Full Stack Web Application
Roster Management

Understanding the Problem

Roster Management

- Sporadic & variable staff work availabilities.
- Inconsistent shift lengths.
- Making rosters can be very time consuming, needing to be done every week.

Expense Management

- Team-members can have different pay rates
- With varying rosters, wage expenses fluctuate, hindering effective management of finances.

Team Management

- Each member's work needs vary
- Keeping track of who has what hours
- Keeping track of who is available
- Larger teams with multiple roles become tedious to allocate & manage.

Our Purpose

Our goal is to create something that will streamline a common and time-consuming ritual found in many industries and sectors. By distinguishing and addressing the underpinning difficulties of this process, we hope to give the people in these various worlds confidence and closure of being on top of this responsibility, and give them back some of the scarce time that they would otherwise spend each day fretting over.

Providing Solutions

Shift Management

- Streamlined shift tailoring and allocation.
- Consolidate shifts for a given time period to provide clear overviews.

Expense Management

- Apply appropriate pay rates to individual employees
- Access projected wage expenses for a given time period.
- Better predict business expenses & manage finances

Team Management

- Create and Access team member details
- Maintain effective staff workload distribution through integrated shift viewing.
- Allocate & Organize team structure for each day more effectively

Target Audience & Industries

Hospitality

- Pubs
- Cafes
- RSLs
- Restaurants
- Cocktail Bars

Emergency Services

- EMTs
- Ambulances
- Fire & Rescue
- Police

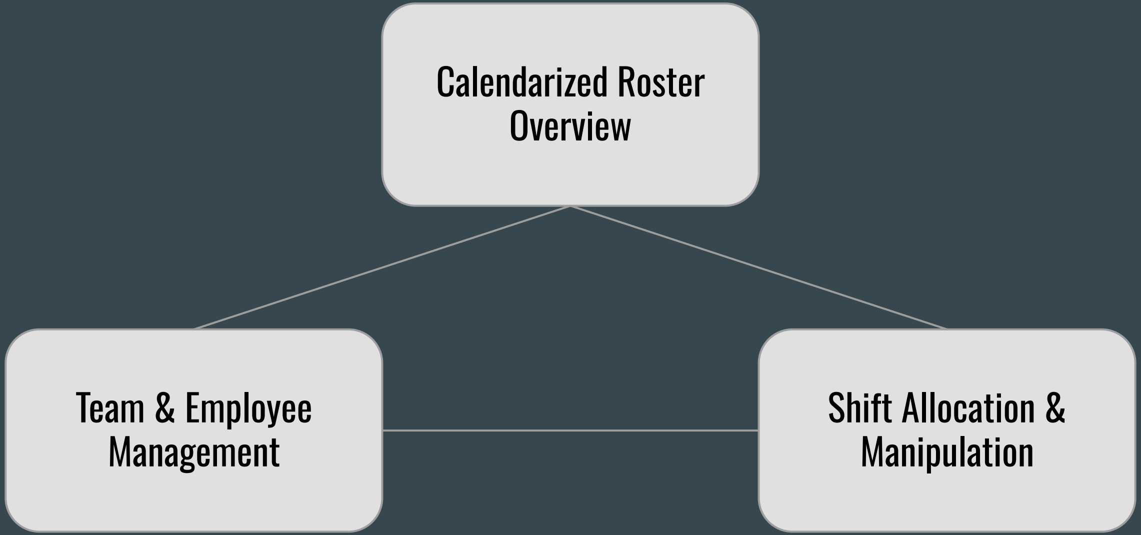
Healthcare

- Nursing
- GP Clinics
- Hospitals
- Emergency Rooms
- Volunteer Services

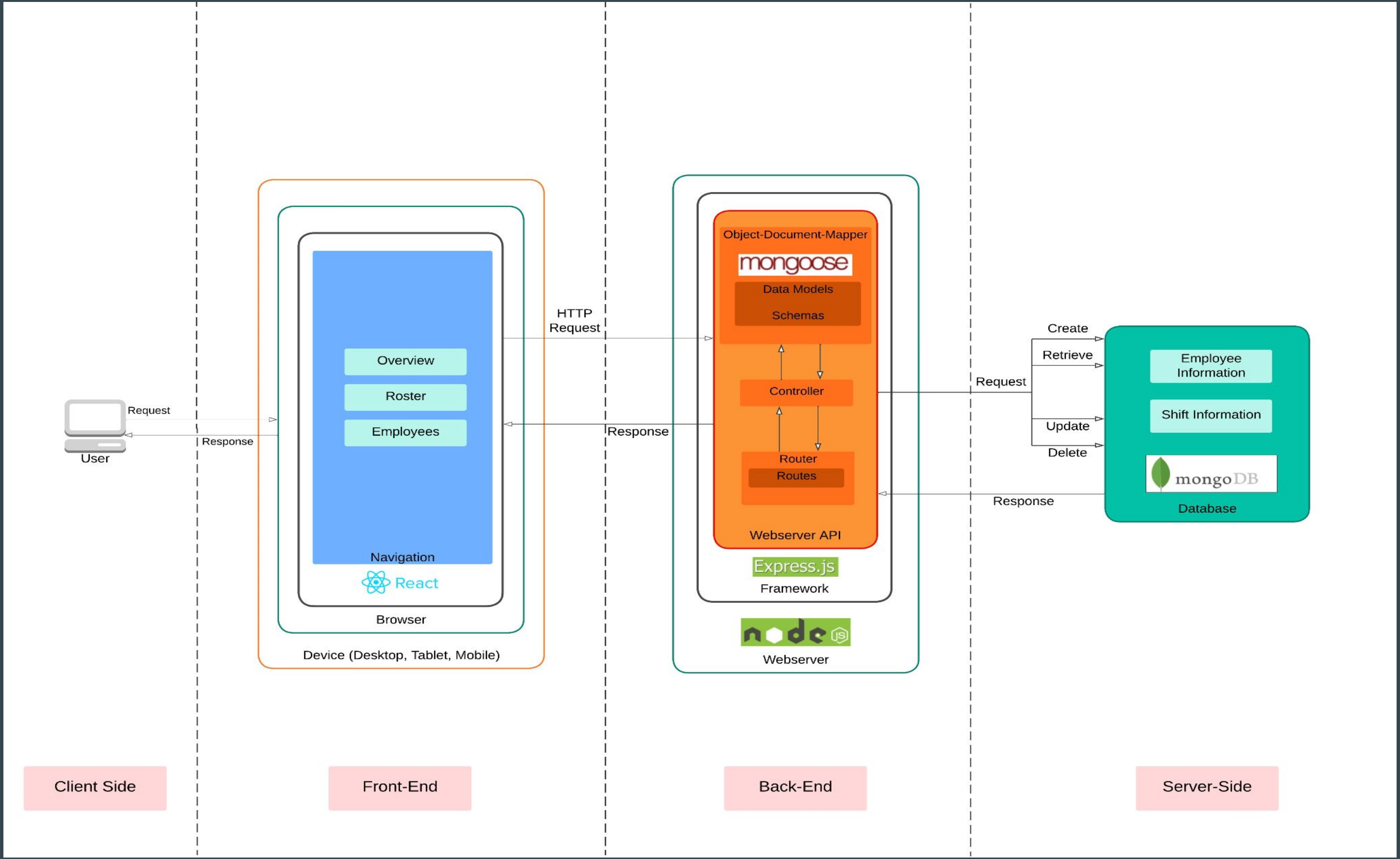
Education & Childcare

- Universities
- TAFE
- Schools
- Daycare
- Preschool

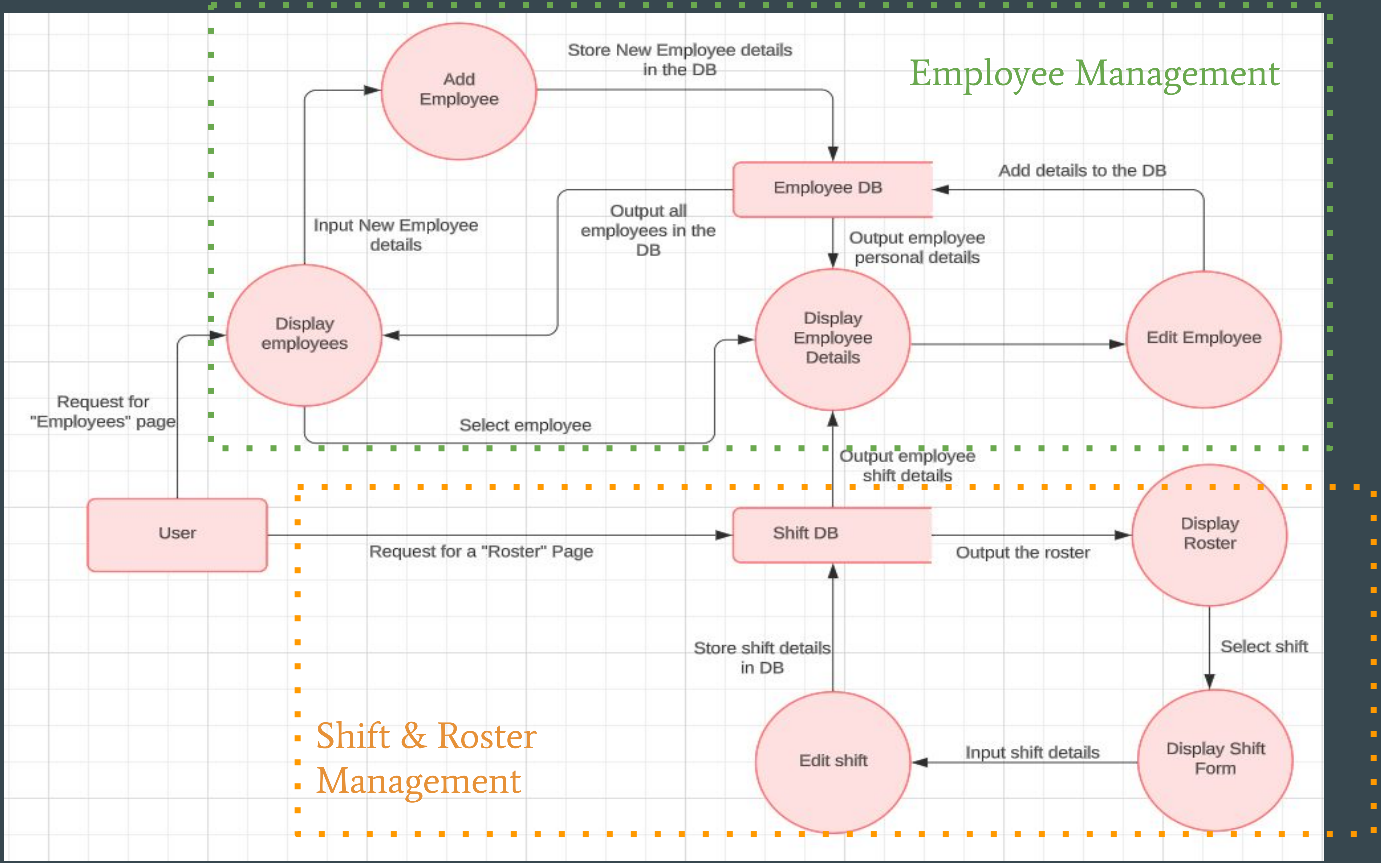
Core Application Features


Calendarized Roster Overview
Team & Employee Management
Shift Allocation & Manipulation

