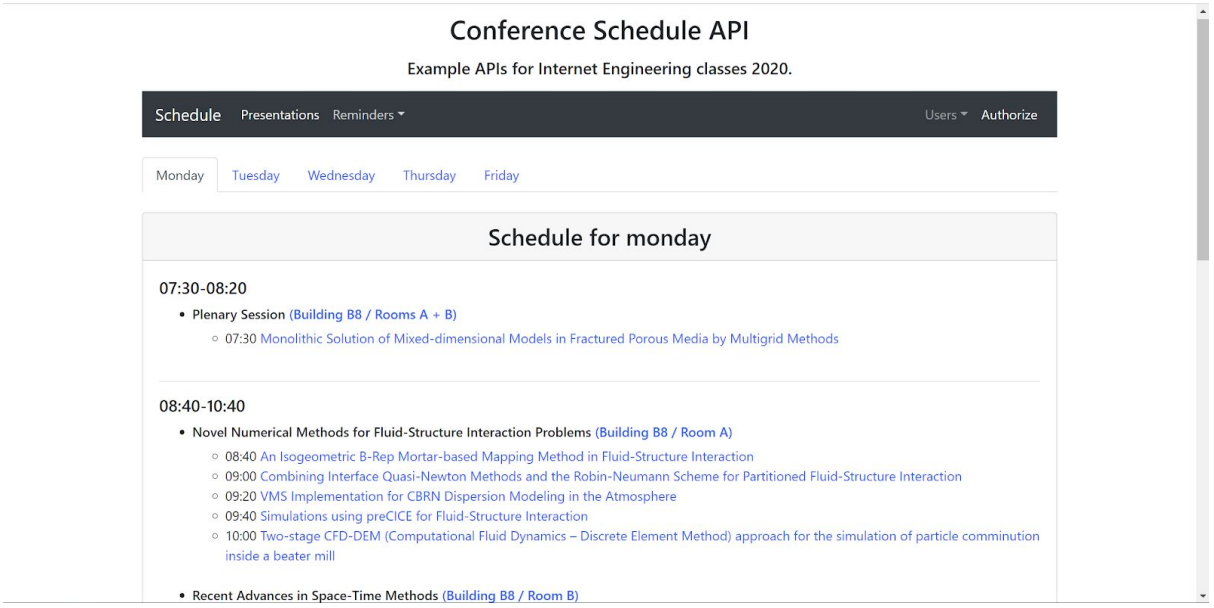
Michał Pałys 299727

Project Report
Front-end application for Conference Schedule API

The application has been written in JavaScript using React as a front-end framework. It is using Axios library to send requests to the backend and Bootstrap for user interface. This report describes in details all the features of the application, as well as the code behind it.

1. Home page - Schedule page

Schedule page contains information about all planned presentations and sessions for each day of the week. Moreover, it provides user information about localisation of each event in form of a link to Google Maps. If a presentation contains an abstract file, a user can access it by simply clicking on the presentation name.



All the information is loaded at the time of mounting the page component:

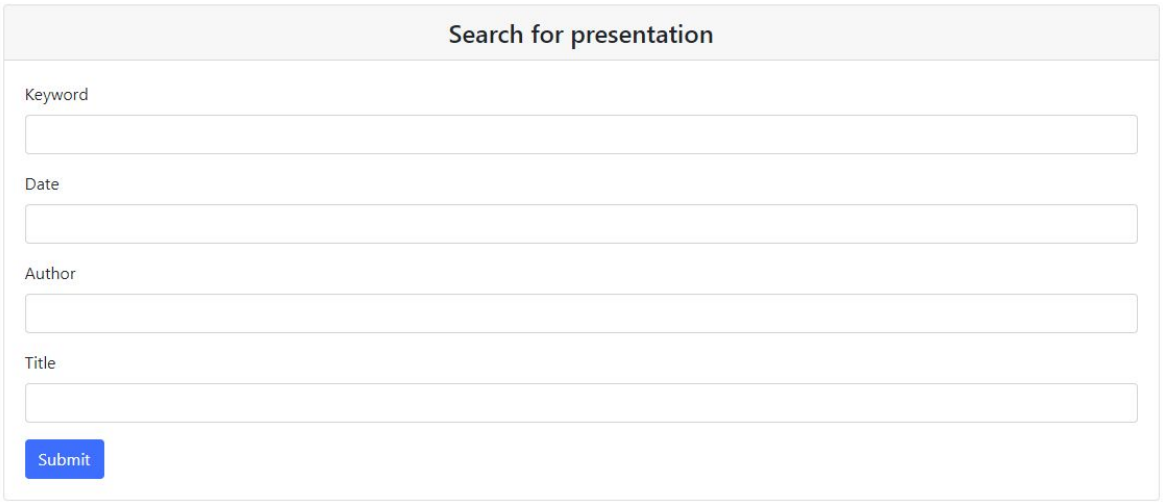
```
componentDidMount() {
    this.getSchedulesFromApi();
    this.getAllSessionsFromApi();
    this.getRoomsFromApi();
    this.getPresentationsFromApi();
}
```

```
getSchedulesFromApi() {
    axios.get(API_BASE_URL + "schedules?day=" + this.state.dayOfTheWeek)
        .then(res => {
            const data = res.data[0][this.state.dayOfTheWeek.toUpperCase()];
            this.setState({
                scheduledSessions: data
            });
        })
        .catch(error => console.log(error));
}

getAllSessionsFromApi() {
    axios.get(API_BASE_URL + "sessions")
        .then(res => {
            const data = res.data;
            this.setState({
                allSessions: data
            });
        })
        .catch(error => console.log(error));
}

getRoomsFromApi() {
    axios.get(API_BASE_URL + "rooms")
        .then(res => {
            const data = res.data;
            this.setState({
                rooms: data
            });
        })
        .catch(error => console.log(error));
}

getPresentationsFromApi() {
    axios.get(`${API_BASE_URL}presentations`)
        .then(res => {
            const presentations = res.data;
            this.setState({ presentations });
        })
        .catch(error => console.log(error));
}
```

