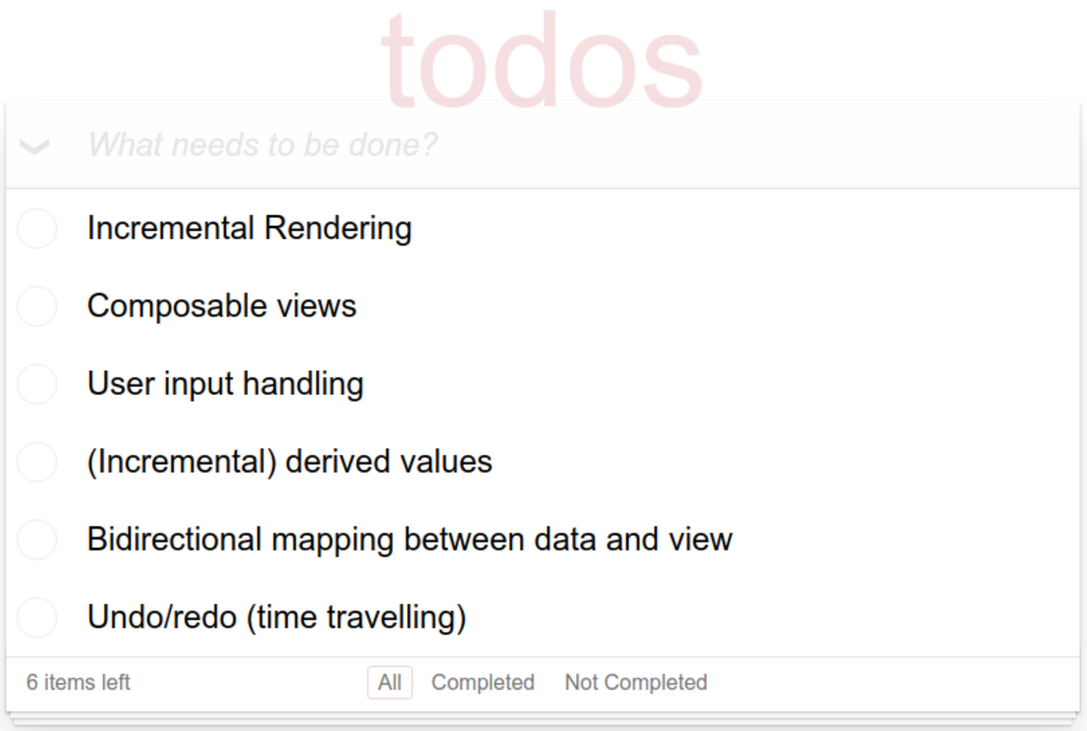
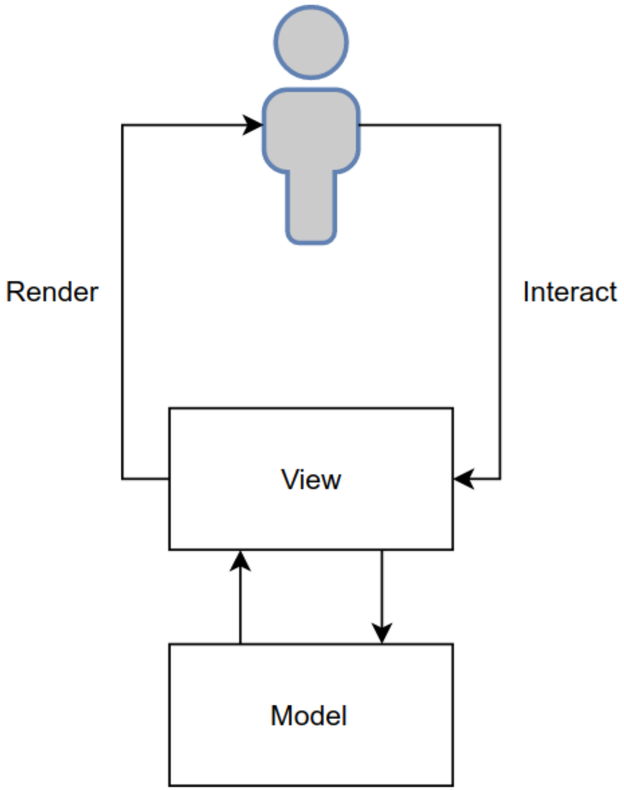
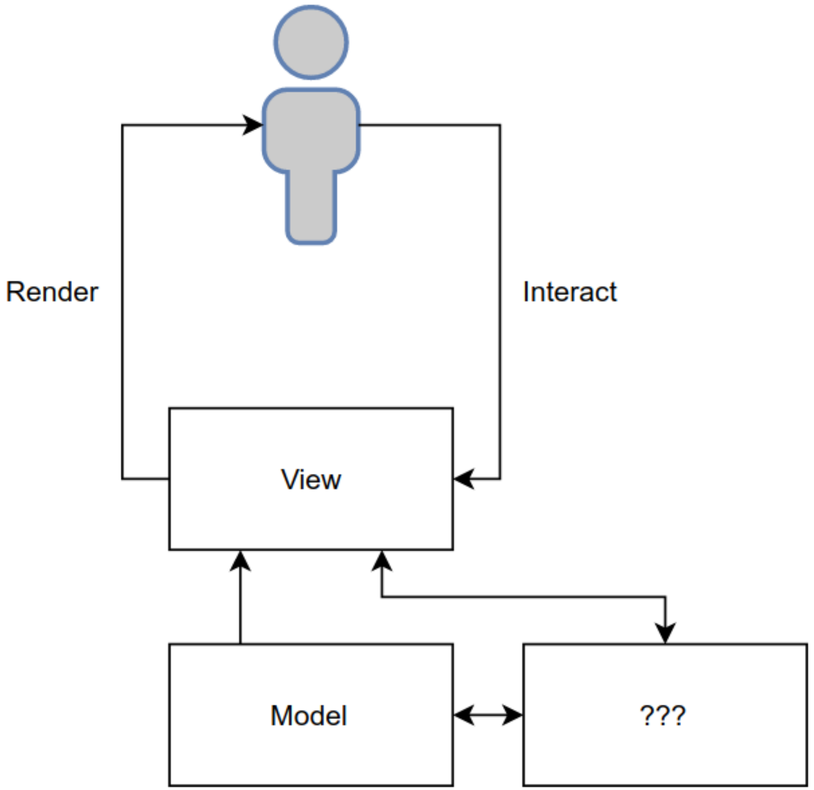
PixieDust: Declarative incremental user interfaces for IceDust



