Creating a CRUD-Enabled Scheduling App with Syncfusion React Scheduler, Node.js, and PostgreSQL

Ram Raju Elaiyaperumal • 10 min read • Nov 19, 2024 • Updated

Sunday
Monday
Tuesday
Wednesday
Thursday
Friday
00 AM
10:00 AM
:00 AM
12:00 AM
node JS

In this blog, we will discuss creating a scheduling application using React, Node.js, and PostgreSQL that demonstrates performing CRUD (create, read, update, delete) operations.

Let’s get started!

Prerequisites

Make sure the following are global installations in your environment:

- Node.js 14 or newer
- npm 5 or newer
- PostgreSQL

Also, ensure that you have Visual Studio Code installed on your machine.

Node.js RESTful API service

This section explains how to create a RESTful API service using Node.js. First, create a back-end folder and initialize a **package.json** file using the **npm init** command.

Then, install the following Node modules:

- Express: A web server module.
- Sequelize: A Node.js ORM for Postgres.
- Pg: Required for PostgreSQL.
- pg-hstore: For converting data into the PostgreSQL hstore format.

Run the following command to install the previously listed modules:

```
npm install express sequelize pg pg-hstore cors --save
```

Set up Express web server

In this example, we will use the Express web server to handle HTTP requests.

Import and initialize the Express server in the **backend/server.js** file. Refer to the following code.

```
const express = require("express");
const bodyParser = require("body-parser");
const cors = require("cors");
const app = express();
var corsOptions = {
  origin: "http://localhost:8081"
};
app.use(cors(corsOptions));
app.use(bodyParser.json());
app.use(bodyParser.urlencoded({ extended: true }));
app.get("/", (req, res) => {
  res.json({ message: "Welcome to Scheduler back-end service." });
});
const PORT = process.env.PORT || 8080;
app.listen(PORT, () => {
  console.log(`Server is running on port ${PORT}.`);
});
```

Run the following command to start the web server.

```
node server.js
```

Now the Node.js Express server is ready to handle the REST API requests.

Configure PostgreSQL database

Define the PostgreSQL database configurations in the **backend/config/db.config.js** file.

```
module.exports = {
  HOST: "localhost",
  USER: "postgres",
  PASSWORD: "admin",
  DB: "testdb",
  dialect: "postgres",
  pool: {
    max: 5,
    min: 0,
    acquire: 30000,
    idle: 10000
  }
};
```

Initialize Sequelize

Define the Scheduler component’s required fields in the **backend/models/scheduler.model.js** file with the following code.

```
module.exports = (sequelize, Sequelize) => {
const SchedulerEvents = sequelize.define("scheduleevents", {
    id: {
        type: Sequelize.INTEGER,
        primaryKey: true,
        autoIncrement: true,
    },
    starttime: {
        type: Sequelize.DATE,
        allowNull: false
    },
    endtime: {
        type: Sequelize.DATE,
        allowNull: false
    },
    subject: {
        type: Sequelize.STRING
    },
    location: {
        type: Sequelize.STRING
    },
    description: {
        type: Sequelize.STRING
    },
    isallday: {
        type: Sequelize.BOOLEAN
    },
    starttimezone: {
        type: Sequelize.STRING
    },
    endtimezone: {
            type: Sequelize.STRING
        },
        recurrencerule: {
            type: Sequelize.STRING
        },
        recurrenceid: {
            type: Sequelize.INTEGER
        },
        recurrenceexception: {
            type: Sequelize.STRING
        },
        followingid: {
            type: Sequelize.INTEGER
        },
        createdAt: {
            type: Sequelize.DATE,
            field: 'created_at'
        },

        updatedAt: {
          type: Sequelize.DATE,
          field: 'updated_at'
        }
    });
    return SchedulerEvents;
};
```

The previous model represents the scheduleevents table in the PostgreSQL database with the columns **id, starttime, endtime, subject, location,**

description, **isallday**, **starttimezone**, **endtimezone**, **recurrencerule**, **recurrenceid**, **recurrenceexception**, **followingid**, **createdAt**, and **updatedAt**.

Refer to the Scheduler component appointment fields documentation to learn more.

