ECE 448/528
Application Software Design

Lecture 22. User Interface Design I
Spring 2025

Won-Jae Yi, Ph.D.

Department of Electrical and Computer Engineering
Illinois Institute of Technology

UI in a Hierarchy

From the Previous Lecture

```
// public/web/members_app.js
class MembersApp extends React.Component {
  ...
  render() {
    return (
      <div className="container">
        ...
            <Members />
        ...
      </div>);
  }
}
window.MembersApp = MembersApp;

// public/web/members.js
function create_members_model(groups) {
  ...
  return that;
}
```

The Members Controller

```
// public/web/members.js
... // definition of function create_members_model(groups)
class Members extends React.Component {
  constructor(props) {
    super(props);
    this.state ={members: create_members_model([
      {name: "A", members: ["a", "b", "c"]},
      {name: "B", members: ["c", "d", "e"]},
      {name: "C", members: ["a", "c", "e"]}])};
  }
  ...
}
window.Members = Members;
```

- A stateful component.
- The members model is part of the Controller state.
  - For now, initialize it with some sample data in the same format as RESTful responses.

React Components

- **State** in React
  - A built-in plain JavaScript object used to contain data/information about the component
  - Used to store the data of the components that need to be rendered to the view
  - Stores component data that needs to be rendered on the view
- **Props** in React
  - A built-in plain JavaScript object used to pass data and event handlers to the children's components
  - Used to pass data and event handler to children’s component

Passing Model Around

```
class Members extends React.Component {
  constructor(props) {
    super(props);
    this.state ={members: create_members_model([
      {name: "A", members: ["a", "b", "c"]},
      {name: "B", members: ["c", "d", "e"]},
      {name: "C", members: ["a", "c", "e"]}])};
  }
  render() {return (<MembersTable members={this.state.members} />);}
}
window.Members = Members;
```

- Create a component of the type MembersTable in render().
  - As the *View*.
  - Recall that a stateful component focuses on the *Model* and *Controller* and should avoid providing the View functionality.
- Pass the members model to the MembersTable.
  - As the argument members, similar to HTML attributes.
  - In HTML-like tags of JSX, use {} to surround JavaScript code.
- **super**: used to call the constructor of the parent class
  - To access variables from the parent class

The MembersTable View

```
// public/web/members_table.js
function MembersTable(props) {
  if (props.members.get_group_names().length == 0)
    return (<div>There are no groups.</div>);
  ...;
}
window.MembersTable = MembersTable;
```

- Stateless components are functions.
  - The name of the function serves as the type of the component.
- The function will take a single parameter, props.
  - Arguments passed by the parent component, like members, can be found inside.
- The function of a stateless component is conceptually the combination of the constructor, and the render function is a stateful component.
  - As you may guess, the props parameter in the constructor of a stateful component serves the same purpose.

Hierarchy of Views

```
// public/web/members_table.js
function MembersTable(props) {
  if (props.members.get_group_names().length == 0)
    return (<div>There are no groups.</div>);
  return (
    <table className="table table-striped table-bordered">
      <Header groupNames={props.members.get_group_names()} />
      <Body members={props.members} />
    </table>);
}
window.MembersTable = MembersTable;
```

- There is no need to create everything all at once – delegate to additional Views!
- Define Header and Body in the same file `members_table.js` before MembersTable so there is no need to export them.
  - A good practice since these two names are so generic that other components may use the same for different purposes.

The Header View

```
function Header(props) {
  ...
  return (
    <thead>
      <tr>
        <th rowSpan="2" width="10%">Members</th>
        <th  colSpan={props.groupNames.length}>Groups</th>
        <th rowSpan="2" width="10%">Remove from All Groups</th>
      </tr>
      <tr>
        {ths}
      </tr>
    </thead>);
}
```

- Most are adapted from the UI mockup code.
- Note that the attribute names rowSpan and colSpan are different than those of HTML (rowspan, colspan).
- The column headers for group names depend on the model – they need to be generated as ths by JavaScript code.