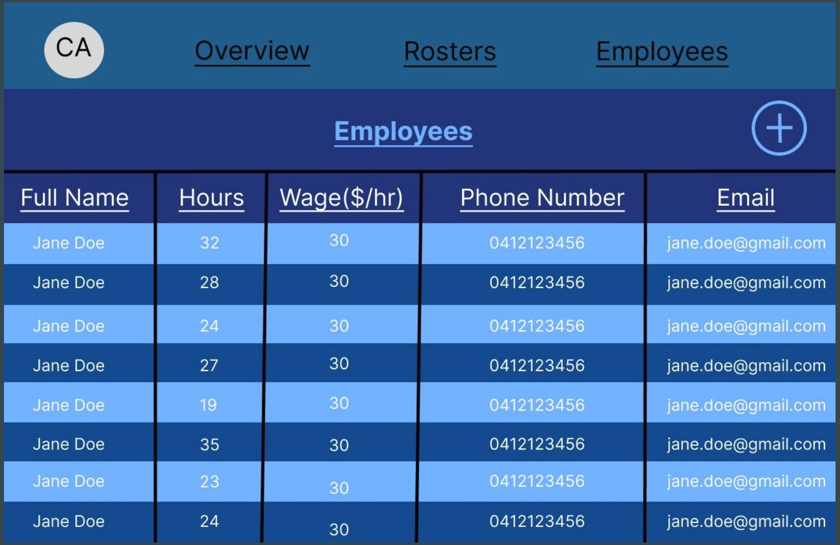
Application Architecture & Tech Stack



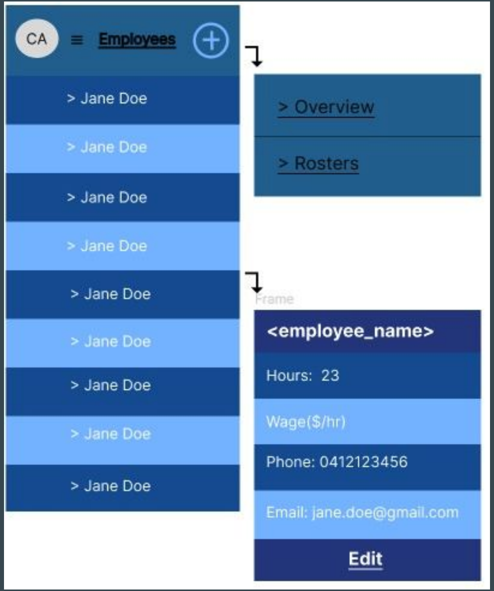
Data Flow


Employee Management
Add Employee
Store New Employee details in the DB
Employee DB
Add details to the DB
Input New Employee details
Output all employees in the DB
Output employee personal details
Display employees
Display Employee Details
Edit Employee
Request for "Employees" page
Select employee
Output employee shift details
User
Shift DB
Display Roster
Request for a "Roster" Page
Output the roster
Store shift details in DB
Select shift
Shift & Roster Management
Edit shift
Input shift details
Display Shift Form

Team Management



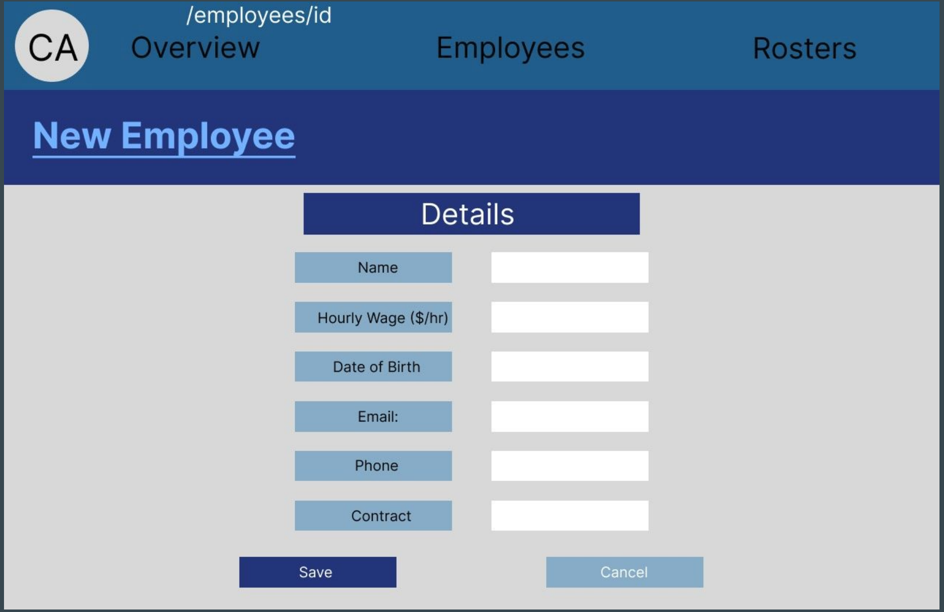
Full Name	Hours	Wage($/hr)	Phone Number	Email
Jane Doe	32	30	0412123456	jane.doe@gmail.com
Jane Doe	28	30	0412123456	jane.doe@gmail.com
Jane Doe	24	30	0412123456	jane.doe@gmail.com
Jane Doe	27	30	0412123456	jane.doe@gmail.com
Jane Doe	19	30	0412123456	jane.doe@gmail.com
Jane Doe	35	30	0412123456	jane.doe@gmail.com
Jane Doe	23	30	0412123456	jane.doe@gmail.com
Jane Doe	24	30	0412123456	jane.doe@gmail.com



<employee_name>
Hours: 23
Wage($/hr)
Phone: 0412123456
Email: jane.doe@gmail.com

- Access team overview
- View weekly work allocation for employees across the board
- Manage workload distribution
- Staff work expectations more effectively

Employee Management

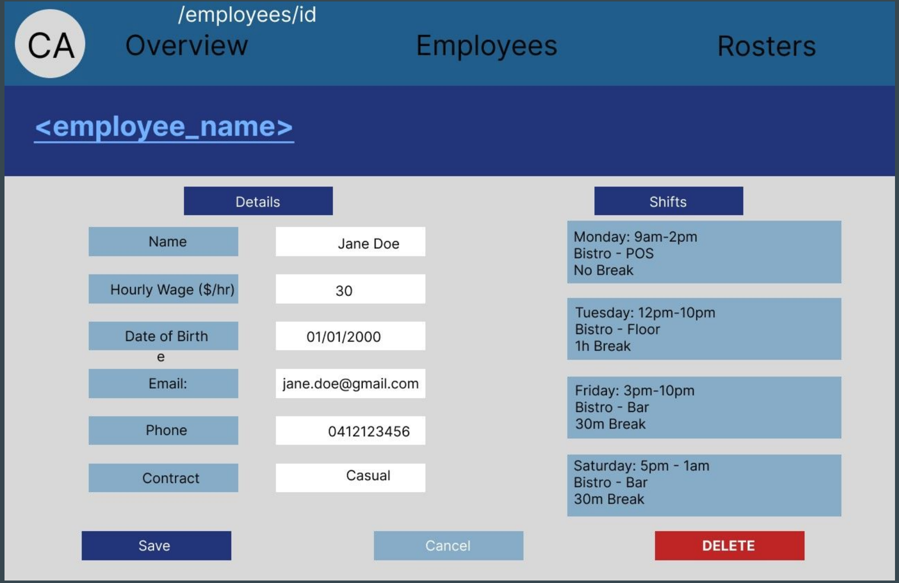


New Employee

Details

Name	
Hourly Wage ($/hr)	
Date of Birth	
Email:	
Phone	
Contract	

- Manage team composition by adding & removing members.
- Specify individual team member key details
- View & access upcoming shifts for the selected employee



<employee_name>

Details

Name	Jane Doe
Hourly Wage ($/hr)	30
Date of Birth e	01/01/2000
Email:	jane.doe@gmail.com
Phone	0412123456
Contract	Casual

Shifts

Monday: 9am-2pm
Bistro - POS
No Break

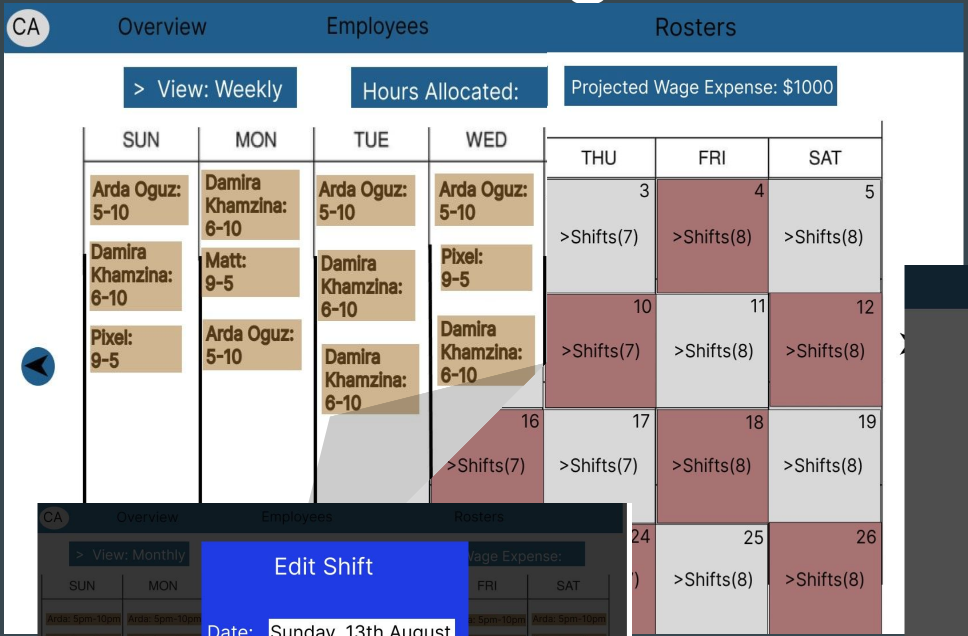
Tuesday: 12pm-10pm
Bistro - Floor
1h Break

Friday: 3pm-10pm
Bistro - Bar
30m Break

Saturday: 5pm - 1am
Bistro - Bar
30m Break

Roster Management

- Select between Weekly & Monthly views to return different overviews
- View specific days to better visualize team composition
- Edit shift details & allocation.


CA Overview Employees Rosters

> View: Weekly

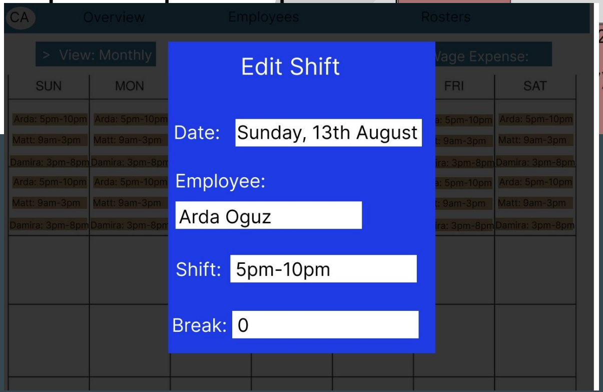
Hours Allocated:

Projected Wage Expense: $1000

SUN	MON	TUE	WED
Arda Oguz: 5-10	Damira Khamzina: 6-10	Arda Oguz: 5-10	Arda Oguz: 5-10
Damira Khamzina: 6-10	Matt: 9-5	Damira Khamzina: 6-10	Pixel: 9-5
Pixel: 9-5	Arda Oguz: 5-10	Damira Khamzina: 6-10	Damira Khamzina: 6-10

THU	FRI	SAT
3 >Shifts(7)	4 >Shifts(8)	5 >Shifts(8)
10 >Shifts(7)	11 >Shifts(8)	12 >Shifts(8)
17 >Shifts(7)	18 >Shifts(8)	19 >Shifts(8)
24	25 >Shifts(8)	26 >Shifts(8)

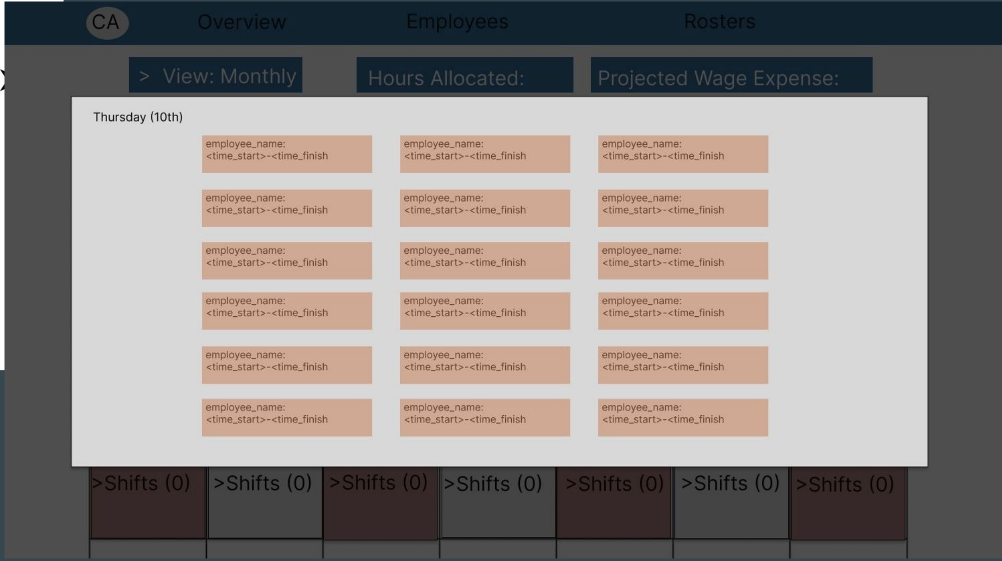
16 >Shifts(7)


