询分类ID下的所有笔记列表*Routers.get(**'/category/article/:id'**, Category.article);  
  
*/\*\*  
 \* 笔记接口  
 \*/  
// 创建笔记*Routers.post(**'/article/create'**, Article.create);  
*// 删除笔记*Routers.**delete**(**'/article/delete/:id'**, Article.**delete**);  
*// 更改笔记*Routers.put(**'/article/update/:id'**, Article.update);  
*// 获取获取列表*Routers.get(**'/article/list'**, Article.list);  
*// 获取笔记详情*Routers.get(**'/article/detail/:id'**, Article.detail);  
  
module.exports = Routers

const moment = require(**'moment'**);  
module.exports = **function** (sequelize, DataTypes) {  
 **return** sequelize.define(**'article'**, {  
 *// 笔记ID* id: {  
 type: DataTypes.INTEGER.UNSIGNED,  
 primaryKey: **true**,  
 allowNull: **true**,  
 autoIncrement: **true**,  
 },  
 *// 笔记标题* title: {  
 type: DataTypes.STRING(100),  
 allowNull: **false**,  
 field: **'title'**,  
 },  
 *// 笔记作者* user\_id: {  
 type: DataTypes.INTEGER.UNSIGNED,  
 field: **'user\_id'**,  
 allowNull: **false** },  
 *// 笔记内容* content: {  
 type: DataTypes.TEXT,  
 allowNull: **false**,  
 field: **'content'** },  
 createdAt: {  
 type: DataTypes.DATE,  
 field: **'created\_at'**,  
 get() {  
 **return** moment(**this**.getDataValue(**'createdAt'**)).format(**'YYYY-MM-DD'**);  
 }  
 },  
 updatedAt: {  
 type: DataTypes.DATE,  
 field: **'updated\_at'**,  
 get() {  
 **return** moment(**this**.getDataValue(**'updatedAt'**)).format(**'YYYY-MM-DD'**);  
 }  
 }  
 }, {  
 freezeTableName: **false** })  
  
}

const moment = require(**'moment'**);  
module.exports = **function** (sequelize, DataTypes) {  
 **return** sequelize.define(**'category'**, {  
 id: {  
 type: DataTypes.INTEGER.UNSIGNED,  
 primaryKey: **true**,  
 allowNull: **true**,  
 autoIncrement: **true**,  
 },  
 *// 分类名字* name: {  
 type: DataTypes.STRING(30),  
 field: **'name'**,  
 allowNull: **false** },  
 user\_id :{  
 type: DataTypes.INTEGER.UNSIGNED,  
 field: **'user\_id'**,  
 allowNull: **false** },  
 createdAt: {  
 type: DataTypes.DATE,  
 field: **'created\_at'**,  
 get() {  
 **return** moment(**this**.getDataValue(**'createdAt'**)).format(**'YYYY-MM-DD'**);  
 }  
 },  
 updatedAt: {  
 field: **'updated\_at'**,  
 type: DataTypes.DATE,  
 get() {  
 **return** moment(**this**.getDataValue(**'updatedAt'**)).format(**'YYYY-MM-DD'**);  
 }  
 }  
 }, {  
 *// 如果为 true 则表的名称和 model 相同，即 category  
 // 为 false MySQL创建的表名称会是复数* freezeTableName: **false** })  
}

const moment = require(**'moment'**);  
  
module.exports = **function** (sequelize, DataTypes) {  
 **return** sequelize.define(**'user'**,{  
 id: {  
 type: DataTypes.INTEGER.UNSIGNED,  
 allowNull: **false**,  
 primaryKey: **true**,  
 autoIncrement: **true** },  
 email: {  
 type: DataTypes.STRING(100),  
 allowNull: **false**,  
 field: **'email'**,  
 },  
 username: {  
 type: DataTypes.STRING(100),  
 field: **'username'**,  
 allowNull: **false** },  
 password: {  
 type: DataTypes.STRING(255),  
 field: **'password'**,  
 allowNull: **false** },  
 createdAt: {  
 type: DataTypes.DATE,  
 field: **'created\_id'**,  
 get() {  
 **return** moment(**this**.getDataValue(**'createdAt'**)).format(**'YYYY-MM-DD'**);  
 }  
 },  
 updatedAt: {  
 field: **'updated\_id'**,  
 type: DataTypes.DATE,  
 get() {  
 **return** moment(**this**.getDataValue(**'updatedAt'**)).format(**'YYYY-MM-DD'**);  
 }  
 }  
 },{  
 freezeTableName: **false** });  
}

const Koa = require(**'koa'**)  
const app = **new** Koa()  
const views = require(**'koa-views'**)  
const json = require(**'koa-json'**)  
const onerror = require(**'koa-onerror'**)  
const bodyparser = require(**'koa-bodyparser'**)  
const jwt = require(**'koa-jwt'**)  
const logger = require(**'koa-logger'**)  
  
const Routers = require(**'./routes/index'**)  
const secret = require(**'./config/secret'**)  
const JWTToken = require(**'./middleware/JWTToken'**)  
  
*// error handler*onerror(app)  
  
*// middlewares*app.use(JWTToken())  
  
*// 此接口列表，过滤不用jwt验证*app.use(jwt({secret: secret.sign}).unless({  
 path: [  
 *// 注册* /^\/api\/user\/register/,  
 *// 登录* /^\/api\/user\/login/  
 ]  
}))  
  
*// middlewares*app.use(bodyparser({  
 enableTypes:[**'json'**, **'form'**, **'text'**]  
}))  
app.use(json())  
app.use(logger())  
app.use(require(**'koa-static'**)(\_\_dirname + **'/public'**))  
  
app.use(views(\_\_dirname + **'/views'**, {  
 extension: **'pug'**}))  
  
*// logger*app.use(async (ctx, next) => {  
 const start = **new** Date()  
 await next()  
 const ms = **new** Date() - start  
 console.log(`${ctx.method} ${ctx.url} - ${ms}ms`)  
})  
  
*// routes*app.use(Routers.routes(), Routers.allowedMethods())  
  
*// error-handling*app.on(**'error'**, (err, ctx) => {  
 console.error(**'server error'**, err, ctx)  
});  
  
module.exports = app

1. 前端代码

**class** Configs {  
 **const** Configs();  
  
*// static String baseUrl = "http://192.168.123.34:3000/api";* **static** String *baseUrl* = **"http://47.101.45.77:3000/api"**;  
  
*// user* **static** String *userApi* = **"/user"**;  
 **static** String *loginUrl* = *userApi* + **"/login"**;  
 **static** String *registerUrl* = *userApi* + **"/register"**;  
 **static** String *infoUrl* = *userApi* + **"/info"**;  
  
*// category* **static** String *categoryApi* = **"/category"**;  
 **static** String *createCategoryUrl* = *categoryApi* + **"/create"**;  
 **static** String *deleteCategoryUrl* = *categoryApi* + **"/delete/"**;  
 **static** String *updateCategoryUrl* = *categoryApi* + **"/update/"**;  
 **static** String *listCategoryUrl* = *categoryApi* + **"/list"**;  
  
 **static** String *articleApi* = **"/article"**;  
 **static** String *createArticleUrl* = *articleApi* + **"/create"**;  
 **static** String *deleteArticleUrl* = *articleApi* + **"/delete/"**;  
 **static** String *updateArticleUrl* = *articleApi* + **"/update/"**;  
 **static** String *listArticleUrl* = *articleApi* + **"/list"**;  
}

**import 'dart:async'**;  
**import 'package:dio/dio.dart'**;  
**import '../config/config.dart'**;  
**import '../utils/HttpUtils.dart'**;  
**import '../model/ArticleModel.dart'**;  
  
**class** Article {  
 **static** Future<int> *create*(  
 String title, String content, int categoryId) **async** {  
 int code = 200;  
 **try** {  
 **await** HttpUtils.*getTokenInstance*().post(Configs.*createArticleUrl*,  
 data: {**"title"**: title, **"content"**: content, **"categoryId"**: categoryId});  
 } **on** DioError **catch** (e) {  
 **if** (e.**response** != **null** && e.**response**.**statusCode** == 401) {  
 HttpUtils.*clearTokenDio*();  
 }  
 code = e.**response**.**statusCode**;  
 }  
 **return** code;  
 }  
  
 **static** Future<int> *delete*(int id) **async** {  
 int code = 200;  
 **try** {  
 **await** HttpUtils.*getTokenInstance*()  
 .delete(Configs.*deleteArticleUrl* + id.toString());  
 } **on** DioError **catch** (e) {  
 **if** (e.**response** != **null** && e.**response**.**statusCode** == 401) {  
 HttpUtils.*clearTokenDio*();  
 }  
 code = e.**response**.**statusCode**;  
 }  
 **return** code;  
 }  
  