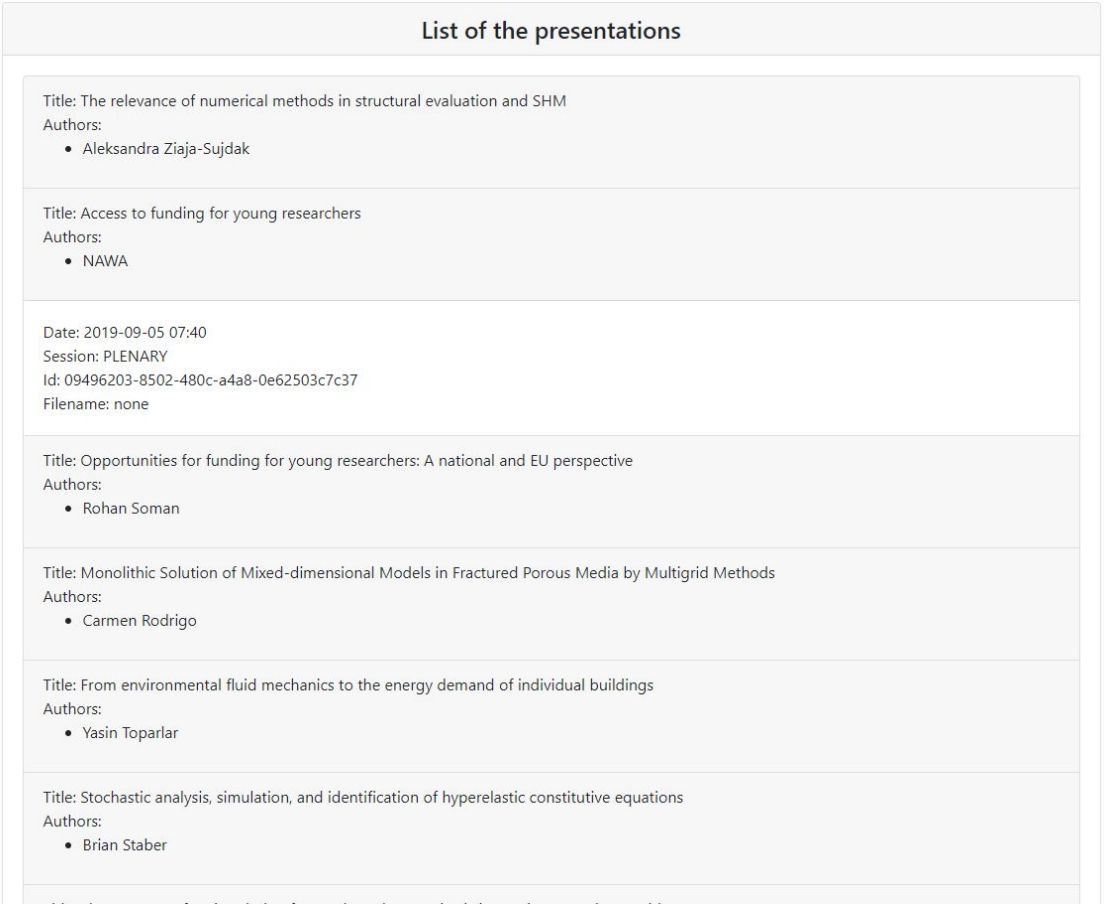
Each information is loaded by making a GET request to the respective end-point.

2. Presentations

The presentations page lets user to list all the presentations present in the system. There is a possibility to filter the results by keywords, a date, authors or a title:



The whole list has been made using Card and Accordeon components from ‘react-bootstrap’. To show information about specific presentation one has to simply clock on it.