The next step is to initialize Sequelize in the **backend/models/index.js** file with the following code.

```
const dbConfig = require("../config/db.config.js");
const Sequelize = require("sequelize");
const sequelize = new Sequelize(dbConfig.DB, dbConfig.USER, dbConfig.PASSWORD,
{
  host: dbConfig.HOST,
  dialect: dbConfig.dialect,
  operatorsAliases: false,
  pool: {
    max: dbConfig.pool.max,
    min: dbConfig.pool.min,
    acquire: dbConfig.pool.acquire,
    idle: dbConfig.pool.idle
  }
});
const db = {};
db.Sequelize = Sequelize;
db.sequelize = sequelize;
db.scheduler = require("./scheduler.model.js")(sequelize, Sequelize);
module.exports = db;
```

Then, import and register Sequelize in the **server.js** file with the help of the sync method.

```
const db = require("./models");
db.sequelize.sync({ force: false }).then(() => {
  console.log("Drop and re-sync db.");
});
```

Create the controller

Create a **backend/controllers/scheduler.controller.js** file to handle the CRUD functions.

```
Const db = require(“../models”);
const SchedulerEvents = db.scheduler;
```

Create a **getData** method to retrieve all events from the database.

```
exports.getData = (req, res) => {
    SchedulerEvents.findAll()
        .then(data => {
            res.send(data);
        })
        .catch(err => {
            res.status(500).send({
                message:
                    err.message || "Some error occurred while retrieving Events."
            });
        });
};
```

Note: By default, the Scheduler component supports loading data on demand. When getting data requests, you can get the current view’s start and end dates in HTTP requests. Based on those parameters, you can filter the events and send the required events alone to the client side to improve the loading performance.

Let us define the **crudActions** methods to handle the create, update, and delete actions.

```
exports.crudActions = (req, res) => {

 if (req.body.added !== null && req.body.added.length > 0) {
    for (var i = 0; i < req.body.added.length; i++) {
        var insertData = req.body.added[i];
        SchedulerEvents.create(insertData)
            .then(data => {
                res.send(data);
            })
            .catch(err => {
                res.status(500).send({
                    message:
                        err.message || "Some error occurred while inserting the events."
                });
            });
      }
  }

  if (req.body.changed !== null && req.body.changed.length > 0) {
    for (var i = 0; i < req.body.changed.length; i++) {
        var updateData = req.body.changed[i];
        SchedulerEvents.update(updateData, { where: { id: updateData.id } })
            .then(num => {
                if (num == 1) {
                    res.send(updateData);
                } else {
                    res.send({
                        message: `Cannot update Event with id=${id}. Maybe Event was not found, or req.body is empty!`
                    });
                }
             })
```

```
                 .catch(err => {
                    res.status(500).send({
                        message: "Error updating Event with id=" + id
                    });
                 });
           }
       }

       if (req.body.deleted !== null && req.body.deleted.length > 0) {
           for (var i = 0; i < req.body.deleted.length; i++) {
             var deleteData = req.body.deleted[i];
             SchedulerEvents.destroy({ where: { id: deleteData.id } })
                .then(num => {
                    if (num == 1) {
                        res.send(deleteData);
                    } else {
                        res.send({
                            message: `Cannot delete Event with id=${id}. Maybe Event was not found!`
                        });
                    }
                 })
                 .catch(err => {
                    res.status(500).send({
                        message: "Could not delete Event with id=" + id
                    });
                 });
             }
        }
};
```

Define routes

Define the router paths in the **backend/routes/scheduler.routes.js** file.

```
module.exports = app => {
    const scheduleService = require("../controllers/scheduler.controller.js");
    var router = require("express").Router();
    router.post("/getData", scheduleService.getData);
    router.post("/crudActions", scheduleService.crudActions);
    app.use('/api/scheduleevents', router);
};
```

Also, include routes in the **server.js** (before app.listen()) file.

```
require("./routes/scheduler.routes")(app);
// set port, listen for requests
const PORT = process.env.PORT || 8080;
app.listen(PORT, () => {
  console.log(`Server is running on port ${PORT}.`);
});
```

Now the web server is ready to handle Scheduler data-binding operations. Next, we'll create the React Scheduler front-end application.