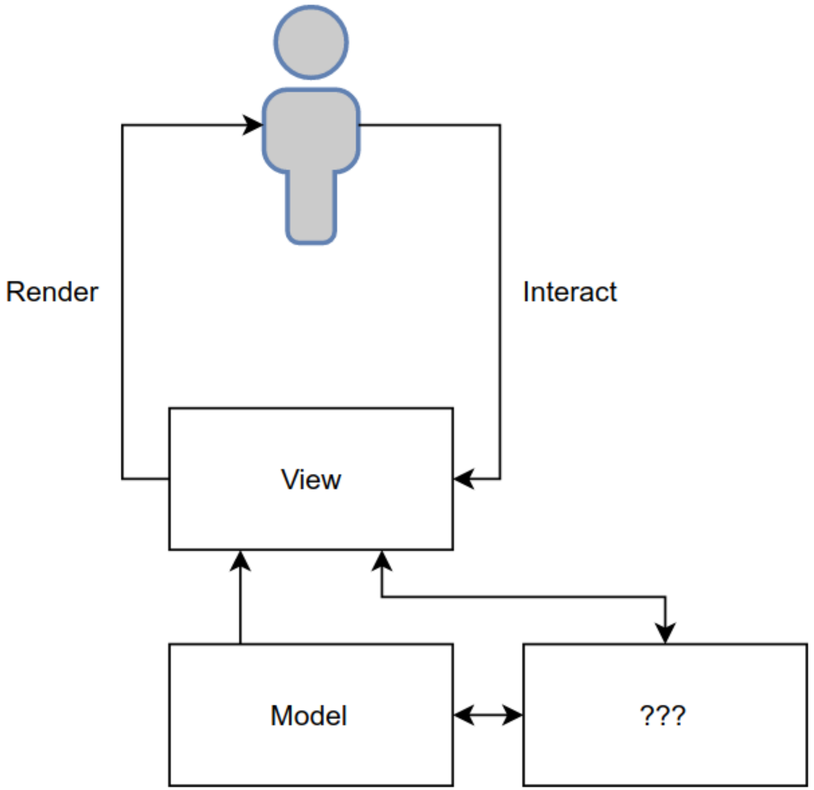
UI Pattern



UI Pattern



UI Pattern



- Incremental Rendering
- Composable views
- User input handling
- (Incremental) derived values
- Bidirectional mapping between data and view
- Undo/redo (time travelling)

Todo

Reset

todo-example

todos

What needs to be done?

