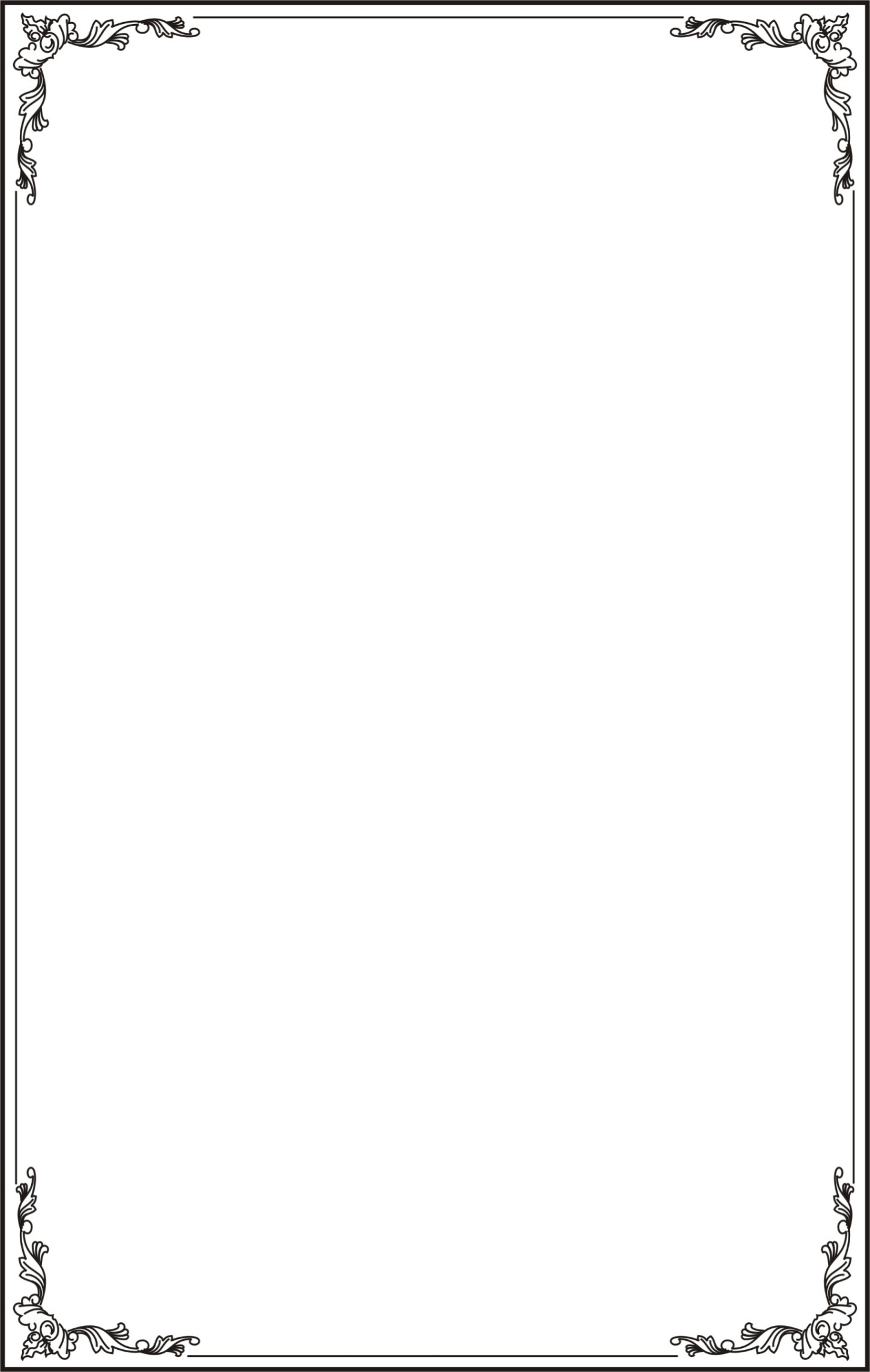
**CÔNG TY CỔ PHẦN**

**VIỄN THÔNG FPT**

****

**BÀI THU HOẠCH**

**ĐÀO TẠO TÂN BINH R&D TUẦN 11 & 12**

|  |  |
| --- | --- |
| Người hướng dẫn: | **MR. TRƯƠNG TẤN SANG** |
| Người thực hiện: | **TRẦN KHÁNH GIA UY** |
| Đơn vị công tác: | **Trung tâm giám sát và đảm bảo dịch vụ (SCC)** |
| Vị trí: | **Open Source Web Software Developer** |
| Bộ phận: | **SCC** |