React Scheduler application

Create a React application in the root folder and initialize the default React Scheduler.

Refer to the getting started with React Scheduler documentation for information on including the React Scheduler component in the React application.

The following code includes a basic Scheduler in the React application.

```
<ScheduleComponent width='100%' height='650px'>
   <Inject services={[Day, Week, WorkWeek, Month, Agenda]}/>
</ScheduleComponent>
```

Initialize Data Manager

Import the Syncfusion Data Manager in the **@syncfusion/ej2-data** Node module and initialize the DataManager with API service URLs, as shown in the following code.

```
import { DataManager,  UrlAdaptor } from '@syncfusion/ej2-data';
```

```
const dataManager = new DataManager({
    url: 'http://localhost:8080/api/scheduleevents/getData',
    crudUrl: 'http://localhost:8080/api/scheduleevents/crudActions',
    adaptor: new UrlAdaptor(),
    crossDomain: true
});
```

The properties are:

- **URL**: Refers to the remote API service URL, which will trigger the Scheduler’s initial loading and date/view navigation actions to get the appointment date.
- **crudUrl**: Refers to the create, update, and delete actions of the Scheduler. In URL parameters, you can get the added, updated, and deleted events’ data. By using that, you can save data in the database.
- **crossDomain**: Refers to enabling cross-domain requests of the application and remote service.

In the **src/App.js** file, remove the existing code and initialize the Scheduler with the Data Manager as shown in the following sample.

```
import './App.css';
import * as React from 'react';
import { ScheduleComponent, Day, Week, WorkWeek, Month, Agenda, Inject } from '@syncfusion/ej2-react-schedule';
import { DataManager,  UrlAdaptor } from '@syncfusion/ej2-data';

import "../node_modules/@syncfusion/ej2-base/styles/material.css";
import "../node_modules/@syncfusion/ej2-buttons/styles/material.css";
import "../node_modules/@syncfusion/ej2-calendars/styles/material.css";
import "../node_modules/@syncfusion/ej2-dropdowns/styles/material.css";
import "../node_modules/@syncfusion/ej2-inputs/styles/material.css";
import "../node_modules/@syncfusion/ej2-navigations/styles/material.css";
import "../node_modules/@syncfusion/ej2-popups/styles/material.css";
import "../node_modules/@syncfusion/ej2-schedule/styles/material.css";

function App() {
  const dataManager = new DataManager({
    url: 'http://localhost:8080/api/scheduleevents/getData',
    crudUrl: 'http://localhost:8080/api/scheduleevents/crudActions',
    adaptor: new UrlAdaptor(),
    crossDomain: true
  });

  return (
    <div className="App">
      <ScheduleComponent width='100%' height='650px' currentView='Month' eventSettings={{ dataSource: dataManager,
       fields: {
         id: 'id',
         subject: { name: 'subject' },
         isAllDay: { name: 'isallday' },
         location: { name: 'location' },
         description: { name: 'description' },
         startTime: { name: 'starttime' },
         endTime: { name: 'endtime' },
         startTimezone: { name: 'starttimezone' },
         endTimezone: { name: 'endtimezone' },
         recurrenceID: {name:'recurrenceid'},
         recurrenceRule:{name:'recurrencerule'},
         recurrenceException: {name:'recurrenceexception'},
         followingID:{name:'followingid'}
        } }}>
        <Inject services={[Day, Week, WorkWeek, Month, Agenda]}/>
      </ScheduleComponent>
    </div>
  );
}

export default App;
```