CA Overview Employees Rosters

> View: Monthly

Edit Shift

Date:	Sunday, 13th August
Employee:	Arda Oguz
Shift:	5pm-10pm
Break:	0


CA Overview Employees Rosters

> View: Monthly

Hours Allocated:

Projected Wage Expense:

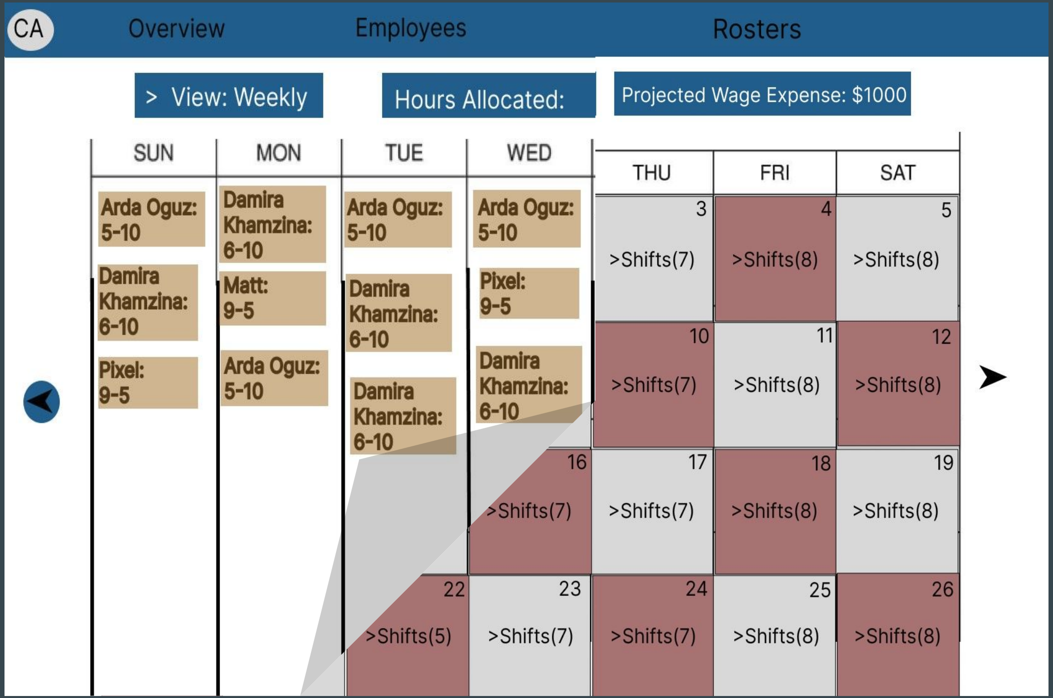
Thursday (10th)

employee_name: <time_start>-<time_finish	employee_name: <time_start>-<time_finish	employee_name: <time_start>-<time_finish
employee_name: <time_start>-<time_finish	employee_name: <time_start>-<time_finish	employee_name: <time_start>-<time_finish
employee_name: <time_start>-<time_finish	employee_name: <time_start>-<time_finish	employee_name: <time_start>-<time_finish
employee_name: <time_start>-<time_finish	employee_name: <time_start>-<time_finish	employee_name: <time_start>-<time_finish
employee_name: <time_start>-<time_finish	employee_name: <time_start>-<time_finish	employee_name: <time_start>-<time_finish
employee_name: <time_start>-<time_finish	employee_name: <time_start>-<time_finish	employee_name: <time_start>-<time_finish

>Shifts (0) >Shifts (0) >Shifts (0) >Shifts (0) >Shifts (0) >Shifts (0) >Shifts (0)

Product Development

Rostering Pre-Development

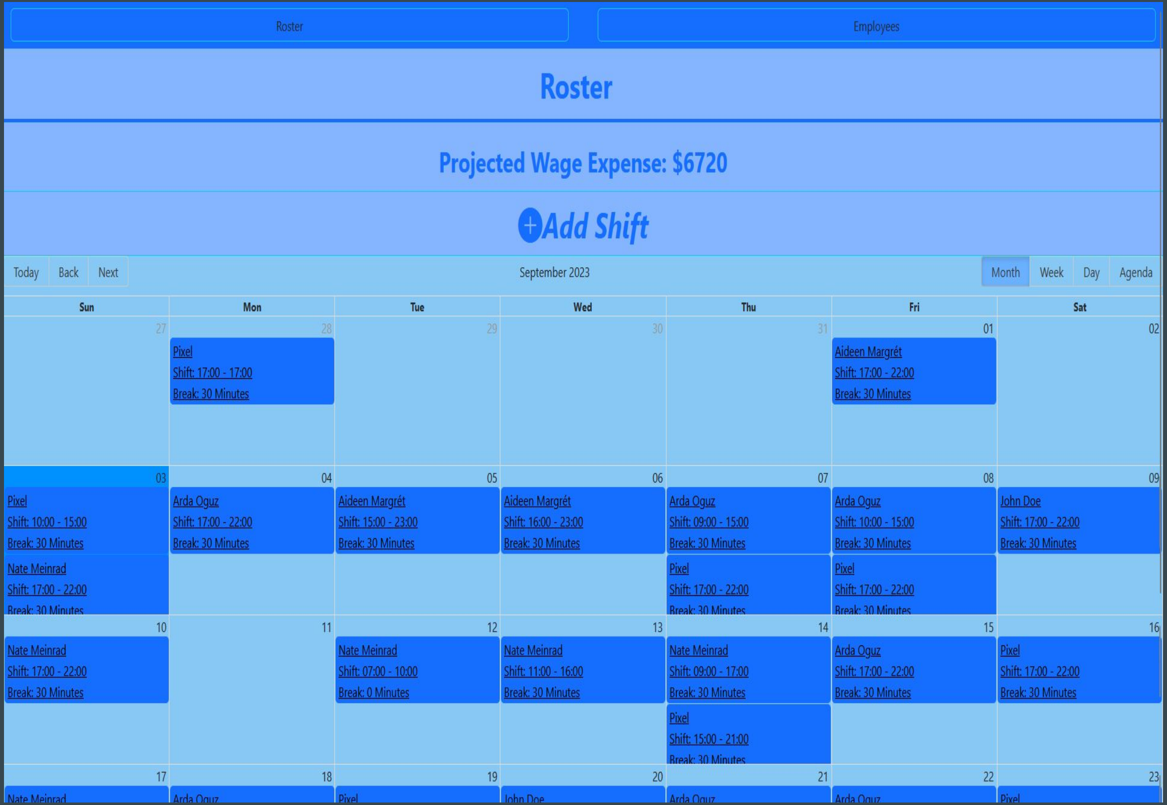


Projected Wage Expense: $1000

SUN	MON	TUE	WED
Arda Oguz: 5-10	Damira Khamzina: 6-10	Arda Oguz: 5-10	Arda Oguz: 5-10
Damira Khamzina: 6-10	Matt: 9-5	Damira Khamzina: 6-10	Pixel: 9-5
Pixel: 9-5	Arda Oguz: 5-10	Damira Khamzina: 6-10	Damira Khamzina: 6-10

TUE	WED	THU	FRI	SAT
		3 >Shifts(7)	4 >Shifts(8)	5 >Shifts(8)
		10 >Shifts(7)	11 >Shifts(8)	12 >Shifts(8)
	16 >Shifts(7)	17 >Shifts(7)	18 >Shifts(8)	19 >Shifts(8)
22 >Shifts(5)	23 >Shifts(7)	24 >Shifts(7)	25 >Shifts(8)	26 >Shifts(8)

Rostering Post-Development



Roster | Employees

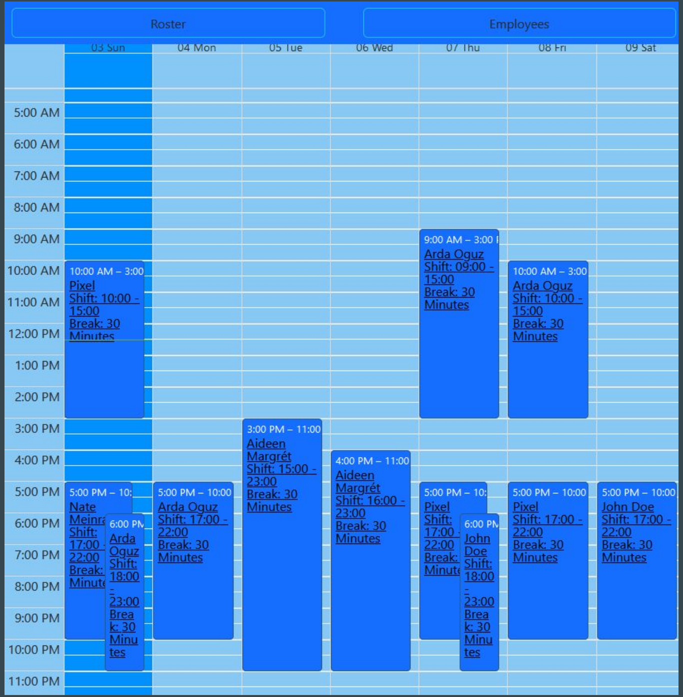
Roster

Projected Wage Expense: $6720

+Add Shift

Today | Back | Next — September 2023 — Month | Week | Day | Agenda

Sun	Mon	Tue	Wed	Thu	Fri	Sat
27	28 Pixel Shift: 17:00 - 17:00 Break: 30 Minutes	29	30	31	01 Aideen Margrét Shift: 17:00 - 22:00 Break: 30 Minutes	02
03 Pixel Shift: 10:00 - 15:00 Break: 30 Minutes Nate Meinrad Shift: 17:00 - 22:00 Break: 30 Minutes	04 Arda Oguz Shift: 17:00 - 22:00 Break: 30 Minutes	05 Aideen Margrét Shift: 15:00 - 23:00 Break: 30 Minutes	06 Aideen Margrét Shift: 16:00 - 23:00 Break: 30 Minutes	07 Arda Oguz Shift: 09:00 - 15:00 Break: 30 Minutes Pixel Shift: 17:00 - 22:00 Break: 30 Minutes	08 Arda Oguz Shift: 10:00 - 15:00 Break: 30 Minutes Pixel Shift: 17:00 - 22:00 Break: 30 Minutes	09 John Doe Shift: 17:00 - 22:00 Break: 30 Minutes
10 Nate Meinrad Shift: 17:00 - 22:00 Break: 30 Minutes	11	12 Nate Meinrad Shift: 07:00 - 10:00 Break: 0 Minutes	13 Nate Meinrad Shift: 11:00 - 16:00 Break: 30 Minutes	14 Nate Meinrad Shift: 09:00 - 17:00 Break: 30 Minutes Pixel Shift: 15:00 - 21:00 Break: 30 Minutes	15 Arda Oguz Shift: 17:00 - 22:00 Break: 30 Minutes	16 Pixel Shift: 17:00 - 22:00 Break: 30 Minutes
17 Nate Meinrad	18 Arda Oguz	19 Pixel	20 John Doe	21 Arda Oguz	22 Arda Oguz	23 Pixel