0 items left All Completed Not Completed

Todo.js

```
export const addTodo = text => ({ type: types.ADD_TODO,
text })
export const deleteTodo = id => ({ type:
types.DELETE_TODO, id })
export const editTodo = (id, text) => ({ type:
types.EDIT_TODO, id, text })
export const completeTodo = id => ({ type:
types.COMPLETE_TODO, id })
export const completeAll = () => ({ type:
types.COMPLETE_ALL })
export const clearCompleted = () => ({ type:
types.CLEAR_COMPLETED })

const initialState = [
  {
    text: 'Use Redux',
    completed: false,
    id: 0
  }
]

export default function todos(state = initialState,
action) {
  switch (action.type) {
    case ADD_TODO:
      return [
        ...state,
        {
          id: state.reduce((maxId, todo) =>
Math.max(todo.id, maxId), -1) + 1,
          completed: false,
          text: action.text
        }
      ]

    case DELETE_TODO:
      return state.filter(todo =>
        todo.id !== action.id
      )

    case EDIT_TODO:
      return state.map(todo =>
        todo.id === action.id ?
        { ...todo, text: action.text } :
          todo
      )

    case COMPLETE_TODO:
      return state.map(todo =>
        todo.id === action.id ?
        { ...todo, completed: !todo.completed } :
          todo
      )

    case COMPLETE_ALL:
      const areAllMarked = state.every(todo =>
todo.completed)
      return state.map(todo => ({
        ...todo,
        completed: !areAllMarked
      }))

    case CLEAR_COMPLETED:
      return state.filter(todo => todo.completed ===
false)

    default:
      return state
  }
}
```

```
export default class TodoTextInput extends Component {
  static propTypes = {
    onSave: PropTypes.func.isRequired,
    text: PropTypes.string,
    placeholder: PropTypes.string,
    editing: PropTypes.bool,
    newTodo: PropTypes.bool
  }

  state = {
    text: this.props.text || ''
  }

  handleSubmit = e => {
    const text = e.target.value.trim()
    if (e.which === 13) {
      this.props.onSave(text)
      if (this.props.newTodo) {
        this.setState({ text: '' })
      }
    }
  }

  handleChange = e => {
    this.setState({ text: e.target.value })
  }

  handleBlur = e => {
    if (!this.props.newTodo) {
      this.props.onSave(e.target.value)
    }
  }

  render() {
    return (
      <input className={
      classnames({
        edit: this.props.editing,
      'new-todo': this.props.newTodo
})}
      type="text"
      placeholder={this.props.placeholder}
      autoFocus="true"
      value={this.state.text}
      onBlur={this.handleBlur}
      onChange={this.handleChange}
      onKeyDown={this.handleSubmit} />
    )
  }
}


const App = ({todos, actions}) => (
<div>
<Header addTodo={actions.addTodo} />
<MainSection todos={todos} actions={actions} />
  </div>
)

App.propTypes = {
  todos: PropTypes.array.isRequired,
  actions: PropTypes.object.isRequired
}

const mapStateToProps = state => ({
  todos: state.todos
})

const mapDispatchToProps = dispatch => ({
  actions: bindActionCreators(TodoActions, dispatch)
})

export default connect(
  mapStateToProps,
  mapDispatchToProps
)(App)
```

```
export default class Footer extends Component {
  static propTypes = {
    completedCount: PropTypes.number.isRequired,
    activeCount: PropTypes.number.isRequired,
    filter: PropTypes.string.isRequired,
    onClearCompleted: PropTypes.func.isRequired,
    onShow: PropTypes.func.isRequired
  }

  renderTodoCount() {
    const { activeCount } = this.props
    const itemWord = activeCount === 1 ? 'item' : 'items'

    return (
      <span className="todo-count">
      <strong>{activeCount || 'No'}</strong> {itemWord}
left
    </span>
    )
  }

  renderFilterLink(filter) {
    const title = FILTER_TITLES[filter]
    const { filter: selectedFilter, onShow } = this.props

    return (
      <a className={classnames({ selected: filter ===
selectedFilter })}
      style={{ cursor: 'pointer' }}
      onClick={() => onShow(filter)}>
      {title}
  </a>
    )
  }

  renderClearButton() {
    const { completedCount, onClearCompleted } =
this.props
    if (completedCount > 0) {
      return (
        <button className="clear-completed"
      onClick={onClearCompleted} >
        Clear completed
      </button>
    )
    }
  }

  render() {
    return (
      <footer className="footer">
      {this.renderTodoCount()}
  <ul className="filters">
      {[ SHOW_ALL, SHOW_ACTIVE, SHOW_COMPLETED
].map(filter =>
    <li key={filter}>
      {this.renderFilterLink(filter)}
  </li>
  )}
  </ul>
    {this.renderClearButton()}
  </footer>
    )
  }
}
```

```
export default class TodoItem extends Component {
  static propTypes = {
    todo: PropTypes.object.isRequired,
    editTodo: PropTypes.func.isRequired,
    deleteTodo: PropTypes.func.isRequired,
    completeTodo: PropTypes.func.isRequired
  }

  state = {
    editing: false
  }

  handleDoubleClick = () => {
    this.setState({ editing: true })
  }

  handleSave = (id, text) => {
    if (text.length === 0) {
      this.props.deleteTodo(id)
    } else {
      this.props.editTodo(id, text)
    }
    this.setState({ editing: false })
  }

  render() {
    const { todo, completeTodo, deleteTodo } = this.props

    let element
    if (this.state.editing) {
      element = (
        <TodoTextInput text={todo.text}
      editing={this.state.editing}
      onSave={(text) => this.handleSave(todo.id, text)} />
    )
    } else {
      element = (
        <div className="view">
        <input className="toggle"
      type="checkbox"
      checked={todo.completed}
      onChange={() => completeTodo(todo.id)} />
    <label onDoubleClick={this.handleDoubleClick}>
      {todo.text}
    </label>
      <button className="destroy"
      onClick={() => deleteTodo(todo.id)} />
    </div>
    )
    }

    return (
      <li className={classnames({
      completed: todo.completed,
      editing: this.state.editing
})}>
      {element}
    </li>
    )
  }
}

export default class Header extends Component {
  static propTypes = {
    addTodo: PropTypes.func.isRequired
  }

  handleSave = text => {
    if (text.length !== 0) {
      this.props.addTodo(text)
    }
  }

  render() {
    return (
      <header className="header">
      <h1>todos</h1>
      <TodoTextInput newTodo
    onSave={this.handleSave}
    placeholder="What needs to be done?" />
      </header>
    )
```

```
export default class MainSection extends Component {
  static propTypes = {
    todos: PropTypes.array.isRequired,
    actions: PropTypes.object.isRequired
  }

  state = { filter: SHOW_ALL }

  handleClearCompleted = () => {
    this.props.actions.clearCompleted()
  }

  handleShow = filter => {
    this.setState({ filter })
  }

  renderToggleAll(completedCount) {
    const { todos, actions } = this.props
    if (todos.length > 0) {
      return (
        <input className="toggle-all"
      type="checkbox"
      checked={completedCount === todos.length}
      onChange={actions.completeAll} />
    )
    }
  }

  renderFooter(completedCount) {
    const { todos } = this.props
    const { filter } = this.state
    const activeCount = todos.length - completedCount

    if (todos.length) {
      return (
        <Footer completedCount={completedCount}
      activeCount={activeCount}
      filter={filter}
      onClearCompleted={this.handleClearCompleted}
      onShow={this.handleShow} />
    )
    }
  }

  render() {
    const { todos, actions } = this.props
    const { filter } = this.state

    const filteredTodos =
todos.filter(TODO_FILTERS[filter])
    const completedCount = todos.reduce((count, todo) =>
      todo.completed ? count + 1 : count,
      0
    )

    return (
      <section className="main">
      {this.renderToggleAll(completedCount)}
  <ul className="todo-list">
      {filteredTodos.map(todo =>
      <TodoItem key={todo.id} todo={todo} {...actions} />
)}
  </ul>
    {this.renderFooter(completedCount)}
  </section>
    )
  }
```