The code behind it:

```
getDataFromApi(query = "") {
    axios.get(`${API_BASE_URL}presentations${query}`)
        .then(res => {
            const presentations = res.data;
            this.setState({ presentations });
        })
        .catch(error => console.log(error));
}

componentDidMount() {
    this.getDataFromApi();
}

handleInputChange(event) {
    const target = event.target;
    const value = target.value;
    const name = target.name;

    this.setState({
        [name]: value
    });
}

handleSubmit(event) {
    event.preventDefault();
    let query = "";

    if (this.state.keyword && this.state.keyword.length > 0) {
        query += ("keyword=" + this.state.keyword);
    }
    if (this.state.date) {
        query += ("&date=" + this.state.date);
    }
    if (this.state.author) {
        query += ("&author=" + this.state.author);
    }
    if (this.state.title) {
        query += ("&title=" + this.state.title);
    }

    if (query.length > 0) query = "?" + query;

    this.getDataFromApi(query);
}
```

Every time user types a letter in the input form the respective state property of the component is changed. After clicking submit button the query to API end-point is concatenated and passed as a query when sending a GET request to API.

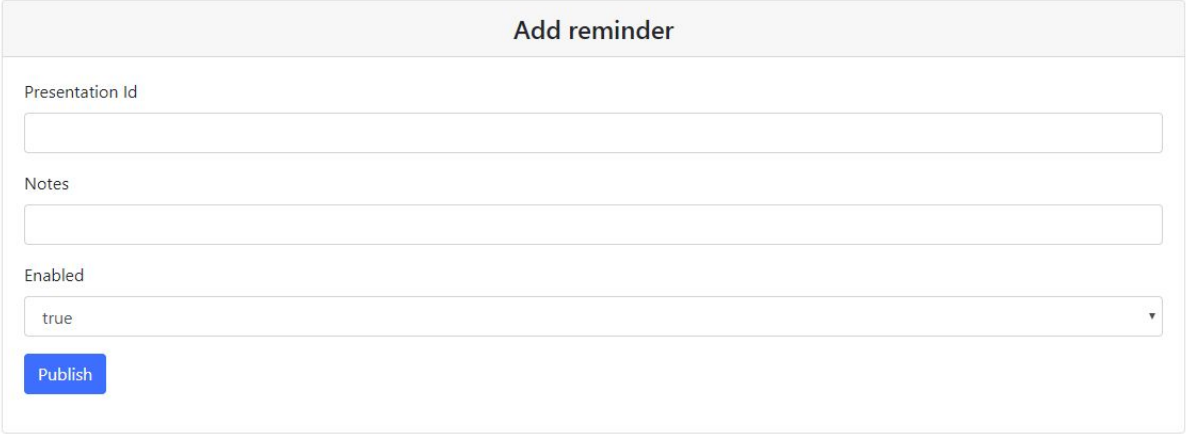
The code behind the presentations list:

```
<Card className="mt-4 mb-5">
    <Card.Header as="h4" className="text-center">
        List of the presentations
    </Card.Header>
    <Card.Body>
        <Accordion>
            {this.state.presentations.map(presentation =>
                <Card key={presentation.id}>
                    <Accordion.Toggle as={Card.Header} eventKey={presentation.id}>
                        Title: {presentation.title}
                        <br />
                        Authors:
                        <ul>
                            {presentation.authors.map(author => <li>{author}</li>)}
                        </ul>
                    </Accordion.Toggle>
                    <Accordion.Collapse eventKey={presentation.id}>
                        <Card.Body>
                            Date: {presentation.date.slice(0,10)} {presentation.date.slice(11,16)}<br />
                            Session: {presentation.session}<br />
                            Id: {presentation.id}<br />
                            Filename: {presentation.filename === "" ? "none" :      any
                                <a href={`${API_BASE_URL}abstracts/${presentation.filename}`}
                                    target="_blank" rel="noopener noreferrer">
                                    {presentation.filename}
                                </a>
                            }
                        </Card.Body>
                    </Accordion.Collapse>
                </Card>
            )}
        </Accordion>
    </Card.Body>
```

