```
/*
 *
 *  University di Pisa − Master’s Degree in Computer Science and Networking
 *
 *  Final Project for the course of Peer to Peer Systems and Blockchains
 *
 *  Teacher: Prof. Laura Ricci
 *
 *  Candidate: Orlando Leombruni, matricola 475727
 *
 *  File: index.js
 *
 *  "Main" application script. It also creates a Material UI palette and applies it to the webapp.
 *
 */

import React from ’react’;
import ReactDOM from ’react−dom’;
import { HashRouter, Route, Switch } from ’react−router−dom’;
import { MuiThemeProvider, createMuiTheme } from ’@material−ui/core/styles’;
import { blue, orange } from ’@material−ui/core/colors’;
import App from ’./components/App’;
import ’./styles/global.css’;
import ’typeface−roboto’;

const mTheme = createMuiTheme({
    palette: {
        primary: blue,
        secondary: orange,
    },
    typography: {
        useNextVariants: true,
    }
});

ReactDOM.render(
    <MuiThemeProvider theme={mTheme}>
        <HashRouter>
            <Switch>
                <Route exact path="/catalog" render={(props) => <App {...props} isCatalog={true} />} />
                <Route render={(props) => <App {...props} isCatalog={false} />} />
            </Switch>
        </HashRouter>
    </MuiThemeProvider>,
    document.getElementById(’root’)
);
```

```
/*
 *
 *  University di Pisa - Master's Degree in Computer Science and Networking
 *
 *  Final Project for the course of Peer to Peer Systems and Blockchains
 *
 *  Teacher: Prof. Laura Ricci
 *
 *  Candidate: Orlando Leombruni, matricola 475727
 *
 *  File: MaterialCustomStyles.js
 *
 *  Contains all the custom styles for each React Component, using the Material UI framework.
 *
 */

import {blue, green, red} from "@material-ui/core/colors";
import {fade} from "@material-ui/core/styles/colorManipulator";

export const AppStyle = theme => ({
    toolbar: theme.mixins.toolbar,
    centered: {
        display: "flex",
        flexDirection: "column",
        justifyContent: "center",
        alignItems: "center",
    },
});

export const AppDrawerStyle = theme => {
    const drawerWidth = 240;
    return ({
    root: {
        display: 'flex',
    },
    mask: {
        display: 'flex',
        width: '100%',
        height: '100%',
    },
    appBar: {
        zIndex: theme.zIndex.drawer + 1,
        transition: theme.transitions.create(['width', 'margin'], {
            easing: theme.transitions.easing.sharp,
            duration: theme.transitions.duration.leavingScreen,
        }),
    },
    appBarShift: {
        marginLeft: drawerWidth,
        width: `calc(100% - ${drawerWidth}px)`,
        transition: theme.transitions.create(['width', 'margin'], {
            easing: theme.transitions.easing.sharp,
            duration: theme.transitions.duration.enteringScreen,
        }),
    },
    menuButton: {
        marginLeft: 12,
        marginRight: 36,
    },
    hide: {
        display: 'none',
    },
    drawer: {
        width: drawerWidth,
        flexShrink: 0,
        whiteSpace: 'nowrap',
    },
    drawerOpen: {
        width: drawerWidth,
        transition: theme.transitions.create('width', {
            easing: theme.transitions.easing.sharp,
            duration: theme.transitions.duration.enteringScreen,
        }),
    },
    drawerClose: {
        transition: theme.transitions.create('width', {
            easing: theme.transitions.easing.sharp,
            duration: theme.transitions.duration.leavingScreen,
        }),
        overflowX: 'hidden',
        width: theme.spacing.unit * 7 + 1,
        [theme.breakpoints.up('sm')]: {
            width: theme.spacing.unit * 9 + 1,
        },
    },
    toolbar: {
        display: 'flex',
        alignItems: 'center',
        justifyContent: 'flex-end',
        padding: '0 8px',
        ...theme.mixins.toolbar,
    },
    content: {
        flexGrow: 1,
        padding: theme.spacing.unit * 3,
    },
    linearProgress: {
        position: 'absolute',
        left: 0,
        zIndex: 1400,
        width: '100%',
        top: 0
    },
    darken: {
        backgroundColor: "rgba(0, 0, 0, 0.5)",
        position: 'fixed',
        zIndex: 1300,
        left: 0,
        right: 0,
        bottom: 0,
        top: 5,
    },
    grow: {
        flexGrow: 1,
    },
    logo: {
        maxHeight: 0,
        "@media (min-width:450px)" : {
            maxHeight: theme.mixins.toolbar.minHeight - (theme.spacing.unit * 4),
        }
    },
    rightGutter: {
        paddingRight: theme.mixins.gutters().paddingRight,
        "@media (min-width:600px)": {
            paddingRight: theme.mixins.gutters()["@media (min-width:600px)"].paddingRight,

        }
    }
})};

export const BuyGiftDialogStyle = theme => ({
    formControl: {
        margin: theme.spacing.unit * 3,
    },
    group: {
        margin: `${theme.spacing.unit}px 0`,
    },
    root: {
        display: 'flex',
    },
    textField: {
        margin: theme.spacing.unit * 2,
        width: 200,
    },
    root2: {
        flexDirection: 'column',
        justifyContent: 'center',
        alignItems: 'center',
        padding: 0,
    }
});

export const BuyGiftPremiumDialogStyle = theme => ({
    root: {
        display: 'flex',
    },
    textField: {
        margin: theme.spacing.unit,
        width: "80%",
    }
});

export const CatalogManagerStyle = theme => ({
    typography: {
        padding: theme.spacing.unit,
    },
    holder: {
        display: 'flex',
        justifyContent: 'center',
    },
    warn: {
        color: theme.palette.getContrastText(red['A700']),
        backgroundColor: red['A700'],
        '&:hover': {
            backgroundColor: red[900],
        }
    },
    extendedIcon: {
        marginRight: theme.spacing.unit,
    },
    button: {
        margin: theme.spacing.unit,
    },
    elements: {
        display: 'flex',
        alignItems: 'center',
        padding: theme.spacing.unit,
    },
    grow: {
        flexGrow: 1,
    }
});

export const ChartsStyle = theme => ({
    root: {
        flexGrow: 1,
    },
    avatarRed: {
        backgroundColor: red[500],
    },
    avatarGreen: {
        backgroundColor: green[500],
    },
    avatarBlue: {
        backgroundColor: blue[500],
    },
    card: {
        padding: theme.spacing.unit * 2,
    },
    button: {
        margin: theme.spacing.unit,
    },
});

export const ContentListStyle = theme => ({
    root: {
        width: "100%",
    },
    heading: {
        fontSize: theme.typography.pxToRem(15),
        fontWeight: theme.typography.fontWeightRegular,
    },
    details: {
        alignItems: "center",
    }
});

export const CreatorAppStyle = theme => ({
    notification: {
        color: theme.palette.text.secondary,
        backgroundColor: theme.palette.secondary.main,
        borderLeft: "8px solid " + theme.palette.secondary.dark,
    },
    textField: {
        margin: theme.spacing.unit,
        width: 400,
    },
    container: {
        display: "flex",
        flexDirection: "row",
        alignItems: "center",
    },
    sameline: {
        marginTop: "-12px",
    },
    warn: {
        margin: theme.spacing.unit,
        color: theme.palette.getContrastText(red['A700']),
        backgroundColor: red['A700'],
        '&:hover': {
            backgroundColor: red[900],
        }
    },
    extendedIcon: {
        marginRight: theme.spacing.unit,
    },
    button: {
        margin: theme.spacing.unit,
    },
    grow: {
        flexGrow: 1,
    }
});
```

```

export const FeedbackFormStyle = theme => ({
    textField: {
        margin: theme.spacing.unit * 2,
        width: 100,
    },
    menu: {
        width: 100,
    },
});

export const LoginPageStyle = theme => ({
    textField: {
        margin: theme.spacing.unit,
        width: '200px',
    },
    grow: {
        flexGrow: 1,
    }
});

export const NewContentFormStyle = theme => ({
    root: {
        ...theme.mixins.gutters(),
        display: 'flex',
        flexWrap: 'wrap',
        flexDirection: 'row',
        alignItems: 'center',
        paddingTop: theme.spacing.unit * 3,
        paddingBottom: theme.spacing.unit * 3,
        height: 'auto',
        margin: theme.spacing.unit * 2,
    },
    textField: {
        margin: theme.spacing.unit,
        width: 180,
    },
    priceField: {
        marginTop: theme.spacing.unit,
        marginLeft: theme.spacing.unit,
        marginBottom: theme.spacing.unit,
        marginRight: 5,
        width: 95,
    },
    currency: {
        marginTop: theme.spacing.unit,
        marginRight: theme.spacing.unit,
        marginBottom: theme.spacing.unit,
        marginLeft: 0,
        width: 80,
    },
    menu: {
        width: 80,
    },
    formDiv: {
        display: "flex",
        alignItems: "center",
        flexWrap: "wrap",
    },
    buttonProgressPrimary: {
        color: theme.palette.secondary.main,
        position: 'absolute',
        top: '50%',
        left: '50%',
        marginTop: -12,
        marginLeft: -12,
    },
    wrapper: {
        display: "inline-block",
        float: "right",
        margin: theme.spacing.unit,
        position: 'relative',
    },
});


export const TransactionConfirmationStyle = theme => ({
    buttonOk: {
        margin: theme.spacing.unit,
    },
    buttonNo: {
        margin: theme.spacing.unit,
        color: red[700],
    }
});

export const UserAppStyle = theme => ({
    notification: {
        color: theme.palette.text.secondary,
        backgroundColor: theme.palette.secondary.main,
        borderLeft: "8px solid " + theme.palette.secondary.dark,
    },
    button: {
        margin: theme.spacing.unit,
    },
    searchField: {
        width: 400,
        margin: theme.spacing.unit,
    },
    sameline: {
        marginTop: "-12px",
    },
    container: {
        display: "flex",
        flexDirection: "row",
        alignItems: "center",
    },
    search: {
        position: 'relative',
        borderRadius: theme.shape.borderRadius,
        backgroundColor: fade(theme.palette.common.white, 0.15),
        '&:hover': {
            backgroundColor: fade(theme.palette.common.white, 0.25),
        },
        marginRight: theme.spacing.unit * 2,
        marginLeft: 0,
        width: '100%',
        [theme.breakpoints.up('sm')]: {
            marginLeft: theme.spacing.unit * 3,
            width: 'auto',
        },
    },
    searchIcon: {
        width: theme.spacing.unit * 5,
        height: '100%',
        position: 'absolute',
        pointerEvents: 'none',
        display: 'flex',
        alignItems: 'center',
        justifyContent: 'center',
    },
    inputRoot: {
        color: 'inherit',
        width: '100%',
    },
    inputInput: {
        paddingTop: theme.spacing.unit,
        paddingRight: theme.spacing.unit,
        paddingBottom: theme.spacing.unit,
        paddingLeft: theme.spacing.unit * 6,
        transition: theme.transitions.create('width'),
        width: '100%',
        [theme.breakpoints.up('md')]: {
            width: 200,
        },
    },
    grow: {
        flexGrow: 1,
    },
    subheading: {
        marginTop: 0,
        width: "100%",
        display: "flex",
        alignItems: "center",
    },
    subMargin: {
        margin: theme.spacing.unit,
        marginTop: 0,
    },
    normalMargin: {
        margin: theme.spacing.unit,
    },
    prFetch: {
        color: theme.palette.secondary.main,
    },
    premium: {
        color: green.A400,
    },
    notPremium: {
        color: red.A400,
    },
    extendedIcon: {
        marginRight: theme.spacing.unit,
    },
});

export const Web3LoginFormStyle = theme => ({
    root: {
        ...theme.mixins.gutters(),
        display: 'flex',
        flexWrap: 'wrap',
        flexDirection: 'column',
        justifyContent: 'center',
        alignItems: 'center',
        paddingTop: theme.spacing.unit * 2,
        paddingBottom: theme.spacing.unit * 2,
        height: 'auto',
        //minHeight: 500,
        "@media (min-height:550px)" : {
            minHeight: 500,
        }
    },
    container: {
        width: 'auto',
        display: 'flex',
        flexWrap: 'wrap',
        justifyContent: 'center',
    },
    textField: {
        margin: theme.spacing.unit * 2,
        width: 200,
    },
    menu: {
        width: 250,
    },
    extendedIcon: {
        marginRight: theme.spacing.unit,
    },
    button: {
        margin: 0,
    },
    buttonProgressPrimary: {
        color: theme.palette.secondary.main,
        position: 'absolute',
        top: '50%',
        left: '50%',
        marginTop: -12,
        marginLeft: -12,
    },
    buttonProgressSecondary: {
        color: theme.palette.primary.main,
        position: 'absolute',
        top: '50%',
        left: '50%',
        marginTop: -12,
        marginLeft: -12,
    },
    wrapper: {
        marginLeft: theme.spacing.unit,
        marginRight: theme.spacing.unit,
        position: 'relative',
    },
    fabProgress: {
        color: theme.palette.primary.main,
        position: 'absolute',
        top: 2,
        left: 2,
        zIndex: 1,
    },
    image: {
        maxWidth: 0,
        "@media (min-height:450px)" : {
            maxWidth: 200,
        }
    }
});

```

```
/*
 *
 *  University di Pisa - Master's Degree in Computer Science and Networking
 *
 *  Final Project for the course of Peer to Peer Systems and Blockchains
 *
 *  Teacher: Prof. Laura Ricci
 *
 *  Candidate: Orlando Leombruni, matricola 475727
 *
 *  File: CatalogManager.js
 *
*/

import React from 'react';
import classNames from 'classnames';
import { withStyles } from '@material-ui/core/styles';
import PropTypes from 'prop-types';
import { Typography,
         Button,
         TextField,
         Slide,
         Dialog,
         DialogContent,
         DialogActions,
         DialogTitle } from '@material-ui/core';
import { LibraryAdd, Delete } from '@material-ui/icons';
import Web3LoginForm from "./Web3LoginForm";
import json from '../solidity/compiled.json';
import TransactionConfirmation from "./TransactionConfirmation";
import AppDrawer from './AppDrawer'
import {getTransactionParameters, makeTitle} from "../Utils";
import { CatalogManagerStyle as styles } from "../styles/MaterialCustomStyles";

/*
 * "Slide up" transition for the animation of a React component.
 */
const Transition = (props) =>
    <Slide direction="up" {...props} />;

/*
 * CatalogManager Class
 *
 * A React Component that provides a very simple interface for creating and deleting Catalog contracts.
 */
class CatalogManager extends React.Component {

    constructor(props) {
        super(props);
        this.state = {
            loggedAccount: null,
            message: "",
            dialogOpen: false,
            catalogToClose: "",
            error: false,
            confirm: false,
            confirmProps: {},
            isClose: false,
        };
        this.abi = JSON.parse(json.contracts["Catalog.sol:Catalog"].abi);
        this.bin = "0x" + json.contracts["Catalog.sol:Catalog"].bin;
        this.props.manageCenter(true);
    }

    /*
     * A callback function invoked when the user has chosen and unlocked an EOA from which to perform Catalog operations.
     */
    unlockCallback = account => {
        this.setState(oldState => ({...oldState, loggedAccount: account}));
        this.props.manageCenter(false);
    };

    /*
     * Begins the creation of a new Catalog contract, estimating the gas cost for the operation.
     */
    createCatalog = () => {
        const { web3 } = this.props;
        const Catalog = new web3.eth.Contract(this.abi, {data: this.bin});
        const d = Catalog.deploy();
        this.setState(oldState => ({...oldState, isClose: false, dialogOpen: false, error: false}));
        getTransactionParameters(web3, d, this.state.loggedAccount).then(
            (result) => this.setState(o => ({...o, confirmProps: result, confirm: true})),
            (error) => console.log(error)
        );
    };

    /*
     * Invoked when the user confirms the transaction (after reviewing it in an informative dialog), this function
     * effectively deploys a new Catalog and then queries it to check if it's been correctly created.
     */
    confirmCreate = () => {
        const {web3} = this.props;
        const Catalog = new web3.eth.Contract(this.abi, {data: this.bin});
        Catalog.deploy().send({
            from: this.state.loggedAccount,
            gasPrice: this.state.confirmProps.gasPrice,
            gas: this.state.confirmProps.gas,
        }).then(
            (result) => {
                result.methods.isActiveCatalog().call({from: this.state.loggedAccount})
                    .then(
                        () => this.setState(oldState => ({...oldState,
                            message: `Catalog created at address ${result.options.address}`})
                        ),
                        () => this.setState(oldState => ({...oldState, message: `Error in creating catalog...`}))
                    );
            },
            (error) => {
                this.setState(oldState => ({...oldState, message: `Error in creating catalog...`}));
                console.log(error)
            }
        );
        this.setState(oldState => ({...oldState, message: "Creating catalog...", confirm: false, confirmProps: {}}));
    };

    /*
     * Shows (or hides) the dialog window where the user can input the address of the Catalog to close.
     */
    handleDialog = () =>
```

```
        this.setState(oldState => ({...oldState, dialogOpen: !oldState.dialogOpen}));

    /*
     * Handles the input of the address of the Catalog to close.
     */
    handleChange = event => {
        const { target } = event;
        this.setState(oldState => ({...oldState, catalogToClose: target.value}));
    };

    /*
     * Begins the closing and deletion of a Catalog contract, estimating the gas cost of the operation.