Todo.pix

```
entity TodoList{ }

entity Todo {
  task     : String
  finished : Boolean
}

relation TodoList.todos * <-> 1 Todo.list
```

Todo.pix

```
component TodoItem(todo: Todo){
  li {
    @BooleanInput(todo.finished)
    @FocusStringInput(todo.task)
  }
}
```

Reset

todo-example

- [ ] Incremental Rendering

Todo.pix

```
component TodoItem(todo: Todo){
  li {
    @BooleanInput(todo.finished)
    @FocusStringInput(todo.task)
    button[onClick=removeTodo()]
  }

  action removeTodo(){
    todo.list { todos = todos \ todo }
  }
}
```

Reset

todo-example

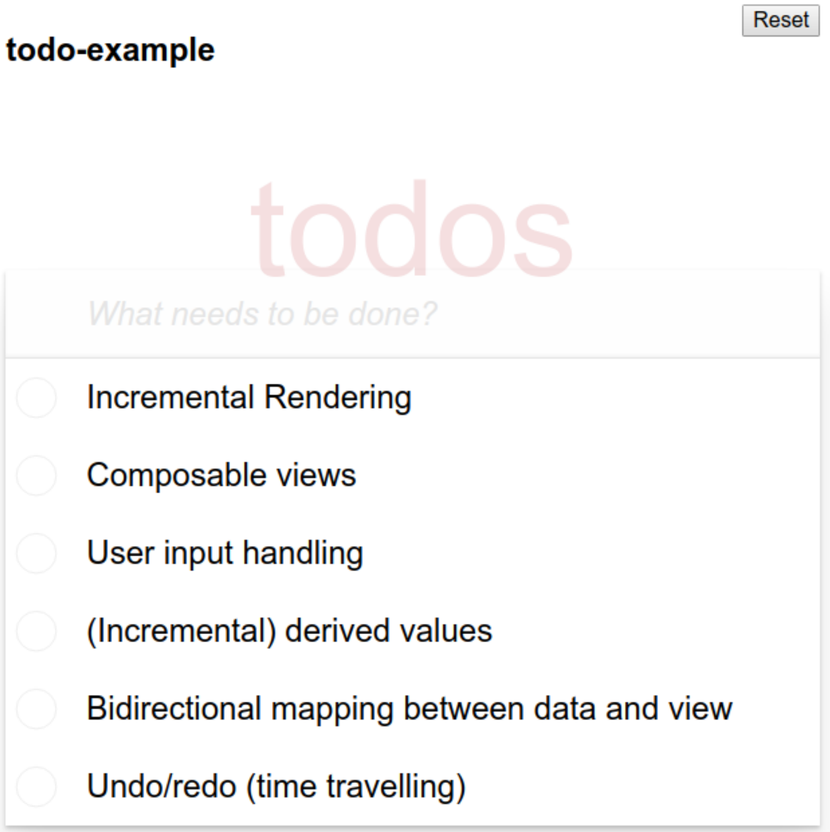
Incremental Rendering

Todo.pix

```
entity TodoList {
  input  : String
}

component TodoList(list: TodoList) {
  h1 { "Todos" }
  @StringInput(list.input)
  ul {
    for(todo in list.todos) @TodoItem(todo)
  }
}
```