 **static** Future<int> *update*(  
 int id, String title, String content, int categoryId) **async** {  
 int code = 200;  
 **try** {  
 **await** HttpUtils.*getTokenInstance*().put(  
 Configs.*updateArticleUrl* + id.toString(),  
 data: {**"title"**: title, **"content"**: content, **"categoryId"**: categoryId});  
 } **on** DioError **catch** (e) {  
 **if** (e.**response** != **null** && e.**response**.**statusCode** == 401) {  
 HttpUtils.*clearTokenDio*();  
 }  
 code = e.**response**.**statusCode**;  
 }  
 **return** code;  
 }  
  
 **static** Future<List<ArticleModel>> *list*() **async** {  
 List<ArticleModel> list = **new** List<ArticleModel>();  
 **try** {  
 Response response =  
 **await** HttpUtils.*getTokenInstance*().get(Configs.*listArticleUrl*);  
 **var** data = response.**data**[**'data'**];  
 **for** (**var** value **in** data) {  
 list.add(ArticleModel.fromJson(value));  
 }  
 } **on** DioError **catch** (e) {  
 **if** (e.**response** != **null** && e.**response**.**statusCode** == 401) {  
 HttpUtils.*clearTokenDio*();  
 }  
 list = **null**;  
 }  
 **return** list;  
 }  
}

**import 'dart:async'**;  
**import 'package:dio/dio.dart'**;  
**import '../config/config.dart'**;  
**import '../model/UserModel.dart'**;  
**import '../utils/HttpUtils.dart'**;  
  
**class** User {  
 **static** Future<int> *login*(String email, String password) **async**{  
 int code = 200;  
 **try** {  
 Response response = **await** HttpUtils.*getInstance*().post(Configs.*loginUrl*,  
 data: {**"email"**: email, **"password"**: password});  
 code = response.**statusCode**;  
 UserModel.*currentLoginUser* = **new** UserModel.fromJson(response.**data**[**'data'**]);  
 UserModel.*currentLoginUser*.**password** = password;  
 } **on** DioError **catch** (e) {  
 *// 打印请求失败相关信息* **if**(e.**response** != **null**) {  
 code = e.**response**.**statusCode**;  
 } **else**{  
 code = 404;  
 }  
 }  
 **return** code;  
 }  
  
 **static** Future<int> *register*(String userName, String email, String password) **async** {  
 int code = 200;  
 **try** {  
 Response response = **await** HttpUtils.*getInstance*().post(Configs.*registerUrl*,  
 data: {**"email"**: email, **"username"**: userName, **"password"**: password});  
 code = response.**statusCode**;  
 } **on** DioError **catch** (e) {  
 *// 打印请求失败相关信息* **if**(e.**response** != **null**) {  
 code = e.**response**.**statusCode**;  
 } **else**{  
 code = 404;  
 }  
 }  
 **return** code;  
 }  
}

**class** ArticleModel {  
 int **id**;  
 String **title**;  
 String **content**;  
 int **categoryId**;  
 String **updateAt**;  
  
 ArticleModel(  
 **this**.**id**, **this**.**title**, **this**.**content**, **this**.**categoryId**, **this**.**updateAt**);  
  
 ArticleModel.fromJson(Map<String, **dynamic**> json)  
 : **id** = json[**'id'**],  
 **title** = json[**'title'**],  
 **content** = json[**'content'**],  
 **categoryId** = json[**'categoryId'**],  
 **updateAt** = DateTime.*parse*(json[**'updated\_at'**].toString()).toLocal().toString().substring(0, 19);  
  
 @override  
 String toString() {  
 **return 'ArticleModel{id:** $**id, title:** $**title, content:** $**content, updateAt:** $**updateAt}'**;  
 }  
}

**class** CategoryModel {  
 int **id**;  
 String **name**;  
  
 CategoryModel(**this**.**id**, **this**.**name**);  
  
 CategoryModel.fromJson(Map<String, **dynamic**> json)  
 : **id** = json[**'id'**],  
 **name** = json[**'name'**];  
  
 @override  
 String toString() {  
 **return 'CategoryModel{id:** $**id, name:** $**name}'**;  
 }  
}

**import 'package:scoped\_model/scoped\_model.dart'**;  
**import '../controller/Category.dart'**;  
**import '../controller/Article.dart'**;  
**import 'CategoryModel.dart'**;  
**import 'ArticleModel.dart'**;  
  
**class** Data {  
 **static** DataScopedModel *model*;  
}  
  
**class** DataScopedModel **extends** Model {  
 List<CategoryModel> **\_category** = **new** List<CategoryModel>();  
 List<ArticleModel> **\_article** = **new** List<ArticleModel>();  
  
 List<CategoryModel> **get category** => **\_category**;  
 List<ArticleModel> **get article** => **\_article**;  
  
 DataScopedModel() {  
 getAll();  
 }  
  
 **void** getAllCategory() **async**{  
 **\_category** = **await** Category.*list*();  
 }  
  
 **void** getAllArticle() **async**{  
 **\_article** = **await** Article.*list*();  
 }  
  
 Future getAll() **async** {  
 **\_category** = **await** Category.*list*();  
 **\_article** = **await** Article.*list*();  
 notifyListeners();  
 }  
  
 **void** deleteCategory(int id) **async**{  
 **for**(int i = 0; i < **\_category**.**length**; i++) {  
 **if**(**\_category**[i].**id** == id) {  
 **await** Category.*delete*(id);  
 **\_category**.removeAt(i);  
 notifyListeners();  
 }  
 }  
 }  
  
 **void** deleteArticle(int id) **async**{  
 **for**(int i = 0; i < **\_article**.**length**; i++) {  
 **if**(**\_article**[i].**id** == id) {  
 **await** Article.*delete*(id);  
 **\_article**.removeAt(i);  
 notifyListeners();  
 }  
 }  
 }  
}

**class** UserModel {  
 **static** UserModel *currentLoginUser*;  
  
 String **username**;  
 String **email**;  
 String **password**;  
 String **token**;  
  
 UserModel(**this**.**username**, **this**.**email**);  
  
 UserModel.fromJson(Map<String, **dynamic**> json)  
 : **username** = json[**'username'**],  
 **email** = json[**'email'**],  
 **token** = json[**'token'**];  
  
 @override  
 String toString() {  
 **return 'UserModel{\_username:** $**username, \_email:** $**email, \_token:** $**token}'**;  
 }  
}

**import 'dart:ui'**;  
  
**import 'package:flutter/cupertino.dart'**;  
  
**class** Colors {  
 **const** Colors();  
  
 **static const** Color *loginGradientStart* = **const** Color(0xFF81D4FA);  
 **static const** Color *loginGradientEnd* = **const** Color(0xFF29B6F6);  
  
 **static const** *primaryGradient* = **const** LinearGradient(  
 colors: **const** [*loginGradientStart*, *loginGradientEnd*],  
 stops: **const** [0.0, 1.0],  
 begin: Alignment.*topCenter*,  
 end: Alignment.*bottomCenter*);  
  
 **static const** *horizontalGradient* = **const** LinearGradient(  
 colors: **const** [*loginGradientStart*, *loginGradientEnd*],  
 begin: **const** FractionalOffset(0.0, 0.0),  
 end: **const** FractionalOffset(1.0, 1.0),  
 stops: [0.0, 1.0],  
 tileMode: TileMode.**clamp**);  
}

**import 'package:flutter/material.dart'**;  
**import '../style/theme.dart' as** Theme;  
**import '../model/DataScopedModel.dart'**;  
**import '../controller/Article.dart'**;  
  
**class** ArticleDetail **extends** StatefulWidget {  
 ArticleDetail(  
 {Key key,  
 @required **this**.**id**,  
 @required **this**.**title**,  
 @required **this**.**categoryId**,  
 @required **this**.**content**})  
 : **super**(key: key);  
 **final** int **id**;  
 **final** String **title**;  
 **final** int **categoryId**;  
 **final** String **content**;  
  
 @override  
 \_ArticleDetail createState() => \_ArticleDetail();  
}  
  
**class** \_ArticleDetail **extends** State<ArticleDetail> {  
 **final** GlobalKey<ScaffoldState> **\_scaffoldKey** = **new** GlobalKey<ScaffoldState>();  
 TextEditingController **articleTitleController** = **new** TextEditingController();  
 TextEditingController **articleContentController** = **new** TextEditingController();  
 **var dropDownButtonValue**;  
  