     */
    deleteCatalog = () => {
        let error = (this.state.catalogToClose === "");
        const { web3 } = this.props;
        const abi = JSON.parse(json.contracts["Catalog.sol:Catalog"].abi);
        const Catalog = new web3.eth.Contract(abi);
        try {
            Catalog.options.address = this.state.catalogToClose;
        } catch (e) {
            error = true;
            console.log(e);
        }
        if (!error) {
            this.setState(oldState => ({...oldState, isClose: true, dialogOpen: false, error: false}));
            getTransactionParameters(web3, Catalog.methods.closeCatalog(), this.state.loggedAccount).then(
                (result) => this.setState(o => ({...o, confirmProps: result, confirm: true})),
                (error) => console.log(error)
            );
        } else {
            this.setState(oldState => ({...oldState, error}));
        }
    };

    /*
     * Invoked when the user confirms the transaction (after reviewing it in an informative dialog), this function
     * checks that the Catalog is valid and active and then proceeds to close and delete it.
     */
    confirmDelete = () => {
        const {web3} = this.props;
        const abi = JSON.parse(json.contracts["Catalog.sol:Catalog"].abi);
        const Catalog = new web3.eth.Contract(abi, this.state.catalogToClose);
        Catalog.methods.isActiveCatalog().call({from: this.state.loggedAccount})
            .then(
                () => {
                    Catalog.methods.closeCatalog().send({
                        from: this.state.loggedAccount,
                        gasPrice: this.state.confirmProps.gasPrice,
                        gas: this.state.confirmProps.gas,
                    }).then(
                        (result) => {
                            Catalog.methods.isActiveCatalog().call({from: this.state.loggedAccount})
                                .then(
                                    () => this.setState(oldState => ({...oldState,
                                        message: `Catalog WAS NOT closed! Make sure you're logged in as the owner.`})
                                    ),
                                    () => this.setState(oldState => ({...oldState, message: `Catalog closed!`}))
                                );
                        },
                        (error) => {
                            this.setState(oldState => ({...oldState, message: `Error in closing catalog`}));
                            console.log(error)
                        }
                    );
                },
                () => this.setState(oldState => ({...oldState, message: `Catalog has already been closed!`}))
            );
        this.setState(oldState => ({...oldState, message: "Closing catalog...", confirm: false, confirmProps: {}}));
    };

    render() {
        const {web3, classes, noLogin} = this.props;
        const { loggedAccount, message, dialogOpen, catalogToClose, error, confirm, confirmProps, isClose} = this.state;

        return (
            <AppDrawer drawer={false} loading={false} writeTitle={() => makeTitle("Catalog Management", classes.grow)} render={() => ""} >
                {Boolean(loggedAccount) ?
                <div>
                    <div className={classes.holder}>
                        <div className={classes.elements}>
                            <Typography variant={"body1"} color={"textPrimary"} className={classes.typography}>
                                Deploy a new Catalog
                            </Typography>
                            <Button variant={"extendedFab"} onClick={this.createCatalog} color={"secondary"} className={classes.button}>
                                <LibraryAdd className={classes.extendedIcon} />
                                Add New
                            </Button>
                        </div>
                        <div className={classes.elements}>
                            <Typography variant={"body1"} color={"textPrimary"} className={classes.typography}>
                                Remove a Catalog
                            </Typography>
                            <Button variant={"extendedFab"}
                                    color={"inherit"}
                                    className={classNames(classes.button, classes.warn)}
                                    onClick={this.handleDialog} >
                                <Delete className={classes.extendedIcon} />
                                Delete
                            </Button>
                        </div>
                    </div>
                    <div className={classes.holder}>
                        <Typography variant={"body1"} color={"textPrimary"} className={classes.typography} >
                            {message}
                        </Typography>
                    </div>
                    <Dialog
                        open={dialogOpen}
                        keepMounted
                        TransitionComponent={Transition}
                        onBackdropClick={this.handleDialog}
                        onEscapeKeyDown={this.handleDialog} >
                        <DialogTitle>Select Catalog to close</DialogTitle>
                        <DialogContent>
                            <TextField
                                id="catalogAddress"
                                required
                                name="Catalog"
                                label="Catalog"
                                className={classes.textField}
                                value={catalogToClose}
                                onChange={this.handleChange}
                                error={error}
                                helperText={error ? "Invalid COBrA Catalog address" : ""}
                            />
                        </DialogContent>
                        <DialogActions>
                            <Button variant={"extendedFab"}
                                    color={"inherit"}
                                    className={classNames(classes.button, classes.warn)}
                                    onClick={this.deleteCatalog} >
                                Delete
                            </Button>
                        </DialogActions>
                    </Dialog>
                    {confirm ?
                        <TransactionConfirmation
                            {...confirmProps}
                            ok={isClose? this.confirmDelete : this.confirmCreate}
                            cancel={() => this.setState(oldState => ({...oldState, confirm: false, confirmProps: {}}))} /> :
                        ""}
                </div> :
                <Web3LoginForm noLogin={noLogin} web3={web3} onUnlock={this.unlockCallback}/>}
            </AppDrawer>
        );
    }
}

CatalogManager.propTypes = {
    noLogin: PropTypes.bool.isRequired,
    web3: PropTypes.object.isRequired,
    classes: PropTypes.object.isRequired,
    manageCenter: PropTypes.func.isRequired,
};

export default withStyles(styles)(CatalogManager);
```

```
/*
 *
 *  University di Pisa - Master’s Degree in Computer Science and Networking
 *
 *  Final Project for the course of Peer to Peer Systems and Blockchains
 *
 *  Teacher: Prof. Laura Ricci
 *
 *  Candidate: Orlando Leombruni, matricola 475727
 *
 *  File: LoginPage.js
 *
 */

import React from ’react’;
import PropTypes from ’prop-types’;
import {
    withStyles,
    Dialog,
    DialogTitle,
    DialogContent,
    DialogContentText,
    DialogActions,
    TextField,
    Slide,
    Button,
} from ’@material-ui/core’;
import ’../styles/login.css’
import Web3LoginForm from "./Web3LoginForm";
import json from ’../solidity/compiled.json’;
import AppDrawer from ’./AppDrawer’
import {makeTitle} from "../Utils";
import { LoginPageStyle as styles } from "../styles/MaterialCustomStyles";

/*
 * "Slide up" transition for the animation of a React component.
 */
const Transition = (props) =>
    <Slide direction="up" {...props} />;

/*
 * LoginPage Class
 *
 * A React Component that shows a login form to the user, allowing them also to select a Catalog contract to connect to.
 */
class LoginPage extends React.Component {

    constructor(props) {
        super(props);
        this.state = {
            logged: false,
            loggedAccount: null,
            catalogAccount: "",
            error: false,
        };
        this.props.manageCenter(true);
    }

    // Changes this component’s state when the Web3 instance unlocks the selected EOA.
    unlockCallback = account => {
            if (account === "error") this.props.onSubmit("error", "error", "error");
            else this.setState(oldState => ({...oldState, loggedAccount: account, logged: true}))
        };

    /*
     * After the desired EOA is unlocked by the Web3 instance and a Catalog account is chosen, this function checks that
     * latter is a valid Catalog and then "reports" to the enclosing component the user EOA, the Catalog account and
     * which app has been selected (Creator or User).
     */
    report = appType => () => {
        this.setState(oldState => ({...oldState, error: false}));
        let error = (this.state.catalogAccount === "");
        const { web3 } = this.props;
        const abi = JSON.parse(json.contracts["Catalog.sol:Catalog"].abi);
        const Catalog = new web3.eth.Contract(abi);
        try {
            Catalog.options.address = this.state.catalogAccount;
        } catch (e) {
            error = true;
        }
        if (!error) {
            Catalog.methods.isActiveCatalog().call({from: this.state.loggedAccount})
                .then(
                    (result) => {
                        this.props.onSubmit(this.state.loggedAccount, this.state.catalogAccount, appType)
                    },
                    (error) => {
                        console.log(error);
                        this.setState(oldState => ({...oldState, error: true}));
                    }
                );
        } else {
            this.setState(oldState => ({...oldState, error}));
        }
    };

    // Handles user changes in the form.
    handleChange = event => {
        const { target } = event;
        this.setState(oldState => ({...oldState, catalogAccount: target.value}));
    };

    render() {
        const { web3, classes, noLogin } = this.props;
        const { logged, catalogAccount, error } = this.state;

        return (
            <AppDrawer loading={false} render={() => ""} drawer={false} writeTitle={() => makeTitle("Login", classes.grow)}>
                {(logged) ?
                <Dialog
                    open={true}
                    disableBackdropClick={true}
                    disableEscapeKeyDown={true}
                    TransitionComponent={Transition}>
                    <DialogTitle>{"Logged in successfully!"}</DialogTitle>
                    <DialogContent>
                        <DialogContentText>
                            COBrA kai successfully connected to the Ethereum network.
                            Please insert the desired Catalog’s address, then
                            choose if you want to use the service as an user or as a content creator.
                        </DialogContentText>
                        <TextField
                            id="catalogAddress"
                            required
                            name="Catalog"
                            label="Catalog"
                            className={classes.textField}
                            value={catalogAccount}
                            onChange={this.handleChange}
                            error={error}
                            helperText={error ? "Invalid COBrA Catalog address" : ""}
                        />
                    </DialogContent>
                    <DialogActions>
                        <Button color={"secondary"} onClick={this.report("user")}>
                            User
                        </Button>
                        <Button color={"secondary"} onClick={this.report("creator")}>
                            Creator
                        </Button>
                    </DialogActions>
                </Dialog> :
                <Web3LoginForm noLogin={noLogin} onUnlock={this.unlockCallback} web3={ web3 }/>}
            </AppDrawer>
        );
    }
}

LoginPage.propTypes = {
    noLogin: PropTypes.bool.isRequired,
    web3: PropTypes.object.isRequired,
    classes: PropTypes.object.isRequired,
    onSubmit: PropTypes.func.isRequired,
    manageCenter: PropTypes.func.isRequired,
};

export default withStyles(styles)(LoginPage);
```

```
/*
 *
 *  University di Pisa - Master's Degree in Computer Science and Networking
 *
 *  Final Project for the course of Peer to Peer Systems and Blockchains
 *
 *  Teacher: Prof. Laura Ricci
 *
 *  Candidate: Orlando Leombruni, matricola 475727
 *
 *  File: AppDrawer.js
 *
 */

import React from 'react';
import PropTypes from 'prop-types';
import classNames from 'classnames';
import {
    withStyles,
    Drawer,
    AppBar,
    Toolbar,
    CssBaseline,
    IconButton,
    LinearProgress,
} from '@material-ui/core';
import {
    Menu as MenuIcon,
    ChevronLeft as ChevronLeftIcon,
    ChevronRight as ChevronRightIcon,
} from '@material-ui/icons';
import secondaryLogo from '../assets/logo2.png';
import { AppDrawerStyle as styles } from "../styles/MaterialCustomStyles";

/*
 * AppDrawer Class
 *
 * A React Component that provides a title bar and (optionally) a drawer menu to a page Component.
 */

class AppDrawer extends React.Component {
    state = {
        open: false,
    };

    /*
     * Manage the opening and closing of the drawer. Functions used as callbacks for the onClick properties of buttons.
     */
    handleDrawerOpen = () => {
        this.setState({ open: true });
    };
    handleDrawerClose = () => {
        this.setState({ open: false });
    };

    render() {
        const { classes, theme, drawer, writeTitle, loading, children } = this.props;

        return (
            <div className={classes.root}>
                <CssBaseline />
                <AppBar
                    position="fixed"
                    className={classNames(classes.appBar, {
                        [classes.appBarShift]: this.state.open,
                    })}>
                    <Toolbar className={classes.rightGutter} disableGutters={drawer && !this.state.open}>
                        {drawer &&
                            <IconButton
                                color="inherit"
                                aria-label="Open drawer"
                                onClick={this.handleDrawerOpen}
                                className={classNames(classes.menuButton, {
                                    [classes.hide]: this.state.open,
                                })}>
                                <MenuIcon />
                            </IconButton>}
                        {writeTitle()}
                        <img src={secondaryLogo} alt={"logo"} className={classes.logo} />
                    </Toolbar>
                </AppBar>
                {drawer? <Drawer
                    variant="permanent"
                    className={classNames(classes.drawer, {
                        [classes.drawerOpen]: this.state.open,
                        [classes.drawerClose]: !this.state.open,
                    })}
                    classes={{
                        paper: classNames({
                            [classes.drawerOpen]: this.state.open,
                            [classes.drawerClose]: !this.state.open,
                        }),
                    }}
                    open={this.state.open}
                >
                    <div className={classes.toolbar}>
                        <IconButton onClick={this.handleDrawerClose}>
                            {theme.direction === 'rtl' ? <ChevronRightIcon /> : <ChevronLeftIcon />}
                        </IconButton>
                    </div>
                    {this.props.render(this.state)}
                </Drawer> : null}
                <main className={classes.content}>
                    {loading && <LinearProgress color={"secondary"} className={classes.linearProgress}/>}
                    {loading && <div className={classes.darken} />}
                    <div className={classes.toolbar} />
                        {children}
                </main>
            </div>
        );
    }
}

AppDrawer.propTypes = {
    classes: PropTypes.object.isRequired,
    theme: PropTypes.object.isRequired,
    drawer: PropTypes.bool.isRequired,
    render: PropTypes.func.isRequired,
    writeTitle: PropTypes.func.isRequired,
    loading: PropTypes.bool.isRequired,
};

export default withStyles(styles, { withTheme: true })(AppDrawer);
```

```javascript
/*
 *
 *  University di Pisa - Master's Degree in Computer Science and Networking
 *
 *  Final Project for the course of Peer to Peer Systems and Blockchains
 *
 *  Teacher: Prof. Laura Ricci
 *
 *  Candidate: Orlando Leombruni, matricola 475727
 *
 *  File: Web3LoginForm.js
 *
 */

import React from 'react';
import { withStyles } from '@material-ui/core/styles';
import PropTypes from 'prop-types';
import { Grid,
         Typography,
         TextField,
         MenuItem,
         FormControl,
         InputLabel,
         Input,
         InputAdornment,
         IconButton,
         Button,
         CircularProgress,
         Paper } from '@material-ui/core';
import { Visibility,
         VisibilityOff,
         Update,
         LockOpen } from "@material-ui/icons";
import mainLogo from '../assets/logo.png';
import { Web3LoginFormStyle as styles } from "../styles/MaterialCustomStyles";

/*
 * Web3LoginForm Class
 *
 * A React Component that queries the Web3 instance for the list of available EOAs and allows the user to unlock one
 * of them in order to be used as the main account for performing operations on the blockchain.
 */
class Web3LoginForm extends React.Component {

    constructor(props) {
        super(props);
        this.state = {
            loading: false,
            account: "",
            password: "",
            accounts: [],
            loaded: false,
            bottomText: "",
            showPassword: false,
        };
    }

    /*
     * This is a React state function that will be called only once, after the component is mounted in the virtual
     * DOM but before it gets rendered.
     *
     * In particular, this function asks the underlying Web3 instance for the list of available EOAs.
     */
    componentDidMount() {
        const { web3, noLogin, onUnlock } = this.props;
        if (noLogin) {
            web3.eth.getAccounts().then(
                (list) => {
                    if (list.length > 0) onUnlock(list[0]);
                    else onUnlock("error");
                },
                (err) => console.log(err)
            );
        } else {
            web3.eth.getAccounts().then(this.showAccounts, (err) => console.log(err));
        }
    }

    /*
     * Populates the list data structure containing the available EOAs.
     */
    showAccounts = accounts => {
        if (accounts && accounts.length > 0) {
            const accountList = accounts.map((account, index) => ({
                    label: `Account #${index+1} (${account.replace(account.slice(5, -3), "...")})`,
                    value: account,
                })
            );
            this.setState(oldState => ({...oldState, accounts: accountList, loaded: true}));
        } else {
            this.setState(oldState => ({...oldState, accounts: [{label: "No accounts found", value: null}]}));
        }
    };

    /*
     * Submits the form; asks the Web3 instance to unlock the selected account, then reports back to the
     * enclosing component.
     */
    submit = event => {
        event.preventDefault();
        const { account, password } = this.state;
        if (!account || !password) return;
        const { web3 } = this.props;
        web3.eth.personal.unlockAccount(account, password, 0).then(
            () => {
                this.setState(oldState => ({...oldState, loading: false}));
                this.props.onUnlock(account);
            },
            (error) => {
                console.log(error);
                this.setState(oldState => ({...oldState, loading: false, bottomText: "Wrong username and/or password"}))
            }
        );
        this.setState(oldState => ({...oldState, loading: true}));
    };

    // Handles the show/hide password button.
    handleShowPassword = () => {
        this.setState(oldState => ({...oldState, showPassword: !oldState.showPassword}));
    };

    // Handles changes made in the form.
    changedForm = event => {
        const { target } = event;
        this.setState(oldState => ({...oldState, [target.name]: target.value}));
    };

    render() {
        const { classes } = this.props;
        const { account, password, accounts, showPassword, loading, loaded } = this.state;

        return (
            <Paper className={classes.root} id="paper" elevation={1}>
                <Grid
                    container
                    spacing={0}
                    display="flex"
                    direction="column"
                    alignItems="center"
                    justify="center">
                    <Grid item style={{width: "fit-content"}} ><img src={mainLogo} alt={"logo"} className={classes.image} /></Grid>
                    <Grid item style={{margin: "8px"}}>
                        <Typography variant="h5" color="textPrimary">
                            Welcome! Please login.