Todo.pix

```
entity TodoList {
  input  : String
}

component TodoList(list: TodoList) {
  h1 { "Todos" }
  @StringInput(list.input, addTodo)
  ul {
    for(todo in list.todos) @TodoItem(todo)
  }

  action addTodo(task: String){
    t: Todo {
      list = list
      task = task
    }
    list { input = "" }
  }
}
```



Todo.pix

```
entity TodoList {
  input  : String
  filter : String
}

component FilterType(name: String, list: TodoList) {
  li {
    a[onClick=setFilter()] { name }
  }
  action setFilter(){
    list { filter = name }
  }
}

component TodoFilters(list: TodoList) {
  ul{
    @FilterType("All", list)
    @FilterType("Completed", list)
    @FilterType("Not Completed", list)
  }
}
```

Todo.pix

```
relation TodoList.finishedTodos =
  todos.filter(todo.finished) <-> Todo

relation TodoList.visibleTodos =
  switch {
    case filter == "All" => todos
    case filter == "Completed" => finishedTodos
    case filter == "Not Completed" =>
      todos \ finishedTodos
  } <-> Todo


for(todo in list.visibleTodos) @TodoItem(todo)
```

Todo.pix

```
allFinished : Boolean = conj(todos.finished)

action toggleAll(){
  list.todos {
    finished = !app.allFinished
  }
}
```

Todo.pix

```
todosLeft  : Int    = (todos \ finishedTodos).count()
itemPlural : String =
  if(todosLeft == 1)
    "item"
  else
    "items"

span {
  "${list.todosLeft} ${list.itemPlural} left"
}
```

Todo.pix

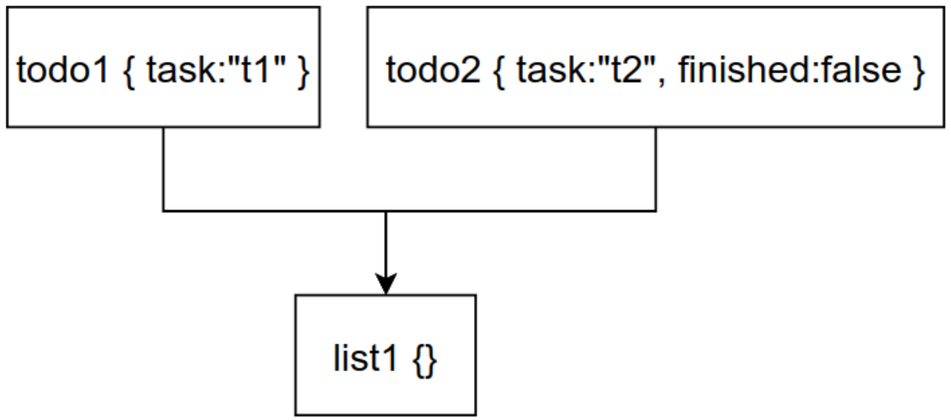
```
action clearCompleted(){
  list { todos = todos \ finishedTodos }
}

if(list.finishedTodos.count() > 0)
  a[onClick=clearCompleted()]{ "Clear completed" }
```

Model

```
entity Todo {
  task     : String
  finished : Boolean = false (default)
  pretty   : String = task + "!"
}
entity TodoList {}

relation Todo.list 1 <-> * TodoList.todos
```

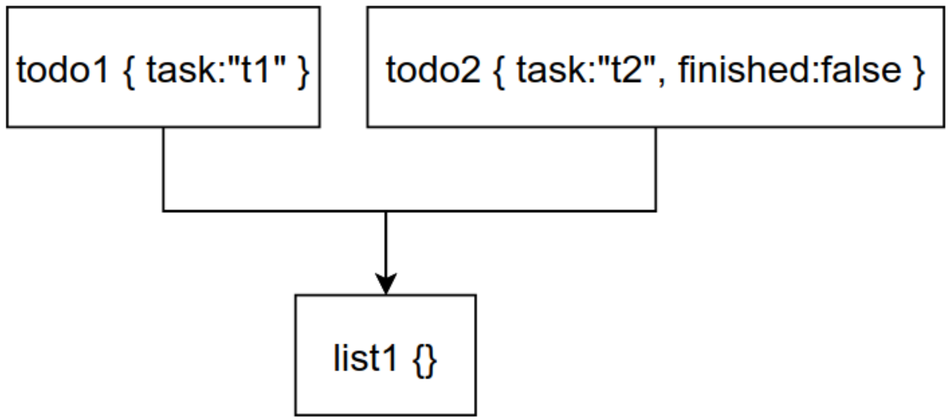
Model

```
entity Todo {
  task     : String
  finished : Boolean = false (default)
  pretty   : String = task + "!"
}
entity TodoList {}

relation Todo.list 1 <-> * TodoList.todos
```