The Header View

```
function Header(props) {
  var ths =[];
  for (var groupName of props.groupNames) {
    ths.push(<th key={groupName}>
      <button className={btnClassAdd}>{groupName}</button>
    </th>);
  }
  ...
```

- An array and a for of loop will do the job.
- For an array of elements, React requires the use of unique key attributes.
  - key, a unique identifier, is required by React to identify which items have changed, added, or removed

The Body View

```
function Body(props) {
  var rows = props.members.get_member_names().map(memberName =>
    <Row key={memberName} memberName={memberName} members={props.members} />);

  return (<tbody>{rows}</tbody>);
}
```

- The map method, together with a lambda function, can work as an alternative to the for of loop.
  - Especially if you are generating one array from another.

The Row View

```
function Row(props) {
  var members = props.members;
  var tds =members.get_group_names().map(groupName => {
    if (members.is_member_in_group(props.memberName, groupName)) {
      return (<td key={groupName}>
        <input type="checkbox" checked/></td>);
    }
    else {
      return (<td key={groupName}>
        <input type="checkbox"/></td>);
    }
  });

  return (
    <tr>
      <td><button className={btnClassAdd}>{props.memberName}</button></td>
      {tds}
      <td><button className={btnClassDel}>X</button></td>
    </tr>
  );
}
```

React Component Lifecycle

React Component Lifecycle

- We use stateful React component to construct controllers.
- As discussed, a controller will be responsible for:
  - Handle events triggered by user inputs.
  - Handle events triggered by incoming network communications.
- The React library will generate additional events for components to handle.
  - Lifecycle events: when certain things happen regarding the component.

Typical Lifecycle Events

- In the context of MVC pattern.
- When creating the controller,
  - Handled by the constructor of the component.
  - A good time to initialize the models in this.state – set default values instead of requesting data from the server backend, in case the server backend fails.
- When the models change,
  - Handled by the render method.
  - Update the view as needed, possibly trigger all child views to be updated hierarchically.
- When the component is ready,
  - e.g., when models are created, and the view is displayed to the user.
  - Handled by the componentDidMount method.
  - Users now have something to look at on the browser– a good time to start contacting the server backend to obtain some data.

Handling componentDidMount Event

```
class Members extends React.Component {
  ...
  componentDidMount() {
    console.info("Members componentDidMount()");
    this.getGroups();
  }
  ...
}
```

- In `public/web/members.js` under branch `lec22-lifecycle`
- Let the getGroups method send the RESTful request.
- At this point, the users already see the view rendered with the initial models.
  - By executing render once with initial this.state.
  - If the RESTful response comes back too quickly, you won't notice such an initial view – can you delay the RESTful request so that the initial view can be seen for learning purposes?

The setState Method

```
class Members extends React.Component {
  ...
  getGroups = () => {
    fetch("api/groups")
      .then(rsp => rsp.json())
      .then(groups => this.showGroups(groups))
      .catch(err => console.error("Members: getGroups", err));
  }
  showGroups = groups => {
    this.setState({members: create_members_model(groups)});
  }
  ...
```

- RESTful request/response are similar to those in Lecture 19.
- In showGroups, instead of updating HTML DOM directly, we need to change the model.
  - And expect React to update the view by render.
- We change the model and notify React at the same time by using the setState method.

Cautions

```
class Members extends React.Component {
  ...
  getGroups = () => {
    ...
  }
  showGroups = groups => {
    this.setState({members: create_members_model(groups)});
  }
  ...
```

- Don't modify this.state directly.
  - React needs to be notified that the state/models are changed.
- Don't try to call render by yourself.
  - The MVC pattern says the view shall be updated when the model changes, and we should allow React to handle that automatically.
- Use arrow functions to define methods that are not lifecycle events.
  - As mentioned before, OOP syntax in JavaScript is somewhat broken.

Order of Lifecycle Events

```
1 MembersApp constructor()
2 MembersApp render()
3 Members constructor()
4 Members render()
5 MembersTable()
6 Members componentDidMount()
7 Members render()
8 MembersTable()
```

- Use logging to understand the order of lifecycle events.
  - Especially when there is a hierarchy of components.
- Line 1 → 2 and 3 → 4: constructor will trigger the first render of the same component.
- Line 2 → 3 and 5: the first render will trigger child components to be created.
- Line 6: componentDidMount is triggered when child components are rendered for the first time.
- Line 7 and 8: subsequent render's are triggered hierarchically when the state/models are changed.
  - No more constructor or componentDidMount are called.