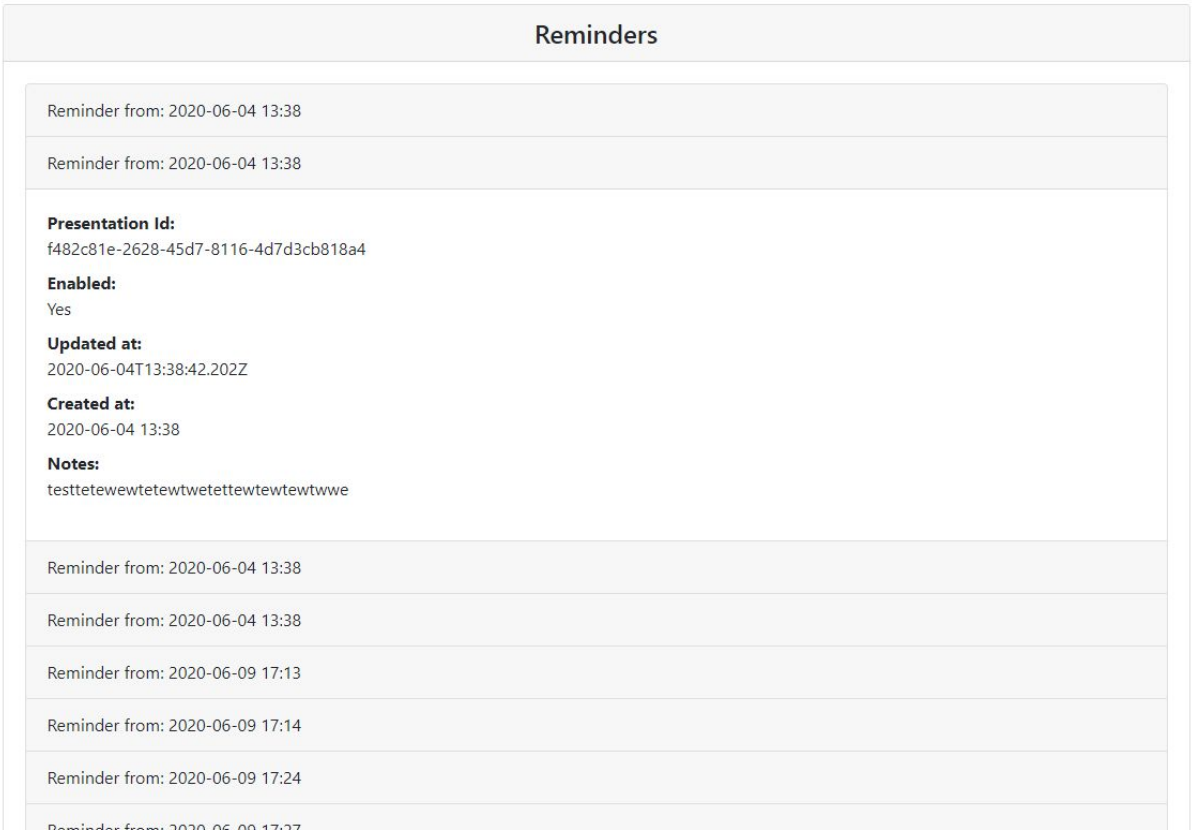
As mentioned before, I have used React Bootstrap components such as Card and Accordion to pleasantly and accurately present a data to the user.

3. Reminders

User has a possibility to list all the reminders that he has made, as well as make a new one:

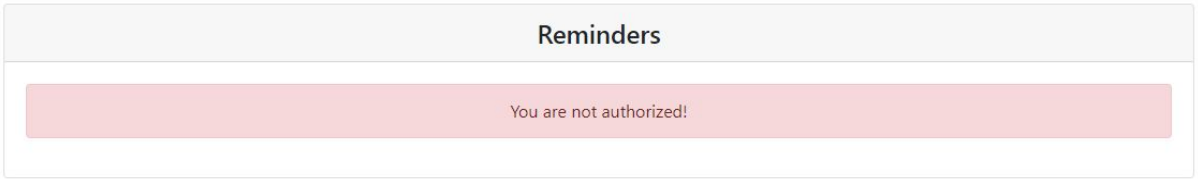


When adding a new reminder, one must remember to pass the correct presentation ID. Otherwise, a suitable warning alert will be displayed. Simillary, a user must specify a note of the reminder and choose whether the reminder is enabled or disabled.

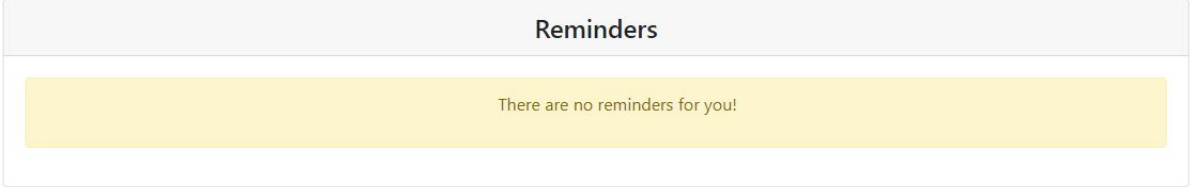


Similarly to presentations list, the all reminders list has also been made using React Bootstrap Card and Accordeon components. Each reminder's information is being displayed after clicking on the Accordeon header, revealing data such as creation and update date, notes text or refering presentation Id.