Model

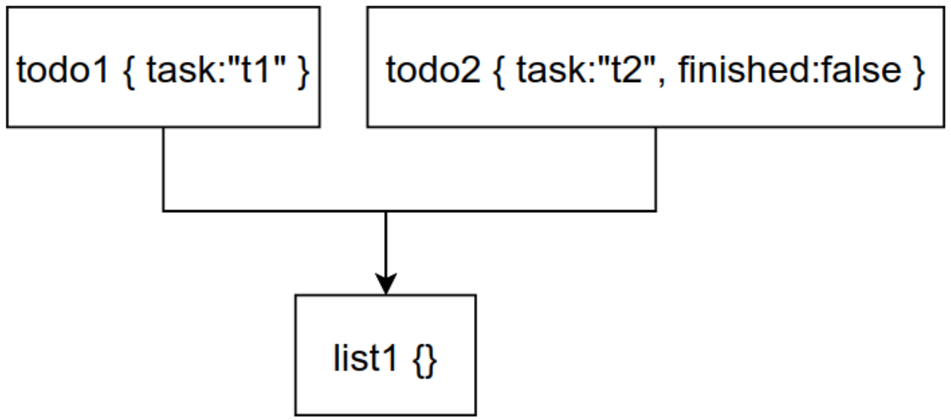
```
entity Todo {
  task     : String
  finished : Boolean = false (default)
  pretty   : String = task + "!"
}
entity TodoList {}

relation Todo.list 1 <-> * TodoList.todos
```




```
Todo:
  todo1 => {id: todo1, task: "t1"}
  todo2 => {id: todo2, task: "t2",
                finished: true}

Todo.finished:
  todo1 => false

Todo.pretty:
  todo1 => "t1!"
  todo2 => "t2!"

TodoList:
  list1 => {id: list1}

Todo.list:
  foo1 => list1
  foo2 => list1

TodoList.todos:
  bar1 => [todo1, todo2]
```

Getter

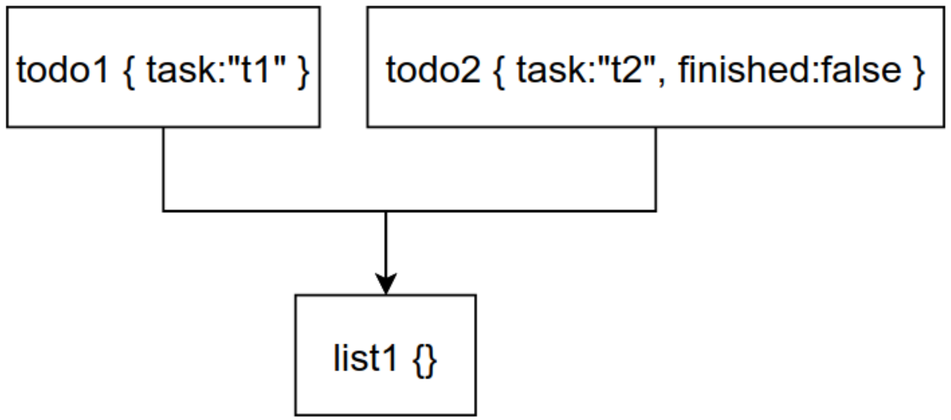
```
entity Todo {
  task     : String
  finished : Boolean = false (default)
  pretty   : String = task + "!"
}
entity TodoList {}

relation Todo.list 1 <-> * TodoList.todos
```




```
function getTodo_task(state, id) {
  return state.Todo.get(id).task;
}
```

Calculate

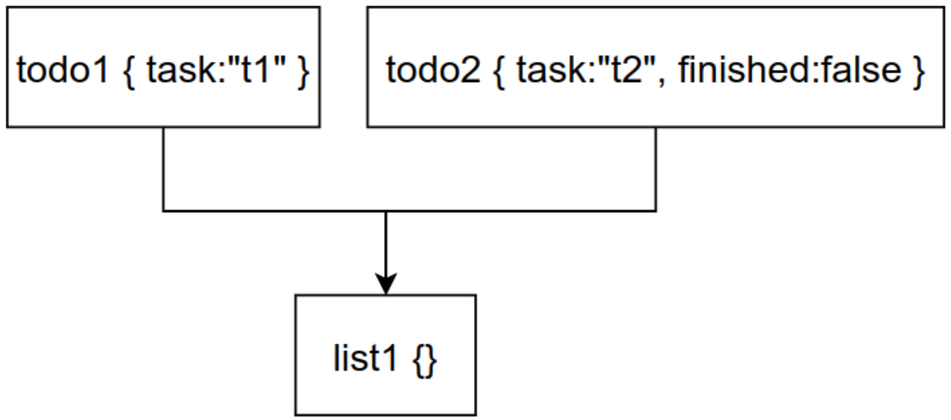
```
entity Todo {
  task     : String
  finished : Boolean = false (default)
  pretty   : String = task + "!"
}
entity TodoList {}

relation Todo.list 1 <-> * TodoList.todos
```




```
function calculateTodo_pretty(state, id){
  var value = getTodo_pretty(state, id);
  if(value === undefined){
    //calculate value
  }
  return {
    state: state
  , value: value
  }
}
```