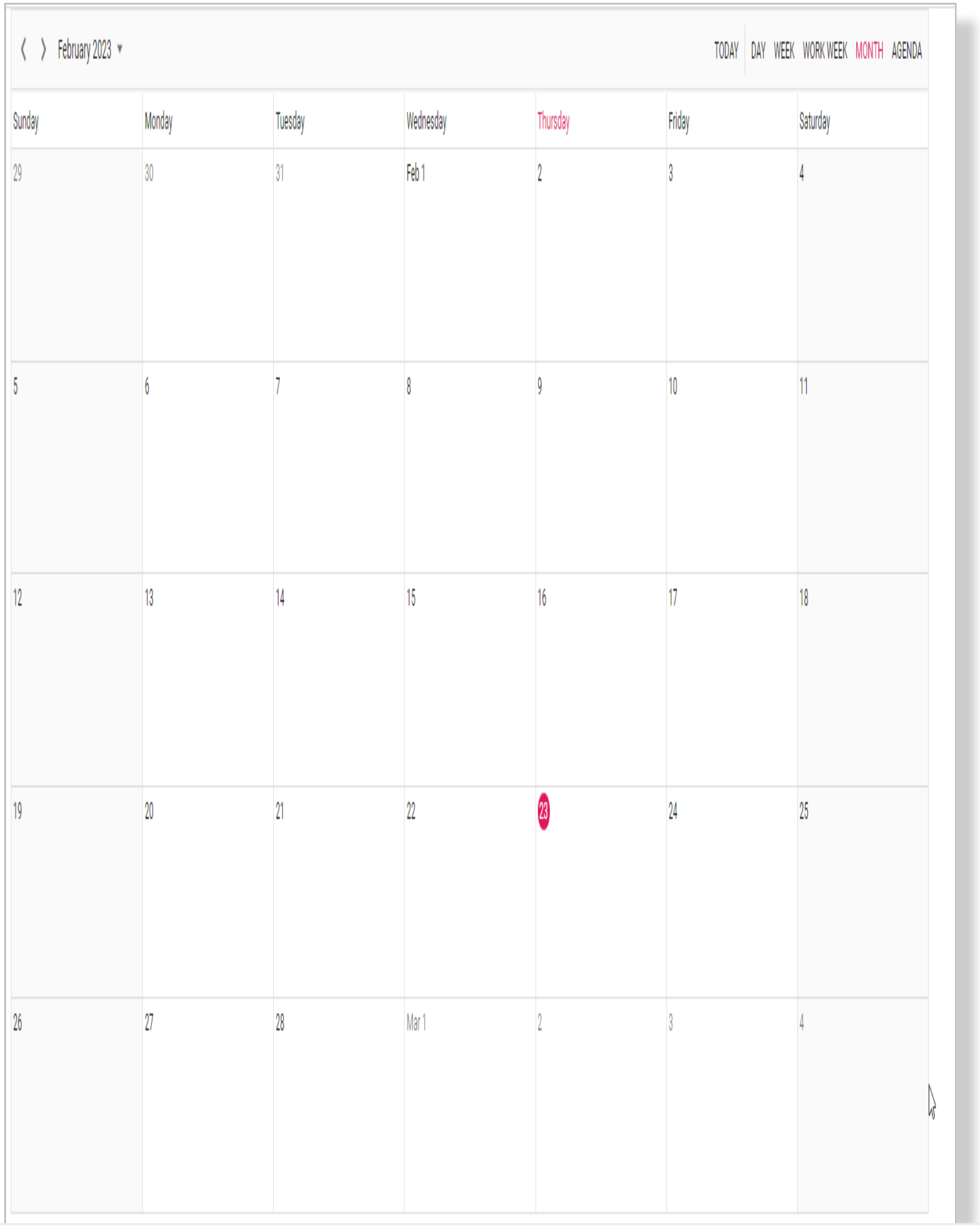
If the Scheduler’s dataSource holds the events collection with different field names, it is necessary to map them with their equivalent field name within the **eventSettings** property.

Map the database field names to Scheduler field names as shown below.

```
fields: {
    id: 'id',
    subject: { name: 'subject' },
    isAllDay: { name: 'isallday' },
        location: { name: 'location' },
        description: { name: 'description' },
        startTime: { name: 'starttime' },
        endTime: { name: 'endtime' },
        startTimezone: { name: 'starttimezone' },
        endTimezone: { name: 'endtimezone' },
        recurrenceID: {name:'recurrenceid'},
        recurrenceRule:{name:'recurrencerule'},
        recurrenceException: {name:'recurrenceexception'},
        followingID:{name:'followingid'}
}
```

It is time to run the application. Run the **npm start** command in the terminal to start the front-end React application. The application will be launched in the browser. The Scheduler will be displayed on the main page, and you will be able to perform CRUD operations in it.