This page can be accessed only by authenticated users. When there is no JWT token on the local storage a user gets appropiate alert:



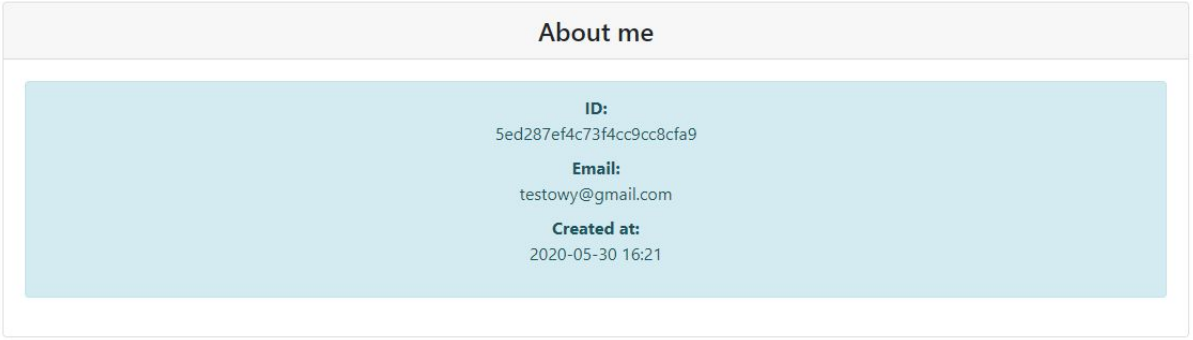
An appropriate alert is also shown if there are no reminders for an authenticated user.



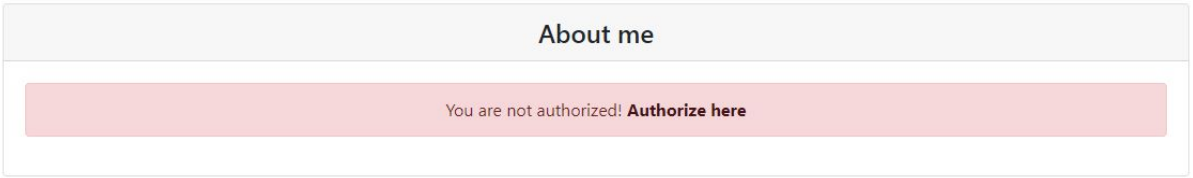
```
<Card>
    <Card.Header as="h4" className="text-center">
        Reminders
    </Card.Header>
    <Card.Body>
        <Alert show={this.state.showWarning} variant='warning' className="text-center">
            <ul>
                {this.state.warningMessage}
            </ul>
        </Alert>
        <Alert show={this.state.showDanger} variant='danger' className="text-center">
            {this.state.dangerMessage}
        </Alert>
        <Accordion>
            {this.state.reminders.map(reminder =>
                <Card key={reminder.id}>
                    <Accordion.Toggle as={Card.Header} eventKey={reminder.id}>
                        Reminder from: {reminder.updatedAt.slice(0, 10)} {reminder.updatedAt.slice(11, 16)}
                    </Accordion.Toggle>
                    <Accordion.Collapse eventKey={reminder.id}>
                        <Card.Body>
                            <dl>
                                <dt>Presentation Id: </dt>
                                <dd>{reminder.presentationId}</dd>

                                <dt>Enabled: </dt>
                                <dd>{reminder.enabled ? "Yes" : "No"}</dd>

                                <dt>Updated at: </dt>
                                <dd>{reminder.updatedAt}</dd>

                                <dt>Created at: </dt>
                                <dd>{reminder.createdAt.slice(0, 10)} {reminder.createdAt.slice(11, 16)}</dd>

                                <dt>Notes: </dt>
                                <dd>{reminder.notes}</dd>
                            </dl>
                        </Card.Body>
                    </Accordion.Collapse>
                </Card>
            )}
        </Accordion>
```

4. Users

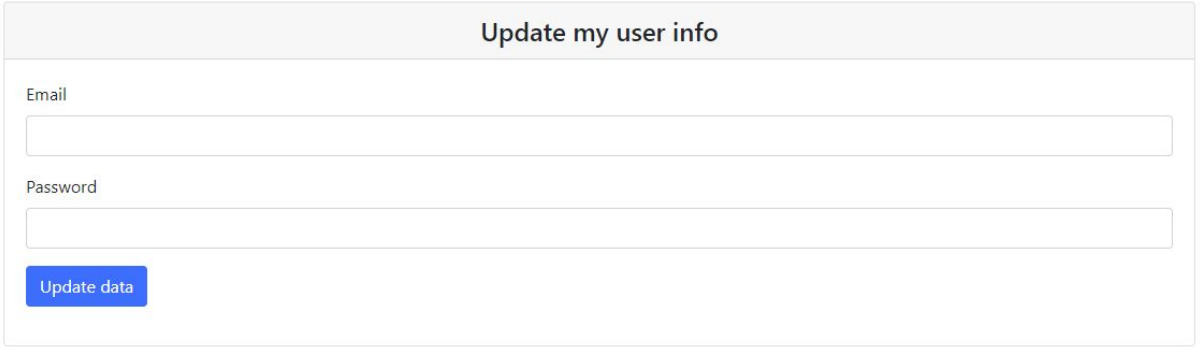
An authenticated user can display his account data by going to About me page. Information such as user ID, email and creation date are displayed.