Setter

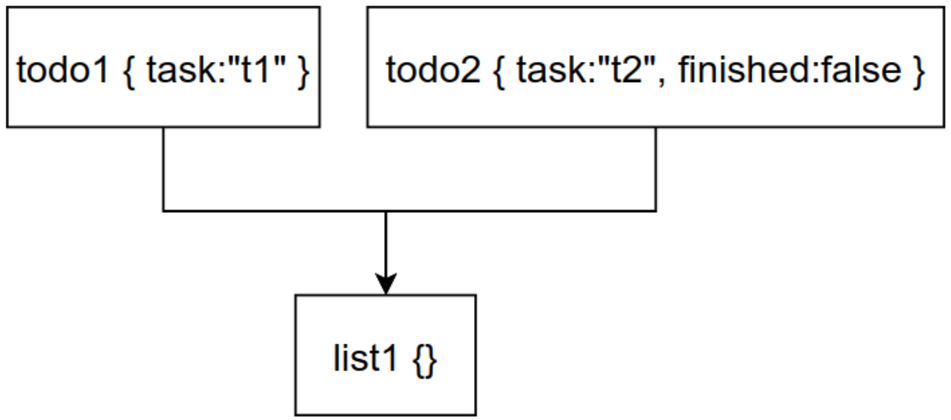
```
entity Todo {
  task     : String
  finished : Boolean = false (default)
  pretty   : String = task + "!"
}
entity TodoList {}

relation Todo.list 1 <-> * TodoList.todos
```




```
function setTodo_task(state, id, value) {
  state = state.Todo.update('task', id, value);
  state = invalidateTodo_task(state, id);
  return state;
}
```

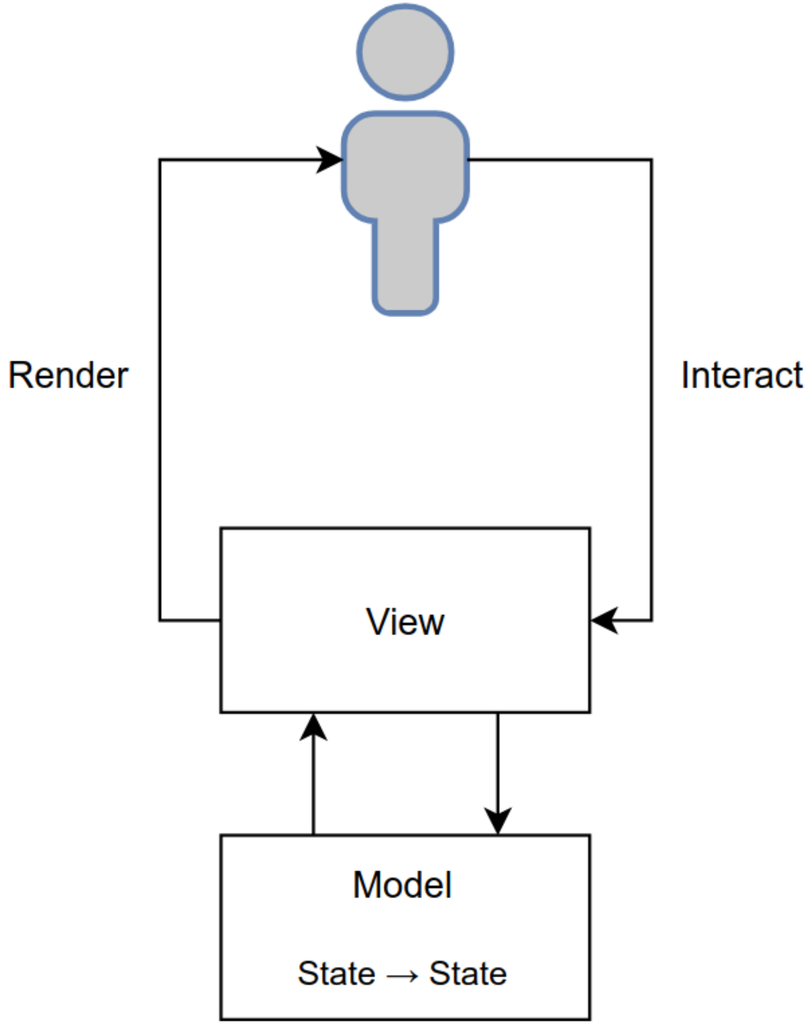
Invalidate

```
entity Todo {
  task     : String
  finished : Boolean = false (default)
  pretty   : String = task + "!"
}
entity TodoList {}

relation Todo.list 1 <-> * TodoList.todos
```

