app.ts

```typescript
// Project Type
enum ProjectStatus {
    Active,
    Finished
}

class Project {
    constructor(
        public id: string,
        public title: string,
        public description: string,
        public people: number,
        public status: ProjectStatus
    ) {
    }
}

// Project State Management
type Listener<T> = (items: T[]) => void;

class State<T> {
    protected listeners: Listener<T>[] = [];

    addListener(listenerFn: Listener<T>) {
        this.listeners.push(listenerFn);
    }
}

class ProjectState extends State<Project> {
    private projects: Project[] = [];
    private static instance: ProjectState;

    private constructor() {
        super();
    }

    static getInstance() {
        if (this.instance) {
            return this.instance;
        }
        this.instance = new ProjectState();
        return this.instance;
    }

    addProject(title: string, description: string, numOfPeople: number) {
        const newProject = new Project(
            Math.random().toString(),
            title,
            description,
            numOfPeople,
            ProjectStatus.Active
        );
        this.projects.push(newProject);
        for (const listenerFn of this.listeners) {
            listenerFn(this.projects.slice());
        }
    }
}

const projectState = ProjectState.getInstance();

// Validation
interface Validatable {
    value: string | number;
    required?: boolean;
    minLength?: number;
    maxLength?: number;
    min?: number;
    max?: number;
}

function validate(validatableInput: Validatable) {
    let isValid = true;
    if (validatableInput.required) {
        isValid = isValid && validatableInput.value.toString().trim().length !== 0;
    }
    if (validatableInput.minLength != null && typeof validatableInput.value === 'string') {
        isValid = isValid && validatableInput.value.length >= validatableInput.minLength;
    }
    if (validatableInput.maxLength != null && typeof validatableInput.value === 'string') {
        isValid = isValid && validatableInput.value.length <= validatableInput.maxLength;
    }
    if (validatableInput.min != null && typeof validatableInput.value === 'number') {
        isValid = isValid && validatableInput.value >= validatableInput.min;
    }
    if (validatableInput.max != null && typeof validatableInput.value === 'number') {
        isValid = isValid && validatableInput.value <= validatableInput.max;
    }
    return isValid;
}

// autobind decorator
function autobind(_: any, _2: string, descriptor: PropertyDescriptor) {
    const originalMethod = descriptor.value;
    const adjDescriptor: PropertyDescriptor = {
        configurable: true,
        get() {
            const boundFn = originalMethod.bind(this);
            return boundFn;
        }
    };
    return adjDescriptor;
}

// Component Base Class
abstract class Component<T extends HTMLElement, U extends HTMLElement> {
    templateElement: HTMLTemplateElement;
    hostElement: T;
    element: U;

    constructor(
        templateId: string,
        hostElementId: string,
        insertAtStart: boolean,
        newElementId?: string
    ) {
        this.templateElement = document.getElementById(templateId)! as HTMLTemplateElement;
        this.hostElement = document.getElementById(hostElementId)! as T;

        const importedNode = document.importNode(this.templateElement.content, true);
        this.element = importedNode.firstElementChild as U;
        if (newElementId) {
            this.element.id = newElementId;
        }

        this.attach(insertAtStart);
    }

    private attach(insertAtBeginning: boolean) {
        this.hostElement.insertAdjacentElement(insertAtBeginning ? 'afterbegin' : 'beforeend', this.element);
    }

    abstract configure(): void;

    abstract renderContent(): void;
}

// ProjectItem Class
class ProjectItem extends Component<HTMLUListElement, HTMLLIElement> {
    private project: Project;

    get persons() {
        if (this.project.people === 1) {
            return '1 person';
        } else {
            return `${this.project.people} persons`;
        }
    }

    constructor(hostId: string, project: Project) {
        super('single-project', hostId, false, project.id);
        this.project = project;

```

```typescript
        this.configure();
        this.renderContent();
    }

    configure() {
    }

    renderContent() {
        this.element.querySelector('h2')!.textContent = this.project.title;
        this.element.querySelector('h3')!.textContent = this.persons + ' assigned';
        this.element.querySelector('p')!.textContent = this.project.description;
    }
}

// ProjectList Class
class ProjectList extends Component<HTMLDivElement, HTMLElement> {
    assignedProjects: Project[];

    constructor(private type: 'active' | 'finished') {
        super('project-list', 'app', false, `${type}-projects`);
        this.assignedProjects = [];

        this.configure();
        this.renderContent();
    }

    configure() {
        projectState.addListener((projects: Project[]) => {
            const relevantProjects = projects.filter(prj => {
                if (this.type === 'active') {
                    return prj.status === ProjectStatus.Active;
                }
                return prj.status === ProjectStatus.Finished;
            });
            this.assignedProjects = relevantProjects;
            this.renderProjects();
        });
    }

    renderContent() {
        const listId = `${this.type}-projects-list`;
        this.element.querySelector('ul')!.id = listId;
        this.element.querySelector('h2')!.textContent = this.type.toUpperCase() + ' PROJECTS';
    }

    private renderProjects() {
        const listEl = document.getElementById(`${this.type}-projects-list`)! as HTMLUListElement;
        listEl.innerHTML = '';
        for (const prjItem of this.assignedProjects) {
            new ProjectItem(this.element.querySelector('ul')!.id, prjItem);
        }
    }
```

```
}

// ProjectInput Class
class ProjectInput extends Component<HTMLDivElement, HTMLFormElement> {
    titleInputElement: HTMLInputElement;
    descriptionInputElement: HTMLInputElement;
    peopleInputElement: HTMLInputElement;

    constructor() {
        super('project-input', 'app', true, 'user-input');
        this.titleInputElement = this.element.querySelector('#title') as HTMLInputElement;
        this.descriptionInputElement = this.element.querySelector('#description') as HTMLInputElement;
        this.peopleInputElement = this.element.querySelector('#people') as HTMLInputElement;
        this.configure();
    }

    configure() {
        this.element.addEventListener('submit', this.submitHandler);
    }

    renderContent() {
    }

    private gatherUserInput(): [string, string, number] | void {
        const enteredTitle = this.titleInputElement.value;
        const enteredDescription = this.descriptionInputElement.value;
        const enteredPeople = this.peopleInputElement.value;

        const titleValidatable: Validatable = {
            value: enteredTitle,
            required: true
        };
        const descriptionValidatable: Validatable = {
            value: enteredDescription,
            required: true,
            minLength: 5
        };
        const peopleValidatable: Validatable = {
            value: +enteredPeople,
            required: true,
            min: 1,
            max: 5
        };

        if (
            !validate(titleValidatable) ||
            !validate(descriptionValidatable) ||
            !validate(peopleValidatable)
        ) {
            alert('Invalid input, please try again!');
            return;
        } else {
```

```typescript
            return [enteredTitle, enteredDescription, +enteredPeople];
        }
    }

    private clearInputs() {
        this.titleInputElement.value = '';
        this.descriptionInputElement.value = '';
        this.peopleInputElement.value = '';
    }

    @autobind
    private submitHandler(event: Event) {
        event.preventDefault();
        const userInput = this.gatherUserInput();
        if (Array.isArray(userInput)) {
            const [title, desc, people] = userInput;
            projectState.addProject(title, desc, people);
            this.clearInputs();
        }
    }
}

const prjInput = new ProjectInput();
const activePrjList = new ProjectList('active');
const finishedPrjList = new ProjectList('finished');
```