 @override  
 **void** initState() {  
 **super**.initState();  
 **articleTitleController**.**text** = **widget**.**title**;  
 **articleContentController**.**text** = **widget**.**content**;  
 **dropDownButtonValue** = **widget**.**categoryId** != -1 ? **widget**.**categoryId** : **null**;  
 }  
  
 @override  
 Widget build(BuildContext context) {  
 double height = MediaQuery.*of*(context).**size**.**height**;  
 **return** Scaffold(  
 key: **\_scaffoldKey**,  
 appBar: PreferredSize(  
 preferredSize: Size(20, 60),  
 child: AppBar(  
 title: Text(  
 **widget**.**title**.**isNotEmpty** ? **"编辑笔记: "** + **widget**.**title** : **"新建笔记"**,  
 style: TextStyle(  
 color: Colors.*white*,  
 fontSize: 20.0,  
 fontFamily: **"WorkSansSemiBold"**)),  
 backgroundColor: Theme.Colors.*loginGradientStart*,  
 elevation: 0,  
 ),  
 ),  
 body: NotificationListener<OverscrollIndicatorNotification>(  
 onNotification: (overscroll) {  
 overscroll.disallowGlow();  
 },  
 child: SingleChildScrollView(  
 child: Container(  
 width: MediaQuery.*of*(context).**size**.**width**,  
 height: height - 80,  
 padding: EdgeInsets.only(top: 20),  
 decoration: **new** BoxDecoration(  
 gradient: **new** LinearGradient(  
 colors: [  
 Theme.Colors.*loginGradientStart*,  
 Theme.Colors.*loginGradientEnd* ],  
 begin: **const** FractionalOffset(0.0, 0.0),  
 end: **const** FractionalOffset(1.0, 1.0),  
 stops: [0.0, 1.0],  
 tileMode: TileMode.**clamp**,  
 ),  
 ),  
 child: Column(  
 mainAxisSize: MainAxisSize.**max**,  
 children: <Widget>[  
 Card(  
 elevation: 2.0,  
 color: Colors.*white*,  
 shape: RoundedRectangleBorder(  
 borderRadius: BorderRadius.circular(8.0),  
 ),  
 child: Container(  
 width: 360.0,  
 child: Column(  
 children: <Widget>[  
 Padding(  
 padding: EdgeInsets.only(  
 top: 0.0, bottom: 20.0, left: 25.0, right: 25.0),  
 child: Row(  
 children: <Widget>[  
 Container(  
 padding: EdgeInsets.only(right: 8),  
 child: Icon(Icons.*category*),  
 ),  
 DropdownButton(  
 items: Data.*model*.**category**.map(  
 (item) {  
 **return** DropdownMenuItem(  
 child: Text(item.**name**),  
 value: item.**id**,  
 );  
 },  
 ).toList(),  
 hint: Text(**"请选择目录"**),  
 onChanged: (T) {  
 setState(() {  
 **dropDownButtonValue** = T;  
 });  
 },  
 style: TextStyle(  
 fontFamily: **"WorkSansSemiBold"**,  
 fontSize: 16.0,  
 color: Colors.*black*,  
 ),  
 value: **dropDownButtonValue**,  
 ),  
 ],  
 ),  
 ),  
 Padding(  
 padding: EdgeInsets.only(  
 top: 0.0, bottom: 0.0, left: 25.0, right: 25.0),  
 child: TextField(  
 controller: **articleTitleController**,  
 keyboardType: TextInputType.*text*,  
 style: TextStyle(  
 fontFamily: **"WorkSansSemiBold"**,  
 fontSize: 16.0,  
 color: Colors.*black*),  
 decoration: InputDecoration(  
 border: InputBorder.*none*,  
 icon: Icon(  
 Icons.*title*,  
 color: Colors.*black*,  
 size: 22.0,  
 ),  
 hintText: **"请输入标题"**,  
 hintStyle: TextStyle(  
 fontFamily: **"WorkSansSemiBold"**,  
 fontSize: 17.0),  
 ),  
 ),  
 ),  
 Container(  
 width: 250.0,  
 height: 1.0,  
 color: Colors.*grey*[400],  
 ),  
 Container(  
 height: 300,  
 padding: EdgeInsets.only(  
 top: 0.0, bottom: 20.0, left: 25.0, right: 25.0),  
 child: TextField(  
 controller: **articleContentController**,  
 keyboardType: TextInputType.*multiline*,  
 maxLines: **null**,  
 style: TextStyle(  
 fontFamily: **"WorkSansSemiBold"**,  
 fontSize: 16.0,  
 color: Colors.*black*),  
 decoration: InputDecoration(  
 border: InputBorder.*none*,  
 icon: Icon(  
 Icons.*code*,  
 size: 22.0,  
 color: Colors.*black*,  
 ),  
 hintText: **"请输入内容"**,  
 hintStyle: TextStyle(  
 fontFamily: **"WorkSansSemiBold"**,  
 fontSize: 17.0),  
 )),  
 ),  
 ],  
 ),  
 ),  
 ),  
 Container(  
 width: 200,  
 margin: EdgeInsets.only(top: 10.0),  
 decoration: **new** BoxDecoration(  
 borderRadius: BorderRadius.all(Radius.circular(5.0)),  
 boxShadow: <BoxShadow>[  
 BoxShadow(  
 color: Theme.Colors.*loginGradientStart*,  
 offset: Offset(1.0, 6.0),  
 blurRadius: 20.0,  
 ),  
 BoxShadow(  
 color: Theme.Colors.*loginGradientEnd*,  
 offset: Offset(1.0, 6.0),  
 blurRadius: 20.0,  
 ),  
 ],  
 gradient: Theme.Colors.*horizontalGradient*,  
 ),  
 child: Center(  
 child: MaterialButton(  
 highlightColor: Colors.*transparent*,  
 splashColor: Theme.Colors.*loginGradientEnd*,  
 child: Padding(  
 padding: **const** EdgeInsets.symmetric(  
 vertical: 10.0,  
 horizontal: 42.0,  
 ),  
 child: Text(  
 **"提 交"**,  
 style: TextStyle(  
 color: Colors.*white*,  
 fontSize: 25.0,  
 fontFamily: **"WorkSansBold"**,  
 ),  
 ),  
 ),  
 onPressed: \_onConfirmArticleButtonPress,  
 ),  
 ),  
 ),  
 ],  
 ),  
 ),  
 ),  
 ),  
 );  
 }  
  
 Future<int> \_onConfirmArticleButtonPress() **async** {  
 int id = **widget**.**id**;  
 int categoryId = **dropDownButtonValue**;  
 String title = **articleTitleController**.**text**;  
 String content = **articleContentController**.**text**;  
 int code = 200;  
 **if** (title == **''**) {  
 showInSnackBar(**"标题不可为空"**);  
 } **else** {  
 **if** (id == -1) {  
 code = **await** Article.*create*(title, content, categoryId);  
 **if** (code == 200) {  
 showInSnackBar(**"创建笔记成功"**);  
 **await** Data.*model*.getAll();  
 Navigator.*pop*(**context**);  
 }  
 } **else** {  
 code = **await** Article.*update*(id, title, content, categoryId);  
 **if** (code == 200) {  
 showInSnackBar(**"笔记更新成功"**);  
 **await** Data.*model*.getAll();  
 Navigator.*pop*(**context**);  
 }  
 }  
 **if** (code != 200) {  
 showInSnackBar(**"服务器好像开小差了"**);  
 }  
 }  
 **return** code;  
 }  
  
 **void** showInSnackBar(String value) {  
 FocusScope.*of*(**context**).requestFocus(**new** FocusNode());  
 **\_scaffoldKey**.**currentState**?.removeCurrentSnackBar();  
 **\_scaffoldKey**.**currentState**.showSnackBar(**new** SnackBar(  
 content: **new** Text(  
 value,  
 textAlign: TextAlign.**center**,  
 style: TextStyle(  
 color: Colors.*white*,  
 fontSize: 16.0,  
 fontFamily: **"WorkSansSemiBold"**),  
 ),  
 backgroundColor: Colors.*blue*,  
 duration: Duration(seconds: 3),  
 ));  
 }  
}

**import 'package:flutter/material.dart'**;  
**import 'package:scoped\_model/scoped\_model.dart'**;  
**import '../style/theme.dart' as** Theme;  
**import '../model/DataScopedModel.dart'**;  
**import 'article\_detail.dart'**;  
  
**class** ArticlePage **extends** StatefulWidget {  
 @override  
 \_ArticlePageState createState() => \_ArticlePageState();  
}  
  