                        </Typography>
                    </Grid>
                    {this.state.bottomText && <Grid item >
                        <Typography variant={"body1"} color={"error"} id="bottomText">{this.state.bottomText}&nbsp;</Typography>
                    </Grid>}
                    <Grid style={{display: "flex", alignItems: "center", flexDirection: "column"}} item xs={6}>
                        <form className={classes.container} noValidate autoComplete="off" onSubmit={this.submit}>
                                <TextField
                                    id="list-accounts"
                                    select
                                    required
                                    name="account"
                                    label="Account"
                                    className={classes.textField}
                                    value={account}
                                    onChange={this.changedForm}
                                    SelectProps={{MenuProps: {className: classes.menu}}}
                                    margin="normal" >
                                    {accounts.map(option => (
                                        <MenuItem key={option.value} value={option.value}>
                                            {option.label}
                                        </MenuItem>
                                    ))}
                                </TextField>
                            <FormControl required className={classes.textField}>
                                <InputLabel htmlFor="password">Password</InputLabel>
                                <Input
                                    id="password"
                                    name="password"
                                    value={password}
                                    onChange={this.changedForm}
                                    type={showPassword ? ’text’ : ’password’}
                                    endAdornment={
                                        <InputAdornment position="end">
                                            <IconButton
                                                aria-label="Toggle password visibility"
                                                onClick={this.handleShowPassword} >
                                                {showPassword ? <Visibility/> : <VisibilityOff/>}
                                            </IconButton>
                                        </InputAdornment>
                                    }
                                />
                            </FormControl>
                            <div style={{display: "flex", alignItems: "center"}}>
                            <div className={classes.wrapper}>
                                <IconButton
                                    onClick={() => {
                                        this.setState(oldState => ({...oldState, loaded: false}));
                                        this.props.web3.eth.getAccounts().then(this.showAccounts);
                                    }}
                                    disabled={!loaded}
                                    className={classes.button}>
                                    <Update/>
                                </IconButton>
                                {!loaded && <CircularProgress size={24} className={classes.buttonProgressSecondary}/>}
                            </div>
                            <div className={classes.wrapper}>
                                <Button
                                    variant="extendedFab"
                                    color="primary"
                                    aria-label="Unlock"
                                    onClick={this.submit}
                                    className={classes.button}
                                    disabled={loading || (!account || !password)}>
                                    <LockOpen className={classes.extendedIcon}/>
                                    Unlock
                                </Button>
                                {loading && <CircularProgress size={24} className={classes.buttonProgressPrimary}/>}
                            </div>
                            </div>
                        </form>
                    </Grid>
                </Grid>
            </Paper>
        );
    }
}



Web3LoginForm.propTypes = {
    noLogin: PropTypes.bool.isRequired,
    web3: PropTypes.object.isRequired,
    onUnlock: PropTypes.func.isRequired,
};


export default withStyles(styles)(Web3LoginForm);
```

```
/*
 *
 *  University di Pisa - Master's Degree in Computer Science and Networking
 *
 *  Final Project for the course of Peer to Peer Systems and Blockchains
 *
 *  Teacher: Prof. Laura Ricci
 *
 *  Candidate: Orlando Leombruni, matricola 475727
 *
 *  File: BuyGiftDialog.js
 *
 */

import React from 'react';
import PropTypes from 'prop-types';
import {
    withStyles,
    Dialog,
    DialogContent,
    DialogActions,
    DialogTitle,
    FormControl,
    FormControlLabel,
    DialogContentText,
    Radio,
    RadioGroup,
    TextField,
    Button,
} from '@material-ui/core';
import {prettifyWei} from "../Utils";
import {BuyGiftDialogStyle as styles} from "../styles/MaterialCustomStyles";

/*
 * BuyGiftDialog Class
 *
 * A React Component that displays a dialog window, reviewing the purchase of a Content and asking the user
 * if they want to buy it for themselves or gift it to another user.
 */
class BuyGiftDialog extends React.Component {

    state = {
        selected: 'me',
        account: '',
    };

    handleChange = event => {
        const { target } = event;
        this.setState(oldState => ({...oldState, selected: target.value}));
    };

    handleAccount = event => {
        const { target } = event;
        this.setState(oldState => ({...oldState, account: target.value}))
    };

    render() {
        const {classes, callback, cancel, price} = this.props;

        return (
            <div className={classes.root}>
            <Dialog
                open={true}
                keepMounted
                onBackdropClick={cancel}
                onEscapeKeyDown={cancel}>
                <DialogTitle>Purchase details</DialogTitle>
                <DialogContent>
                    <DialogContentText>You need to pay {prettifyWei(price)} to access this content.</DialogContentText>
                    <FormControl component={"fieldset"} className={classes.formControl}>
                        <RadioGroup
                            name={"recipient"}
                            className={classes.group}
                            value={this.state.selected}
                            onChange={this.handleChange}>
                            <FormControlLabel control={<Radio/>} label={"Buy it for me"} value={"me"}/>
                            <FormControlLabel control={<Radio/>} label={
                                <div style={{display: "flex", alignItems: "center"}}>
                                    Buy it for someone else:
                                <TextField
                                    name={"account"}
                                    type={"string"}
                                    placeholder={"Account address..."}
                                    value={this.state.account}
                                    className={classes.textField}
                                    onChange={event => this.handleAccount(event)}
                                    onClick={() => this.setState(oldState => ({...oldState, selected: 'other'}))}
                                />
                                </div>
                            } value={"other"}/>
                        </RadioGroup>
                    </FormControl>
                </DialogContent>
                <DialogActions>
                    <Button color={"inherit"} onClick={cancel}>Cancel</Button>
                    <Button color={"secondary"} onClick={this.state.selected === 'me' ? callback() : callback(this.state.account)}>Ok</Button>
                </DialogActions>
            </Dialog>
            </div>
        );
    }

}

BuyGiftDialog.propTypes = {
    classes: PropTypes.object.isRequired,
    callback: PropTypes.func.isRequired,
    cancel: PropTypes.func.isRequired,
    price: PropTypes.string.isRequired,
};

export default withStyles(styles)(BuyGiftDialog);
```

```
/*
 *
 *  University di Pisa - Master's Degree in Computer Science and Networking
 *
 *  Final Project for the course of Peer to Peer Systems and Blockchains
 *
 *  Teacher: Prof. Laura Ricci
 *
 *  Candidate: Orlando Leombruni, matricola 475727
 *
 *  File: ContentList.js
 *
 */
import React from 'react';
import PropTypes from 'prop-types';
import CopyToClipboard from 'react-copy-to-clipboard';
import { withStyles } from '@material-ui/core/styles';
import {
    ExpansionPanel,
    ExpansionPanelActions,
    ExpansionPanelDetails,
    ExpansionPanelSummary,
    Typography,
    Button,
    Tooltip
} from '@material-ui/core';
import { ExpandMore } from '@material-ui/icons';
import { ContentListStyle as styles } from "../styles/MaterialCustomStyles";

/*
 * CopyWrapper Class
 *
 * A React Component that wraps another component and makes it clickable; when clicked, the component's text
 * gets copied into the clipboard
 */
const CopyWrapper = ({condition, placement, what, onCopy, children}) => (
    condition ?
        <Tooltip title={"Click to copy"} placement={placement}>
            <CopyToClipboard text={what} onCopy={onCopy}>
                {children}
            </CopyToClipboard>
        </Tooltip> :
        <div>
            {children}
        </div>
);

/*
 * ContentList Class
 *
 * A React Component that displays a list where each item has several (displayable) attributes.
 * The component is completely generic and can be used in a multitude of ways.
 */

class ContentList extends React.Component {

    constructor(props) {
        super(props);
        this.divRef = React.createRef();
        this.handleResize = this.handleResize.bind(this);
    }


    state = {
        smallerBox: false,
    };

    /*
     * Function that handles the resizing of the browser window.
     */
    handleResize() {
        if (this.divRef) {
            this.setState({smallerBox: (this.divRef.getBoundingClientRect().width < 531)});
        }
    }

    // React lifecycle methods to add/remove a 'resize' event listener when the component gets mounted/unmounted.
    componentDidMount() {
        this.handleResize();
        window.addEventListener('resize', this.handleResize);
    }

    componentWillUnmount() {
        window.removeEventListener('resize', this.handleResize);
    }

    render() {
        const { classes, className, list, action, attributes, actionName, action2, action2Name, notify } = this.props;
        const percent = (100/attributes.length) + "%";
        return (
            <div className={className ? className : classes.root} ref={element => this.divRef = element}>
                {list.map((item, key) =>
                    <ExpansionPanel key={key}>
                        <ExpansionPanelSummary expandIcon={<ExpandMore/>}>
                            <Typography className={classes.heading}>{item.description}</Typography>
                        </ExpansionPanelSummary>
                        <ExpansionPanelDetails className={classes.details}>
                            {attributes.map((attr, index) => (
                                <div key={index} style={{flexBasis: percent}}>
                                    <Typography variant={"subtitle2"} style={{marginRight: 12}}>{attr.description}</Typography>
                                    <CopyWrapper
                                        condition={attr.copy}
                                        placement={"bottom"}
                                        what={item[attr.name]}
                                        onCopy={() => notify("Copied!", "Copied to clipboard", "success")}>
                                        <Typography style={{cursor: attr.copy? "pointer" : "default"}}>
                                            {attr.extra ?
                                                (this.state.smallerBox ?
                                                    attr.makeSmall(attr.extra(item[attr.name])) :
                                                    attr.extra(item[attr.name])) :
                                                (this.state.smallerBox ?
                                                    attr.makeSmall(item[attr.name]) :
                                                    item[attr.name])}
                                        </Typography>
                                    </CopyWrapper>
                                </div>
                            ))}
                        </ExpansionPanelDetails>
                            <ExpansionPanelActions>
                                {Boolean(action) &&
                                <Button size={"small"} color={"secondary"} onClick={action(item.index)}>
                                    {actionName}
                                </Button>}
                                {Boolean(action2) &&
                                <Button size={"small"} color={"default"} onClick={action2(item.index)}>
                                    {action2Name}
                                </Button>}
                            </ExpansionPanelActions>
                    </ExpansionPanel>
                ))}
            </div>
        );
    }

}

ContentList.propTypes = {
    classes: PropTypes.object.isRequired,
    list: PropTypes.arrayOf(PropTypes.object).isRequired,
    attributes: PropTypes.arrayOf(PropTypes.object).isRequired,
    action: PropTypes.func,
    actionName: PropTypes.string,
    action2: PropTypes.func,
    action2Name: PropTypes.string,
    notify: PropTypes.func.isRequired,
    className: PropTypes.string,
};

export default withStyles(styles)(ContentList);
```

```
/*
 *
 *  University di Pisa - Master’s Degree in Computer Science and Networking
 *
 *  Final Project for the course of Peer to Peer Systems and Blockchains
 *
 *  Teacher: Prof. Laura Ricci
 *
 *  Candidate: Orlando Leombruni, matricola 475727
 *
 *  File: Charts.js
 *
 */

import React from ’react’;
import PropTypes from ’prop-types’;
import {
    withStyles,
    Grid,
    Card,
    Button,
    CardHeader,
    CardContent,
    CardActions,
    Chip,
    Typography,
    Avatar,
    List,
    ListItem,
    ListItemText,
    Divider,
    Menu,
    MenuItem,
} from ’@material-ui/core’;
import {
    DonutSmall,
    FavoriteBorder,
    RecordVoiceOver,
    AttachMoney,
} from ’@material-ui/icons’;
import { ChartsStyle as styles } from "../styles/MaterialCustomStyles";

/*
 * Feedbacks Class
 *
 * A React Component that lists the top Contents for each feedback category.
 */

class Feedbacks extends React.Component {

    render() {
        const { avg, appreciation, fairness, suggest } = this.props;

        return (
            <div style={{flexGrow: 1}}>
                <Typography variant={"subtitle1"}>
                    Top rated contents
                </Typography>
                <List>
                    <Typography variant={"subtitle2"}>
                        Average
                    </Typography>
                    <ListItem>
                        <Avatar><DonutSmall/></Avatar>
                        <ListItemText primary={avg}/>
                    </ListItem>
                    <Typography variant={"subtitle2"}>
                        Appreciation
                    </Typography>
                    <ListItem>
                        <Avatar><FavoriteBorder/></Avatar>
                        <ListItemText primary={appreciation}/>
                    </ListItem>
                    <Typography variant={"subtitle2"}>
                        Price fairness
                    </Typography>
                    <ListItem>
                        <Avatar><AttachMoney/></Avatar>
                        <ListItemText primary={fairness}/>
                    </ListItem>
                    <Typography variant={"subtitle2"}>
                        Likeliness to suggest to other users
                    </Typography>
                    <ListItem>
                        <Avatar><RecordVoiceOver/></Avatar>
                        <ListItemText primary={suggest}/>
                    </ListItem>
                </List>
            </div>
        );
    }
}

Feedbacks.propTypes = {
    avg: PropTypes.string.isRequired,
    appreciation: PropTypes.string.isRequired,
    fairness: PropTypes.string.isRequired,
    suggest: PropTypes.string.isRequired,
};

// Default text shown while waiting for the actual results.
const defaultValue = "Loading...";

/*
 * Charts Class
 *
 * A React Component that shows to the user various information about Contents in the Catalog, such as the top rated
 * or the most recent ones.
 */
class Charts extends React.Component {

    constructor(props) {
        super(props);
        this.state.author.name = this.props.authors[0];
        this.state.genre.name = this.props.genres[0];
    }


    state = {
        global: {
            recentContents: [defaultValue, defaultValue, defaultValue, defaultValue, defaultValue],
            topRatedAvg: defaultValue,
            topRatedAppreciation: defaultValue,
            topRatedFairness: defaultValue,
            topRatedSuggest: defaultValue,
        },
        author: {
            name: "",
            mostRecent: defaultValue,
            mostViewed: defaultValue,
            topRatedAvg: defaultValue,
            topRatedAppreciation: defaultValue,
            topRatedFairness: defaultValue,
            topRatedSuggest: defaultValue,
        },
        genre: {
            name: "",
            mostRecent: defaultValue,
            mostViewed: defaultValue,
            topRatedAvg: defaultValue,
            topRatedAppreciation: defaultValue,
            topRatedFairness: defaultValue,
            topRatedSuggest: defaultValue,
        },
        authorAnchorEl: null,
        genreAnchorEl: null,
    };

    /*
     * These functions control the behaviour of the "change author/genre" menus.
     */
    showAuthors = event => {
        const target = event.currentTarget;
        this.setState(o => ({...o, authorAnchorEl: target}))
    };

    showGenres = event => {
        const target = event.currentTarget;
        this.setState(o => ({...o, genreAnchorEl: target}))
    };

    authorClose = () =>
        this.setState(o => ({...o, authorAnchorEl: null}));

    genreClose = () =>
        this.setState(o => ({...o, genreAnchorEl: null}));

    handleAuthorMenu = (event, index) => {
        this.setState(o => ({
            ...o,
            author: {
                name: this.props.authors[index],
                mostRecent: "Loading...",
                mostViewed: "Loading...",
                topRatedAvg: "Loading...",
                topRatedAppreciation: "Loading...",
                topRatedFairness: "Loading...",
                topRatedSuggest: "Loading...",
            }
        }), this.refreshAuthor);
    };

    handleGenreMenu = (event, index) => {
        this.setState(o => ({
            ...o,
            genre: {
                name: this.props.genres[index],
                mostRecent: "Loading...",
                mostViewed: "Loading...",
                topRatedAvg: "Loading...",
                topRatedAppreciation: "Loading...",
                topRatedFairness: "Loading...",
                topRatedSuggest: "Loading...",
            }
        }), this.refreshGenre);
    };

    /*
     * This function loads information about top/most recent contents of a specified genre.
     */
    refreshGenre = () => {
        this.setState(o => ({...o, genreAnchorEl: null}));
        const {Catalog, web3, account} = this.props;
        const genre = web3.utils.asciiToHex(this.state.genre.name);
        Catalog.methods.getLatestByGenre(genre).call({from: account}).then(
            (result) => {
                this.setState(o => ({...o, genre: {...o.genre, mostRecent: web3.utils.hexToAscii(result)}}));
            },
            (error) => {
                console.log(`Error in retrieving latest content with genre ${this.state.genre.name}`);
                this.setState(o => ({...o, genre: {...o.genre, mostRecent: "Error!"}}));
            }
        );
        Catalog.methods.getMostPopularByGenre(genre).call({from: account}).then(
            (result) => {
                this.setState(o => ({...o, genre: {...o.genre, mostViewed: web3.utils.hexToAscii(result)}}));
            },
            (error) => {
                console.log(`Error in retrieving most popular content with genre ${this.state.genre.name}`);
                this.setState(o => ({...o, genre: {...o.genre, mostViewed: "Error!"}}));
            }
        );
        Catalog.methods.getMostRatedByGenre(genre).call({from: account}).then(
            (result) => {
                this.setState(o => ({...o, genre: {...o.genre, topRatedAvg: web3.utils.hexToAscii(result)}}));
            },
            (error) => {
                console.log(`Error in retrieving most rated (average) content with genre ${this.state.genre.name}`);
                this.setState(o => ({...o, genre: {...o.genre, topRatedAvg: "Error!"}}));
            }
        );
        ["Appreciation", "Fairness", "Suggest"].forEach((category, idx) => {
            Catalog.methods.getMostRatedByGenre(genre, idx+1).call({from: account}).then(
                (result) => {
                    this.setState(o => ({...o, genre: {...o.genre, ["topRated" + category]: web3.utils.hexToAscii(result)}}));
                },
                (error) => {
                    console.log(`Error in retrieving most rated (${category}) content with genre ${this.state.genre.name}`);
                    this.setState(o => ({...o, genre: {...o.genre, ["topRated" + category]: "Error!"}}));
                }
            );
        });
    };

    /*
     * This function loads information about top/most recent contents of a specified author.