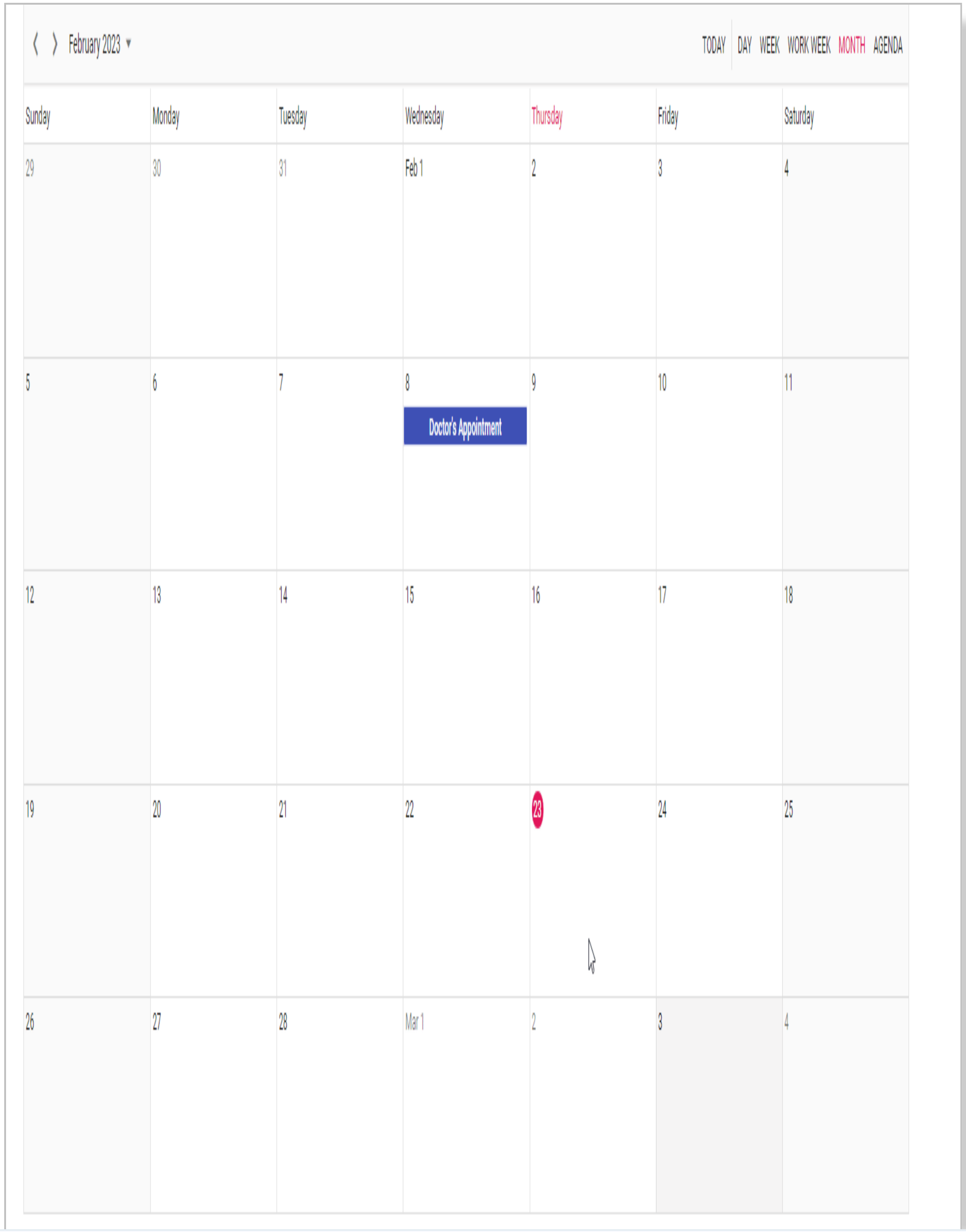
Create action: You can create events using either quick info or a more detailed editor window. Clicking on a cell will open a quick popup, prompting new event creation. Double-clicking on a cell will open the editor window. You can enter desired field values and then click the Save button to create an event.