***Thành phố Hồ Chí Minh, tháng 07 năm 2021***

# MỤC LỤC

[MỤC LỤC 2](#_Toc78130045)

[CHƯƠNG 1: TÌM HIỂU ANT DESIGN PRO 3](#_Toc78130046)

[1.1 Ant design pro 3](#_Toc78130047)

[1.2 Demo bizchart và tạo table chi tiết số lượng chat trên Ant design pro 4](#_Toc78130048)

[1.3 Source code 10](#_Toc78130049)

[1.4 Demo 10](#_Toc78130050)

[CHƯƠNG 2: TÌM HIỂU VỀ HỆ THỐNG INCIDENT MANAGEMENT 11](#_Toc78130051)

[2.1 Giới thiệu 11](#_Toc78130052)

[2.1.1 Giới thiệu nghiệp vụ 11](#_Toc78130053)

[2.1.2 Cài đặt cơ bản 11](#_Toc78130054)

[2.2 Cài đặt hiển thị thông tin sự cố 12](#_Toc78130055)

[2.2.1 Vấn đề 12](#_Toc78130056)

[2.2.2 Xử lý vấn đề 12](#_Toc78130057)

[2.3 Cài đặt hiển thị thông tin khi check incident 14](#_Toc78130058)

[2.3.1 Vấn đề 14](#_Toc78130059)

[2.3.2 Xử lý vấn đề 14](#_Toc78130060)

[2.4 Source code 15](#_Toc78130061)

[2.5 Demo 15](#_Toc78130062)

# CHƯƠNG 1: THỰC HIỆN CÁC TASK TRONG INCIDENT MANAGEMENT

## Bổ sung thêm trường *Level* của ticket

* + 1. **Yêu cầu**
* Ở trong *Ticket notify* và *Ticket list*, bổ sung thêm 2 trường *Level*. Wireframe:

**Graphical user interface, text, application

Description automatically generated**

**Graphical user interface

Description automatically generated with low confidence**

* + 1. **Thực hiện**
* Level của ticket notify đã được lưu trong redux, chính vì thế chỉ cần lấy dữ liệu từ redux vào thêm vào trường level của ticket:

**Text

Description automatically generated**

* Tương tự như vậy với ticket list
  1. **Chức năng Search/Filter ticket theo trường thông tin của ticket**
     1. **Yêu cầu**
* Ở các trường: *Ticket ID, Ticket name* sẽ có khung search để người dùng có thể tìm kiếm. Các trường *Level, Department, Status* sẽ có filter tương ứng. Thực hiện yêu cầu này với *Ticket notify* và *Ticket list*. Wireframe:

Graphical user interface, text, application

Description automatically generatedGraphical user interface, text, application, chat or text message

Description automatically generated

Graphical user interface, text, application, Word

Description automatically generatedGraphical user interface, application

Description automatically generatedGraphical user interface, application

Description automatically generated

* + 1. **Thực hiện**
* Trường *Ticket ID* đã được làm sẵn nên bỏ qua trường này
* Để lấy được dữ liệu tương ứng với các filter, em sẽ gọi API với query *must* có thêm các filter tương ứng. Ví dụ:

Text

Description automatically generated

* Bây giờ em sẽ lấy nội dung search/filter và đặt nó vào query để lấy dữ liệu tương ứng
  + - 1. **Ticket name**
* Tương tự với Ticket ID, em sẽ sửa lại biến để Ticket name có thể sử dụng chung. Sau khi lấy được nội dung tìm kiếm, em sẽ gọi API để tìm kiếm dữ liệu tương ứng.
  + - 1. **Level**
* Level em sử dụng Checkbox group trong Ant design để tạo giao diện filter. Khi người dùng nhấn nút *OK* sẽ lấy thông tin filter và lưu vào redux để gọi API.

function onChangeLevel(checkedValues, setSelectedKeys, setLevelChecked) {

  setSelectedKeys(checkedValues);

  setLevelChecked(checkedValues);