     */
    refreshAuthor = () => {
        this.setState(o => ({...o, authorAnchorEl: null}));
        const { Catalog, web3, account } = this.props;
        const author = web3.utils.asciiToHex(this.state.author.name);
        Catalog.methods.getLatestByAuthor(author).call({from: account}).then(
            (result) => {
                this.setState(o => ({...o, author: {...o.author, mostRecent: web3.utils.hexToAscii(result)}}));
            },
            (error) => {
                console.log(`Error in retrieving latest content with author ${this.state.author.name}`);
                this.setState(o => ({...o, author: {...o.author, mostRecent: "Error!"}}));
            }
        );
        Catalog.methods.getMostPopularByAuthor(author).call({from: account}).then(
            (result) => {
                this.setState(o => ({...o, author: {...o.author, mostViewed: web3.utils.hexToAscii(result)}}));
            },
            (error) => {
                console.log(`Error in retrieving most popular content with author ${this.state.author.name}`);
                this.setState(o => ({...o, author: {...o.author, mostViewed: "Error!"}}));
            }
        );
        Catalog.methods.getMostRatedByAuthor(author).call({from: account}).then(
            (result) => {
                this.setState(o => ({...o, author: {...o.author, topRatedAvg: web3.utils.hexToAscii(result)}}));
            },
            (error) => {
                console.log(`Error in retrieving most rated (average) content with author ${this.state.author.name}`);
                this.setState(o => ({...o, author: {...o.author, topRatedAvg: "Error!"}}));
            }
        );
        ["Appreciation", "Fairness", "Suggest"].forEach((category, idx) => {
            Catalog.methods.getMostRatedByAuthor(author, idx+1).call({from: account}).then(
                (result) => {
                    this.setState(o => ({...o, author: {...o.author, ["topRated" + category]: web3.utils.hexToAscii(result)}}));
                },
                (error) => {
                    console.log(`Error in retrieving most rated (${category}) content with author ${this.state.author.name}`);
                    this.setState(o => ({...o, author: {...o.author, ["topRated" + category]: "Error!"}}));
                }
            );
        });
    };

    /*
     * This is a React state function that will be called only once, after the component is mounted in the virtual
     * DOM but before it gets rendered.
     *
     * In particular, this function asks the Catalog contract for information about "global" (i.e. not author-
     * or genre-related) recent and top Contents, then loads the information about the default author's and genre's
     * Contents. (The default author and genre are usually the ones of the first Content inserted in the Catalog).
     */
    componentDidMount() {
        const { Catalog, web3, account } = this.props;
        Catalog.methods.getNewContentsList(5).call({from: account}).then(
            (result) => {
                const list = result.map(item => web3.utils.hexToString(item));
                if (list.length < 5) {
                    list.concat(new Array(5 - list.length).fill("N/A"));
                }
                this.setState(o => ({...o, global: {...o.global, recentContents: list}}))
            },
            (error) => {
                console.log("Error in retrieving recent contents");
                const list = [1,2,3,4,5].map(() => "Error!");
                this.setState(o => ({...o, global: {...o.global, recentContents: list}}))
            }
        );
        Catalog.methods.getMostRated().call({from: account}).then(
            (result) => {
                this.setState(o => ({...o, global: {...o.global, topRatedAvg: web3.utils.hexToString(result)}}))
            },
            (error) => {
                console.log("Error in retrieving top rated content (average)");
                this.setState(o => ({...o, global: {...o.global, topRatedAvg: "Error!"}}))
            }
        );
        ["Appreciation", "Fairness", "Suggest"].forEach((category, idx) => {
            Catalog.methods.getMostRated(idx+1).call({from: account}).then(
                (result) => {
                    this.setState(o => ({
                        ...o,
                        global: {
                            ...o.global,
                            ["topRated" + category]: web3.utils.hexToString(result),
                        }
                    }));
                },
                (error) => {
                    console.log(`Error in retrieving top rated content (${category})`);
                    this.setState(o => ({...o, global: {...o.global, ["topRated" + category]: "Error!"}}))
                }
            );
        });
        this.refreshAuthor();
        this.refreshGenre();
    }

    render() {
        const { classes } = this.props;
        const {
            global,
            author,
            genre,
            authorAnchorEl,
            genreAnchorEl,
        } = this.state;

        const authorOpen = Boolean(authorAnchorEl), genreOpen = Boolean(genreAnchorE
l);

        return (
            <Grid container className={classes.root} spacing={24}>
                <Grid item xs={12}>
                    <Card className={classes.card}>
                        <CardHeader
                            avatar={<Avatar className={classes.avatarRed}>O</Avatar>}
                            title={"Overview"}
                            subheader={"Information about all contents"}
                        />
                        <CardContent>
                            <div style={{display: 'flex', flexDirection: 'row'}}>
                                <div style={{flexGrow: 1}}>
                                    <Typography variant={"subtitle1"}>
                                        Latest contents added to the catalog
                                    </Typography>
                                    <List>
                                        {global.recentContents.map((item, idx) => (
                                            <ListItem key={idx}>
                                                <Avatar>{idx+1}</Avatar>
                                                <ListItemText primary={item}/>
                                            </ListItem>
                                        ))}
                                    </List>
                                </div>
                                <Divider/>
                                <Feedbacks
                                    avg={global.topRatedAvg}
                                    appreciation={global.topRatedAppreciation}
                                    fairness={global.topRatedFairness}
                                    suggest={global.topRatedSuggest}
                                />
                            </div>
                        </CardContent>
                    </Card>
                </Grid>
                <Grid item xs={6}>
                    <Card className={classes.card}>
                        <CardHeader
                            avatar={<Avatar className={classes.avatarGreen}>A</Avatar>}
                            title={"Author"}
                            subheader={"Information about contents by a specific author"}
                            action={
                                <Chip
                                    onClick={this.showAuthors}
                                    variant={"outlined"}
                                    color={"secondary"}
                                    label={author.name}
                                    avatar={<Avatar>{author.name[0].toUpperCase()}</Avatar>} />
                            }
                        />
                        <Menu
                            id={"-menu"}
                            anchorEl={authorAnchorEl}
                            open={authorOpen}
                            onClose={this.authorClose}
                            PaperProps={{
                                style: {
                                    maxHeight: 48 * 4.5,
                                    width: 200,
                                }
                            }}>
                            {this.props.authors.map((item, idx) => (
                                <MenuItem key={item} onClick={event => this.handleAuthorMenu(event, idx)}>
                                    {item}
                                </MenuItem>
                            ))}
                        </Menu>
                        <CardContent>
                            <Grid container>
                                <Grid item xs={6}>
                                    <Typography variant={"subtitle1"}>
                                        Latest
                                    </Typography>
                                    <List>
                                        <ListItem>
                                            <Avatar>{author.mostRecent[0].toUpperCase()}</Avatar>
                                            <ListItemText primary={author.mostRecent}/>
                                        </ListItem>
                                    </List>
                                </Grid>
                                <Grid item xs={6}>
                                    <Typography variant={"subtitle1"}>
                                        Most viewed
                                    </Typography>
                                    <List>
                                        <ListItem>
                                            <Avatar>{author.mostViewed[0].toUpperCase()}</Avatar>
                                            <ListItemText primary={author.mostViewed}/>
                                        </ListItem>
                                    </List>
                                </Grid>
                                <Grid item xs={12}>
                                    <Feedbacks
                                        avg={author.topRatedAvg}
                                        appreciation={author.topRatedAppreciation}
                                        fairness={author.topRatedFairness}
                                        suggest={author.topRatedSuggest}/>
                                </Grid>
                            </Grid>
                        </CardContent>
                        <CardActions style={{display: "block", width: "100%"}}>
                            <Button
                                className={classes.button}
                                variant={"contained"}
                                onClick={() => this.props.addSubAuthor(author.name)}>
                                Notify about new content by {author.name}
                            </Button>
                        </CardActions>
                    </Card>
                </Grid>
                <Grid item xs={6}>
                    <Card className={classes.card}>
                        <CardHeader
                            avatar={<Avatar className={classes.avatarBlue}>G</Avatar>}
                            title={"Genre"}
                            subheader={"Information about contents of a specific genre"}
                            action={
                                <Chip
```

```
                                    onClick={this.showGenres}
                                    variant={"outlined"}
                                    color={"secondary"}
                                    label={genre.name}
                                    avatar={<Avatar>{genre.name[0].toUpperCase()}</Avatar>}/>
                            }
                        />
                        <Menu
                            id={"-menu"}
                            anchorEl={genreAnchorEl}
                            open={genreOpen}
                            onClose={this.genreClose}
                            PaperProps={{
                                style: {
                                    maxHeight: 48 * 4.5,
                                    width: 200,
                                }
                            }}>
                            {this.props.genres.map((item, idx) => (
                                <MenuItem key={item} onClick={event => this.handleGenreMenu(event, idx)}>
                                    {item}
                                </MenuItem>
                            ))}
                        </Menu>
                        <CardContent>
                            <Grid container>
                                <Grid item xs={6}>
                                    <Typography variant={"subtitle1"}>
                                        Latest
                                    </Typography>
                                    <List>
                                        <ListItem>
                                            <Avatar>{genre.mostRecent[0].toUpperCase()}</Avatar>
                                            <ListItemText primary={genre.mostRecent}/>
                                        </ListItem>
                                    </List>
                                </Grid>
                                <Grid item xs={6}>
                                    <Typography variant={"subtitle1"}>
                                        Most viewed
                                    </Typography>
                                    <List>
                                        <ListItem>
                                            <Avatar>{genre.mostViewed[0].toUpperCase()}</Avatar>
                                            <ListItemText primary={genre.mostViewed}/>
                                        </ListItem>
                                    </List>
                                </Grid>
                                <Grid item xs={12}>
                                    <Feedbacks
                                        avg={genre.topRatedAvg}
                                        appreciation={genre.topRatedAppreciation}
                                        fairness={genre.topRatedFairness}
                                        suggest={genre.topRatedSuggest}/>
                                </Grid>
                            </Grid>
                        </CardContent>
                        <CardActions style={{display: "block", width: "100%"}}>
                            <Button
                                className={classes.button}
                                variant={"contained"}
                                onClick={() => this.props.addSubGenre(genre.name)}>
                                Notify about new content of genre {genre.name}
                            </Button>
                        </CardActions>
                    </Card>
                </Grid>
            </Grid>
        );
    }
}

Charts.propTypes = {
    classes: PropTypes.object.isRequired,
    authors: PropTypes.array.isRequired,
    genres: PropTypes.array.isRequired,
    addSubAuthor: PropTypes.func.isRequired,
    addSubGenre: PropTypes.func.isRequired,
    web3: PropTypes.object.isRequired,
    Catalog: PropTypes.object.isRequired,
    account: PropTypes.string.isRequired,
};

export default withStyles(styles)(Charts);
```

```
/*
 *
 *  University di Pisa - Master's Degree in Computer Science and Networking           )
 *
 *  Final Project for the course of Peer to Peer Systems and Blockchains
 *
 *  Teacher: Prof. Laura Ricci
 *
 *  Candidate: Orlando Leombruni, matricola 475727
 *
 *  File: NewContentForm.js
 *
*/

import React from 'react';
import PropTypes from 'prop-types';
import BN from 'bn.js';
import {
    withStyles,
    Card,
    CardContent,
    CardActions,
    TextField,
    Button,
    MenuItem,
    CircularProgress,
} from '@material-ui/core';
import { NewContentFormStyle as styles } from "../styles/MaterialCustomStyles";

/*
 * NewContentForm Class
 *
 * A React Component that provides the user with a form for creating a new managed C
ontent.
 */
class NewContentForm extends React.Component {

    constructor(props) {
        super(props);
        this.currencies = [
            {label: "wei",    value: "1"},
            {label: "kwei",   value: "1000"},
            {label: "Mwei",   value: "1000000"},
            {label: "Gwei",   value: "1000000000"},
            {label: "szabo",  value: "1000000000000"},
            {label: "finney", value: "1000000000000000"},
            {label: "ether",  value: "1000000000000000000"}
        ];
        this.state = {
            name: "",
            genre: "",
            author: "",
            price: 1,
            currency: this.currencies[0].value,
        };
    }

    // Handles user changes in the form.
    handleChange = event => {
        const { target } = event;
        this.setState(oldState => ({...oldState, [target.name]: target.value}))
    };

    // Validates the form if all input is correct.
    checkForm = () => {
        const { name, genre, author, price, currency } = this.state;
        if (name && genre && author && (+price > 0) && currency)
            this.props.submitForm(name, genre, author, new BN(currency).muln(+price)
    };

    render() {
        const { classes, loading } = this.props;
        const { name, genre, author, price, currency } = this.state;
        return (
            <Card className={classes.root}>
                <CardContent>
                <form onSubmit={this.checkForm} style={{width: "100%"}}>
                    <div className={classes.formDiv}>
                    <TextField
                        className={classes.textField}
                        name={"name"}
                        label={"Content description"}
                        InputLabelProps={{shrink: true}}
                        required
                        margin={"normal"}
                        value={this.state.name}
                        type={"string"}
                        onChange={this.handleChange}
                    />
                    <TextField
                        className={classes.textField}
                        name={"genre"}
                        label={"Genre"}
                        InputLabelProps={{shrink: true}}
                        required
                        margin={"normal"}
                        type={"string"}
                        value={genre}
                        onChange={this.handleChange}
                    />
                    <TextField
                        className={classes.textField}
                        name={"author"}
                        label={"Author"}
                        InputLabelProps={{shrink: true}}
                        required
                        margin={"normal"}
                        type={"string"}
                        value={author}
                        onChange={this.handleChange}
                    />
                    <div>
                    <TextField
                        className={classes.priceField}
                        name={"price"}
                        label={"Price"}
                        InputLabelProps={{shrink: true}}
                        inputProps={{min: 1, step: 1, style: {textAlign: "right"}}}
                        required
                        error={+price <= 0}
                        type={"number"}
                        margin={"normal"}
                        value={price}
                        onChange={this.handleChange}
                    />
                    <TextField
                        select
                        required
                        InputLabelProps={{shrink: true}}
                        className={classes.currency}
                        name={"currency"}
```

```
                            label={"Unit"}
                            margin={"normal"}
                            value={currency}
                            SelectProps={{
                                MenuProps: {
                                    className: classes.menu,
                                }
                            }}
                            onChange={this.handleChange}>
                            {this.currencies.map((option, item) =>
                                <MenuItem key={item} value={option.value}>
                                    {option.label}
                                </MenuItem>
                            )}
                        </TextField>
                        </div>
                        </div>
                    </form>
                    </CardContent>
                    <CardActions style={{display: "block", width: "100%"}}>
                        <div className={classes.wrapper}>
                        <Button
                            variant={"contained"}
                            className={classes.button}
                            color={"secondary"}
                            disabled={(!(name && genre && author && (+price > 0) && curr
ency)) || loading}
                            onClick={this.checkForm}>Add</Button>
                            {loading && <CircularProgress size={24} className={classes.b
uttonProgressPrimary}/>}
                        </div>
                    </CardActions>

                </Card>
            );
        }
}

NewContentForm.propTypes = {
    classes: PropTypes.object.isRequired,
    loading: PropTypes.bool.isRequired,
    submitForm: PropTypes.func.isRequired,
};

export default withStyles(styles)(NewContentForm);
```

```
/*
 *
 *  University di Pisa - Master's Degree in Computer Science and Networking
 *
 *  Final Project for the course of Peer to Peer Systems and Blockchains
 *
 *  Teacher: Prof. Laura Ricci
 *
 *  Candidate: Orlando Leombruni, matricola 475727
 *
 *  File: FeedbackForm.js
 *
 */

import React from 'react';
import PropTypes from 'prop-types';
import {
    withStyles,
    Dialog,
    DialogTitle,
    DialogContent,
    DialogContentText,
    DialogActions,
    TextField,
    MenuItem,
    Button
} from '@material-ui/core';
import { FeedbackFormStyle as styles } from "../styles/MaterialCustomStyles";

/*
 * FeedbackForm Class
 *
 * A React Component that allows the user to input feedback (as three numerical grad
es) for the Content.