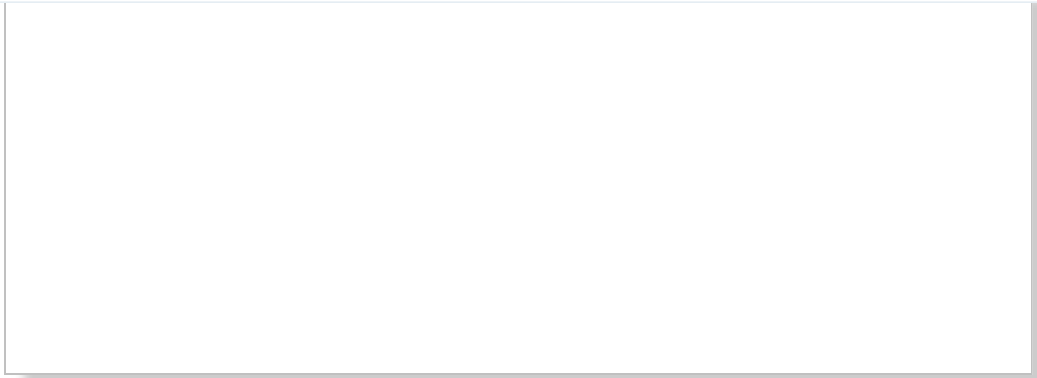
Sunday	Monday	Tuesday	Wednesday	Thursday	Friday	Saturday
29	30	31	Feb 1	2	3	4
5	6	7	8	9	10	11
12	13	14	15	16	17	18
19	20	21	22	23	24	25
26	27	28	Mar 1	2	3	4

Update action: You can open the default editor window filled with appointment details by double-clicking on an event. It is prefilled with event details such as subject, location, start and end times, all-day, time zone, description, and recurrence options. You can edit the desired field values and then click Save button to update them.

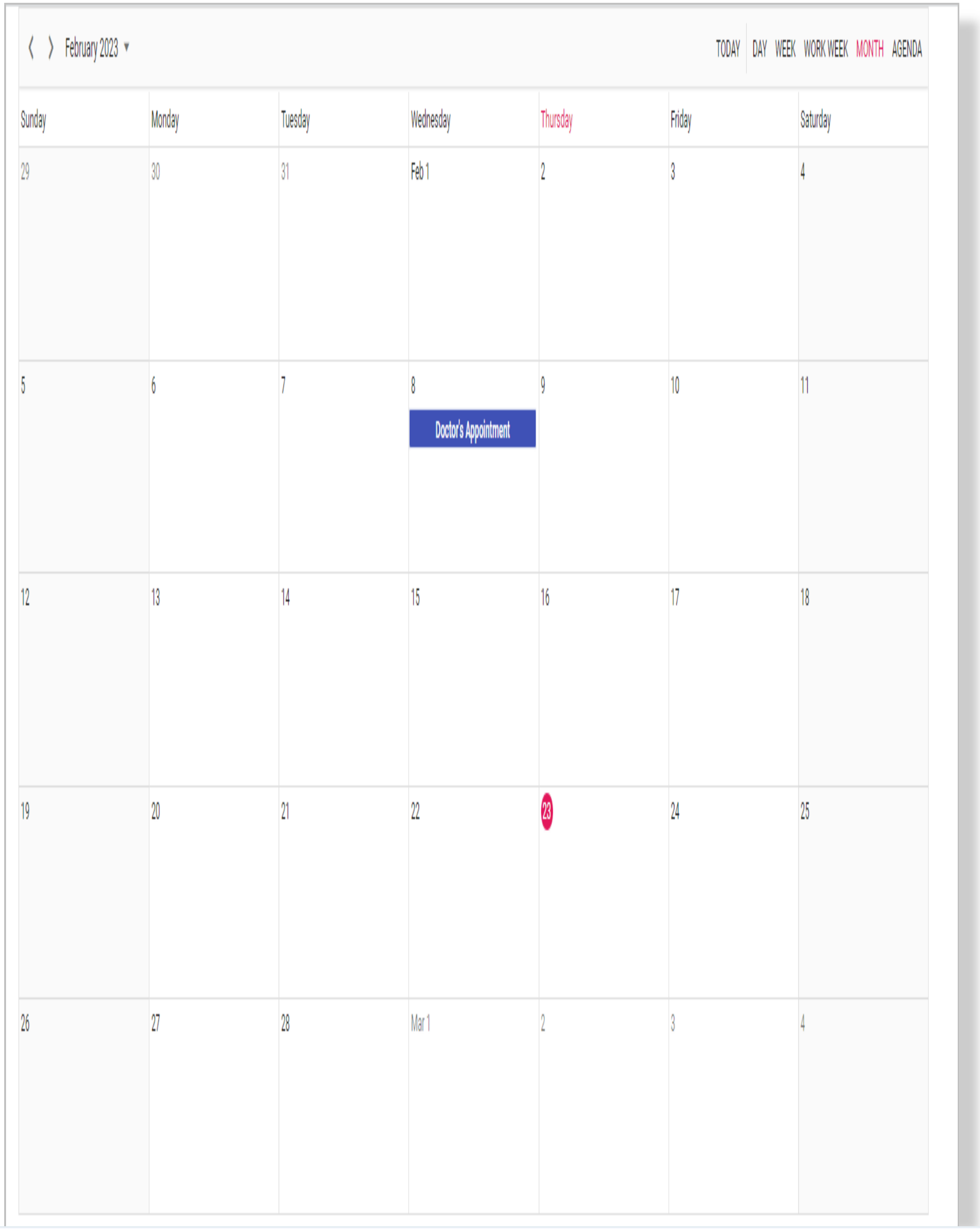
You can also perform dragging or resizing actions to reschedule events quickly.