}

function handleFilterLevel(selectedKeys, confirm, setFilter) {

  setFilter({ Level: selectedKeys });

  confirm();

}

function handleResetFilterLevel(clearFilters, setFilter, setLevelChecked) {

  clearFilters();

  setFilter({ Level: [1, 2, 3, 4, 5, 6] });

  setLevelChecked([]);

}

const levelOptions = [

  { label: '1', value: '1' },

  { label: '2', value: '2' },

  { label: '3', value: '3' },

  { label: '4', value: '4' },

  { label: '5', value: '5' },

  { label: '6', value: '6' },

];

export const customUI\_level = {

  searching: (setFilter, levelChecked, setLevelChecked) => ({

    filterDropdown: ({ setSelectedKeys, selectedKeys, confirm, clearFilters }) => (

      <div style={{ padding: 8, width: 100 }}>

        <Checkbox.Group

          style={{ width: '100%' }}

          options={levelOptions}

          value={levelChecked}

          onChange={checkedValues => onChangeLevel(checkedValues, setSelectedKeys, setLevelChecked)}

        ></Checkbox.Group>

        <Divider style={{ marginTop: 5, marginBottom: 0, marginLeft: 0, marginRight: 0 }} />

        <Button

          type="link"

          size="small"

          style={{ width: 25, marginRight: 8, textAlign: 'right' }}

          onClick={() => handleFilterLevel(selectedKeys, confirm, setFilter)}

        >

          OK

        </Button>

        <Button

          size="small"

          type="link"

          style={{ width: 50, textAlign: 'left' }}

          onClick={() => handleResetFilterLevel(clearFilters, setFilter, setLevelChecked)}

        >

          Reset

        </Button>

      </div>

    ),

    filterIcon: filtered => <FilterOutlined style={{ color: filtered ? '#1890ff' : undefined }} />,

  }),

};

* + - 1. **Department**
* Người dùng có thể chọn các queue trong danh sách, bên cạnh đó, người dùng có thể tìm kiếm trong danh sách đó giúp cho việc chọn lựa nhanh hơn.
* Để tối ưu cho việc tìm kiếm, em sẽ sử dụng *debounce* từ thư viện lodash để có thể tìm kiếm sau khi người dùng gõ xong.

const SearchCheckbox = ({ queues, confirm, clearFilters, setFilterQueues }) => {

  const [userQuery, setUserQuery] = useState('');

  const [queueList, setQueueList] = useState(

    queues.map(item => {

      return { label: item.Name, value: item.ID, checked: false };

    }),

  );

  const [queueRender, setQueueRender] = useState([]);

  const sendQuery = query => {

    const searchedList = queueList.filter(item => {

      return item.label.toLowerCase().indexOf(query.trim().toLowerCase()) !== -1;

    });

    setQueueRender(searchedList);

  };

  const updateQuery = () => sendQuery(userQuery);

  const delayedQuery = useCallback(debounce(updateQuery, 500), [userQuery]);

  const onChange = e => {

    setUserQuery(e.target.value);

  };

  useEffect(() => {

    delayedQuery();

    return delayedQuery.cancel;

  }, [queueRender, userQuery, delayedQuery]);

  const onChangeQueue = checkedValue => {

    let value = checkedValue.target.value;

    let tempQueueRender = queueRender;

    for (let item of tempQueueRender) {

      if (item.value === value) {

        item.checked = !item.checked;

        break;

      }

    }

    setQueueRender(tempQueueRender);

  };

  function handleFilterQueue(confirm, setFilter) {

    let result = [];

    for (let item of queueList) {

      if (item.checked === true) {

        result.push(item.value);

      }

    }

    if (result.length !== 0) {

      setFilter({ Queue: result });

    } else {

      setFilter({ Queue: undefined });

    }

    confirm();

  }

  function handleResetFilterQueue(clearFilters, setFilter) {

    clearFilters();

    setFilter({ Queue: undefined });

    setQueueList(

      queues.map(item => {

        return { label: item.Name, value: item.ID, checked: false };

      }),

    );

    setUserQuery('');

  }

  return (

    <div>

      <Input

        placeholder="Search"

        onChange={e => onChange(e)}

        value={userQuery}

        bordered={false}

        style={{ width: 100, marginBottom: 8, display: 'block' }}

      />

      <div style={{ width: 130, height: 200, overflow: 'auto' }}>

        {queueRender.map(item => {

          return (

            <Row>

              <Checkbox value={item.value} checked={item.checked} onChange={onChangeQueue}>

                {item.label}

              </Checkbox>

            </Row>

          );