 */
class FeedbackForm extends React.Component {
    state = {
        appreciation: -1,
        fairness: -1,
        suggest: -1
    };

    // Handles changes in the form.
    changeFields = event => {
        const { target } = event;
        this.setState(oldState => ({...oldState, [target.name]: target.value}))
    };

    render() {

        const { classes, cancel, confirm } = this.props;

        const categories = [
            {
                label: "0 (worst)",
                value: 0,
            },
            {
                label: "1",
                value: 1,
            },
            {
                label: "2",
                value: 2,
            },
            {
                label: "3",
                value: 3,
            },
            {
                label: "4",
                value: 4,
            },
            {
                label: "5 (best)",
                value: 5,
            }
        ];

        return (
            <Dialog
                open={true}
                keepMounted
                onBackdropClick={cancel}
                onEscapeKeyDown={cancel} >
                <DialogTitle>Rate your experience</DialogTitle>
                <DialogContent>
                    <ul>
                        <li>
                            <div>
                                <DialogContentText>How much did you enjoy the conten
t?</DialogContentText>
                                <TextField
                                    id={"appreciation"}
                                    select
                                    required
                                    name={"appreciation"}
                                    className={classes.textField}
                                    value={this.state.appreciation}
                                    onChange={this.changeFields}
                                    SelectProps={{
                                        MenuProps: {
                                            className: classes.menu,
                                        }
                                    }}
                                    margin={"normal"} >
                                    {categories.map((item) => (
                                        <MenuItem key={item.value} value={item.value
}>{item.label}</MenuItem>
                                    ))}
                                </TextField>
                            </div>
                        </li>
                        <li>
                            <div>
                                <DialogContentText>Do you think that the price was f
air?</DialogContentText>
                                <TextField
                                    id={"fairness"}
                                    select
                                    required
                                    name={"fairness"}
                                    className={classes.textField}
                                    value={this.state.fairness}
                                    onChange={this.changeFields}
                                    SelectProps={{
                                        MenuProps: {
                                            className: classes.menu,
                                        }
                                    }}
                                    margin={"normal"} >
                                    {categories.map((item) => (
                                        <MenuItem key={item.value} value={item.value}>{item.label}</MenuItem>
                                    ))}
                                </TextField>
                            </div>
                        </li>
                        <li>
                            <div>
                                <DialogContentText>How likely are you to recommend this content to others?</DialogContentText>
                                <TextField
                                    id={"suggest"}
                                    select
                                    required
                                    name={"suggest"}
                                    className={classes.textField}
                                    value={this.state.suggest}
                                    onChange={this.changeFields}
                                    SelectProps={{
                                        MenuProps: {
                                            className: classes.menu,
                                        }
                                    }}
                                    margin={"normal"} >
                                    {categories.map((item) => (
                                        <MenuItem key={item.value} value={item.value}>{item.label}</MenuItem>
                                    ))}
                                </TextField>
                            </div>
                        </li>
                    </ul>
                </DialogContent>
                <DialogActions>
                    <Button
                        color={"inherit"}
                        className={classes.button}
                        onClick={cancel}>Cancel</Button>
                    <Button
                        color={"secondary"}
                        disabled={!((this.state.appreciation > -1) && (this.state.fairness > -1) && (this.state.suggest > -1))}
                        className={classes.button}
                        onClick={confirm(this.state)}>Ok</Button>
                </DialogActions>
            </Dialog>
        );
    }
}

FeedbackForm.propTypes = {
    classes: PropTypes.object.isRequired,
    cancel: PropTypes.func.isRequired,
    confirm: PropTypes.func.isRequired,
};

export default withStyles(styles)(FeedbackForm);
```

```
/*
 *
 *  University di Pisa - Master's Degree in Computer Science and Networking
 *
 *  Final Project for the course of Peer to Peer Systems and Blockchains
 *
 *  Teacher: Prof. Laura Ricci
 *
 *  Candidate: Orlando Leombruni, matricola 475727
 *
 *  File: CreatorApp.js
 *
 */

import React from 'react';
import { withStyles } from '@material-ui/core';
import PropTypes from 'prop-types';
import ReactNotification from 'react-notifications-component';
import AppDrawer from './AppDrawer'
import ContentList from './ContentList'
import TransactionConfirmation from './TransactionConfirmation';
import NewContentForm from './NewContentForm';
import {
    List,
    ListItemIcon,
    ListItemText,
    ListItem,
    Tooltip,
    Divider,
    TextField,
    Button,
    Dialog,
    DialogTitle,
    DialogContent,
    DialogContentText,
    DialogActions,
    Slide,
} from '@material-ui/core';
import {
    List as ListIcon,
    LibraryAdd,
    Delete,
    Update,
    Publish,
} from '@material-ui/icons';
import CopyToClipboard from 'react-copy-to-clipboard';
import 'react-notifications-component/dist/theme.css';
import 'animate.css';
import json from '../solidity/compiled.json';
import {getTransactionParameters, makeTitle, prettifyWei} from "../Utils";
import { CreatorAppStyle as styles } from "../styles/MaterialCustomStyles";

/*
 * "Slide up" transition for the animation of a React component.
 */
const Transition = (props) =>
    <Slide direction="up" {...props} />;

/*
 * Enum-like object for the various sections of the application.
 */
const Panels = Object.freeze({
    LIST_CONTENT: Symbol("list"),
    NEW_CONTENT: Symbol("new"),
    DEL_CONTENT: Symbol("delete"),
    PUBLISH: Symbol("publish"),
});

/*
 * CreatorApp Class
 *
 * A React Component that represents the entire Creators portion for the app.
 * It allows a Creator to publish new contents, delete existing ones, and request monetization for their work.
 */
class CreatorApp extends React.Component {

    constructor(props) {
        super(props);
        this.notificationRef = React.createRef();
        this.notify = this.notify.bind(this);
        this.web3 = this.props.web3;
        this.props.manageCenter(false);
        const abi = JSON.parse(json.contracts["Catalog.sol:Catalog"].abi);
        this.Catalog = new this.web3.eth.Contract(abi, this.props.catalog);
        this.newContent = {};
        this.registered = [];
    }

    state = {
        panel: Panels.LIST_CONTENT,
        contentList: [],
        transactionProps: {},
        transactPopup: false,
        error: false,
        selectedContent: "",
        creatingContent: false,
        publishingContent: false,
        closing: false,
        createdContent: "",
        createdPopup: false,
        loading: false,
        monetize: false,
    };

    // Changes the current panel (section of the app).
    changePanel = (panel) => () => {
        this.setState(oldState => ({...oldState, panel: panel}));
    };

    // Displays a notification in the upper right portion of the screen.
    notify(title, message, type, time) {
        this.notificationRef.current.addNotification({
            title,
            message,
            type,
            insert: "top",
            container: "top-right",
            animationIn: ["animated", "fadeIn"],
            animationOut: ["animated", "fadeOut"],
            dismiss: { duration: time || 2000 },
            dismissable: { click: true },
        });
    }

    // Callback function to handle user input for the publishing/deletion of a content.
    handleContent = (event) => {
        const { target } = event;
        this.setState(oldState => ({...oldState, selectedContent: target.value}))
    };

    // Closes the "new Content created" popup dialog.
    handleDialog = () =>
        this.setState(oldState => ({...oldState, createdPopup: false, createdContent: ""}));

    /*
     * Begins the deletion of a ContentManagementBase contract, checking the correctness of the operation
     * (i.e. if the content exists and is owned by the current selected EOA) and estimating the gas cost.
     */
    initiateDelete = () => {
        let error = (this.state.selectedContent === "");
        const abi = JSON.parse(json.contracts["ContentManagementBase.sol:ContentManagementBase"].abi);
        const CMB = new this.web3.eth.Contract(abi);
        try {
            CMB.options.address = this.state.selectedContent;
        } catch (err) {
            error = true;
            console.log(err);
        }
        if (error) {
            this.setState(oldState => ({...oldState, error: true}));
            this.notify("Error", "Invalid Content address", "danger");
        } else {
            this.setState(oldState => ({...oldState, closing: true, error: false}));
            getTransactionParameters(this.web3, CMB.methods.closeContract(), this.props.account).then(
                (result) => this.setState(o => ({...o, transactionProps: result, transactPopup: true})),
                (error) => {
                    console.log("Error in getting parameters for initiateDelete");
                    console.log(error);
                    this.notify("Error", error.stackTrace.message.toString(), "danger");
                }
            );
        }
    };

    /*
     * Invoked when the user confirms the transaction (after reviewing it in an informative dialog), this function
     * effectively closes and deletes a content, checking afterwards that the operation was successful.
     */
    confirmDelete = () => {
        const abi = JSON.parse(json.contracts["ContentManagementBase.sol:ContentManagementBase"].abi);
        const CMB = new this.web3.eth.Contract(abi, this.state.selectedContent);
        CMB.methods.closeContract().send({
            from: this.props.account,
            gasPrice: this.state.transactionProps.gasPrice,
            gas: this.state.transactionProps.gas,
        }).then(
            (result) => {
                CMB.methods.isActiveContent().call({from: this.props.account}).then(
                    () => this.notify("Error", "Contract NOT closed! Are you the owner?", "danger"),
                    () => this.notify("Success!", "Content contract was closed!", "success")
                );
                this.setState(o => ({...o, loading: false}));
            },
            (error) => {
                console.log(error);
                this.notify("Error", "Could not close contract...", "danger");
                this.setState(o => ({...o, loading: false}));
            }
        );
        this.setState(oldState => ({...oldState, loading: true, closing: false, transactionProps: {}, transactPopup: false}));
    };

    // Starts a request for monetization of a Content to the Catalog, estimating the gas cost.
    requestMonetization = (key) => () => {
        const address = this.state.contentList[key].address;
        getTransactionParameters(this.web3, this.Catalog.methods.collectPayment(address), this.props.account).then(
            (result) => this.setState(o => ({...o, transactionProps: result, transactPopup: true})),
            (error) => {
                console.log("Error in getting parameters for requestMonetization");
                console.log(error.stackTrace);
                switch (error.type) {
                    case "estimateGas":
                        this.notify("Error", "Could not request payment, maybe there’s no payment available", "danger");
                        break;
                    case "getBalance":
                        this.notify("Error", "Could not check account balance, maybe connection is down", "danger");
                        break;
                    case "getGasPrice":
                        this.notify("Error", "Could not check gas price from the blockchain, maybe connection is down", "danger");
                        break;
                    default:
                        this.notify("Error", error.stackTrace.message.toString(), "danger");
                }
                this.setState(o => ({...o, monetize: ""}));
            }
        );
        this.setState(o => ({...o, monetize: address}));
    };

    /*
     * Invoked when the user confirms the transaction (after reviewing it in an informative dialog), this function
     * effectively sends a request for payment to the Catalog and (upon success) shows the Creator their new balance.
     */
    confirmMonetize = () => {
        this.Catalog.methods.collectPayment(this.state.monetize).send({
            from: this.props.account,
            gas: this.state.transactionProps.gas,
            gasPrice: this.state.transactionProps.gasPrice
        }).then(
            () => {
                this.web3.eth.getBalance(this.props.account).then(
                    (result) => {
                        this.notify("Success!",
                            ‘You’ve been paid for views of your content, balance is now ${prettifyWei(result.toString())}‘,
                            "success");
                    },
                    (error) => {
                        this.notify("Success",
                            ‘You’ve been paid for views of your content, but could n
```

```
ot fetch new balance`,
                            "material");
                    }
                );
                this.setState(o => ({...o, loading: false}));
            },
            (error) => {
                console.log("Could not monetize", error);
                this.notify("Error", "There's been a problem with monetization, check console logs", "danger");
                this.setState(o => ({...o, loading: false}));
            }
        );
        this.setState(o => ({...o, transactionProps: {}, transactPopup: false, monetize: "", loading: true}));
    };

    // Begins the creation of a new ContentManagementBase contract, estimating its gas cost.
    createContent = (name, genre, author, price) => {
        const abi = JSON.parse(json.contracts["ContentManagementBase.sol:ContentManagementBase"].abi);
        const bin = "0x" + json.contracts["ContentManagementBase.sol:ContentManagementBase"].bin;
        const CMB = new this.web3.eth.Contract(abi);
        this.newContent = CMB.deploy({
            data: bin,
            arguments: [
                this.web3.utils.asciiToHex(name),
                this.web3.utils.asciiToHex(genre),
                this.web3.utils.asciiToHex(author),
                price.toString(),
            ]});
        getTransactionParameters(this.web3, this.newContent, this.props.account).then(
            (result) => this.setState(o => ({...o, transactionProps: result, transactPopup: true})),
            (error) =>  {
                console.log("Error in getting parameters for createContent");
                console.log(error.stackTrace);
                this.notify("Error", `Could not complete operation (error in ${error.type})`, "danger");
                this.setState(o => ({...o, creatingContent: false}));
            }
        );
        this.setState(oldState => ({...oldState, closing: false, creatingContent: true}))
    };

    /*
     * Invoked when the user confirms the transaction (after reviewing it in an informative dialog), this function
     * effectively creates a new ContentManagementBase contract and checks if it has been correctly deployed.
     *
     * This function WILL NOT publish the new Content on the Catalog, but will ask the user if they want to.
     */
    confirmCreate = () => {
        this.newContent.send({
            from: this.props.account,
            gasPrice: this.state.transactionProps.gasPrice,
            gas: this.state.transactionProps.gas,
        }).then(
            (result) => {
                result.methods.isActiveContent().call({from: this.props.account}).then(
                    () => {
                        this.setState(oldState => ({...oldState, createdContent: result.options.address, createdPopup: true, creatingContent: false}));
                        this.newContent = result;
                    },
                    () => {
                        this.notify("Error", "Contract not created...", "danger");
                        this.setState(o => ({...o, loading: false, creatingContent: false}));
                    }
                )
            },
            (error) => {
                console.log(error);
                this.notify("Error", "Could not create contract...", "danger");
                this.setState(o => ({...o, loading: false, creatingContent: false}));
            }
        );
        this.setState(oldState => ({...oldState, loading: true, closing: false, transactionProps: {}, transactPopup: false}));
    };

    /*
     * Begins the publishing of a Content (either a freshly-created or a user-provided one) to the Catalog,
     * estimating the operation's gas cost.
     */
    publishContent = () => {
        this.setState(oldState => ({...oldState, publishingContent: true}));
        getTransactionParameters(this.web3, this.newContent.methods.publish(this.props.catalog), this.props.account).then(
            (result) => this.setState(o => ({...o, transactionProps: result, transactPopup: true})),
            (error) => {
                console.log("Error in getting parameters for publishContent");
                console.log(error);
                if (error.type === "estimateGas")
                    this.notify("Error", "Could not publish content, maybe it's already in the catalog?", "danger");
                else
                    this.notify("Error", `Could not complete operation (error in ${error.type})`, "danger");
                this.setState(o => ({...o, loading: false, publishingContent: false, createdPopup: false, createdContent: ""}));
            }
        );
    };

    /*
     * Invoked when the user confirms the transaction (after reviewing it in an informative dialog), this function
     * effectively publishes a Content to the Catalog.
     */
    confirmPublish = () => {
        this.newContent.methods.publish(this.props.catalog).send({
            from: this.props.account,
            gas: this.state.transactionProps.gas,
            gasPrice: this.state.transactionProps.gasPrice,
        }).then(
            (result) => {
                this.notify("Success!", `Contract published! Refresh the list to see it`, "success");
                this.setState(o => ({...o, loading: false}));
            },
            (error) => {
                this.notify("Error", "Error in publishing contract!", "danger");
                this.setState(o => ({...o, loading: false}));
            }
        );
        this.newContent = {};
        this.setState(oldState => ({...oldState, loading: true, transactionProps:{}, publishingContent: false, transactPopup: false, createdPopup: false, createdContent: ""}))
    };

    // Renders the drawer menu.
    menu = () =>
        <List>
            <Tooltip title={"List this account’s Contents"} placement={"right"}>
                <ListItem button key={"list"} onClick={this.changePanel(Panels.LIST_CONTENT)}>
                    <ListItemIcon><ListIcon /></ListItemIcon>
                    <ListItemText secondary={"My Contents"}/>
                </ListItem>
            </Tooltip>
            <Tooltip title={"Create a new Content"} placement={"right"}>
                <ListItem button key={"new"} onClick={this.changePanel(Panels.NEW_CONTENT)}>
                    <ListItemIcon><LibraryAdd /></ListItemIcon>
                    <ListItemText secondary={"New Content"}/>
                </ListItem>
            </Tooltip>
            <Tooltip title={"Publish an already-existing content"} placement={"right"}>
                <ListItem button key={"publish"} onClick={this.changePanel(Panels.PUBLISH)}>
                    <ListItemIcon><Publish/></ListItemIcon>
                    <ListItemText secondary={"Publish a Content"}/>
                </ListItem>
            </Tooltip>
            <Divider/>
            <Tooltip title={"Delete a Content from the Catalog"} placement={"right"}>
                <ListItem button key={"delete"} onClick={this.changePanel(Panels.DEL_CONTENT)}>
                    <ListItemIcon><Delete/></ListItemIcon>
                    <ListItemText secondary={"Delete a Content"}/>
                </ListItem>
            </Tooltip>
            {(this.state.panel === Panels.LIST_CONTENT) && <Divider/>}
            {(this.state.panel === Panels.LIST_CONTENT) &&
                <Tooltip title={"Update Content list"} placement={"right"}>
                    <ListItem button key={"update"} onClick={this.updateContentList}>
                        <ListItemIcon><Update/></ListItemIcon>
                        <ListItemText secondary={"Update List"}/>
                    </ListItem>
                </Tooltip>}
        </List>;

    // Renders the app bar title (depending on the current panel).
    renderTitle = () => {
        const { classes } = this.props;
        switch (this.state.panel) {
            case Panels.LIST_CONTENT: return [makeTitle("My Contents", classes.grow)];
            case Panels.NEW_CONTENT: return [makeTitle("Add a new Content", classes.grow)];
            case Panels.PUBLISH: return [makeTitle("Publish a Content", classes.grow)];
            case Panels.DEL_CONTENT: return [makeTitle("Delete a Content", classes.grow)];
            default: return [makeTitle("", classes.grow)];
        }
    };

    /*
     * Updates the list of content belonging to the current Creator, also registering callbacks for the "Content
     * Provided" Solidity events.
     */
    updateContentList = () => {
        this.Catalog.methods.getCreatorContentList().call({from: this.props.account})
            .then(
                (result) => {
                    if (!result[0] || !result[1]) {
                        this.notify("Error", "Couldn’t fetch Catalog content", "danger");
                        console.log(result);
                    }
                    else {
                        const list = [];
                        for (let i = 0; i < result[0].length; i++) {
                            list.push({
                                index: i,
                                description: this.web3.utils.hexToAscii(result[0][i]),
                                address: result[1][i],
                            });
                            this.registerProvide(result[1][i]);
                        }
                        this.setState(oldState => ({...oldState, contentList: list}));
                        this.notify("List updated!", "List of content has been updated", "material");
                    }
                },
                (error) => {
                    this.notify("Error", "Couldn’t fetch Catalog content", "danger");
                    console.log(error);
                }
            )
    };

    // Registers a callback for the "Provide Content" event relative to the selected Content.
    registerProvide = (address) => {
        if (!this.registered.some(item => item === address)) {
            const abi = JSON.parse(json.contracts["ContentManagementBase.sol:ContentManagementBase"].abi);
            const CMB = new this.web3.eth.Contract(abi, address);

            CMB.events.ProvideContent({filter: {content: address}, fromBlock: "latest"})
                .on(’error’, error => console.log("Error in event Providecontent", error))
                .on(’data’, event => {
                    const content = this.state.contentList.find((item) => item.address === address);
                    if (content) {
                        this.notify("Notice!", ‘Content ${address} provided to user ${event.returnValues.user}!`, "material", 10000);
                    }
                });

            this.registered.push(address);
        }
    };

    /*
     * This is a React state function that will be called only once, after the component is mounted in the virtual
     * DOM but before it gets rendered.