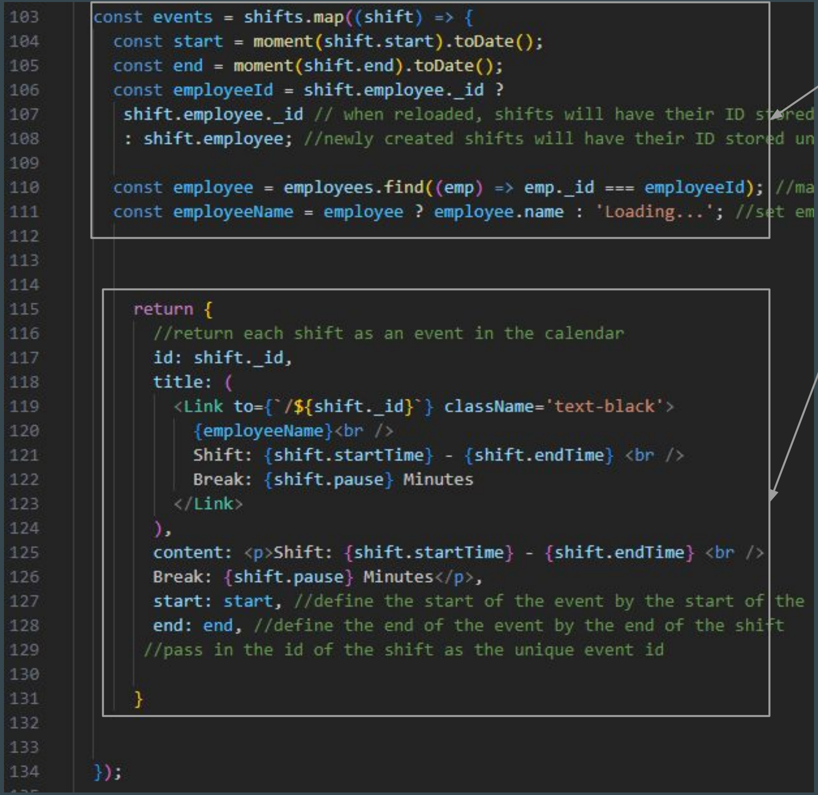


Roster | Employees

	03 Sun	04 Mon	05 Tue	06 Wed	07 Thu	08 Fri	09 Sat
	10:00 AM – 3:00 Pixel Shift: 10:00 - 15:00 Break: 30 Minutes 5:00 PM – 10: Nate Meinra Shift: 17:00 - 22:00 Break: Minute 6:00 PM Arda Oguz Shift: 18:00 - 23:00 Break: 30 Minutes	5:00 PM – 10:00 Arda Oguz Shift: 17:00 - 22:00 Break: 30 Minutes	3:00 PM – 11:00 Aideen Margrét Shift: 15:00 - 23:00 Break: 30 Minutes	4:00 PM – 11:00 Aideen Margrét Shift: 16:00 - 23:00 Break: 30 Minutes	9:00 AM – 3:00 Arda Oguz Shift: 09:00 - 15:00 Break: 30 Minutes 5:00 PM – 10: Pixel Shift: 17:00 - 22:00 Break: Minute 6:00 PM John Doe Shift: 18:00 - 23:00 Break: 30 Minutes	10:00 AM – 3:00 Arda Oguz Shift: 10:00 - 15:00 Break: 30 Minutes 5:00 PM – 10:00 Pixel Shift: 17:00 - 22:00 Break: 30 Minutes	5:00 PM – 10:00 John Doe Shift: 17:00 - 22:00 Break: 30 Minutes

5:00 AM, 6:00 AM, 7:00 AM, 8:00 AM, 9:00 AM, 10:00 AM, 11:00 AM, 12:00 PM, 1:00 PM, 2:00 PM, 3:00 PM, 4:00 PM, 5:00 PM, 6:00 PM, 7:00 PM, 8:00 PM, 9:00 PM, 10:00 PM, 11:00 PM

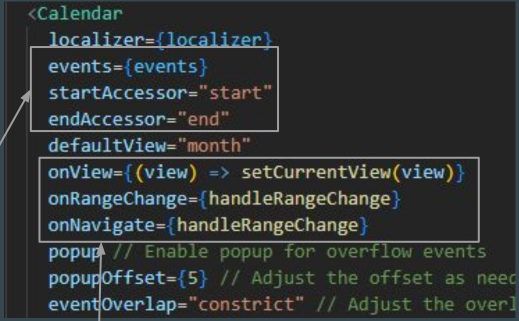
Roster: Calendar



```
const events = shifts.map((shift) => {
  const start = moment(shift.start).toDate();
  const end = moment(shift.end).toDate();
  const employeeId = shift.employee._id ?
   shift.employee._id // when reloaded, shifts will have their ID stored
   : shift.employee; //newly created shifts will have their ID stored un

  const employee = employees.find((emp) => emp._id === employeeId); //ma
  const employeeName = employee ? employee.name : 'Loading...'; //set em



    return {
     //return each shift as an event in the calendar
     id: shift._id,
     title: (
       <Link to={`/${shift._id}`} className='text-black'>
         {employeeName}<br />
         Shift: {shift.startTime} - {shift.endTime} <br />
         Break: {shift.pause} Minutes
       </Link>
     ),
     content: <p>Shift: {shift.startTime} - {shift.endTime} <br />
     Break: {shift.pause} Minutes</p>,
     start: start, //define the start of the event by the start of the
     end: end, //define the end of the event by the end of the shift
    //pass in the id of the shift as the unique event id

    }


});
```

The Roster component is designed to iterate over each shift in the database, and create its own 'event' in the calendar

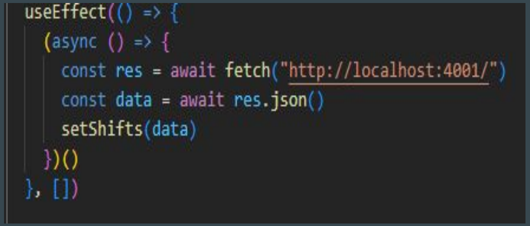
Events are then passed into the 'react-big-calendar', which then maps each shifts start and end time, and facilitates different calendar views



```
<Calendar
  localizer={localizer}
  events={events}
  startAccessor="start"
  endAccessor="end"
  defaultView="month"
  onView={(view) => setCurrentView(view)}
  onRangeChange={handleRangeChange}
  onNavigate={handleRangeChange}
  popup // Enable popup for overflow events
  popupOffset={5} // Adjust the offset as need
  eventOverlap="constrict" // Adjust the overl
```

calendar props are configured to trigger re-renders as the user changes views or navigates to other time periods. This also ensures that the wage projection is re-calculated when views change

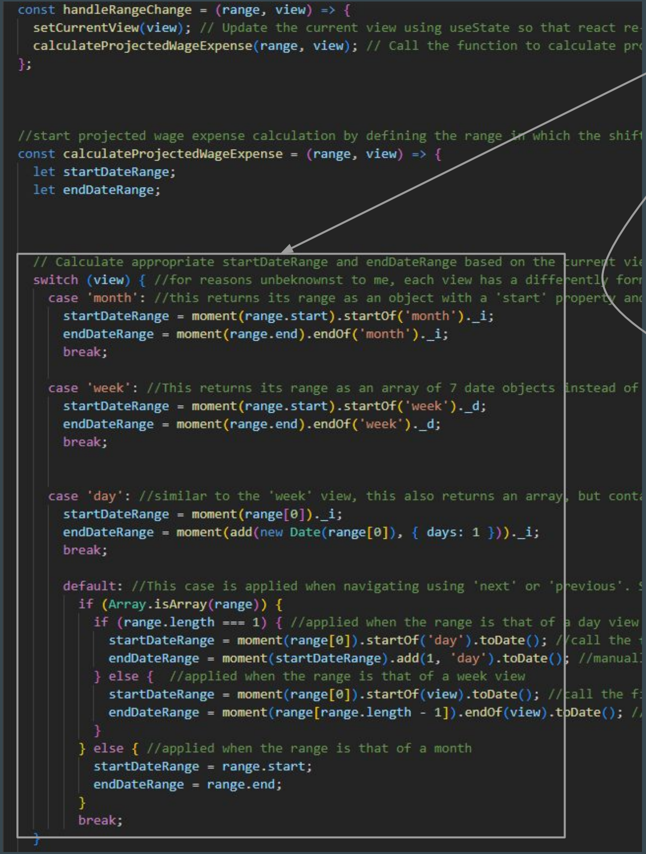
useEffect and useState hooks are utilized to fetch shift data upon during the initial render, and trigger re-renders by setting the state



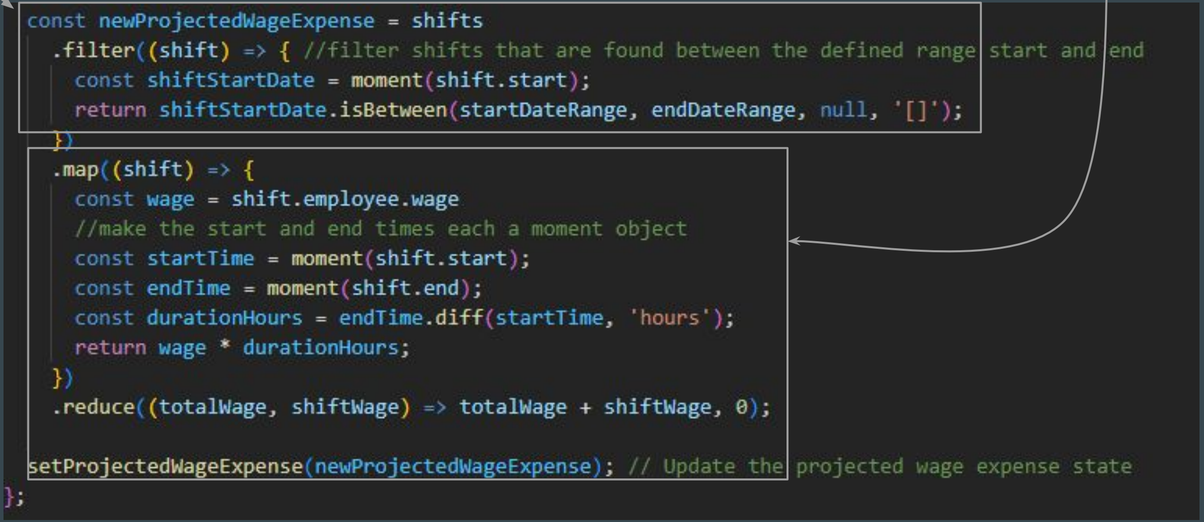
```
useEffect(() => {
  (async () => {
    const res = await fetch("http://localhost:4001/")
    const data = await res.json()
    setShifts(data)
  })()
}, [])
```

Roster: Wage Projection

- Wage Projection first involves the selected view, determining the time-range within which shifts are included in the calculation
- From there, shifts are filtered based on the time range.
- Each shift's approximate expense is calculated based on the duration of the shift and the wage of the employee assigned, and compiled to produce