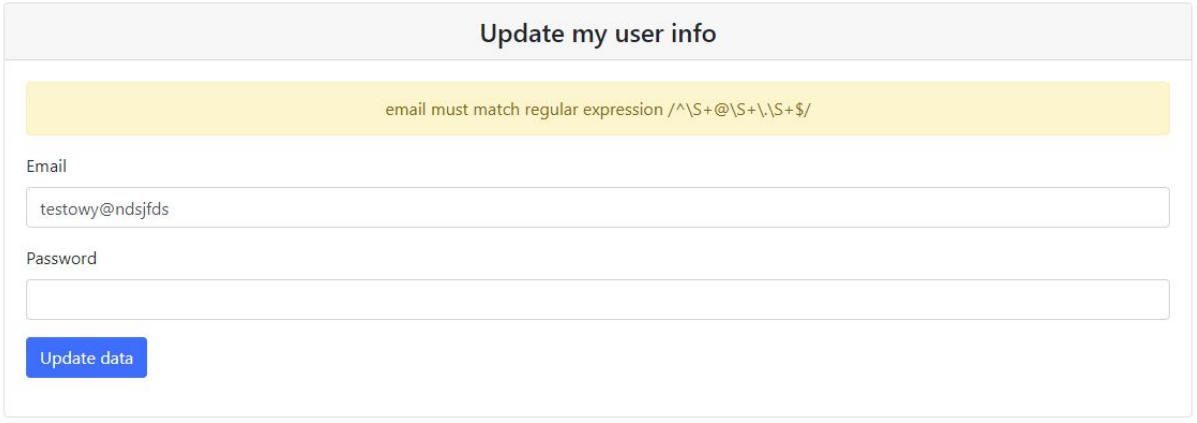
If the user is not authenticated he is shown an appropriate alert with a link to authentication page.



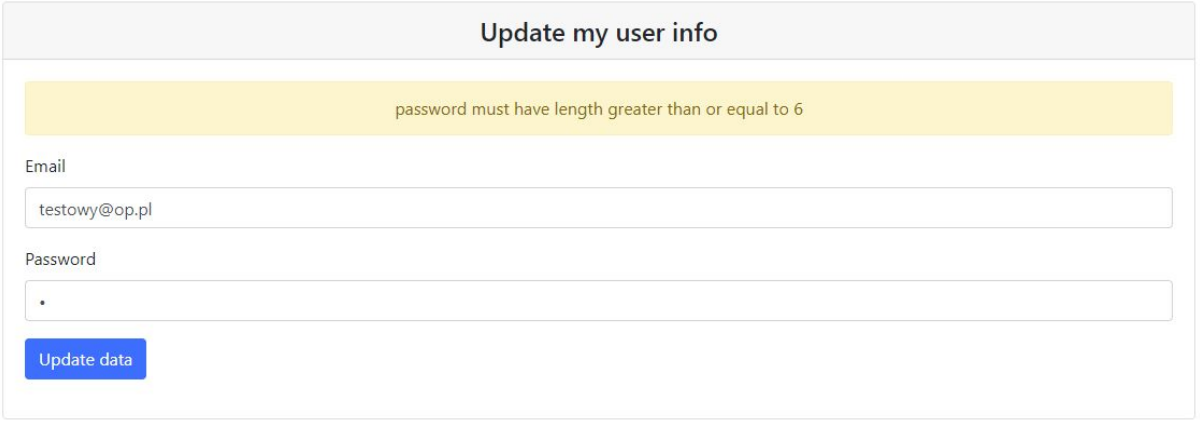
An authenticated user credentials can also be updated in the /update page by passing correct email and password.