     *
     * In particular, this function adds an event listener that reacts to Solidity "Payment Available" events.
     */
    componentDidMount() {
        this.updateContentList();
        this.Catalog.events.PaymentAvailable({fromBlock: "latest"})
            .on(’error’, error => console.log("PaymentAvailable error: ", error))
            .on(’data’, event => {
                const content = this.state.contentList.find((item) => item.address === event.returnValues.contentPayable);
                if (content) {
                    this.notify("Notice!", ‘Payment is available for content ${content.description}!‘, "material", 10000);
                }
            });
    };

    // Starts the publishing of the Content selected by the Creator.
    initiatePublish = () => {
        const abi = JSON.parse(json.contracts["ContentManagementBase.sol:ContentManagementBase"].abi);
        this.newContent =  new this.web3.eth.Contract(abi, this.state.selectedContent);
        this.publishContent();
    };

    render() {
        const { classes } = this.props;
        const {
            panel,
            contentList,
            transactionProps,
            transactPopup,
            selectedContent,
            error,
            creatingContent,
            publishingContent,
            closing,
            createdPopup,
            createdContent,
            loading,
            monetize,
        } = this.state;

        const attributes = [
            {
                name: "address",
                description: "Account address",
                copy: true,
                makeSmall: (what) => what.replace(what.slice(5, -3), "..."),
            }
        ];

        return (
            <AppDrawer drawer={true} writeTitle={this.renderTitle} render={this.menu} loading={loading}>
                <ReactNotification
                    ref={this.notificationRef}
                    types={[{
                        htmlClasses: classes.notification,
                        name: "material",
                    }]} />
                {(() => {
                    switch (panel) {
                        case Panels.LIST_CONTENT: {
                            return (
                                <div>
                                    <ContentList
                                        attributes={attributes}
                                        action={this.requestMonetization}
                                        actionName={"Request Monetization"}
                                        list={contentList}
                                        notify={this.notify}
                                    />
                                </div>
                            );
                        }
                        case Panels.NEW_CONTENT:
                            return (
                                <div>
                                    <NewContentForm loading={creatingContent} submitForm={this.createContent}/>
                                </div>
                            );
                        case Panels.PUBLISH:
                            return (
                                <div>
                                    <Tooltip
                                        title={‘Make sure your content isn’t already published‘}
                                        placement={"bottom"} >
                                        <TextField
                                            id="contentAddress"
                                            required
                                            name="Content"
                                            label="Content Address"
                                            className={classes.textField}
                                            value={selectedContent}
                                            onChange={this.handleContent}
                                            error={error}
                                        />
                                    </Tooltip>
                                    <Button color={"primary"} className={classes.button} variant={"extendedFab"} onClick={this.initiatePublish}>
                                        <Publish className={classes.extendedIcon} />
                                        Publish
                                    </Button>
                                </div>
                            );
                        case Panels.DEL_CONTENT:
                            return (
                                <div>
                                    <Tooltip
                                        title={‘You can copy a Content’s address from the "My Contents" section‘}
                                        placement={"bottom"} >
                                        <TextField
                                            id="contentAddress"
                                            required
                                            name="Content"
                                            label="Content Address"
                                            className={classes.textField}
                                            value={selectedContent}
                                            onChange={this.handleContent}
                                            error={error}
                                        />
                                    </Tooltip>
                                    <Button className={classes.warn} variant={"extendedFab"} onClick={this.initiateDelete}>
                                        <Delete className={classes.extendedIcon} />
                                        Delete
                                    </Button>
                                </div>
                            );
                        default:
                            return "";
                    }
                })()}
                {transactPopup ? <TransactionConfirmation
                    {...transactionProps}
                    ok={(createdPopup || publishingContent)? this.confirmPublish :
                        (monetize? this.confirmMonetize :
                        (closing? this.confirmDelete : this.confirmCreate))}
                    cancel={() => this.setState(oldState => ({...oldState, closing: false, transactionProps: {}, transactPopup: false}))}
                /> : ""}
                <Dialog
                    open={createdPopup}
                    keepMounted
                    TransitionComponent={Transition}
                    onEscapeKeyDown={this.handleDialog}
                    onBackdropClick={this.handleDialog} >
                    <DialogTitle>Content contract created!</DialogTitle>
                    <DialogContent>
                        <DialogContentText>The desired Content contract has been created at address:</DialogContentText>
                        <Tooltip title={"Click to copy"} placement={"right"}>
                            <CopyToClipboard text={createdContent} onCopy={() => this.notify("Success!", "Copied to clipboard", "success")}>
                                <DialogContentText style={{cursor: "pointer"}}>{createdContent}</DialogContentText>
                            </CopyToClipboard>
                        </Tooltip>
                        <DialogContentText>Do you want to publish it right away?</DialogContentText>
                        <DialogContentText>
                            Make sure to hold on to the contract address if you want to publish it later.
                        </DialogContentText>
                        <DialogContentText>
                            You can click on the address in this window to copy it into the clipboard.
                        </DialogContentText>
                    </DialogContent>
                    <DialogActions>
                        <Button
                            color={"primary"}
                            disabled={publishingContent}
                            onClick={this.handleDialog}
                            className={classes.button}>
                            Cancel
                        </Button>
                        <Button
                            color={"secondary"}
                            disabled={publishingContent}
                            onClick={this.publishContent}
                            className={classes.button}>
                            Publish
                        </Button>
                    </DialogActions>
                </Dialog>
            </AppDrawer>
        );
    }
}

CreatorApp.propTypes = {
    classes: PropTypes.object.isRequired,
    web3: PropTypes.object.isRequired,
    catalog: PropTypes.string.isRequired,
    account: PropTypes.string.isRequired,
    manageCenter: PropTypes.func.isRequired,
};

export default withStyles(styles)(CreatorApp);
```

```
/*
 *
 *  University di Pisa - Master’s Degree in Computer Science and Networking
 *
 *  Final Project for the course of Peer to Peer Systems and Blockchains
 *
 *  Teacher: Prof. Laura Ricci
 *
 *  Candidate: Orlando Leombruni, matricola 475727
 *
 *  File: BuyGiftPremiumDialog.js
 *
 */

import React from ’react’;
import PropTypes from ’prop-types’;
import {
    withStyles,
    Dialog,
    DialogTitle,
    DialogContent,
    DialogContentText,
    DialogActions,
    TextField, Button,
} from ’@material-ui/core’;
import { prettifyWei } from ’../Utils’;
import { BuyGiftPremiumDialogStyle as styles} from "../styles/MaterialCustomStyles";

/*
 * BuyGiftPremiumDialog Class
 *
 * A React Component that displays a dialog window, reviewing the purchase or renewal of a Premium subscription
 * and asking the user if they want to buy it for themselves or gift it to another user.
 */
class BuyGiftPremiumDialog extends React.Component {


    constructor(props) {
        super(props);
        this.state = {
            giftTo: ’’,
            retrieving: true,
            price: 0,
        };
    }

    handleChange = (event) => {
        const { target } = event ;
        this.setState(oldState => ({...oldState, giftTo: target.value}));
    };

    /*
     * This is a React state function that will be called only once, after the component is mounted in the virtual
     * DOM but before it gets rendered.
     *
     * In particular, this function asks the Catalog contract for the price of a Premium subscription.
     */
    componentDidMount() {
        this.props.catalog.methods.getPremiumPrice().call({from: this.props.account}).then(
            (result) => {
                this.setState(oldState => ({giftTo: oldState.giftTo, retrieving: false, price: result}))
            },
            (error) => {
                console.log(error);
                this.props.cancel(true);
            }
        );
    }

    render() {
        const { classes, gifting, confirm, cancel } = this.props;
        const { giftTo, retrieving, price } = this.state;

        return (
            <div className={classes.root}>
                <Dialog
                    open={true}
                    keepMounted
                    onBackdropClick={cancel(false)}
                    onEscapeKeyDown={cancel(false)}>
                    <DialogTitle>Purchase Premium subscription</DialogTitle>
                    <DialogContent>
                        <DialogContentText>
                            {retrieving ?
                                "Fetching subscription price..." :
                                ‘The subscription costs ${prettifyWei(price)}.‘}
                        </DialogContentText>
                        <div style={{display: (gifting? "block" : "none")}}>
                            <DialogContentText>Insert the account to gift.</DialogContentText>
                            <form onChange={this.handleChange}>
                                <TextField
                                    name={"account"}
                                    type={"string"}
                                    placeholder={"Account address..."}
                                    value={giftTo}
                                    className={classes.textField}
                                />
                            </form>
                        </div>
                        <DialogContentText>Purchase?</DialogContentText>
                    </DialogContent>
                    <DialogActions>
                        <Button color={"inherit"} onClick={cancel(false)}>Cancel</Button>
                        {gifting ? <Button
                            disabled={retrieving || !giftTo}
                            color={"secondary"}
                            onClick={confirm(price, giftTo)}>
                            Gift
                        </Button> :
                            <Button
                                disabled={retrieving}
                                color={"secondary"}
                                onClick={confirm(price)}>
                                Ok
                            </Button>
                        }
                    </DialogActions>
                </Dialog>
            </div>
        );
    }
}

BuyGiftPremiumDialog.propTypes = {
    classes: PropTypes.object.isRequired,
    gifting: PropTypes.bool.isRequired,
    confirm: PropTypes.func.isRequired,
    cancel: PropTypes.func.isRequired,
    catalog: PropTypes.object.isRequired,
    account: PropTypes.string.isRequired,
};

export default withStyles(styles)(BuyGiftPremiumDialog);
```

```
/*
 *
 *  University di Pisa - Master’s Degree in Computer Science and Networking
 *
 *  Final Project for the course of Peer to Peer Systems and Blockchains
 *
 *  Teacher: Prof. Laura Ricci
 *
 *  Candidate: Orlando Leombruni, matricola 475727
 *
 *  File: TransactionConfirmation.js
 *
 */

import React from ’react’;
import PropTypes from ’prop-types’;
import BN from ’bn.js’;
import {
    withStyles,
    Dialog,
    DialogTitle,
    DialogContent,
    DialogContentText,
    DialogActions,
    Button,
} from ’@material-ui/core’;
import { prettifyWei } from "../Utils";
import { TransactionConfirmationStyle as styles } from "../styles/MaterialCustomStyles";

/*
 * TransactionConfirmation Class
 *
 * A React Component that shows to the user a dialog window that contains information about the transaction
 * they’re going to perform: estimated gas cost, estimated transaction cost and current account balance.
 * The user can then choose to perform or decline the transaction.
 */
class TransactionConfirmation extends React.Component {

    render() {
        const { classes, gas, gasPrice, balance, ok, cancel } = this.props;
        const valid = gas * gasPrice < balance;
        return (
            <Dialog
                open={true}
                keepMounted
                onBackdropClick={cancel}
                onEscapeKeyDown={cancel} >
                <DialogTitle>{valid ? "Perform the transaction?" : "Insufficient funds"}</DialogTitle>
                <DialogContent>
                        <ul>
                            <li><DialogContentText>Estimated gas is {gas}</DialogContentText></li>
                            <li><DialogContentText>Estimated transation cost is {prettifyWei(new BN(gasPrice).mul(new BN(gas)).toString())}</DialogContentText></li>
                            {valid ?
                                <li><DialogContentText>Account balance is {prettifyWei(balance)}</DialogContentText></li>:
                                <li><DialogContentText>Insufficient funds in the account ({prettifyWei(balance)})</DialogContentText></li>}
                        </ul>
                </DialogContent>
                <DialogActions>
                        <Button color={"primary"} disabled={!valid} className={classes.buttonOk} onClick={ok}>Ok</Button>
                        <Button color={"inherit"} className={classes.buttonNo} onClick={cancel}>Cancel</Button>
                    </DialogActions>
                </Dialog>
            );
    }

}

TransactionConfirmation.propTypes = {
    classes: PropTypes.object.isRequired,
    gas: PropTypes.string.isRequired,
    gasPrice: PropTypes.string.isRequired,
    balance: PropTypes.string.isRequired,
    ok: PropTypes.func.isRequired,
    cancel: PropTypes.func.isRequired,
};

export default withStyles(styles)(TransactionConfirmation);
```

```
/*
 *
 *  University di Pisa - Master's Degree in Computer Science and Networking
 *
 *  Final Project for the course of Peer to Peer Systems and Blockchains
 *
 *  Teacher: Prof. Laura Ricci
 *
 *  Candidate: Orlando Leombruni, matricola 475727
 *
 *  File: UserApp.js
 *
 */
import React from 'react';
import PropTypes from 'prop-types';
import {
    withStyles,
    List,
    ListItem,
    ListItemText,
    ListItemIcon,
    Tooltip,
    Divider,
    Typography,
    InputBase,
    Button,
} from '@material-ui/core';
import {
    List as ListIcon,
    Search,
    Stars,
    Subscriptions,
    BarChart,
    ShopTwo,
    Update,
    CardGiftcard,
    Timelapse,
    QueuePlayNext
} from '@material-ui/icons';
import ReactNotification from 'react-notifications-component';
import 'react-notifications-component/dist/theme.css';
import AppDrawer from "./AppDrawer";
import ContentList from "./ContentList";
import json from '../solidity/compiled.json';
import {getTransactionParameters, makeTitle, prettifyWei} from "../Utils";
import BuyGiftDialog from "./BuyGiftDialog";
import TransactionConfirmation from "./TransactionConfirmation";
import BuyGiftPremiumDialog from "./BuyGiftPremiumDialog";
import FeedbackForm from "./FeedbackForm";
import Charts from "./Charts";
import { UserAppStyle as styles } from "../styles/MaterialCustomStyles";

/*
 * Enum-like object for the various sections of the application.
 */
const Panels = Object.freeze({
    FULL_LIST: Symbol("list"),
    CHARTS: Symbol("charts"),
    PLAY: Symbol("play"),
    FEEDBACK: Symbol("feedback"),
    BUY: Symbol("buy")
});

/*
 * UserApp Class
 *
 * A React Component that represents the entire User (consumer) portion of the app.
 *
 * It allows Users to check trending content, purchase premium subscriptions and acc
ess to Contents, and leave
 * feedback.
 */
class UserApp extends React.Component {

    constructor(props) {
        super(props);
        this.notify = this.notify.bind(this);
        this.notificationRef = React.createRef();
        this.props.manageCenter(false);
        this.web3 = this.props.web3;
        const abi = JSON.parse(json.contracts["Catalog.sol:Catalog"].abi);
        this.Catalog = new this.web3.eth.Contract(abi, this.props.catalog);
        this.premiumUpdates = null;
    }

    state = {
        authorList: [],
        genreList: [],
        loading: false,
        panel: Panels.FULL_LIST,
        contentList: [],
        searchTerm: "",
        filteredList: [],
        boughtContent: [],
        feedbackToLeave: [],
        feedbackLeaving: null,
        feedback: null,
        feedbackOpen: false,
        buyingPrice: 0,
        buyingAction: false,
        consuming: null,
        gifting: "",
        premiumFetching: true,
        isPremium: false,
        buyingPremium: false,
        premiumMethod: null,
        giftingPremium: false,
        transactPopup: false,
        transactionProps: {},
    };

    // State not used in rendering (doesn't have to be changed by React.Component.se
tState)
    otherState = {
        contentBuying: -1,
        isPremiumTransaction: false,
        authorSubs: [],
        genreSubs: [],
    };

    // Displays a notification in the upper right portion of the screen.
    notify(title, message, type, time) {
        this.notificationRef.current.addNotification({
            title,
            message,
            type,
            insert: "top",
            container: "top-right",
            animationIn: ["animated", "fadeIn"],
            animationOut: ["animated", "fadeOut"],
            dismiss: { duration: time || 2000 },
            dismissable: { click: true },
        });
    }

    // Changes the current panel (section of the app).
    changePanel = (panel) => () => {
        this.setState(oldState => ({...oldState, panel: panel}));
    };

    // Renders the drawer menu.
    menu = () =>
        <List>
            <Tooltip title={"List all contents in the Catalog"} placement={"right"}>
                <ListItem button key={"list"} onClick={this.changePanel(Panels.FULL_LIST)}>
                    <ListItemIcon><ListIcon /></ListItemIcon>
                    <ListItemText secondary={"Show all Contents"}/>
                </ListItem>
            </Tooltip>
            <Tooltip title={"Discover trending content"} placement={"right"}>
                <ListItem button key={"charts"} onClick={this.changePanel(Panels.CHARTS)}>
                    <ListItemIcon><BarChart/></ListItemIcon>
                    <ListItemText secondary={"Trending"}/>
                </ListItem>
            </Tooltip>
            <Divider/>
            <Tooltip title={"Obtain a Content that you've bought"} placement={"right"}>
                <ListItem button key={"play"} onClick={this.changePanel(Panels.PLAY)}>
                    <ListItemIcon><Subscriptions/></ListItemIcon>
                    <ListItemText secondary={"Consume Content"}/>
                </ListItem>
            </Tooltip>
            <Tooltip title={"Leave feedback for consumed Contents"} placement={"right"}>
                <ListItem button key={"feedback"} onClick={this.changePanel(Panels.FEEDBACK)}>
                    <ListItemIcon><Stars/></ListItemIcon>
                    <ListItemText secondary={"Leave Feedback"}/>
                </ListItem>
            </Tooltip>
            <Tooltip title={"Buy or gift a Premium subscription"} placement={"right"}>
                <ListItem button key={"buy"} onClick={this.changePanel(Panels.BUY)}>
                    <ListItemIcon><ShopTwo/></ListItemIcon>
                    <ListItemText secondary={"Buy Premium"}/>
                </ListItem>
            </Tooltip>
            {(this.state.panel === Panels.FULL_LIST) && <Divider/>}
            {(this.state.panel === Panels.FULL_LIST) &&
                <Tooltip title={"Update Content list"} placement={"right"}>
                    <ListItem button key={"update"} onClick={this.updateContentList}>
                        <ListItemIcon><Update/></ListItemIcon>
                        <ListItemText secondary={"Update List"}/>
                    </ListItem>
                </Tooltip>
            }
        </List>;

    // Renders the app bar title (depending on the current panel).