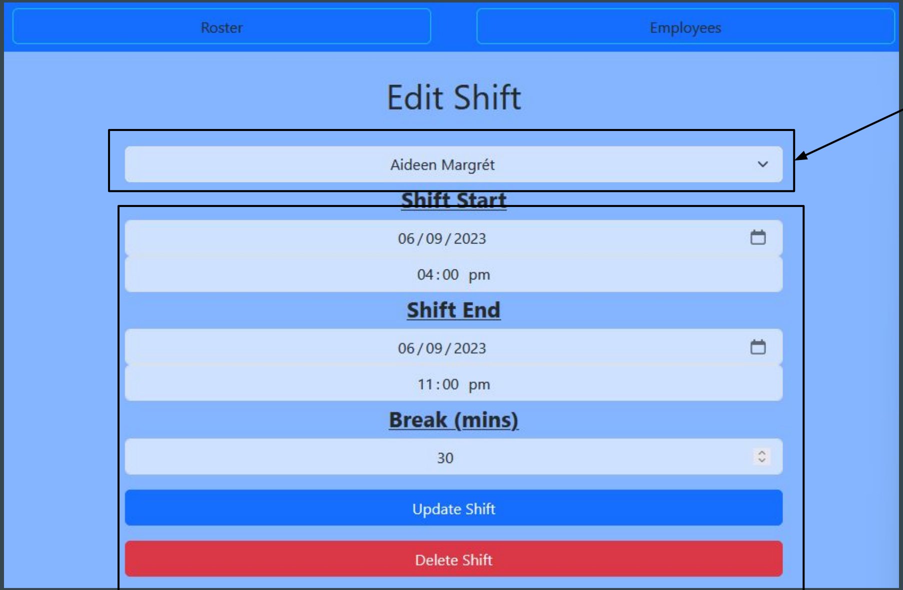


```javascript
const handleRangeChange = (range, view) => {
  setCurrentView(view); // Update the current view using useState so that react re
  calculateProjectedWageExpense(range, view); // Call the function to calculate pr
};



//start projected wage expense calculation by defining the range in which the shif
const calculateProjectedWageExpense = (range, view) => {
  let startDateRange;
  let endDateRange;



  // Calculate appropriate startDateRange and endDateRange based on the current vie
  switch (view) { //for reasons unbeknownst to me, each view has a differently for
    case 'month': //this returns its range as an object with a 'start' property an
      startDateRange = moment(range.start).startOf('month')._i;
      endDateRange = moment(range.end).endOf('month')._i;
      break;

    case 'week': //This returns its range as an array of 7 date objects instead of
      startDateRange = moment(range.start).startOf('week')._d;
      endDateRange = moment(range.end).endOf('week')._d;
      break;


    case 'day': //similar to the 'week' view, this also returns an array, but conta
      startDateRange = moment(range[0])._i;
      endDateRange = moment(add(new Date(range[0]), { days: 1 }))._i;
      break;

      default: //This case is applied when navigating using 'next' or 'previous'. S
        if (Array.isArray(range)) {
          if (range.length === 1) { //applied when the range is that of a day view
            startDateRange = moment(range[0]).startOf('day').toDate(); //call the f
            endDateRange = moment(startDateRange).add(1, 'day').toDate(); //manual
          } else {  //applied when the range is that of a week view
            startDateRange = moment(range[0]).startOf(view).toDate(); //call the fi
            endDateRange = moment(range[range.length - 1]).endOf(view).toDate(); //
          }
        } else { //applied when the range is that of a month
          startDateRange = range.start;
          endDateRange = range.end;
        }
        break;
  }
```



```javascript
const newProjectedWageExpense = shifts
  .filter((shift) => { //filter shifts that are found between the defined range start and end
    const shiftStartDate = moment(shift.start);
    return shiftStartDate.isBetween(startDateRange, endDateRange, null, '[]');
  })
  .map((shift) => {
    const wage = shift.employee.wage
    //make the start and end times each a moment object
    const startTime = moment(shift.start);
    const endTime = moment(shift.end);
    const durationHours = endTime.diff(startTime, 'hours');
    return wage * durationHours;
  })
  .reduce((totalWage, shiftWage) => totalWage + shiftWage, 0);

setProjectedWageExpense(newProjectedWageExpense); // Update the projected wage expense state
};
```

Roster: Shift Management



Roster

Employees

Edit Shift

Aideen Margrét

Shift Start

06 / 09 / 2023

04 : 00 pm

Shift End

06 / 09 / 2023

11 : 00 pm

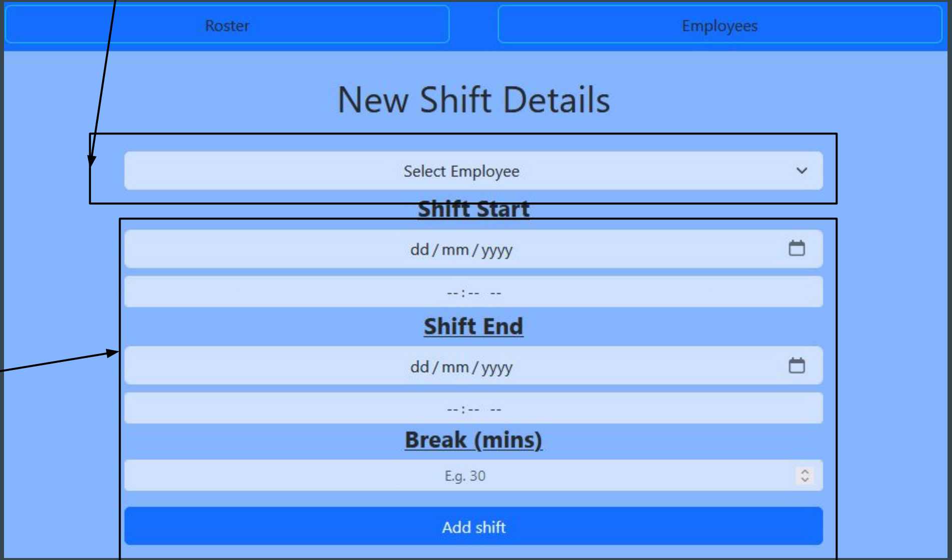
Break (mins)

30

Update Shift

Delete Shift

Employees can be assigned & reallocated to shifts, allowing for simple shift swapping



Roster

Employees

New Shift Details

Select Employee

Shift Start

dd / mm / yyyy

-- : -- --

Shift End

dd / mm / yyyy

-- : -- --

Break (mins)

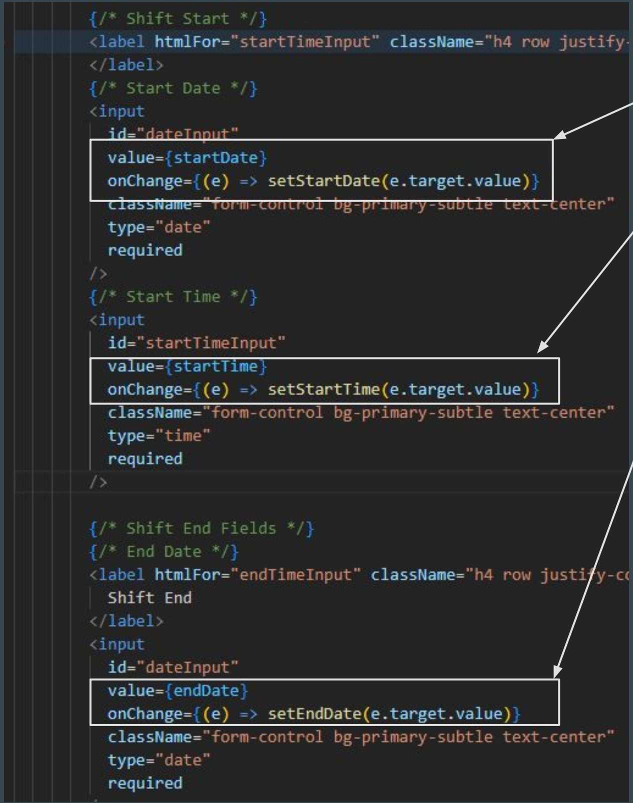
E.g. 30

Add shift

Shifts can be created and adjusted as necessary

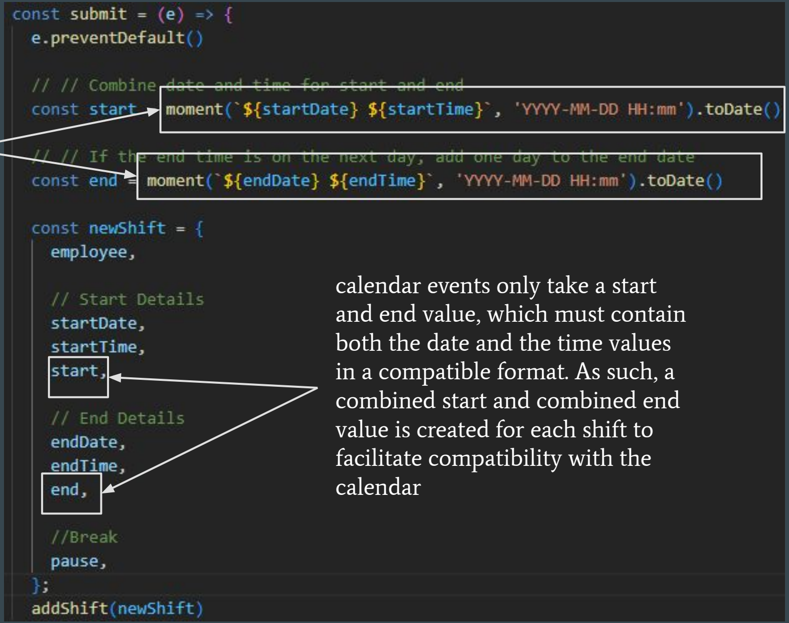
Roster: Shift Management

Each input sets the value of a new / updated shift's specified parameter.



```
{/* Shift Start */}
<label htmlFor="startTimeInput" className="h4 row justify-
</label>
{/* Start Date */}
<input
  id="dateInput"
  value={startDate}
  onChange={(e) => setStartDate(e.target.value)}
  className="form-control bg-primary-subtle text-center"
  type="date"
  required
/>
{/* Start Time */}
<input
  id="startTimeInput"
  value={startTime}
  onChange={(e) => setStartTime(e.target.value)}
  className="form-control bg-primary-subtle text-center"
  type="time"
  required
/>

{/* Shift End Fields */}
{/* End Date */}
<label htmlFor="endTimeInput" className="h4 row justify-c
  Shift End
</label>
<input
  id="dateInput"
  value={endDate}
  onChange={(e) => setEndDate(e.target.value)}
  className="form-control bg-primary-subtle text-center"
  type="date"
  required
```

When submitted, the shift's details are processed to produce data that is compatible with both the database and the calendar

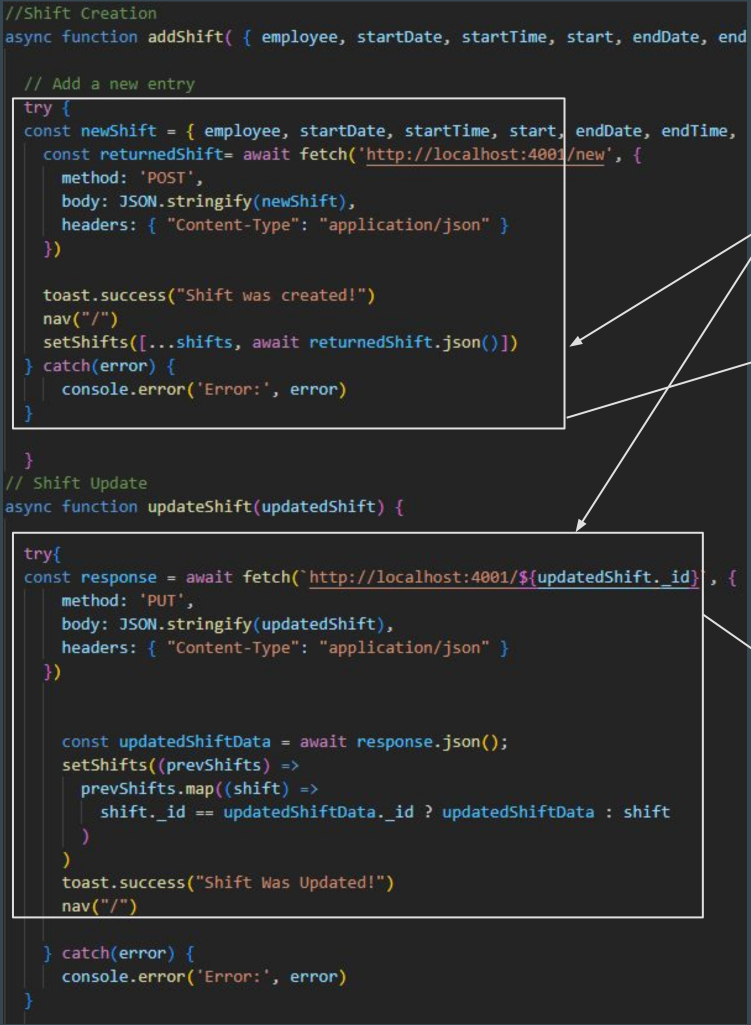


```
const submit = (e) => {
  e.preventDefault()

  // // Combine date and time for start and end
  const start = moment(`${startDate} ${startTime}`, 'YYYY-MM-DD HH:mm').toDate()

  // // If the end time is on the next day, add one day to the end date
  const end = moment(`${endDate} ${endTime}`, 'YYYY-MM-DD HH:mm').toDate()

  const newShift = {
    employee,

    // Start Details
    startDate,
    startTime,
    start,

    // End Details
    endDate,
    endTime,
    end,

    //Break
    pause,
  };
  addShift(newShift)
```