**class** \_ArticlePageState **extends** State<ArticlePage> {  
 **final** GlobalKey<ScaffoldState> **\_scaffoldKey** = **new** GlobalKey<ScaffoldState>();  
  
 Widget build(BuildContext context) {  
 **return** ScopedModelDescendant<DataScopedModel>(builder: (context, \_, model) {  
 List<Widget> tabBars = **new** List<Widget>();  
 tabBars.add(  
 Tab(  
 text: **"全部"**,  
 ),  
 );  
 List<Widget> tabBarViews = **new** List<Widget>();  
 ArticleList articleList = ArticleList();  
 tabBarViews.add(  
 articleList,  
 );  
 **for** (**var** value **in** model.**category**) {  
 tabBars.add(  
 Tab(  
 text: value.**name**,  
 ),  
 );  
 List<Widget> list = **new** List<Widget>();  
 **for** (**var** item **in** model.**article**) {  
 **if** (item.**categoryId** == value.**id**) {  
 list.add(  
 ArticleItem(  
 id: item.**id**,  
 title: item.**title**,  
 content: item.**content**,  
 categoryId: item.**categoryId**,  
 updateAt: item.**updateAt**,  
 ),  
 );  
 }  
 }  
 **if** (list.**length** == 0) {  
 list.add(ListTile(  
 title: Text(**"当前目录还没有文章"**),  
 leading: Icon(Icons.*announcement*),  
 ));  
 }  
 tabBarViews.add(ListView(  
 children: list,  
 ));  
 }  
 **return** DefaultTabController(  
 initialIndex: 0,  
 length: tabBarViews.**length**,  
 child: Scaffold(  
 key: **\_scaffoldKey**,  
 appBar: PreferredSize(  
 child: Container(  
 decoration: **new** BoxDecoration(  
 gradient: Theme.Colors.*horizontalGradient*,  
 ),  
 child: AppBar(  
 bottom: TabBar(  
 tabs: tabBars,  
 isScrollable: **true**,  
 ),  
 centerTitle: **true**,  
 backgroundColor: Colors.*transparent*,  
 elevation: 0,  
 ),  
 ),  
 preferredSize: Size(20, 50),  
 ),  
 body: TabBarView(children: tabBarViews),  
 floatingActionButton: FloatingActionButton(  
 child: Icon(Icons.*add*),  
 onPressed: () {  
 **if** (model.**category**.**length** == 0) {  
 showInSnackBar(**"请先创建目录!"**);  
 } **else** {  
 Navigator.*push*(  
 context,  
 MaterialPageRoute(  
 builder: (BuildContext context) {  
 **return** ArticleDetail(  
 id: -1,  
 title: **""**,  
 content: **""**,  
 categoryId: -1,  
 );  
 },  
 ),  
 );  
 }  
 },  
 ),  
 ),  
 );  
 });  
 }  
  
 **void** showInSnackBar(String value) {  
 FocusScope.*of*(**context**).requestFocus(**new** FocusNode());  
 **\_scaffoldKey**.**currentState**?.removeCurrentSnackBar();  
 **\_scaffoldKey**.**currentState**.showSnackBar(**new** SnackBar(  
 content: **new** Text(  
 value,  
 textAlign: TextAlign.**center**,  
 style: TextStyle(  
 color: Colors.*white*,  
 fontSize: 16.0,  
 fontFamily: **"WorkSansSemiBold"**),  
 ),  
 backgroundColor: Colors.*blue*,  
 duration: Duration(seconds: 3),  
 ));  
 }  
}  
  
**class** ArticleList **extends** StatelessWidget {  
 @override  
 Widget build(BuildContext context) {  
 **return** ScopedModelDescendant<DataScopedModel>(  
 builder: (context, \_, model) {  
 List<Widget> list = **new** List<Widget>();  
 **for** (**var** value **in** model.**article**) {  
 list.add(  
 ArticleItem(  
 id: value.**id**,  
 title: value.**title**,  
 content: value.**content**,  
 categoryId: value.**categoryId**,  
 updateAt: value.**updateAt**,  
 ),  
 );  
 }  
 **if** (list.**length** == 0) {  
 list.add(ListTile(  
 title: Text(**"当前目录还没有文章"**),  
 leading: Icon(Icons.*announcement*),  
 ));  
 }  
 **return** ListView(  
 children: list,  
 );  
 },  
 );  
 }  
}  
  
**class** ArticleItem **extends** StatefulWidget {  
 ArticleItem(  
 {Key key,  
 @required **this**.**id**,  
 @required **this**.**title**,  
 @required **this**.**content**,  
 @required **this**.**categoryId**,  
 @required **this**.**updateAt**,})  
 : **super**(key: key);  
 **final** int **id**;  
 **final** String **title**;  
 **final** String **content**;  
 **final** int **categoryId**;  
 **final** String **updateAt**;  
  
 @override  
 \_ArticleItem createState() => \_ArticleItem();  
}  
  
**class** \_ArticleItem **extends** State<ArticleItem> {  
 @override  
 Widget build(BuildContext context) {  
 **return** GestureDetector(  
 onTap: () {  
 Navigator.*push*(  
 context,  
 MaterialPageRoute(  
 builder: (BuildContext context) {  
 **return** ArticleDetail(  
 id: **widget**.**id**,  
 title: **widget**.**title**,  
 content: **widget**.**content**,  
 categoryId: **widget**.**categoryId**,  
 );  
 },  
 ),  
 );  
 },  
 child: ListTile(  
 title: Text(**widget**.**title**),  
 subtitle: Text(**"最后更新: "** + **widget**.**updateAt**),  
 trailing: ScopedModelDescendant<DataScopedModel>(  
 rebuildOnChange: **false**,  
 builder: (context, \_, model) => GestureDetector(  
 child: Icon(  
 Icons.*delete\_forever*,  
 ),  
 onTap: () {  
 showGeneralDialog(  
 context: context,  
 pageBuilder: (context, \_, \_\_) => AlertDialog(  
 title: Text(**"提示"**),  
 content: SingleChildScrollView(  
 child: ListBody(  
 children: <Widget>[  
 Text(**"确定删除笔记: '**${**widget**.**title**}**' 吗?"**),  
 Text(**'一旦删除数据不可恢复!'**),  
 ],  
 ),  
 ),  
 actions: <Widget>[  
 FlatButton(  
 child: Text(**"取消"**),  
 onPressed: () {  
 Navigator.*of*(context).pop();  
 },  
 ),  
 FlatButton(  
 child: Text(**"确定"**),  
 onPressed: () {  
 model.deleteArticle(**widget**.**id**);  
 Navigator.*of*(context).pop();  
 },  
 )  
 ],  
 ),  
 barrierDismissible: **false**,  
 transitionDuration: Duration(milliseconds: 200));  
 },  
 ),  
 ),  
 ),  
 );  
 }  
}

**import 'package:flutter/material.dart'**;  
**import 'dart:async'**;  
**import 'package:flutter/services.dart'**;  
**import '../style/theme.dart' as** Theme;  
**import '../controller/Category.dart'**;  
**import '../model/DataScopedModel.dart'**;  
  
**class** CategoryDetail **extends** StatefulWidget {  
 CategoryDetail({Key key,@required **this**.**id**, @required **this**.**name**}) : **super**(key: key);  
 **final** int **id**;  
 **final** String **name**;  
  
 @override  
 \_CategoryDetail createState() => \_CategoryDetail();  
}  
  
**class** \_CategoryDetail **extends** State<CategoryDetail> {  
 TextEditingController **categoryNameController** = **new** TextEditingController();  
 **final** GlobalKey<ScaffoldState> **\_scaffoldKey** = **new** GlobalKey<ScaffoldState>();  
  
 @override  
 **void** initState() {  
 **super**.initState();  
 **if** (**widget**.**name** != **''**) {  
 **categoryNameController**.**text** = **widget**.**name**;  
 }  
 }  
  