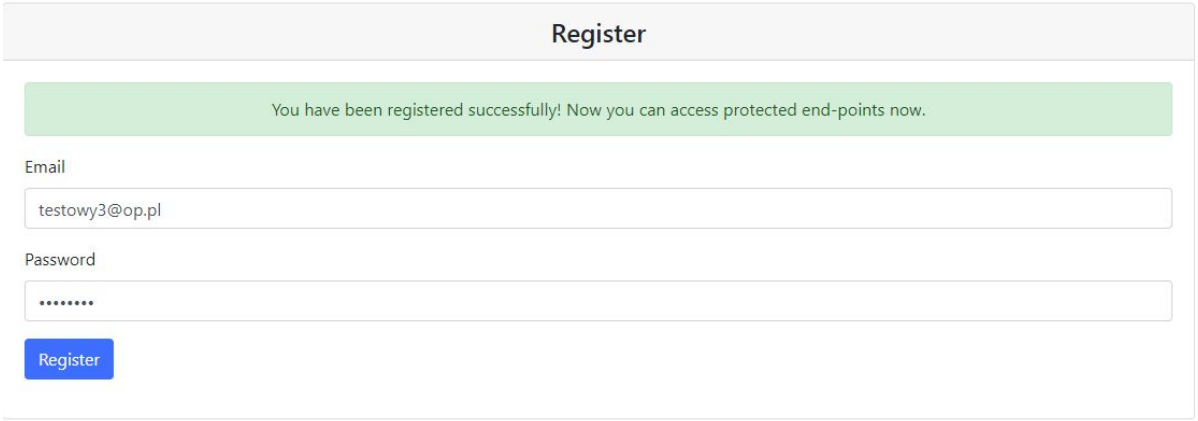
Upon entering incorrect data the user will get an alert containing error message.



or:



A new user can sign up by accessing the /register page. The form contains email and password fields and produces same alert messages as shown above in case of incorrect data passage.



The code behind handling alert messages for registration page:

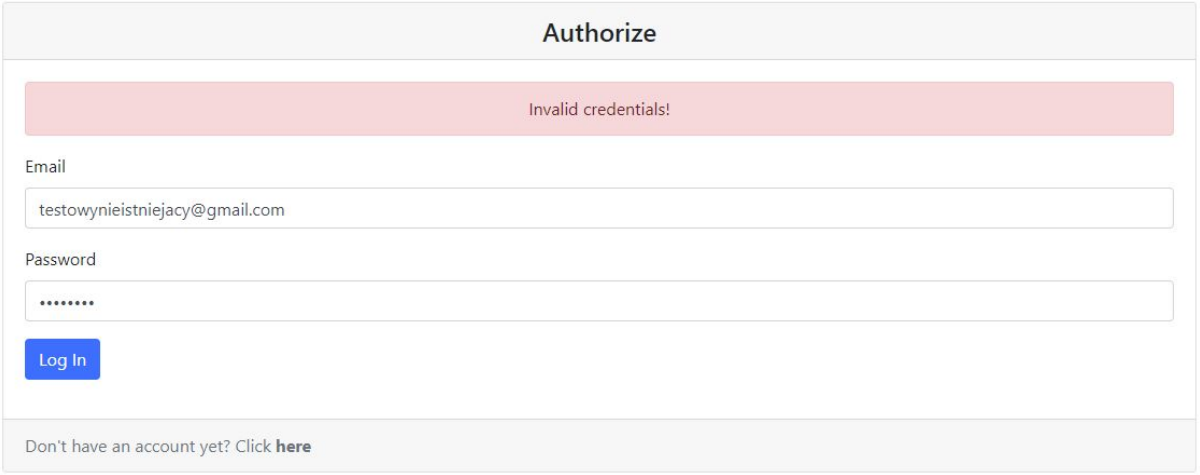
```
axios.post(`${API_BASE_URL}users`, {email: this.state.email, password: this.state.password})
    .then(res => {
        if (res.status === 201) {
            const token = res.data.token;
            localStorage.setItem("JWT", token);

            this.setState({
                showSuccess: true,
                showWarning: false,
                showDanger: false
            });
        }
    })
    .catch(error => {
        if (error.response) {
            if (error.response.status === 400) {
                const message = error.response.data.message;
                this.setState({
                    showSuccess: false,
                    showWarning: true,
                    showDanger: false,
                    warningMessage: message
                })
            }
            else if (error.response.status === 409) {
                const message = error.response.data.message;;
                this.setState({
                    showSuccess: false,
                    showWarning: false,
                    showDanger: true,
                    dangerMessage: message
                })
            }
        }
    });
```

The alert messages are kept in component's state, as well as their respective boolean type variables switching an alert on/off.