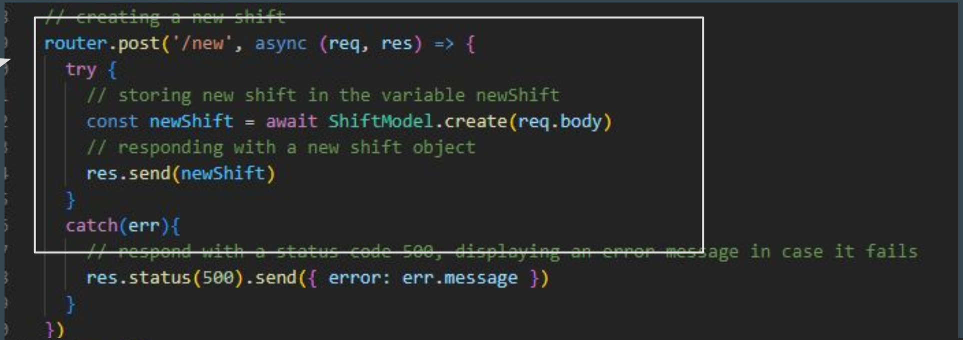
calendar events only take a start and end value, which must contain both the date and the time values in a compatible format. As such, a combined start and combined end value is created for each shift to facilitate compatibility with the calendar

Roster: Shift Management

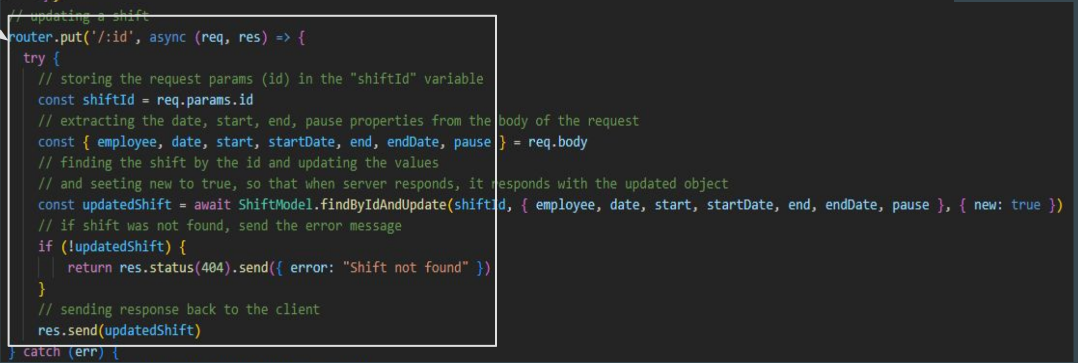
Shift data is then passed into their respective function found in the app component, where the appropriate fetch request is made with the data payload as its body. Once the request is completed, the user is notified and redirected back to the roster page



```
//Shift Creation
async function addShift( { employee, startDate, startTime, start, endDate, end

  // Add a new entry
  try {
  const newShift = { employee, startDate, startTime, start, endDate, endTime,
    const returnedShift= await fetch('http://localhost:4001/new', {
      method: 'POST',
      body: JSON.stringify(newShift),
      headers: { "Content-Type": "application/json" }
    })

    toast.success("Shift was created!")
    nav("/")
    setShifts([...shifts, await returnedShift.json()])
  } catch(error) {
      console.error('Error:', error)
  }

  }
// Shift Update
async function updateShift(updatedShift) {

  try{
  const response = await fetch(`http://localhost:4001/${updatedShift._id}`, {
      method: 'PUT',
      body: JSON.stringify(updatedShift),
      headers: { "Content-Type": "application/json" }
    })


      const updatedShiftData = await response.json();
      setShifts((prevShifts) =>
        prevShifts.map((shift) =>
          shift._id == updatedShiftData._id ? updatedShiftData : shift
        )
      )
      toast.success("Shift Was Updated!")
      nav("/")

    } catch(error) {
      console.error('Error:', error)
  }
```

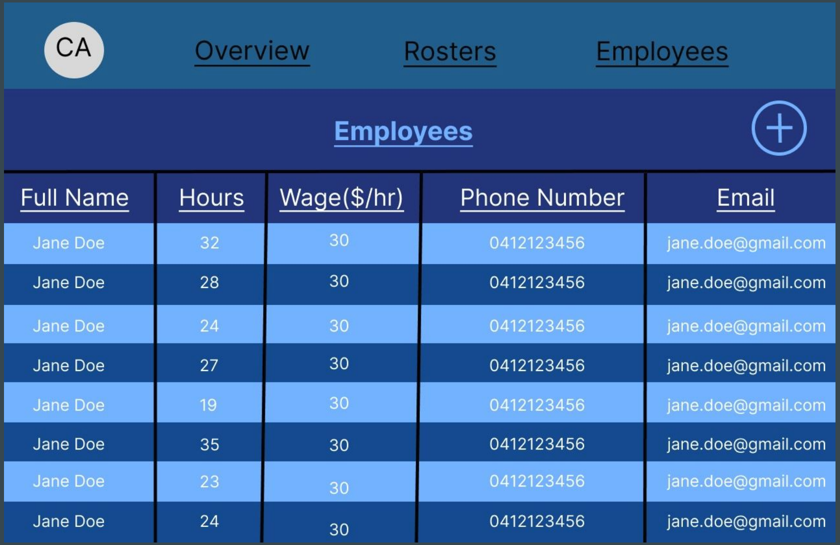


```
  // creating a new shift
  router.post('/new', async (req, res) => {
    try {
      // storing new shift in the variable newShift
      const newShift = await ShiftModel.create(req.body)
      // responding with a new shift object
      res.send(newShift)
    }
    catch(err){
      // respond with a status code 500, displaying an error message in case it fails
      res.status(500).send({ error: err.message })
    }
  })
```



```
// updating a shift
router.put('/:id', async (req, res) => {
  try {
    // storing the request params (id) in the "shiftId" variable
    const shiftId = req.params.id
    // extracting the date, start, end, pause properties from the body of the request
    const { employee, date, start, startDate, end, endDate, pause } = req.body
    // finding the shift by the id and updating the values
    // and seeting new to true, so that when server responds, it responds with the updated object
    const updatedShift = await ShiftModel.findByIdAndUpdate(shiftId, { employee, date, start, startDate, end, endDate, pause }, { new: true })
    // if shift was not found, send the error message
    if (!updatedShift) {
      return res.status(404).send({ error: "Shift not found" })
    }
    // sending response back to the client
    res.send(updatedShift)
} catch (err) {
```

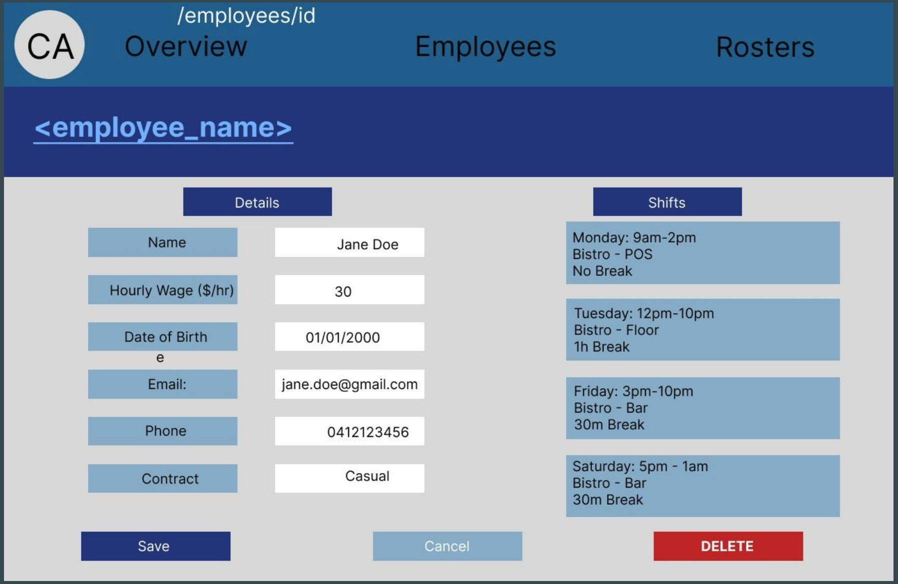
Team Management Pre-Development



CA Overview Rosters Employees

Employees

Full Name	Hours	Wage($/hr)	Phone Number	Email
Jane Doe	32	30	0412123456	jane.doe@gmail.com
Jane Doe	28	30	0412123456	jane.doe@gmail.com
Jane Doe	24	30	0412123456	jane.doe@gmail.com
Jane Doe	27	30	0412123456	jane.doe@gmail.com
Jane Doe	19	30	0412123456	jane.doe@gmail.com
Jane Doe	35	30	0412123456	jane.doe@gmail.com
Jane Doe	23	30	0412123456	jane.doe@gmail.com
Jane Doe	24	30	0412123456	jane.doe@gmail.com



/employees/id

CA Overview Employees Rosters

<employee_name>

Details

Name	Jane Doe
Hourly Wage ($/hr)	30
Date of Birth e	01/01/2000
Email:	jane.doe@gmail.com
Phone	0412123456
Contract	Casual