 @override  
 Widget build(BuildContext context) {  
 **return** Scaffold(  
 key: **\_scaffoldKey**,  
 appBar: AppBar(  
 title: Text(**widget**.**name**.**isNotEmpty** ? **"编辑分类: "** + **widget**.**name** : **"新建分类"**,  
 style: TextStyle(  
 color: Colors.*white*,  
 fontSize: 20.0,  
 fontFamily: **"WorkSansSemiBold"**)),  
 backgroundColor: Theme.Colors.*loginGradientStart*,  
 elevation: 0),  
 body: Container(  
 padding: EdgeInsets.only(top: 23.0),  
 width: MediaQuery.*of*(context).**size**.**width**,  
 height: MediaQuery.*of*(context).**size**.**height**,  
 decoration: **new** BoxDecoration(  
 gradient: **new** LinearGradient(  
 colors: [  
 Theme.Colors.*loginGradientStart*,  
 Theme.Colors.*loginGradientEnd* ],  
 begin: **const** FractionalOffset(0.0, 0.0),  
 end: **const** FractionalOffset(1.0, 1.0),  
 stops: [0.0, 1.0],  
 tileMode: TileMode.**clamp**),  
 ),  
 child: Center(  
 child: Column(  
 children: <Widget>[  
 Stack(  
 alignment: Alignment.*topCenter*,  
 overflow: Overflow.**visible**,  
 children: <Widget>[  
 Card(  
 elevation: 2.0,  
 color: Colors.*white*,  
 shape: RoundedRectangleBorder(  
 borderRadius: BorderRadius.circular(8.0),  
 ),  
 child: Container(  
 width: 360,  
 child: Column(  
 children: <Widget>[  
 Padding(  
 padding: EdgeInsets.only(  
 top: 20.0,  
 bottom: 20.0,  
 left: 25.0,  
 right: 25.0),  
 child: TextField(  
 controller: **categoryNameController**,  
 keyboardType: TextInputType.*text*,  
 style: TextStyle(  
 fontFamily: **"WorkSansSemiBold"**,  
 fontSize: 16.0,  
 color: Colors.*black*),  
 decoration: InputDecoration(  
 border: InputBorder.*none*,  
 icon: Icon(  
 Icons.*category*,  
 color: Colors.*black*,  
 size: 22.0,  
 ),  
 hintText: **"请输入分类名"**,  
 hintStyle: TextStyle(  
 fontFamily: **"WorkSansSemiBold"**,  
 fontSize: 17.0),  
 ),  
 ),  
 ),  
 ],  
 ),  
 ),  
 ),  
 Container(  
 margin: EdgeInsets.only(top: 70.0),  
 decoration: **new** BoxDecoration(  
 borderRadius: BorderRadius.all(Radius.circular(5.0)),  
 boxShadow: <BoxShadow>[  
 BoxShadow(  
 color: Theme.Colors.*loginGradientStart*,  
 offset: Offset(1.0, 6.0),  
 blurRadius: 20.0,  
 ),  
 BoxShadow(  
 color: Theme.Colors.*loginGradientEnd*,  
 offset: Offset(1.0, 6.0),  
 blurRadius: 20.0,  
 ),  
 ],  
 gradient: Theme.Colors.*horizontalGradient*,  
 ),  
 child: MaterialButton(  
 highlightColor: Colors.*transparent*,  
 splashColor: Theme.Colors.*loginGradientEnd*,  
 child: Padding(  
 padding: **const** EdgeInsets.symmetric(  
 vertical: 10.0, horizontal: 42.0),  
 child: Text(  
 **"提 交"**,  
 style: TextStyle(  
 color: Colors.*white*,  
 fontSize: 25.0,  
 fontFamily: **"WorkSansBold"**),  
 ),  
 ),  
 onPressed: \_onConfirmCategoryButtonPress),  
 ),  
 ],  
 ),  
 ],  
 ),  
 ),  
 ),  
 );  
 }  
  
 Future<int> \_onConfirmCategoryButtonPress() **async** {  
 int id = **widget**.**id**;  
 String name = **categoryNameController**.**text**;  
 int code = 200;  
 **if** (name == **''**) {  
 showInSnackBar(**"分类名不可为空"**);  
 } **else** {  
 **if** (id == -1) {  
 code = **await** Category.*create*(name);  
 **if** (code == 200) {  
 showInSnackBar(**"创建分类成功"**);  
 **await** Data.*model*.getAll();  
 Navigator.*pop*(**context**);  
 }  
 } **else** {  
 code = **await** Category.*update*(id, name);  
 **if** (code == 200) {  
 showInSnackBar(**"分类更新成功"**);  
 **await** Data.*model*.getAll();  
 Navigator.*pop*(**context**);  
 }  
 }  
 **if** (code != 200) {  
 showInSnackBar(**"服务器好像开小差了"**);  
 }  
 }  
 **return** code;  
 }  
  
 **void** showInSnackBar(String value) {  
 FocusScope.*of*(**context**).requestFocus(**new** FocusNode());  
 **\_scaffoldKey**.**currentState**?.removeCurrentSnackBar();  
 **\_scaffoldKey**.**currentState**.showSnackBar(**new** SnackBar(  
 content: **new** Text(  
 value,  
 textAlign: TextAlign.**center**,  
 style: TextStyle(  
 color: Colors.*white*,  
 fontSize: 16.0,  
 fontFamily: **"WorkSansSemiBold"**),  
 ),  
 backgroundColor: Colors.*blue*,  
 duration: Duration(seconds: 3),  
 ));  
 }  
}

**import 'package:flutter/material.dart'**;  
**import 'package:scoped\_model/scoped\_model.dart'**;  
**import '../model/DataScopedModel.dart'**;  
**import 'category\_detail.dart'**;  
  
**class** CategoryPage **extends** StatelessWidget {  
 Widget build(BuildContext context) {  
 **return** Scaffold(  
 body: CategoryList(),  
 floatingActionButton: FloatingActionButton(  
 child: Icon(Icons.*add*),  
 onPressed: () {  
 Navigator.*push*(context,  
 MaterialPageRoute(builder: (BuildContext context) {  
 **return** CategoryDetail(  
 name: **''**,  
 id: -1,  
 );  
 }));  
 }),  
 );  
 }  
}  
  
**class** CategoryList **extends** StatelessWidget {  
 @override  
 Widget build(BuildContext context) {  
 **return** ScopedModelDescendant<DataScopedModel>(  
 builder: (context, \_, model) {  
 List<Widget> list = **new** List<Widget>();  
 **for** (**var** value **in** model.**category**) {  
 list.add(CategoryItem(id: value.**id**, name: value.**name**));  
 }  
 **if** (list.**length** == 0) {  
 list.add(ListTile(  
 title: Text(**"当前账号还没有目录"**),  
 leading: Icon(Icons.*announcement*),  
 ));  
 }  
 **return** ListView(  
 children: list,  
 );  
 },  
 );  
 }  
}  
  
**class** CategoryItem **extends** StatefulWidget {  
 CategoryItem({Key key, @required **this**.**id**, @required **this**.**name**})  
 : **super**(key: key);  
 **final** int **id**;  
 **final** String **name**;  
  
 @override  
 \_CategoryItem createState() => \_CategoryItem();  
}  
  
**class** \_CategoryItem **extends** State<CategoryItem> {  
 @override  
 Widget build(BuildContext context) {  
 **return** GestureDetector(  
 onTap: () {  
 Navigator.*push*(  
 context,  
 MaterialPageRoute(  
 builder: (BuildContext context) {  
 **return** CategoryDetail(  
 id: **widget**.**id**,  
 name: **widget**.**name**,  
 );  
 },  
 ),  
 );  
 },  
 child: ListTile(  
 title: Text(**widget**.**name**),  
*// leading: Icon(Icons.category),* trailing: ScopedModelDescendant<DataScopedModel>(  
 rebuildOnChange: **false**,  
 builder: (context, \_, model) => GestureDetector(  
 child: Icon(  
 Icons.*delete\_forever*,  
 ),  
 onTap: () {  
 int flag = 0;  
 **for** (**var** value **in** model.**article**) {  
 **if** (value.**categoryId** == **widget**.**id**) {  
 flag = 1;  
 **break**;  
 }  
 }  
 **if** (flag == 1) {  
 showDialog(  
 context: context,  
 builder: (\_) => SimpleDialog(  
 title: Text(**"删除失败"**),  
 children: <Widget>[  
 Padding(  
 padding: **const** EdgeInsets.all(8.0),  
 child: Center(  
 child: Row(  
 children: <Widget>[  
 Icon(Icons.*error*),  
 Text(**"分类下还有笔记,不可删除"**),  
 ],  
 ),  
 ),  
 ),  
 ],  
 ),  
 );  
 } **else** {  
 showGeneralDialog(  
 context: context,  
 pageBuilder: (context, \_, \_\_) => AlertDialog(  
 title: Text(**"提示"**),  
 content: SingleChildScrollView(  
 child: ListBody(  
 children: <Widget>[  
 Text(**"确定删除: '**${**widget**.**name**}**' 目录吗?"**),  
 Text(**'一旦删除数据不可恢复!'**),  
 ],  
 ),  
 ),  
 actions: <Widget>[  
 FlatButton(  
 child: Text(**"取消"**),  
 onPressed: () {  
 Navigator.*of*(context).pop();  
 },  
 ),  
 FlatButton(  
 child: Text(**"确定"**),  
 onPressed: () {  
 model.deleteCategory(**widget**.**id**);  
 Navigator.*of*(context).pop();  
 },  
 )  
 ],  
 ),  
 barrierDismissible: **false**,  
 transitionDuration: Duration(milliseconds: 200),  
 );  
 }  
 },  
 ),  
 ),  
 ),  
 );  
 }  
}