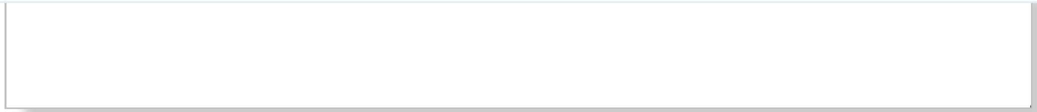
Sunday	Monday	Tuesday	Wednesday	Thursday	Friday	Saturday
29	30	31	Feb 1	2	3	4
5	6	7	8 Doctor's Appointment	9	10	11
12	13	14	15	16	17	18
19	20	21	22	23	24	25
26	27	28	Mar 1	2	3	4



Delete action: When you double-click an event, the default editor window will open. This window includes a Delete button at the bottom left to allow you to delete that appointment. When deleting an appointment through this editor window, the delete alert confirmation will not be displayed, and the event will be deleted immediately. Also, you can select an appointment and press the Delete key to delete the appointment.



Sunday	Monday	Tuesday	Wednesday	Thursday	Friday	Saturday
29	30	31	Feb 1	2	3	4
5	6	7	8 Doctor's Appointment	9	10	11
12	13	14	15	16	17	18
19	20	21	22	23	24	25
26	27	28	Mar 1	2	3	4



Check out the documentation to learn more about performing CRUD actions in the React Scheduler.

GitHub reference

You can check out the complete working example of this React Scheduler CRUD application on GitHub.

Summary

This blog explained how to create a React application with the Scheduler component and perform CRUD operations with Node.js and a PostgreSQL database. I hope you found this useful.

Follow the steps in this blog on your own and share your feedback in the comments section below.

You can also contact us through our support forums, support portal, or feedback portal. We are always happy to assist you!

Related blogs

- React Router vs. React Router DOM
- React Multicolumn MultiSelect Dropdown Component
- Create a Redux Form with Syncfusion React Components
- Restrict Editing of Word Documents Based on User in a Web Application

MEET THE AUTHOR

Ram Raju Elaiyaperumal

Ram Raju Elaiyaperumal is a software engineer at Syncfusion. He develops Syncfusion’s web components. With a passion for web technologies, his current focus centers around Angular, React, and Vue frameworks.

CONTACT US

Fax: +1 919.573.0306

US: +1 919.481.1974

UK: +44 20 7084 6215

Toll Free (USA):

1-888-9DOTNET

sales@syncfusion.com

Facebook-icon Twitter-icon Linkedin-icon Youtube-icon

39K+ 12K+ 15K+ 27K+

Pinterest-icon Instagram-icon Threads-icon

Privacy Policy | Cookie Policy | Terms of Use

Security Policy | Responsible Disclosure | Ethics Policy

altibox-imageviewer