Shifts

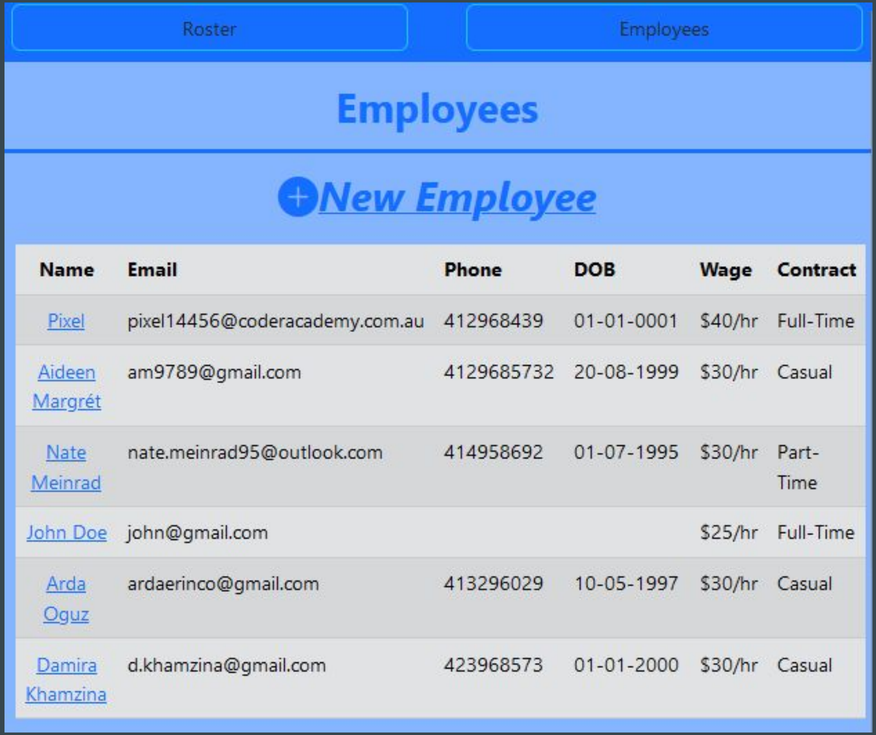
Monday: 9am-2pm
Bistro - POS
No Break

Tuesday: 12pm-10pm
Bistro - Floor
1h Break

Friday: 3pm-10pm
Bistro - Bar
30m Break

Saturday: 5pm - 1am
Bistro - Bar
30m Break

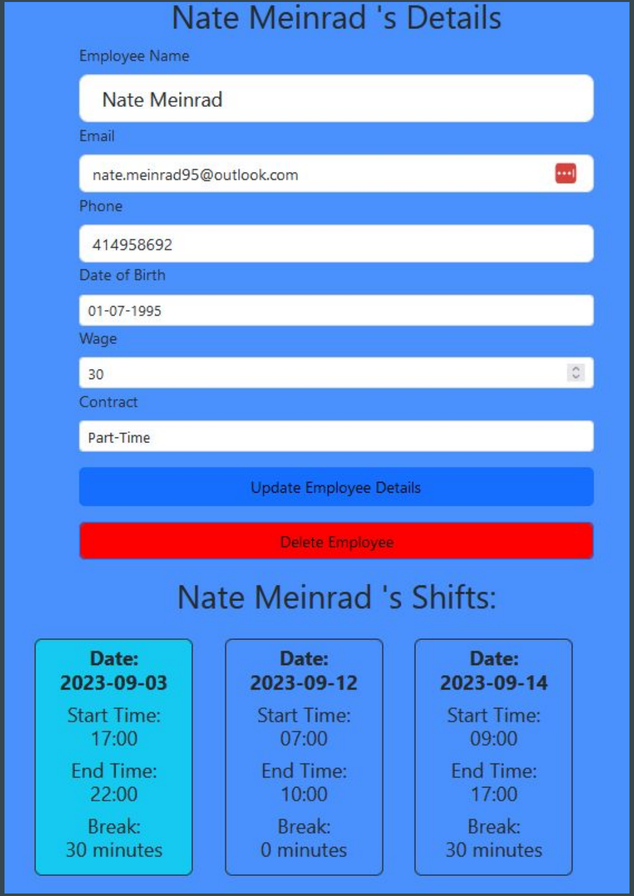
Team Management Post-Development



Employees

New Employee

Name	Email	Phone	DOB	Wage	Contract
Pixel	pixel14456@coderacademy.com.au	412968439	01-01-0001	$40/hr	Full-Time
Aideen Margrét	am9789@gmail.com	4129685732	20-08-1999	$30/hr	Casual
Nate Meinrad	nate.meinrad95@outlook.com	414958692	01-07-1995	$30/hr	Part-Time
John Doe	john@gmail.com			$25/hr	Full-Time
Arda Oguz	ardaerinco@gmail.com	413296029	10-05-1997	$30/hr	Casual
Damira Khamzina	d.khamzina@gmail.com	423968573	01-01-2000	$30/hr	Casual



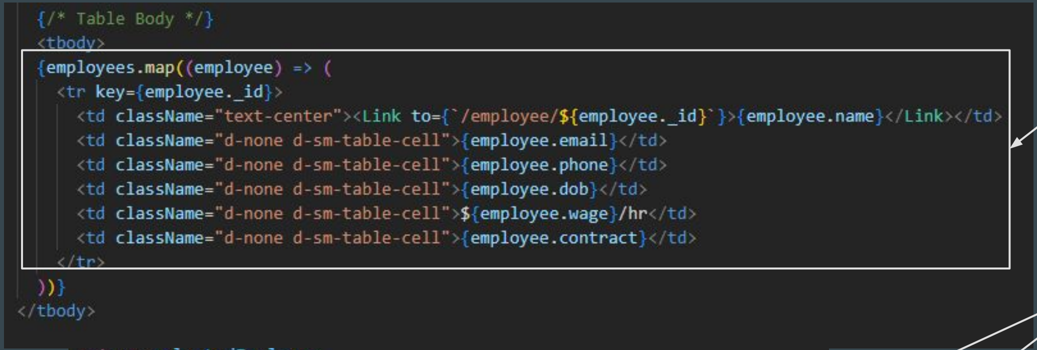
Nate Meinrad 's Details

Employee Name	Nate Meinrad
Email	nate.meinrad95@outlook.com
Phone	414958692
Date of Birth	01-07-1995
Wage	30
Contract	Part-Time

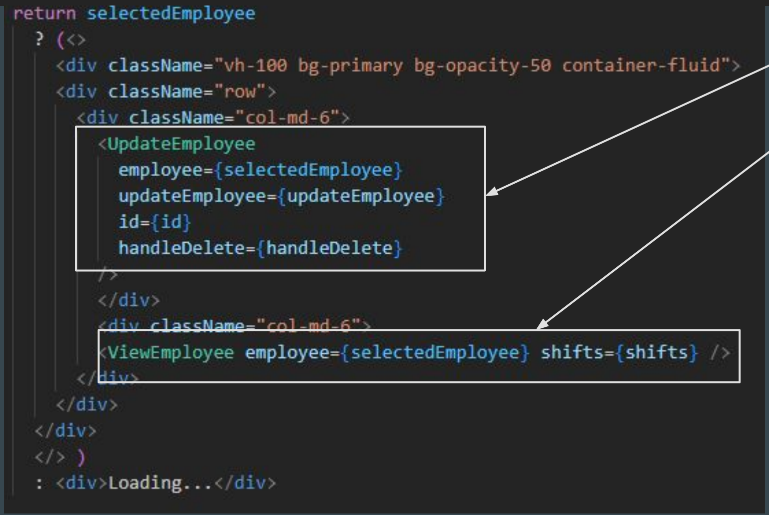
Nate Meinrad 's Shifts:

Date	Start Time	End Time	Break
2023-09-03	17:00	22:00	30 minutes
2023-09-12	07:00	10:00	0 minutes
2023-09-14	09:00	17:00	30 minutes

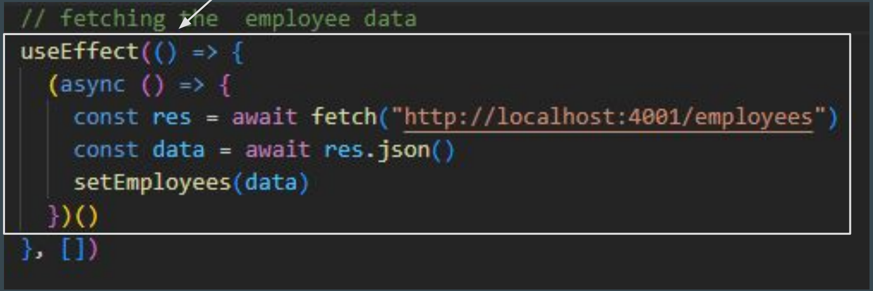
Team Management



```
  {/* Table Body */}
  <tbody>
  {employees.map((employee) => (
    <tr key={employee._id}>
      <td className="text-center"><Link to={`/employee/${employee._id}`}>{employee.name}</Link></td>
      <td className="d-none d-sm-table-cell">{employee.email}</td>
      <td className="d-none d-sm-table-cell">{employee.phone}</td>
      <td className="d-none d-sm-table-cell">{employee.dob}</td>
      <td className="d-none d-sm-table-cell">${employee.wage}/hr</td>
      <td className="d-none d-sm-table-cell">{employee.contract}</td>
    </tr>
  ))}
</tbody>
```



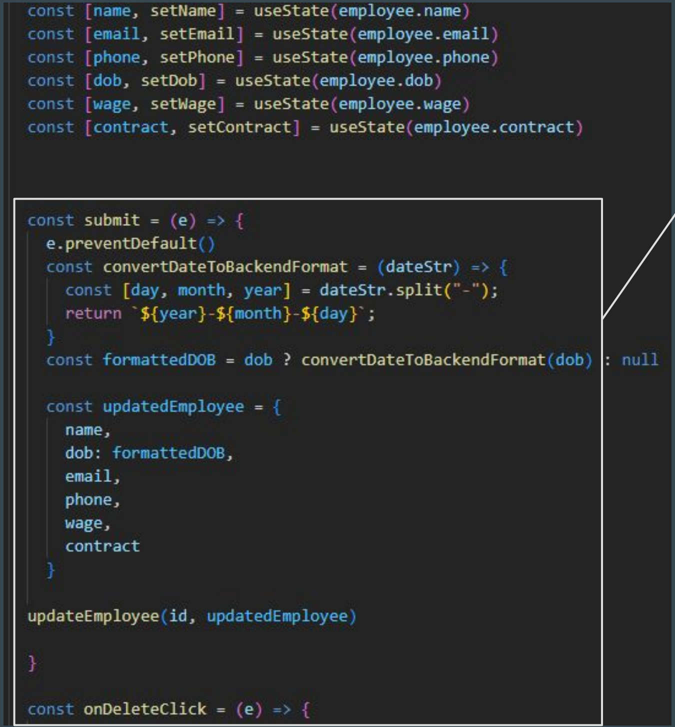
```
return selectedEmployee
  ? (<>
    <div className="vh-100 bg-primary bg-opacity-50 container-fluid">
    <div className="row">
      <div className="col-md-6">
        <UpdateEmployee
          employee={selectedEmployee}
          updateEmployee={updateEmployee}
          id={id}
          handleDelete={handleDelete}
        />
        </div>
        <div className="col-md-6">
        <ViewEmployee employee={selectedEmployee} shifts={shifts} />
      </div>
    </div>
  </div>
  </> )
  : <div>Loading...</div>
```



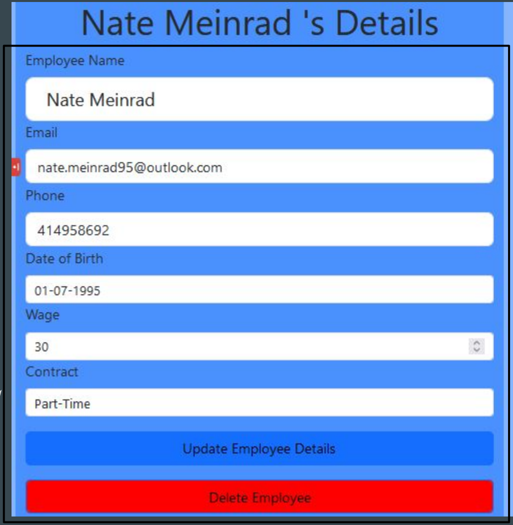
```
// fetching the  employee data
useEffect(() => {
  (async () => {
    const res = await fetch("http://localhost:4001/employees")
    const data = await res.json()
    setEmployees(data)
  })()
}, [])
```

- The employees registered in the database are iterated over, with their data populating the contents of the table. Each employee can be inspected on selection.
- Employee information and shifts are compartmentalized and passed into a single component that makes up their page
- useEffect and useState hooks are utilized to fetch employee data upon during the initial render, and trigger re-renders by setting the state

Team Management



```
const [name, setName] = useState(employee.name)
const [email, setEmail] = useState(employee.email)
const [phone, setPhone] = useState(employee.phone)
const [dob, setDob] = useState(employee.dob)
const [wage, setWage] = useState(employee.wage)
const [contract, setContract] = useState(employee.contract)


const submit = (e) => {
  e.preventDefault()
  const convertDateToBackendFormat = (dateStr) => {
    const [day, month, year] = dateStr.split("-");
    return `${year}-${month}-${day}`;
  }
  const formattedDOB = dob ? convertDateToBackendFormat(dob) : null

  const updatedEmployee = {
    name,
    dob: formattedDOB,
    email,
    phone,
    wage,
    contract
  }

updateEmployee(id, updatedEmployee)

}

const onDeleteClick = (e) => {
```