**import 'dart:async'**;  
**import 'package:scoped\_model/scoped\_model.dart'**;  
**import 'package:flutter/material.dart'**;  
**import '../model/UserModel.dart'**;  
**import '../model/DataScopedModel.dart'**;  
**import '../style/theme.dart' as** Theme;  
**import '../controller/Category.dart'**;  
**import '../main.dart'**;  
**import 'article\_page.dart'**;  
**import 'category\_page.dart'**;  
  
**class** HomeApp **extends** StatelessWidget {  
 @override  
 Widget build(BuildContext context) {  
 **return new** MaterialApp(  
 title: **'BubyNotebook'**,  
 theme: **new** ThemeData(primarySwatch: Colors.*blue*),  
 home: **new** HomePage());  
 }  
}  
  
**class** HomePage **extends** StatefulWidget {  
 HomePage({Key key}) : **super**(key: key);  
  
 @override  
 \_HomePageState createState() => **new** \_HomePageState();  
}  
  
**class** \_HomePageState **extends** State<HomePage> {  
 @override  
 Widget build(BuildContext context) {  
 *//* ***TODO: implement build* return** ScopedModel(  
 model: Data.*model*,  
 child: **new** DefaultTabController(  
 length: 2,  
 child: **new** Scaffold(  
 appBar: PreferredSize(  
 child: Container(  
 decoration: **new** BoxDecoration(  
 gradient: Theme.Colors.*horizontalGradient*,  
 ),  
 child: AppBar(  
 title: Text(**"BuBy Notebook"**,  
 style: TextStyle(  
 color: Colors.*white*,  
 fontSize: 30.0,  
 fontFamily: **"WorkSansSemiBold"**)),  
 centerTitle: **true**,  
 backgroundColor: Colors.*transparent*,  
 elevation: 0)),  
 preferredSize: Size(20, 60)),  
 drawer: Drawer(  
 child: ListView(  
 padding: EdgeInsets.all(0),  
 children: <Widget>[  
 UserAccountsDrawerHeader(  
 accountEmail: Text(  
 UserModel.*currentLoginUser*.**email**,  
 style: TextStyle(color: Colors.*white*),  
 ),  
 accountName: Text(  
 UserModel.*currentLoginUser*.**username**,  
 style: TextStyle(color: Colors.*white*),  
 ),  
 currentAccountPicture: CircleAvatar(  
 backgroundImage: **new** AssetImage(**'assets/img/lg.png'**),  
 ),  
 decoration: **new** BoxDecoration(  
 gradient: Theme.Colors.*horizontalGradient*,  
 ),  
 ),  
 ListTile(  
 title: Text(  
 **"注销登录"**,  
 ),  
 trailing: Icon(Icons.*exit\_to\_app*),  
 onTap: \_onSingOutButtonPress),  
 ListTile(  
 title: Text(  
 **"设置"**,  
 ),  
  
 onTap: \_onSingOutButtonPress),  
  
 ],  
 ),  
 ),  
 body: TabBarView(  
 children: <Widget>[  
 ArticlePage(),  
 CategoryPage(),  
 ],  
 ),  
 bottomNavigationBar: Container(  
 height: 60,  
 decoration: **new** BoxDecoration(  
 gradient: Theme.Colors.*horizontalGradient*,  
 ),  
 child: TabBar(  
 labelStyle: TextStyle(  
 color: Colors.*white*,  
 fontSize: 10.0,  
 fontFamily: **"WorkSansBold"**),  
 tabs: <Widget>[  
 Tab(  
 icon: Icon(Icons.*mode\_edit*),  
 text: **"笔记"**,  
 ),  
 Tab(  
 icon: Icon(Icons.*category*),  
 text: **"分类"**,  
 )  
 ],  
 ),  
 ),  
 ),  
 ),  
 );  
 }  
  
 \_onSingOutButtonPress() {  
 Navigator.*of*(**context**).pushAndRemoveUntil(  
 **new** MaterialPageRoute(builder: (context) => **new** MyApp()),  
 (route) => route == **null**);  
 }  
  
 Future<int> list() **async** {  
 print(**await** Category.*list*());  
 **return null**;  
 }  
}

**import 'dart:async'**;  
**import 'package:flutter/material.dart'**;  
**import 'package:flutter/services.dart'**;  
**import 'package:font\_awesome\_flutter/font\_awesome\_flutter.dart'**;  
**import '../style/theme.dart' as** Theme;  
**import '../utils/bubble\_indication\_painter.dart'**;  
**import '../model/DataScopedModel.dart'**;  
**import '../controller/User.dart'**;  
**import '../ui/home\_page.dart'**;  
  
**class** LoginPage **extends** StatefulWidget {  
 LoginPage({Key key}) : **super**(key: key);  
  
 @override  
 \_LoginPageState createState() => **new** \_LoginPageState();  
}  
  
**class** \_LoginPageState **extends** State<LoginPage>  
 **with** SingleTickerProviderStateMixin {  
 **final** GlobalKey<ScaffoldState> **\_scaffoldKey** = **new** GlobalKey<ScaffoldState>();  
  
 **final** FocusNode **myFocusNodeEmailLogin** = FocusNode();  
 **final** FocusNode **myFocusNodePasswordLogin** = FocusNode();  
  
 **final** FocusNode **myFocusNodePassword** = FocusNode();  
 **final** FocusNode **myFocusNodeEmail** = FocusNode();  
 **final** FocusNode **myFocusNodeName** = FocusNode();  
  
 TextEditingController **loginEmailController** = **new** TextEditingController();  
 TextEditingController **loginPasswordController** = **new** TextEditingController();  
  
 bool **\_obscureTextLogin** = **true**;  
 bool **\_obscureTextSignup** = **true**;  
 bool **\_obscureTextSignupConfirm** = **true**;  
  
 TextEditingController **signupEmailController** = **new** TextEditingController();  
 TextEditingController **signupNameController** = **new** TextEditingController();  
 TextEditingController **signupPasswordController** = **new** TextEditingController();  
 TextEditingController **signupConfirmPasswordController** =  
 **new** TextEditingController();  
  
 PageController **\_pageController**;  
  
 Color **left** = Colors.*black*;  
 Color **right** = Colors.*white*;  
  
 @override  
 Widget build(BuildContext context) {  
 **return new** Scaffold(  
 key: **\_scaffoldKey**,  
 body: NotificationListener<OverscrollIndicatorNotification>(  
 onNotification: (overscroll) {  
 overscroll.disallowGlow();  
 },  
 child: SingleChildScrollView(  
 child: Container(  
 width: MediaQuery.*of*(context).**size**.**width**,  
 height: MediaQuery.*of*(context).**size**.**height**,  
 decoration: **new** BoxDecoration(  
 gradient: **new** LinearGradient(  
 colors: [  
 Theme.Colors.*loginGradientStart*,  
 Theme.Colors.*loginGradientEnd* ],  
 begin: **const** FractionalOffset(0.0, 0.0),  
 end: **const** FractionalOffset(1.0, 1.0),  
 stops: [0.0, 1.0],  
 tileMode: TileMode.**clamp**),  
 ),  
 child: Column(  
 mainAxisSize: MainAxisSize.**max**,  
 children: <Widget>[  
 Padding(  
 padding: EdgeInsets.only(top: 40.0),  
 child: **new** Image(  
 width: 180.0,  
 height: 140.0,  
 fit: BoxFit.**fill**,  
 image: **new** AssetImage(**'assets/img/login\_logo.png'**)),  
 ),  
 Padding(  
 padding: EdgeInsets.only(top: 10.0),  
 child: \_buildMenuBar(context),  
 ),  
 Expanded(  
 flex: 2,  
 child: PageView(  
 controller: **\_pageController**,  
 onPageChanged: (i) {  
 **if** (i == 0) {  
 setState(() {  
 **right** = Colors.*white*;  
 **left** = Colors.*black*;  
 });  
 } **else if** (i == 1) {  
 setState(() {  
 **right** = Colors.*black*;  
 **left** = Colors.*white*;  
 });  
 }  
 },  
 children: <Widget>[  
 **new** ConstrainedBox(  
 constraints: **const** BoxConstraints.expand(),  
 child: \_buildSignIn(context),  
 ),  
 **new** ConstrainedBox(  
 constraints: **const** BoxConstraints.expand(),  
 child: \_buildSignUp(context),  
 ),  
 ],  
 ),  
 ),  
 ],  
 ),  
 ),  
 ),  
 ),  
 );  
 }  
  