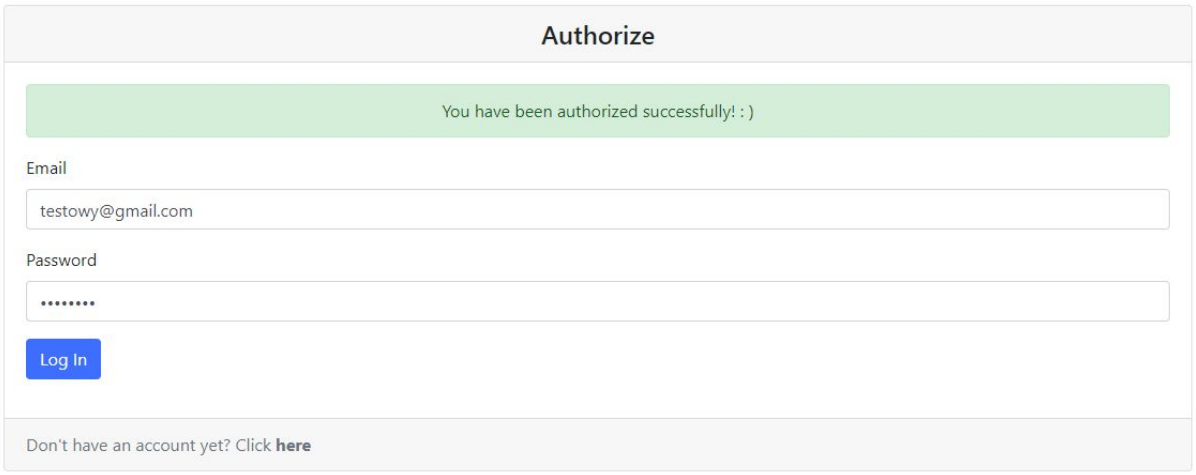
```
<Card.Body>
    <Alert show={this.state.showSuccess} variant='success' className="text-center">
        You have been registered successfully! Now you can access protected end-points now.
    </Alert>
    <Alert show={this.state.showWarning} variant='warning' className="text-center">
        {this.state.warningMessage}
    </Alert>
    <Alert show={this.state.showDanger} variant='danger' className="text-center">
        {this.state.dangerMessage}
    </Alert>
    <Form onSubmit={this.handleSubmit} className="mb-3">
        <Form.Group>
            <Form.Label>Email</Form.Label>
            <Form.Control type="email" name="email" value={this.state.email} onChange={this.handleInputChange} />
        </Form.Group>
        <Form.Group>
            <Form.Label>Password</Form.Label>
            <Form.Control type="password" name="password" value={this.state.password} onChange={this.handleInputChange} />
        </Form.Group>
        <Button variant="primary" type="submit">
            Register
        </Button>
    </Form>
</Card.Body>
```

5. Authorize

The registered user can authenticate (send a request to the end-point in order to receive an JWT token and store it in local storage) by accessing the Authorize page. Two required fields are email and password. If there is no such user in the database, an appropriate alert will be shown.



After successful authorization:



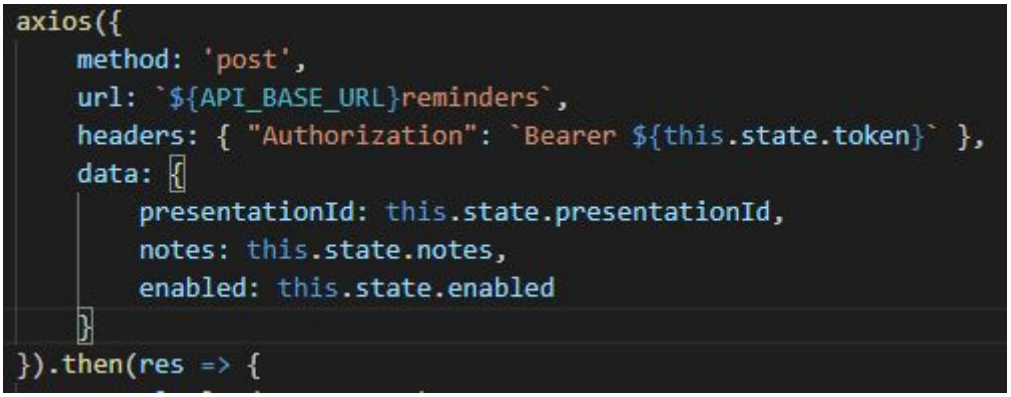
The received JWT token is stored in the local storage and used later in several other pages to access protected API's end-points.

```
axios.post(`${API_BASE_URL}auth`, {}, { headers })
    .then(res => {
        if (res.status === 201) {
            const token = res.data.token;
            localStorage.setItem("JWT", token);

            this.setState({
                showSuccess: true,
                showWarning: false,
                showDanger: false
            });
        }
    })
```

For example:

```
getDataFromApi() {
    const token = localStorage.getItem("JWT");
    this.setState({
        token: token,
    },
        () => {
            axios.get(`${API_BASE_URL}reminders`, { headers: { "Authorization": `Bearer ${this.state.token}` } })
                .then(res => {
                    if (res.status === 200) {
                        if (res.data === null || res.data.length === 0) {
                            this.setState({
                                warningMessage: "There are no reminders for you!",
                                showWarning: true,
                                showDanger: false
                            });
```

JWT token is needed to get all reminders list for a given user. Otherwise, an error with status code 401 is received.



JWT token is also needed when adding a new reminder. Custom headers and body have to be passed to the end-point in order to successfully add a new reminder.