Nate Meinrad 's Details

Employee Name

Nate Meinrad

Email

nate.meinrad95@outlook.com

Phone

414958692

Date of Birth

01-07-1995

Wage

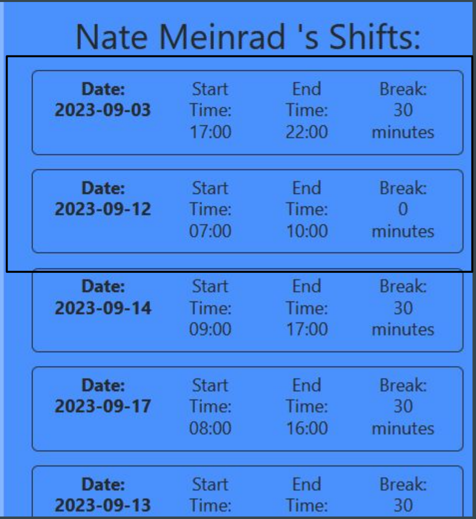
30

Contract

Part-Time

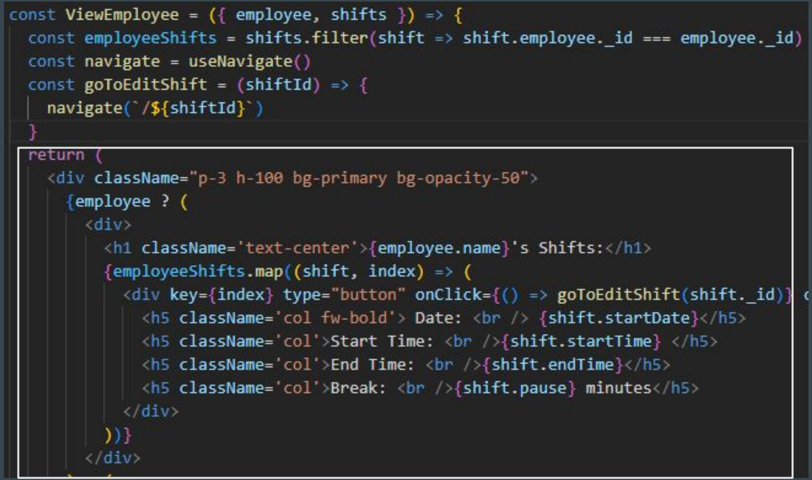
Update Employee Details

Delete Employee



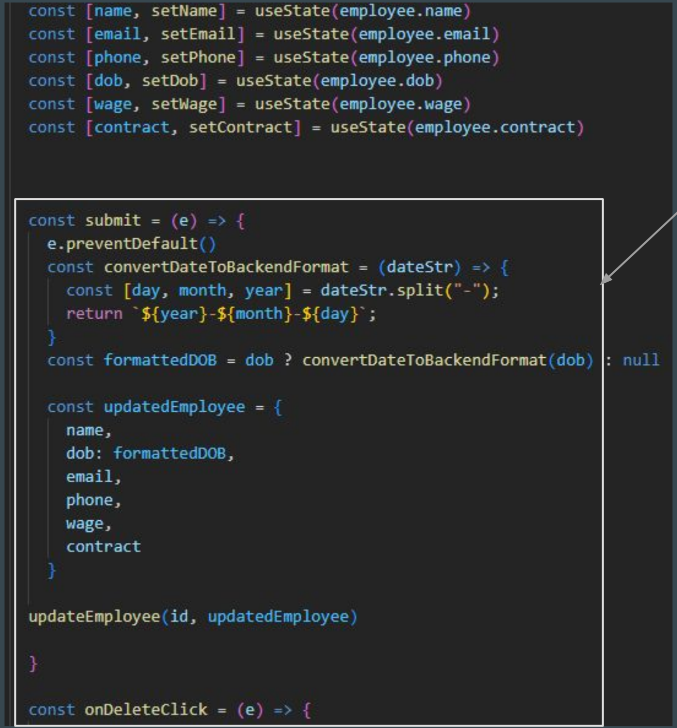
Nate Meinrad 's Shifts:

Date: 2023-09-03	Start Time: 17:00	End Time: 22:00	Break: 30 minutes
Date: 2023-09-12	Start Time: 07:00	End Time: 10:00	Break: 0 minutes
Date: 2023-09-14	Start Time: 09:00	End Time: 17:00	Break: 30 minutes
Date: 2023-09-17	Start Time: 08:00	End Time: 16:00	Break: 30 minutes
Date: 2023-09-13	Start Time:	End Time:	Break: 30



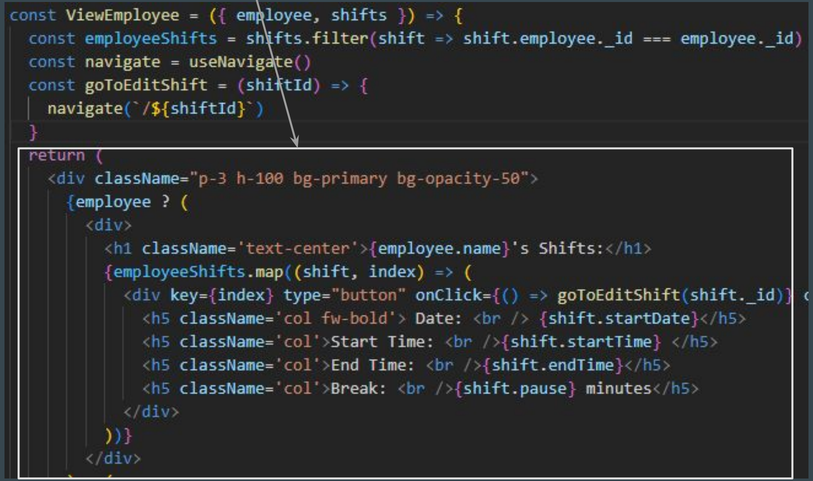
```
const ViewEmployee = ({ employee, shifts }) => {
  const employeeShifts = shifts.filter(shift => shift.employee._id === employee._id)
  const navigate = useNavigate()
  const goToEditShift = (shiftId) => {
    navigate(`/${shiftId}`)
  }
  return (
    <div className="p-3 h-100 bg-primary bg-opacity-50">
      {employee ? (
        <div>
          <h1 className='text-center'>{employee.name}'s Shifts:</h1>
          {employeeShifts.map((shift, index) => (
            <div key={index} type="button" onClick={() => goToEditShift(shift._id)}
              <h5 className='col fw-bold'> Date: <br /> {shift.startDate}</h5>
              <h5 className='col'>Start Time: <br />{shift.startTime} </h5>
              <h5 className='col'>End Time: <br />{shift.endTime}</h5>
              <h5 className='col'>Break: <br />{shift.pause} minutes</h5>
            </div>
          ))}
        </div>
```

Team Management



```
const [name, setName] = useState(employee.name)
const [email, setEmail] = useState(employee.email)
const [phone, setPhone] = useState(employee.phone)
const [dob, setDob] = useState(employee.dob)
const [wage, setWage] = useState(employee.wage)
const [contract, setContract] = useState(employee.contract)


const submit = (e) => {
  e.preventDefault()
  const convertDateToBackendFormat = (dateStr) => {
    const [day, month, year] = dateStr.split("-");
    return `${year}-${month}-${day}`;
  }
  const formattedDOB = dob ? convertDateToBackendFormat(dob) : null

  const updatedEmployee = {
    name,
    dob: formattedDOB,
    email,
    phone,
    wage,
    contract
  }

updateEmployee(id, updatedEmployee)

}

const onDeleteClick = (e) => {
```

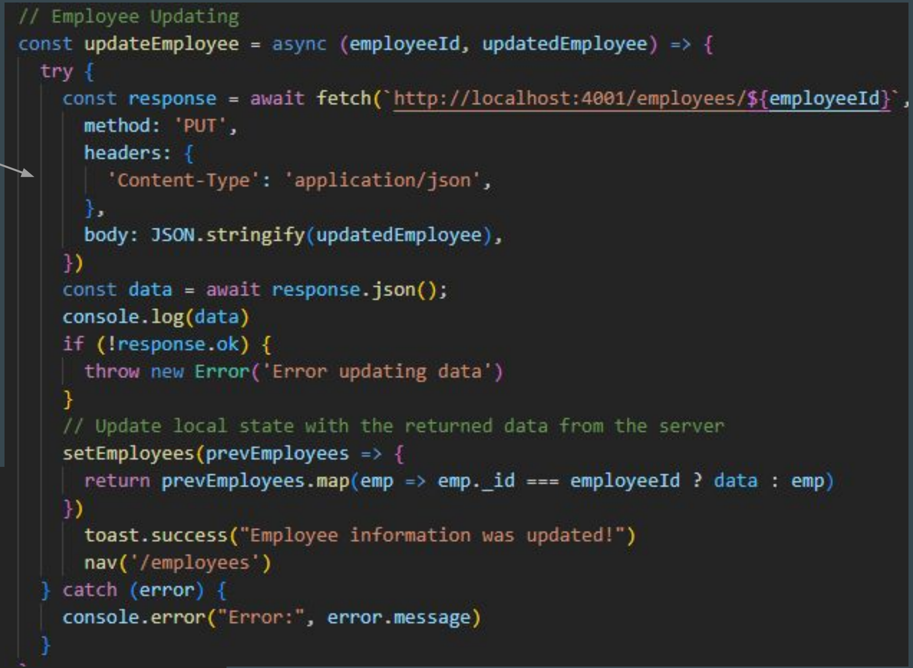
- Employee details automatically fill the input zones, and changes update both their employee profile as well as their shifts to reflect the changes. When an employee is deleted, their shifts are also deleted as well
- Shifts are filtered based on employee ID, and are iterated over to provide a list of the employee’s shifts, linking to the Shift Edit component



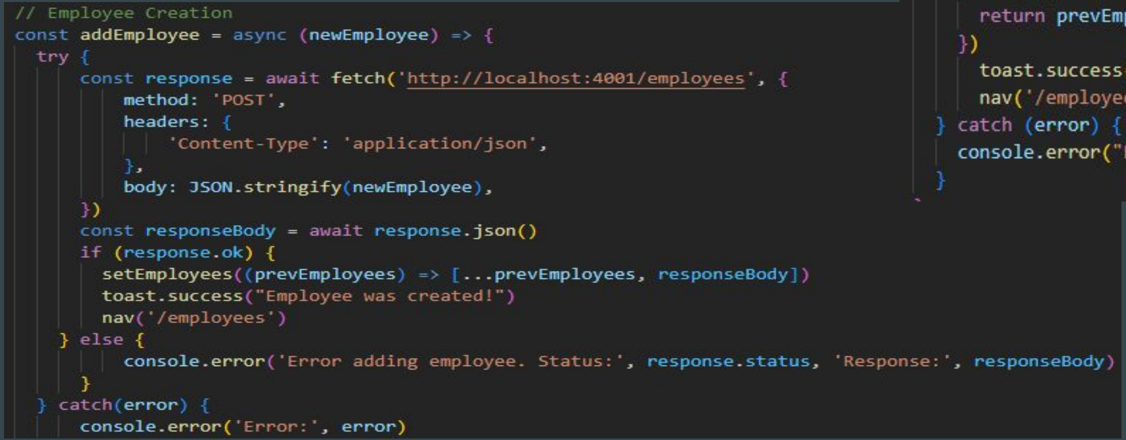
```
const ViewEmployee = ({ employee, shifts }) => {
  const employeeShifts = shifts.filter(shift => shift.employee._id === employee._id)
  const navigate = useNavigate()
  const goToEditShift = (shiftId) => {
    navigate(`/${shiftId}`)
  }
  return (
    <div className="p-3 h-100 bg-primary bg-opacity-50">
      {employee ? (
        <div>
          <h1 className='text-center'>{employee.name}'s Shifts:</h1>
          {employeeShifts.map((shift, index) => (
            <div key={index} type="button" onClick={() => goToEditShift(shift._id)} c
              <h5 className='col fw-bold'> Date: <br /> {shift.startDate}</h5>
              <h5 className='col'>Start Time: <br />{shift.startTime} </h5>
              <h5 className='col'>End Time: <br />{shift.endTime}</h5>
              <h5 className='col'>Break: <br />{shift.pause} minutes</h5>
            </div>
          ))}
        </div>
```

Team Management

- New / updated employee information is passed into the app component as the body of their respective requests.
- Once a response is received, the user is notified and redirected to the Employees page.



```
// Employee Updating
const updateEmployee = async (employeeId, updatedEmployee) => {
  try {
    const response = await fetch(`http://localhost:4001/employees/${employeeId}`,
      method: 'PUT',
      headers: {
        'Content-Type': 'application/json',
      },
      body: JSON.stringify(updatedEmployee),
    })
    const data = await response.json();
    console.log(data)
    if (!response.ok) {
      throw new Error('Error updating data')
    }
    // Update local state with the returned data from the server
    setEmployees(prevEmployees => {
      return prevEmployees.map(emp => emp._id === employeeId ? data : emp)
    })
      toast.success("Employee information was updated!")
      nav('/employees')
  } catch (error) {
    console.error("Error:", error.message)
  }
```



```
// Employee Creation
const addEmployee = async (newEmployee) => {
  try {
    const response = await fetch('http://localhost:4001/employees', {
      method: 'POST',
      headers: {
        'Content-Type': 'application/json',
      },
      body: JSON.stringify(newEmployee),
    })
    const responseBody = await response.json()
    if (response.ok) {
      setEmployees((prevEmployees) => [...prevEmployees, responseBody])
      toast.success("Employee was created!")
      nav('/employees')
  } else {
      console.error('Error adding employee. Status:', response.status, 'Response:', responseBody)
    }
  } catch(error) {
    console.error('Error:', error)
```

Thank You!