 @override  
 **void** dispose() {  
 **myFocusNodePassword**.dispose();  
 **myFocusNodeEmail**.dispose();  
 **myFocusNodeName**.dispose();  
 **\_pageController**?.dispose();  
 **super**.dispose();  
 }  
  
 @override  
 **void** initState() {  
 **super**.initState();  
  
 SystemChrome.*setPreferredOrientations*([  
 DeviceOrientation.**portraitUp**,  
 DeviceOrientation.**portraitDown**,  
 ]);  
  
 **\_pageController** = PageController();  
 }  
  
 **void** showInSnackBar(String value) {  
 FocusScope.*of*(**context**).requestFocus(**new** FocusNode());  
 **\_scaffoldKey**.**currentState**?.removeCurrentSnackBar();  
 **\_scaffoldKey**.**currentState**.showSnackBar(**new** SnackBar(  
 content: **new** Text(  
 value,  
 textAlign: TextAlign.**center**,  
 style: TextStyle(  
 color: Colors.*white*,  
 fontSize: 16.0,  
 fontFamily: **"WorkSansSemiBold"**),  
 ),  
 backgroundColor: Colors.*blue*,  
 duration: Duration(seconds: 3),  
 ));  
 }  
  
 Widget \_buildMenuBar(BuildContext context) {  
 **return** Container(  
 width: 300.0,  
 height: 50.0,  
 decoration: BoxDecoration(  
 color: Color(0x552B2B2B),  
 borderRadius: BorderRadius.all(Radius.circular(25.0)),  
 ),  
 child: CustomPaint(  
 painter: TabIndicationPainter(pageController: **\_pageController**),  
 child: Row(  
 mainAxisAlignment: MainAxisAlignment.**spaceEvenly**,  
 children: <Widget>[  
 Expanded(  
 child: FlatButton(  
 splashColor: Colors.*transparent*,  
 highlightColor: Colors.*transparent*,  
 onPressed: \_onSignInButtonPress,  
 child: Text(  
 **"已有账号"**,  
 style: TextStyle(  
 color: **left**,  
 fontSize: 16.0,  
 fontFamily: **"WorkSansSemiBold"**),  
 ),  
 ),  
 ),  
 Expanded(  
 child: FlatButton(  
 splashColor: Colors.*transparent*,  
 highlightColor: Colors.*transparent*,  
 onPressed: \_onSignUpButtonPress,  
 child: Text(  
 **"新建账号"**,  
 style: TextStyle(  
 color: **right**,  
 fontSize: 16.0,  
 fontFamily: **"WorkSansSemiBold"**),  
 ),  
 ),  
 ),  
 ],  
 ),  
 ),  
 );  
 }  
  
 Widget \_buildSignIn(BuildContext context) {  
 **return** Container(  
 padding: EdgeInsets.only(top: 13.0),  
 child: Column(  
 children: <Widget>[  
 Stack(  
 alignment: Alignment.*topCenter*,  
 overflow: Overflow.**visible**,  
 children: <Widget>[  
 Card(  
 elevation: 2.0,  
 color: Colors.*white*,  
 shape: RoundedRectangleBorder(  
 borderRadius: BorderRadius.circular(8.0),  
 ),  
 child: Container(  
 width: 300.0,  
 height: 190.0,  
 child: Column(  
 children: <Widget>[  
 Padding(  
 padding: EdgeInsets.only(  
 top: 20.0, bottom: 20.0, left: 25.0, right: 25.0),  
 child: TextField(  
 focusNode: **myFocusNodeEmailLogin**,  
 controller: **loginEmailController**,  
 keyboardType: TextInputType.*emailAddress*,  
 style: TextStyle(  
 fontFamily: **"WorkSansSemiBold"**,  
 fontSize: 16.0,  
 color: Colors.*black*),  
 decoration: InputDecoration(  
 border: InputBorder.*none*,  
 icon: Icon(  
 FontAwesomeIcons.*envelope*,  
 color: Colors.*black*,  
 size: 22.0,  
 ),  
 hintText: **"请输入邮箱地址"**,  
 hintStyle: TextStyle(  
 fontFamily: **"WorkSansSemiBold"**, fontSize: 17.0),  
 ),  
 ),  
 ),  
 Container(  
 width: 250.0,  
 height: 1.0,  
 color: Colors.*grey*[400],  
 ),  
 Padding(  
 padding: EdgeInsets.only(  
 top: 20.0, bottom: 20.0, left: 25.0, right: 25.0),  
 child: TextField(  
 focusNode: **myFocusNodePasswordLogin**,  
 controller: **loginPasswordController**,  
 obscureText: **\_obscureTextLogin**,  
 style: TextStyle(  
 fontFamily: **"WorkSansSemiBold"**,  
 fontSize: 16.0,  
 color: Colors.*black*),  
 decoration: InputDecoration(  
 border: InputBorder.*none*,  
 icon: Icon(  
 FontAwesomeIcons.*lock*,  
 size: 22.0,  
 color: Colors.*black*,  
 ),  
 hintText: **"请输入密码"**,  
 hintStyle: TextStyle(  
 fontFamily: **"WorkSansSemiBold"**, fontSize: 17.0),  
 suffixIcon: GestureDetector(  
 onTap: \_toggleLogin,  
 child: Icon(  
 FontAwesomeIcons.*eye*,  
 size: 15.0,  
 color: Colors.*black*,  
 ),  
 ),  
 ),  
 ),  
 ),  
 ],  
 ),  
 ),  
 ),  
 Container(  
 margin: EdgeInsets.only(top: 170.0),  
 decoration: **new** BoxDecoration(  
 borderRadius: BorderRadius.all(Radius.circular(5.0)),  
 boxShadow: <BoxShadow>[  
 BoxShadow(  
 color: Theme.Colors.*loginGradientStart*,  
 offset: Offset(1.0, 6.0),  
 blurRadius: 20.0,  
 ),  
 BoxShadow(  
 color: Theme.Colors.*loginGradientEnd*,  
 offset: Offset(1.0, 6.0),  
 blurRadius: 20.0,  
 ),  
 ],  
 gradient: Theme.Colors.*horizontalGradient*,  
 ),  
 child: MaterialButton(  
 highlightColor: Colors.*transparent*,  
 splashColor: Theme.Colors.*loginGradientEnd*,  
 child: Padding(  
 padding: **const** EdgeInsets.symmetric(  
 vertical: 10.0, horizontal: 42.0),  
 child: Text(  
 **"登 录"**,  
 style: TextStyle(  
 color: Colors.*white*,  
 fontSize: 25.0,  
 fontFamily: **"WorkSansBold"**),  
 ),  
 ),  
 onPressed: \_onConfirmSignInButtonPress),  
 ),  
 ],  
 ),  
 ],  
 ),  
 );  
 }  
  