        })}

      </div>

      <Divider style={{ marginTop: 5, marginBottom: 0, marginLeft: 0, marginRight: 0 }} />

      <Button

        type="link"

        size="small"

        style={{ width: 50, marginRight: 8 }}

        onClick={() => handleFilterQueue(confirm, setFilterQueues)}

      >

        OK

      </Button>

      <Button

        size="small"

        type="link"

        style={{ width: 50 }}

        onClick={() => handleResetFilterQueue(clearFilters, setFilterQueues)}

      >

        Reset

      </Button>

    </div>

  );

};

export const customUI\_department = {

  searching: (queues, setFilterQueues) => ({

    filterDropdown: ({ confirm, clearFilters }) => (

      <div style={{ padding: 8 }}>

        <SearchCheckbox

          queues={queues}

          confirm={confirm}

          clearFilters={clearFilters}

          setFilterQueues={setFilterQueues}

        />

      </div>

    ),

    filterIcon: filtered => <FilterOutlined style={{ color: filtered ? '#1890ff' : undefined }} />,

  }),

};

* Sau khi người dùng không gõ tiếp trong vòng nửa giây, hệ thống sẽ tìm kiếm queue tương ứng.

Graphical user interface, application, table

Description automatically generated

* Để lưu lại những queue mà người dùng chọn qua các lần search, em sẽ sử dụng state *queueList* với trường *checked* là *true* mỗi khi người dùng chọn queue đó và *false* mỗi khi người dùng bỏ chọn.
* Cuối cùng khi nhấn *OK* hệ thống sẽ lấy những queue có *checked* là *true* để gọi API.
  + - 1. **Status**
* Tương tự với *Level* em cũng sẽ sử dụng Checkbox group để lấy thông tin filter.

function onChangeStatus(checkedValues, setSelectedKeys, setStatusChecked) {

  setSelectedKeys(checkedValues);

  setStatusChecked(checkedValues);

}

function handleFilterStatus(selectedKeys, confirm, setFilter) {

  setFilter({ TicketStatus: selectedKeys });

  confirm();

}

function handleResetFilterStatus(clearFilters, setFilter, setStatusChecked) {

  clearFilters();

  setFilter({ TicketStatus: [-2, -1, 0, 2, 3, 5, 8, 9, 10] });

  setStatusChecked([]);

}

const statusOptions = [

  { label: 'NEW', value: '-1' },

  { label: 'CHANGED', value: '2' },

  { label: 'FORWARD', value: '3' },

  { label: 'ASSIGNMENT', value: '9' },

  { label: 'PENDING', value: '5' },

  { label: 'RESOLVED', value: '10' },

  { label: 'REOPEN', value: '8' },

  { label: 'NOTIFIED', value: '-2' },

];

export const customUI\_status = {

  searching: (setFilterStatus, statusChecked, setStatusChecked) => ({

    filterDropdown: ({ setSelectedKeys, selectedKeys, confirm, clearFilters }) => (

      <div style={{ padding: 8, width: 150 }}>

        <Checkbox.Group

          style={{ width: '100%' }}

          options={statusOptions}

          value={statusChecked}

          onChange={checkedValues =>

            onChangeStatus(checkedValues, setSelectedKeys, setStatusChecked)

          }

        ></Checkbox.Group>

        <Divider style={{ marginTop: 5, marginBottom: 0, marginLeft: 0, marginRight: 0 }} />

        <Button

          type="link"

          size="small"

          style={{ width: 55, marginRight: 8, justifyContent: 'right' }}

          onClick={() => handleFilterStatus(selectedKeys, confirm, setFilterStatus)}

        >

          OK

        </Button>

        <Button

          size="small"

          type="link"

          style={{ width: 55, justifyContent: 'left' }}

          onClick={() => handleResetFilterStatus(clearFilters, setFilterStatus, setStatusChecked)}

        >

          Reset

        </Button>

      </div>

    ),

    filterIcon: filtered => <FilterOutlined style={{ color: filtered ? '#1890ff' : undefined }} />,

  }),

};