    renderTitle = () => {
        const { classes } = this.props;
        const { panel, searchTerm } = this.state;
        switch(panel) {
            case Panels.FULL_LIST:
                return (
                    [
                    makeTitle("List of available Content", classes.grow),
                    <div key={"search"} className={classes.search}>
                        <div className={classes.searchIcon}>
                            <Search />
                        </div>
                        <InputBase
                            placeholder="Searchâ\200¦"
                            classes={{
                                root: classes.inputRoot,
                                input: classes.inputInput,
                            }}
                            onChange={this.filterContent}
                            value={searchTerm}
                        />
                    </div>
                    ]
                );
            case Panels.PLAY:
                return [makeTitle("Consume Content", classes.grow)];
            case Panels.BUY:
                return [makeTitle("Premium Subscriptions", classes.grow)];
            case Panels.FEEDBACK:
                return [makeTitle("Rate your Contents", classes.grow)];
            case Panels.CHARTS:
                return [makeTitle("Discover trending", classes.grow)];
            default:
                return [makeTitle("Something went wrong", classes.grow)];
        }
    };

    // Filters the content list in real-time according to user input in the search bar.
    filterContent = (event) => {
        const { target } = event;
        const filtered = this.state.contentList.filter(value => (
            value.description.toLowerCase().includes(target.value.toLowerCase())
            || value.genre.toLowerCase().includes(target.value.toLowerCase())
            || value.author.toLowerCase().includes(target.value.toLowerCase())
        ));
        this.setState(oldState => ({...oldState, filteredList: filtered, searchTerm: target.value}));
    };

    // Refreshes the content list (and also the list of authors and genres).
    updateContentList = () => {
        this.setState(o => ({...o, loading: true}));
        this.Catalog.methods.getStatistics().call({from: this.props.account})
            .then(
                (result) => {
                    if (!result[0] || !result[1] || !result[2] || !result[3] || !result[4] ) {
                        this.notify("Error", "Couldn't fetch Catalog content", "danger");
                        console.log(result);
                    } else {
                        const list = [];
                        for (let i = 0; i < result[0].length; i++) {
                            list.push({
                                index: i,
                                description: this.web3.utils.hexToAscii(result[0][i]),
                                genre: this.web3.utils.hexToAscii(result[1][i]),
                                author: this.web3.utils.hexToAscii(result[2][i]),
                                price: result[3][i],
                                views: result[4][i],
                            })
                        }
                        const authors = new Set(), genres = new Set();
                        for (let j = 0; j < list.length; j++) {
                            authors.add(list[j].author);
                            genres.add(list[j].genre);
                        }
                        this.setState(oldState => ({
                            ...oldState,
                            loading: false,
                            contentList: list,
                            filteredList: list,
                            authorList: [...authors],
                            genreList: [...genres],
                        }));
                        this.notify("List updated!", "List of content has been updated", "material");
                    }
                },
                (error) => {
                    this.notify("Error", "Couldn’t fetch Catalog content", "danger");
                    console.log(error);
                    this.setState(o => ({...o, loading: false}));
                }
            )
    };

    // Refreshes the list of Content bought by the User.
    refreshBought = () => {
        this.Catalog.methods.grantsAvailable().call({from: this.props.account}).then(
            (result) => {
                if (!result) {
                    this.notify("Error", "Could not retrieve bought content", "danger");
                } else {
                    const list = [];
                    for (let i = 0; i < result.length; i++) {
                        list.push({
                            index: i,
                            description: this.web3.utils.hexToAscii(result[i]),
                        })
                    }
                    this.setState(oldState => ({...oldState, loading: false, boughtContent: list}));
                    this.notify("Success!", "Bought Content list updated!", "success");
                }
            },
            (error) => {
                console.log(error);
                this.notify("Error", "Could not retrieve bought content", "danger");
                this.setState(o => ({...o, loading: false}))
            }
        );
        this.setState(o => ({...o, loading: true}));
    };

    // Refreshes the list of Content for which the User can leave feedback.
    refreshFeedback = () => {
        this.Catalog.methods.feedbackAvailable().call({from: this.props.account}).then(
            (result) => {
                if (!result[0] || !result[1]) {
                    this.notify("Error", "Could not retrieve list of content to leave feedback for", "danger");
                } else {
                    const list = [];
                    for (let i = 0; i < result[0].length; i++) {
                        list.push({
                            index: i,
                            description: this.web3.utils.hexToAscii(result[0][i]),
                            address: result[1][i],
                        })
                    }
                    this.setState(oldState => ({...oldState, loading: false, feedbackToLeave: list}));
                    this.notify("Success!", "List of feedbacks to leave updated!", "success");
                }
            },
            (error) => {
                console.log(error);
                this.notify("Error", "Could not retrieve list of content to leave feedback for", "danger");
                this.setState(o => ({...o, loading: false}));
            }
        );
        this.setState(o => ({...o, loading: true}))
    };

    // Starts the "buy" operation (for both premium and non-premium users), estimating its gas cost.
    buy = acc => () => {
        const { contentList } = this.state;
        const { contentBuying, isPremiumTransaction: premium } = this.otherState;
        if (contentBuying < 0) {
            this.notify("Error", "No selected content", "danger");
        } else {
            const content = this.web3.utils.asciiToHex(contentList[contentBuying].description);
            const method = (premium)? this.Catalog.methods.getContentPremium(content) :
                ((acc) ? this.Catalog.methods.giftContent(content, acc) : this.Catalog.methods.getContent(content));

            const value = premium ? 0 : contentList[contentBuying].price;
            getTransactionParameters(this.web3, method, this.props.account, {value}).then(
                (result) => {
                    this.setState(o => ({...o, transactionProps: result, transactPopup: true}))
                },
                (error) => {
                    console.log("Error in getting parameters for buy");
                    console.log(error.stackTrace);
                    switch(error.type) {
                        case "estimateGas":
                            this.notify("Error",
                                (premium) ? "Unable to obtain access to Content, check if you have a subscription" :
                                "Unable to buy Content, check if already bought or if recipient has a premium subscription",
                                "danger");
                            break;
                        case "getGasPrice":
                            this.notify("Error", "Could not get gas price from the blockchain, check the connection", "danger");
                            break;
                        case "getBalance":
                            this.notify("Error", "Could not get balance for the current account, check the connection", "danger");
                            break;
                        default:
                            this.notify("Error", error.stackTrace.message.toString(), "danger");
                    }
                    this.otherState.contentBuying = -1;
                    this.otherState.isPremiumTransaction = false;
                    this.setState(oldState => ({
                        ...oldState,
                        transactionProps: {},
                        transactPopup: false,
                        buyingAction: false,
                        gifting: "",
                    }));
                }
            );
            this.setState(oldState => ({...oldState, gifting: acc, buyingPrice: 0, buyingAction: true}));
        }
    };

    /*
     * Invoked when the user confirms the transaction (after reviewing it in an informative dialog), this function
     * effectively asks the Catalog for access to the content, paying for it if the user is not a Premium subscriber.
     */
    confirmBuy = () => {
        const { contentList, gifting, transactionProps } = this.state;
        const { contentBuying, isPremiumTransaction } = this.otherState;
        if (contentBuying < 0) {
            this.notify("Error", "No selected content", "danger");
        } else {
            const content = this.web3.utils.asciiToHex(contentList[contentBuying].description);
            const method = (isPremiumTransaction)? this.Catalog.methods.getContentPremium(content) :
                ((gifting) ? this.Catalog.methods.giftContent(content, gifting) : this.Catalog.methods.getContent(content));

            method.send({
                from: this.props.account,
                gas: transactionProps.gas,
                gasPrice: transactionProps.gasPrice,
                value: (isPremiumTransaction ? 0 : contentList[contentBuying].price),
            }).then(
                (result) => this.setState(o => ({...o, loading: false})),
                (error) => {
                    console.log(error);
                    this.notify("Error", "Could not complete purchase", "danger");
                    this.setState(o => ({...o, loading: false}));
                }
            );

        }
        this.otherState = {contentBuying: -1, isPremiumTransaction: false};
        this.setState(oldState => ({...oldState, loading: true, gifting: "", buyingAction: false, transactionProps: {}, transactPopup: false}))
    };

    // Starts the "leave feedback for a content" operation, estimating its gas cost.
    leaveFeedback = (idx) => () => {
        const content = this.state.feedbackToLeave[idx];
        if (content.address) {
            this.setState(o => ({...o, feedbackLeaving: idx, feedbackOpen: true}));
        }
    };

    feedbackFormSubmitted = (feedback) => () => {
        const content = this.state.feedbackToLeave[this.state.feedbackLeaving];
        if (content.address) {
            getTransactionParameters(this.web3, this.Catalog.methods.leaveFeedback(content.address, 5, 5, 5), this.props.account).then(
                (result) => this.setState(o => ({
                    ...o,
                    feedback: feedback,
                    transactionProps: result,
                    transactPopup: true
                })),
                (error) => {
                    console.log("Error in getting parameters for leaveFeedback");
                    console.log(error);
                    this.notify("Error", `Could not complete operation (error in ${error.type})`, "danger");
                    this.setState(o => ({...o, feedbackLeaving: null, feedbackOpen: false}));
                }
            )
        }
    };

    /*
    * Invoked when the user confirms the transaction (after reviewing it in an informative dialog), this function
    * effectively writes the user’s feedback for the current Content to the Catalog.
    */
    confirmedFeedback = () => {
        const { feedbackLeaving, feedbackToLeave, transactionProps, feedback } = this.state;
        const { appreciation, fairness, suggest } = feedback;
        this.Catalog.methods
            .leaveFeedback(feedbackToLeave[feedbackLeaving].address, appreciation, fairness, suggest)
            .send({from: this.props.account, gas: transactionProps.gas, gasPrice: transactionProps.gasPrice})
            .then(
                (result) => {
                    this.notify("Success!", `Left feedback for content ${feedbackToLeave[feedbackLeaving].description}`, "success");
                    const feedList = feedbackToLeave.filter((item, idx) => idx !== feedbackLeaving).map((item, idx) =>
                        ({...item, index: idx})
                    );
                    this.setState(o => ({...o, loading: false, feedbackLeaving: null, feedback: null, feedbackToLeave: feedList}));
                },
                (error) => {
                    console.log("error in confirm feedback");
                    console.log(error);
                    this.notify("Error", `Could not leave feedback for ${feedbackToLeave[feedbackLeaving].description}`, "danger");
                    this.setState(o => ({...o, loading: false, feedbackLeaving: null, feedback: null}));
                }
            );
        this.setState(o => ({...o, loading: true, feedbackOpen: false, transactionProps: {}, transactPopup: false}));
    };

    // Starts the consumption of a bought Content, estimating its gas cost.
    consume = (idx) => () => {
        const content = this.state.boughtContent[idx];
        if (content.address) {
            const abi = JSON.parse(json.contracts["ContentManagementBase.sol:ContentManagementBase"].abi);
            const CMB = new this.web3.eth.Contract(abi, content.address);
            getTransactionParameters(this.web3, CMB.methods.consumeContent(), this.props.account).then(
                (result) => this.setState(o => ({...o, transactionProps: result, transactPopup: true})),
                (error) => {
                    console.log("Error in getting parameters for consume");
                    console.log(error);
                    this.notify("Error", ‘Could not complete operation (error in ${error.type})‘, "danger");
                    this.setState(o => ({...o, consuming: null}))
                }
            );
            this.setState(oldState => ({...oldState, consuming: CMB}));
        } else {
            this.Catalog.methods.getContentAddress(this.web3.utils.asciiToHex(content.description)).call(
                {from: this.props.account}
            ).then(
                (result) => {
                    this.setState(oldState => ({
                        ...oldState,
                        boughtContent: oldState.boughtContent.map((v, i) =>
                            (i === idx) ? {...v, address: result.toString()} : v
                        )
                    }));
                    this.consume(idx)();
                },
                (error) => {
                    console.log(error);
                    this.notify("Error", "Could not get Content’s address", "danger");
                }
            )
        }
    };

    /*
    * Invoked when the user confirms the transaction (after reviewing it in an informative dialog), this function
    * effectively asks the Content for consumption, also registering callbacks for the "Content Provider" and "Can
    * Leave Feedback" solidity events.
    */
    confirmConsume = () => {
        this.state.consuming.once(’ProvideContent’, {filter: {user: this.props.account}, fromBlock: "latest"},
            (error, event) => {
                if (error) console.log("provideContent error: " + error);
                else {
                    const contentName = this.web3.utils.hexToAscii(event.returnValues.content);
                    this.notify("Success!", ‘Content "${contentName}" has been provided!‘, "success", 10000);
                    const bought = this.state.boughtContent
                        .filter(content => content.description !== contentName)
                        .map((item, idx) => ({...item, index: idx}));
                    this.setState(oldState => ({...oldState, boughtContent: bought}));
                }
            }
        );
        this.state.consuming.once(’CanLeaveFeedback’, {filter: {user: this.props.account}, fromBlock: "latest"},
            (error, event) => {
                if (error) console.log("canLeaveFeedback error: " + error);
                else  {
                    const contentName = this.web3.utils.hexToAscii(event.returnValues.content);
                    this.notify("Leave feedback", ‘You can now leave feedback for ${contentName}‘, "material", 10000);
                    const feedbacks = [...this.state.feedbackToLeave, {
                        index: this.state.feedbackToLeave.length,
                        description: contentName,
                        address: event.address,
                    }];
                    this.setState(oldState => ({...oldState, feedbackToLeave: feedbacks}));
                }
            }
        );
        this.state.consuming.methods.consumeContent().send({
            from: this.props.account,
            gas: this.state.transactionProps.gas,
            gasPrice: this.state.transactionProps.gasPrice,
        }).then(
            (result) => {
                this.setState(o => ({...o, loading: false}));
                this.notify("Success!", "Request for access sent", "success");
            },
            (error) => {
                console.log(error);
                this.setState(o => ({...o, loading: false}));
                this.notify("Error", "Request for access denied or other error, check console log", "danger");
            }
        );
        this.setState(oldState => ({...oldState, loading: true, consuming: null, transactionProps: {}, transactPopup: false}));
    };

    // Starts the purchase or renewal of a Premium subscription, estimating the gas cost of the operation.
    buyPremium = (price, acc) => () => {
        const method = (acc) ? this.Catalog.methods.giftPremium(acc) : this.Catalog.methods.buyPremium();
        getTransactionParameters(this.web3, method, this.props.account, {value: price}).then(
            (result) => this.setState(o => ({...o, transactionProps: {price, ...result}, transactPopup: true})),
            (error) => {
                console.log("Error in getting parameters for buyPremium");
                console.log(error);
                this.notify("Error", ‘Could not complete operation (error in ${error.type})‘, "danger");
                this.setState(o => ({...o, giftingPremium: false, premiumMethod: null}));
            }
        );
        this.setState(o => ({...o, buyingPremium: false, premiumMethod: method}));
    };

    /*
    * Invoked when the user confirms the transaction (after reviewing it in an informative dialog), this function
```

```
     * effectively purchases or renews a Premium subscription (either for the user or gifted to another user) from the
     * Catalog.
     */
    confirmPremium = () => {
        this.state.premiumMethod.send({
            from: this.props.account,
            gas: this.state.transactionProps.gas,
            gasPrice: this.state.transactionProps.gasPrice,
            value: this.state.transactionProps.price
        }).then(
            (result) => {
                let premium;
                if (this.state.giftingPremium) {
                    this.notify("Success!", "Gifted a premium subscription to the other user", "success");
                    premium = this.state.isPremium;
                } else {
                    this.notify("Success!",
                        `You ${this.state.isPremium ? "extended your" : "bought a"} Premium subscription!`, "success");
                    premium = true;
                }
                this.setState(o => ({...o, loading: false, isPremium: premium, premiumMethod: null, giftingPremium: false}));
            },
            (error) => {
                console.log(error);
                this.notify("Error", "Could not buy/gift content", "danger");
                this.setState(o => ({...o, loading: false, premiumMethod: null, giftingPremium: false}));
            }
        );
        this.setState(o => ({...o, transactionProps: {}, transactPopup: false, loading: true}));
    };

    // Adds an author to the list of subscriptions for notifications
    addSubAuthor = (author) => {
        if (this.otherState.authorSubs.includes(author)) {
            this.notify("Warning", `You was already subscribed to new content by ${author}`, "material");
        } else {
            this.otherState.authorSubs.push(author);
            this.notify("Success!", `You've subscribed to new content by ${author}!`, "success");
        }
    };

    // Adds a genre to the list of subscriptions for notifications
    addSubGenre = (genre) => {
        if (this.otherState.genreSubs.includes(genre)) {
            this.notify("Warning", `You was already subscribed to new content with genre ${genre}`, "material");
        } else {
            this.otherState.genreSubs.push(genre);
            this.notify("Success!", `You've subscribed to new content with genre ${genre}!`, "success");
        }
    };

    /*
     * This is a React state function that will be called only once, after the component is mounted in the virtual
     * DOM but before it gets rendered.