 Widget \_buildSignUp(BuildContext context) {  
 **return** Container(  
 padding: EdgeInsets.only(top: 13.0),  
 child: Column(  
 children: <Widget>[  
 Stack(  
 alignment: Alignment.*topCenter*,  
 overflow: Overflow.**visible**,  
 children: <Widget>[  
 Card(  
 elevation: 2.0,  
 color: Colors.*white*,  
 shape: RoundedRectangleBorder(  
 borderRadius: BorderRadius.circular(8.0),  
 ),  
 child: Container(  
 width: 300.0,  
*// height: 360.0,* child: Column(  
 children: <Widget>[  
 Padding(  
 padding: EdgeInsets.only(  
 top: 10.0, bottom: 10.0, left: 25.0, right: 25.0),  
 child: TextField(  
 focusNode: **myFocusNodeName**,  
 controller: **signupNameController**,  
 keyboardType: TextInputType.*text*,  
 textCapitalization: TextCapitalization.**words**,  
 style: TextStyle(  
 fontFamily: **"WorkSansSemiBold"**,  
 fontSize: 16.0,  
 color: Colors.*black*),  
 decoration: InputDecoration(  
 border: InputBorder.*none*,  
 icon: Icon(  
 FontAwesomeIcons.*user*,  
 color: Colors.*black*,  
 ),  
 hintText: **"请输入昵称"**,  
 hintStyle: TextStyle(  
 fontFamily: **"WorkSansSemiBold"**, fontSize: 16.0),  
 ),  
 ),  
 ),  
 Container(  
 width: 250.0,  
 height: 1.0,  
 color: Colors.*grey*[400],  
 ),  
 Padding(  
 padding: EdgeInsets.only(  
 top: 10.0, bottom: 10.0, left: 25.0, right: 25.0),  
 child: TextField(  
 focusNode: **myFocusNodeEmail**,  
 controller: **signupEmailController**,  
 keyboardType: TextInputType.*emailAddress*,  
 style: TextStyle(  
 fontFamily: **"WorkSansSemiBold"**,  
 fontSize: 16.0,  
 color: Colors.*black*),  
 decoration: InputDecoration(  
 border: InputBorder.*none*,  
 icon: Icon(  
 FontAwesomeIcons.*envelope*,  
 color: Colors.*black*,  
 ),  
 hintText: **"请输入邮箱地址"**,  
 hintStyle: TextStyle(  
 fontFamily: **"WorkSansSemiBold"**, fontSize: 16.0),  
 ),  
 ),  
 ),  
 Container(  
 width: 250.0,  
 height: 1.0,  
 color: Colors.*grey*[400],  
 ),  
 Padding(  
 padding: EdgeInsets.only(  
 top: 10.0, bottom: 10.0, left: 25.0, right: 25.0),  
 child: TextField(  
 focusNode: **myFocusNodePassword**,  
 controller: **signupPasswordController**,  
 obscureText: **\_obscureTextSignup**,  
 style: TextStyle(  
 fontFamily: **"WorkSansSemiBold"**,  
 fontSize: 16.0,  
 color: Colors.*black*),  
 decoration: InputDecoration(  
 border: InputBorder.*none*,  
 icon: Icon(  
 FontAwesomeIcons.*lock*,  
 color: Colors.*black*,  
 ),  
 hintText: **"请输入密码"**,  
 hintStyle: TextStyle(  
 fontFamily: **"WorkSansSemiBold"**, fontSize: 16.0),  
 suffixIcon: GestureDetector(  
 onTap: \_toggleSignup,  
 child: Icon(  
 FontAwesomeIcons.*eye*,  
 size: 15.0,  
 color: Colors.*black*,  
 ),  
 ),  
 ),  
 ),  
 ),  
 Container(  
 width: 250.0,  
 height: 1.0,  
 color: Colors.*grey*[400],  
 ),  
 Padding(  
 padding: EdgeInsets.only(  
 top: 10.0, bottom: 10.0, left: 25.0, right: 25.0),  
 child: TextField(  
 controller: **signupConfirmPasswordController**,  
 obscureText: **\_obscureTextSignupConfirm**,  
 style: TextStyle(  
 fontFamily: **"WorkSansSemiBold"**,  
 fontSize: 16.0,  
 color: Colors.*black*),  
 decoration: InputDecoration(  
 border: InputBorder.*none*,  
 icon: Icon(  
 FontAwesomeIcons.*lock*,  
 color: Colors.*black*,  
 ),  
 hintText: **"请再次输入密码"**,  
 hintStyle: TextStyle(  
 fontFamily: **"WorkSansSemiBold"**, fontSize: 16.0),  
 suffixIcon: GestureDetector(  
 onTap: \_toggleSignupConfirm,  
 child: Icon(  
 FontAwesomeIcons.*eye*,  
 size: 15.0,  
 color: Colors.*black*,  
 ),  
 ),  
 ),  
 ),  
 ),  
 ],  
 ),  
 ),  
 ),  
 Container(  
 margin: EdgeInsets.only(top: 260.0),  
 decoration: **new** BoxDecoration(  
 borderRadius: BorderRadius.all(Radius.circular(5.0)),  
 boxShadow: <BoxShadow>[  
 BoxShadow(  
 color: Theme.Colors.*loginGradientStart*,  
 offset: Offset(1.0, 6.0),  
 blurRadius: 20.0,  
 ),  
 BoxShadow(  
 color: Theme.Colors.*loginGradientEnd*,  
 offset: Offset(1.0, 6.0),  
 blurRadius: 20.0,  
 ),  
 ],  
 gradient: Theme.Colors.*horizontalGradient*,  
 ),  
 child: MaterialButton(  
 highlightColor: Colors.*transparent*,  
 splashColor: Theme.Colors.*loginGradientEnd*,  
 *//shape: RoundedRectangleBorder(borderRadius: BorderRadius.all(Radius.circular(5.0))),* child: Padding(  
 padding: **const** EdgeInsets.symmetric(  
 vertical: 10.0, horizontal: 42.0),  
 child: Text(  
 **"注 册"**,  
 style: TextStyle(  
 color: Colors.*white*,  
 fontSize: 25.0,  
 fontFamily: **"WorkSansBold"**),  
 ),  
 ),  
 onPressed: \_onConfirmSignUpButtonPress),  
 ),  
 ],  
 ),  
 ],  
 ),  
 );  
 }  
  
 **void** \_onSignInButtonPress() {  
 **\_pageController**.animateToPage(0,  
 duration: Duration(milliseconds: 500), curve: Curves.*decelerate*);  
 }  
  
 **void** \_onSignUpButtonPress() {  
 **\_pageController**?.animateToPage(1,  
 duration: Duration(milliseconds: 500), curve: Curves.*decelerate*);  
 }  
  
 **void** \_toggleLogin() {  
 setState(() {  
 **\_obscureTextLogin** = !**\_obscureTextLogin**;  
 });  
 }  
  
 **void** \_toggleSignup() {  
 setState(() {  
 **\_obscureTextSignup** = !**\_obscureTextSignup**;  
 });  
 }  
  
 **void** \_toggleSignupConfirm() {  
 setState(() {  
 **\_obscureTextSignupConfirm** = !**\_obscureTextSignupConfirm**;  
 });  
 }  
  
 Future \_onConfirmSignInButtonPress() **async** {  
 **var** email = **loginEmailController**.**text**;  
 **var** password = **loginPasswordController**.**text**;  
 **if** (email == **''**) {  
 showInSnackBar(**"邮箱地址不可为空"**);  
 } **else if** (password == **''**) {  
 showInSnackBar(**"密码不可为空"**);  
 } **else** {  
 int code = **await** User.*login*(email, password);  
 **if** (code == 200) {  
 showInSnackBar(**"登录成功"**);  
 Data.*model* = DataScopedModel();  
 **await** Data.*model*.getAll();  
 Navigator.*of*(**context**).pushAndRemoveUntil(  
 **new** MaterialPageRoute(builder: (context) => **new** HomeApp()),  
 (route) => route == **null**);  
 } **else if** (code == 403) {  
 showInSnackBar(**"账号不存在"**);  
 } **else if** (code == 401) {  
 showInSnackBar(**"账号或密码不正确"**);  
 } **else** {  
 showInSnackBar(**"服务器好像开小差了"**);  
 }  
 }  
 }  
  
 Future \_onConfirmSignUpButtonPress() **async** {  
 **var** name = **signupNameController**.**text**;  
 **var** email = **signupEmailController**.**text**;  
 **var** password = **signupPasswordController**.**text**;  
 **var** confirmPassword = **signupConfirmPasswordController**.**text**;  
 **if** (name == **''**) {  
 showInSnackBar(**"昵称不可为空"**);  
 } **else if** (email == **''**) {  
 showInSnackBar(**"邮箱地址不可为空"**);  
 } **else if** (password == **''**) {  
 showInSnackBar(**"密码不可为空"**);  
 } **else if** (confirmPassword == **''**) {  
 showInSnackBar(**"确认密码不可以为空"**);  
 } **else if** (password != confirmPassword) {  
 showInSnackBar(**"两次输入密码不一致"**);  
 } **else** {  
 int code = **await** User.*register*(name, email, password);  
 **if** (code == 200) {  
 \_onSignInButtonPress();  
 showInSnackBar(**"注册成功,快登录使用吧!"**);  
 } **else if** (code == 403) {  
 showInSnackBar(**"邮箱已经存在,请换个邮箱试试吧!"**);  
 } **else** {  
 showInSnackBar(**"服务器好像开小差了"**);  
 }  
 }  
 }  
}

**import 'package:flutter/material.dart'**;  
**import './ui/login\_page.dart'**;  
  
**void** main() => runApp(**new** MyApp());  
  
**class** MyApp **extends** StatelessWidget {  
 @override  
 Widget build(BuildContext context) {  
 **return new** MaterialApp(  
 title: **'BubyNotebook'**,  
 theme: **new** ThemeData(  
 primarySwatch: Colors.*blue*,  
 ),  
 home: **new** LoginPage(),  
 );  
 }  
}