     *
     * First, this function populates the content list. Then it registers some callbacks for Solidity events
     * fired when there's a new content available, when the Catalog grants access to a Content, and when a(n extension
     * to the) Premium subscription is made available for the user. Finally, it registers a repeating function that
     * checks if the user is still a Premium subscriber every 20 seconds.
     */
    componentDidMount() {
        this.updateContentList();
        this.Catalog.events.NewContentAvailable({fromBlock: "latest"}).on('data', (event) => {
            let show = false;
            if (this.otherState.genreSubs.includes(this.web3.utils.hexToAscii(event.returnValues.genre))
                || this.otherState.authorSubs.includes(this.web3.utils.hexToAscii(event.returnValues.author)))
                show = true;
            if (show) {
                this.notify("New content!",
                    `Content ${this.web3.utils.hexToAscii(event.returnValues.descr)} has been published on the Catalog!`,
                    "material", 10000);
            }
            if (!this.state.contentList.some((content) =>
                content.description === this.web3.utils.hexToAscii(event.returnValues.descr)
            )) {
                const list = [...this.state.contentList, {
                    index: this.state.contentList.length,
                    description: this.web3.utils.hexToAscii(event.returnValues.descr),
                    genre: this.web3.utils.hexToAscii(event.returnValues.genre),
                    author: this.web3.utils.hexToAscii(event.returnValues.author),
                    price: event.returnValues.price,
                    views: 0,
                }];
                this.setState(oldState => ({...oldState, contentList: list, filteredList: list}));
            }
        }).on('error', console.log);
        this.Catalog.events.GrantedAccess({filter: {user: this.props.account}, fromBlock: "latest"})
            .on('error', error => console.log("Error in GrantedAccess: " + error))
            .on('data', event => {
                const { boughtContent, contentList } = this.state;
                const description = this.web3.utils.hexToAscii(event.returnValues.description);
                this.notify("Access!", `Obtained access to "${description}"!`, "success", 10000);
                const refresh = [...boughtContent, {
                    description: description,
                    index: boughtContent.length,
                    address: event.returnValues.requested
                }];
                const newList = contentList.map((item) =>
                    (item.description === description) ?
                        {...item, address: event.returnValues.requested} :
                        item
                );
                this.setState(oldState => ({...oldState, contentList: newList, boughtContent: refresh}));
            });
        this.Catalog.events.GotPremium({filter: {user: this.props.account}, fromBlock: "latest"})
            .on('error', error => console.log("GotPremium: " + error))
            .on(’data’, event => {
                console.log(event);
                this.notify("Premium!", ‘Obtained or extended a Premium subscription!`, "success", 10000);
                this.setState(oldState => ({...oldState, premiumFetching: false, isPremium: true}));
            });
        const premiumfun = () => {
            this.setState(oldState => ({...oldState, premiumFetching: true}));
            this.Catalog.methods.isPremium(this.props.account).call({from: this.props.account}).then(
                (result) => {
                    this.setState(oldState => ({...oldState, premiumFetching: false, isPremium: result}));
                },
                (error) => {
                    this.notify("Error", "Could not retrieve premium subscription status", "danger");
                    this.setState(oldState => ({...oldState, premiumFetching: false}))
                }
            );
        };
        this.premiumUpdates = window.setInterval(premiumfun, 20000);
        premiumfun();
    }

    // Unregisters the "check for premium status" function.
    componentWillUnmount() {
        if (this.premiumUpdates) window.clearInterval(this.premiumUpdates);
    }

    render() {
        const {
            classes,
        } = this.props;

        const {
            authorList,
            genreList,
            panel,
            filteredList,
            contentList,
            boughtContent,
            feedbackToLeave,
            buyingPrice,
            buyingAction,
            consuming,
            premiumFetching,
            isPremium,
            transactionProps,
            transactPopup,
            buyingPremium,
            premiumMethod,
            giftingPremium,
            feedbackOpen,
            feedback,
        } = this.state;

        const defaultFun = x => x;

        const attributes = [
            {
                name: "genre",
                description: "Genre",
                copy: false,
                makeSmall: defaultFun
            },
            {
                name: "author",
                description: "Author",
                copy: false,
                makeSmall: defaultFun,
            },
            {
                name: "price",
                description: "Price",
                copy: false,
                makeSmall: defaultFun,
                extra: prettifyWei,
            },
            {
                name: "views",
                description: "Number of views",
                copy: false,
                makeSmall: defaultFun,

            },
        ];

        return (
            <div>
                <AppDrawer drawer={true} writeTitle={this.renderTitle} render={this.menu} loading={this.state.loading}>
                    <ReactNotification
                        ref={this.notificationRef}
                        types={[{
                            htmlClasses: classes.notification,
                            name: "material",
                        }]}
                    />
                    {(() => {
                        switch (panel) {
                            case Panels.FULL_LIST:
                                return(
                                    <div>
                                        <ContentList
                                            attributes={attributes}
                                            action={(i) => () => {
                                                this.otherState.contentBuying = i;
                                                this.setState(oldState => ({...oldState, buyingPrice: contentList[i].price}));
                                            }}
                                            actionName={"Buy"}
                                            action2={(i) => () => {
                                                this.otherState = { contentBuying: i, isPremiumTransaction: true };
                                                this.buy()()}
                                            }
                                            action2Name={"Obtain (Premium)"}
                                            list={filteredList}
                                            notify={this.notify}
                                        />
                                    </div>
                                );
                            case Panels.PLAY:
                                return(
                                    <div>
                                        <div className={classes.subheading}>
                                            <Typography variant={"body1"} className={classes.subMargin}>
                                                The list may not be synced with the
```

```
server.
                        </Typography>
                        <Button variant={"contained"}
                                size={"small"}
                                onClick={this.refreshBought}
                                color={"secondary"}
                                className={classes.subMargin}>
                            Refresh now
                        </Button>
                    </div>
                    <ContentList
                        className={classes.normalMargin}
                        attributes={[]}
                        action={this.consume}
                        actionName={"Consume"}
                        list={boughtContent}
                        notify={this.notify}
                    />
                </div>
            );
        case Panels.FEEDBACK:
            return(
                <div>
                    <div className={classes.subheading}>
                        <Typography variant={"body1"} className={classes.subMargin}>
                            The list may not be synced with the server.
                        </Typography>
                        <Button variant={"contained"}
                                size={"small"}
                                onClick={this.refreshFeedback}
                                color={"secondary"}
                                className={classes.subMargin}>
                            Refresh now
                        </Button>
                    </div>
                    <ContentList
                        className={classes.normalMargin}
                        attributes={[]}
                        action={this.leaveFeedback}
                        actionName={"Leave Feedback"}
                        list={feedbackToLeave}
                        notify={this.notify}
                    />
                </div>
            );
        case Panels.BUY:
            return (
                <div>
                    <div style={{display: "flex"}}>
                        <Typography variant={"subtitle2"}>Premium subscription:</Typography>
                        <Typography
                            style={{marginLeft: 16}}
                            variant={"subtitle2"}
                            color={"inherit"}
                            className={premiumFetching ?
                                classes.prFetch :
                                (isPremium ? classes.premium : classes.notPremium)}>
                            {premiumFetching ?
                                "Fetching..." :
                                (isPremium ? "Active" : "Not active")}
                        </Typography>
                    </div>
                    <div style={{display: "flex", justifyContent: "center"}}>
                        <Button
                            color={"primary"}
                            variant={"extendedFab"}
                            className={classes.button}
                            onClick={() => this.setState(o => ({...o, buyingPremium: true, giftingPremium: false}))} >
                            {isPremium ?
                                <Timelapse className={classes.extendedIcon}/> :
                                <QueuePlayNext className={classes.extendedIcon}/>}
                            {isPremium ? "Extend" : "Buy"}
                        </Button>
                        <Button
                            color={"primary"}
                            variant={"extendedFab"}
                            className={classes.button}
                            onClick={() => this.setState(o => ({...o, buyingPremium: true, giftingPremium: true}))} >
                            <CardGiftcard className={classes.extendedIcon}/>
                            Gift
                        </Button>
                    </div>
                </div>
            );
        case Panels.CHARTS:
            return(
                <Charts
                    authors={authorList}
                    genres={genreList}
                    web3={this.web3}
                    addSubAuthor={this.addSubAuthor}
                    addSubGenre={this.addSubGenre}
                    Catalog={this.Catalog}
                    account={this.props.account}/>
            );
        default:
            return (
                <div>
                    Not implemented.
                </div>
            );
    }
})()}
{feedbackOpen ?
    <FeedbackForm
        cancel={ () =>  this.setState(o => ({...o, transactionProps: {}, feedback: null, feedbackOpen: false, feedbackLeaving: null})) }
        confirm={this.feedbackFormSubmitted}
    /> : null
}
{buyingPrice ?
<BuyGiftDialog
    price={buyingPrice}
    cancel={() => this.setState(oldState => ({...oldState, buyingPrice: 0, buyingAction: false}))}
    callback={this.buy}
/> : null}
{buyingPremium &&
<BuyGiftPremiumDialog
    account={this.props.account}
    catalog={this.Catalog}
```

```
                        confirm={this.buyPremium}
                        gifting={giftingPremium}
                        cancel={(error) => () => {
                            this.setState(o => ({...o, buyingPremium: false, giftingPremium: false}));
                            error && this.notify("Error", "Could not fetch Premium subscription price from Catalog", "danger");
                        }}
                    />}
                    {transactPopup &&
                    <TransactionConfirmation
                        {...transactionProps}
                        ok={buyingAction ? this.confirmBuy :
                            (premiumMethod? this.confirmPremium :
                                (consuming? this.confirmConsume :
                                    (feedback ? this.confirmedFeedback : null)))}
                        cancel={() => this.setState(oldState => ({...oldState, buyingAction: false, consuming: false, transactionProps: {}, transactPopup: false}))}
                    />}
                </AppDrawer>
            </div>
        );
    }
}

UserApp.propTypes = {
    classes: PropTypes.object.isRequired,
    web3: PropTypes.object.isRequired,
    account: PropTypes.string.isRequired,
    catalog: PropTypes.string.isRequired,
    manageCenter: PropTypes.func.isRequired,
};

export default withStyles(styles)(UserApp);
```

```
/*
 *
 *  University di Pisa - Master's Degree in Computer Science and Networking
 *
 *  Final Project for the course of Peer to Peer Systems and Blockchains
 *
 *  Teacher: Prof. Laura Ricci
 *
 *  Candidate: Orlando Leombruni, matricola 475727
 *
 *  File: App.js
 *
 */

import React from 'react';
import { withStyles } from '@material-ui/core/styles';
import { Dialog, Typography } from '@material-ui/core';
import classNames from 'classnames';
import LoginPage from './LoginPage';
import UserApp from './UserApp';
import CreatorApp from './CreatorApp';
import Web3 from 'web3';
import PropTypes from "prop-types";
import CatalogManager from "./CatalogManager";
import { AppStyle as styles } from '../styles/MaterialCustomStyles';

/*
 *
 * App Class
 *
 * A React Component that acts as the "foundation" for the web application. It manages the connection
 * to a Web3 instance and provides the other Components with needed data such as the EOA used for the operations.
 *
 */

class App extends React.Component {

    constructor(props) {
        super(props);
        this.state = {
            centerContent: true,
            connecting: true,
            account: "",
            catalog: "",
            app: "none",
            noLogin: false,
            error: false,
        };

        // Initialize the Web3 instance
        var web3;
        if (window.ethereum) { // Web3 provided by latest versions of browsers/extensions (e.g. Metamask, Mist)
            web3 = new Web3(window.ethereum);
            window.ethereum.enable().then(
                () => this.setState(oldState => ({...oldState, noLogin: true, connecting: false})),
                () => this.setState(oldState => ({error: true}))
            );
        } else { // Web3 injected by old browsers/extensions, or with an instance of a compatible client
            web3 = new Web3(Web3.givenProvider || new Web3.providers.WebsocketProvider("ws://localhost:8545"));
            web3.eth.net.isListening()
                .then(() => this.setState(oldState => ({...oldState, connecting: false})),
                      () => this.setState(oldState => ({error: true})));
        }
        /*
         * Workaround for latest versions of Web3 ( >= 1.0.0-beta.33 )
         * This fixes a bug where Solidity events or functions that return 0 or empty arrays cause Web3 to throw error
         * See https://github.com/ethereum/web3.js/issues/1916
         * Should be fixed in Web3.js version 1.0.0-beta.37
         */
        console.log("using web3 ver. ", web3.version);
        const version = +web3.version.slice(-2);
        if (version >= 33 && version < 37) {
            console.log("Applying fix patch");
            web3.eth.abi.decodeParameters = function(outputs, bytes) {
                if (bytes === '0x') {
                    bytes = '0x00';
                }
                return web3.eth.abi.__proto__.decodeParameters(outputs, bytes)
            }
        }
        this.web3 = web3;
        this.submit = this.submit.bind(this);
    };

    /*
     *  Takes values input by the user (which EOA to use, which Catalog to connect to, which application to use)
     *  and triggers the start of the chosen application (user interface/creator interface).
     *  Used as a callback by the LoginPage component.
     */
    submit(account, catalog, app) {
        if (app === "error") this.setState(o => ({...o, error: true}));
        this.setState(oldState => ({...oldState, account, catalog, app}));
    };

    /*
     *  Centers the graphic elements in the window, if required.
     *  Used as a callback by various components.
     */
    fixCenter = (what) =>
        this.setState(oldState => ({...oldState, centerContent: what}));

    render() {
        const { connecting, account, catalog, app, error, centerContent, noLogin } = this.state;
        const { isCatalog, classes } = this.props;
        return (
            <div className={(centerContent ? classNames("pagewide", classes.centered) : "pagewide")}>
                {(error) ?
                    <Dialog open={true} disableBackdropClick={true} disableEscapeKeyDown={true}>
                        <Typography id="error" variant={"body1"} color={"textPrimary"}>
                            Could not connect to an Ethereum node. Check your provider (geth, Metamask, ...).
                        </Typography>
                    </Dialog> :
                    (connecting) ?
                        <Typography variant={"body1"} color={"textPrimary"}>
                            Connecting to Ethereum node...
                        </Typography> :
                        (isCatalog) ?
```

```
                            <CatalogManager web3={this.web3} noLogin={noLogin} manageCenter={this.fixCenter} /> :
                            (app === "none") ?
                                <LoginPage onSubmit={this.submit} noLogin={noLogin} web3={this.web3} manageCenter={this.fixCenter} /> :
                                (app === "user") ?
                                    <UserApp web3={this.web3} account={account} catalog={catalog} manageCenter={this.fixCenter} /> :
                                    <CreatorApp web3={this.web3} account={account} catalog={catalog} manageCenter={this.fixCenter} />
                }
            </div>
        );
    };
}

App.propTypes = {
    classes: PropTypes.object.isRequired,
    isCatalog: PropTypes.bool.isRequired,
};

export default withStyles(styles)(App);
```

```
/*
 *
 *  University di Pisa - Master’s Degree in Computer Science and Networking
 *
 *  Final Project for the course of Peer to Peer Systems and Blockchains
 *
 *  Teacher: Prof. Laura Ricci
 *
 *  Candidate: Orlando Leombruni, matricola 475727
 *
 *  File: Utils.js
 *
 *  Collection of utility functions, used throughout the webapp.
 */

import BN from ’bn.js’;
import Typography from "@material-ui/core/Typography/Typography";
import React from "react";

// Given a numerical amount of wei (e.g. 5003000000000000000) returns a human-readable string (e.g. "5.003 ether").
export const prettifyWei = wei => {
    const weiBN = new BN(wei);
    const mweiBN = weiBN.divn(1000000);
    const finneyBN = mweiBN.div(new BN(1000000000));
    if (finneyBN.gtn(0)) {
        return (finneyBN.toNumber() / 1000) + " ether";
    } else
    if (mweiBN.gtn(0)) {
        return (mweiBN.toNumber() / 1000) + " gwei";
    } else return weiBN.toString() + " wei";
};

// Bundles together text and style for app bar titles.
export const makeTitle = (title, className) =>
    <Typography key={"title"} variant="h6" color="inherit" className={className} noWrap>
        {title}
    </Typography>;

/*
 * Given a Solidity method (accessed through a Web3 instance), its parameters and the EOA from which the operation
 * must be performed, asks the blockchain for a gas estimate, current average gas cost and EOA balance, then delivers
 * it to the asking function with a Promise.
 */
export const getTransactionParameters = (web3, method, account, otherParams) =>
    new Promise((resolve, reject) => {
        let gas = null, balance = null, gasPrice = null;
        const params = {
            from: account,
            ...otherParams
        };
        method.estimateGas(params).then(
            (result) => {
                gas = result.toString();
                if (balance && gasPrice) resolve({balance, gas, gasPrice});
            },
            (error) => {
                reject({type: "estimateGas", stackTrace: error});
            }
        );
        web3.eth.getBalance(account).then(
            (result) => {
                balance = result.toString();
                if (gas && gasPrice) resolve({balance, gas, gasPrice});
            },
            (error) => {
                reject({type: "getBalance", stackTrace: error});
            }
        );
        web3.eth.getGasPrice().then(
            (result) => {
                gasPrice = result.toString();
                if (gas && balance) resolve({balance, gas, gasPrice});
            },
            (error) => {
                reject({type: "getGasPrice", stackTrace: error});
            }
        )
    });
```