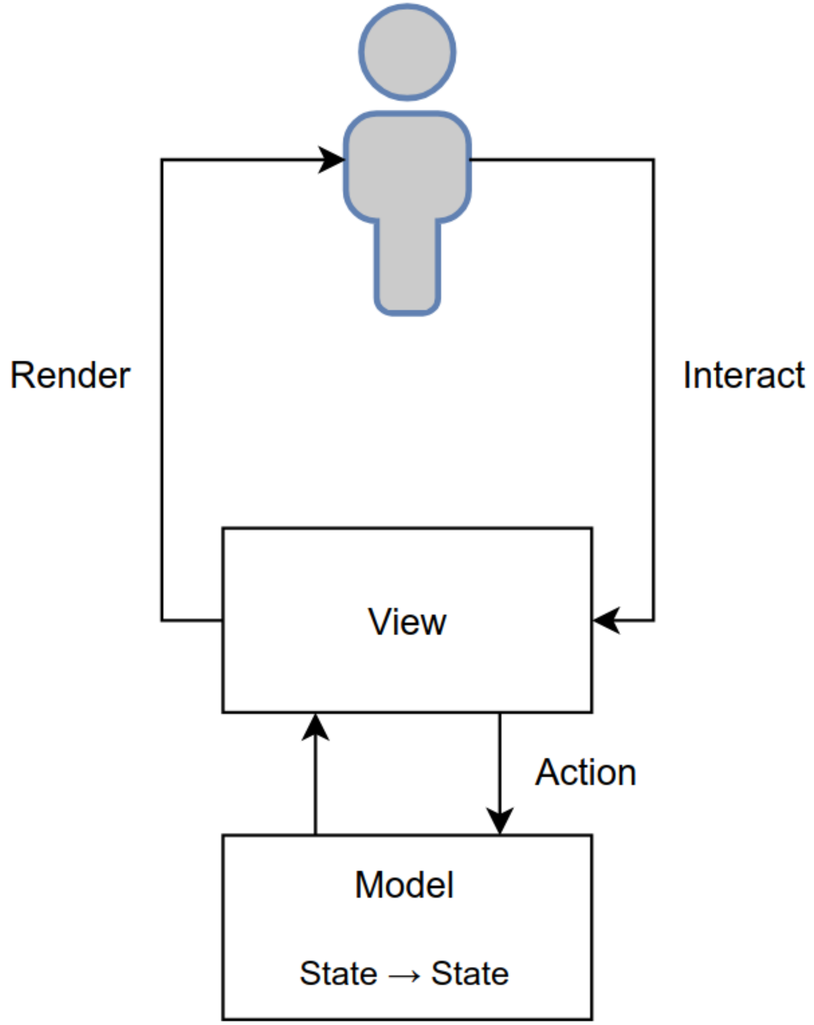



```
function invalidateTodo_task(state, id){
  state = invalidateTodo_pretty(state, id);
  return state;
}

function invalidateTodo_pretty(state, id) {
  state = state.Todo_pretty.remove(id);
  return state;
}
```

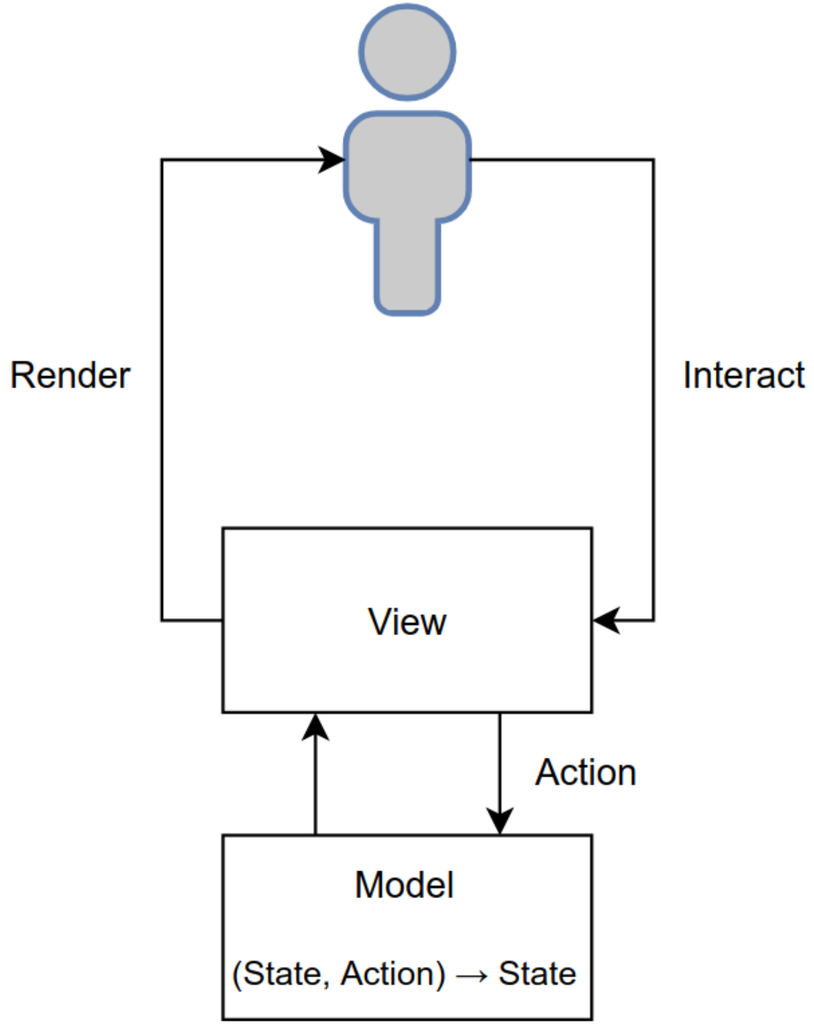
Actions



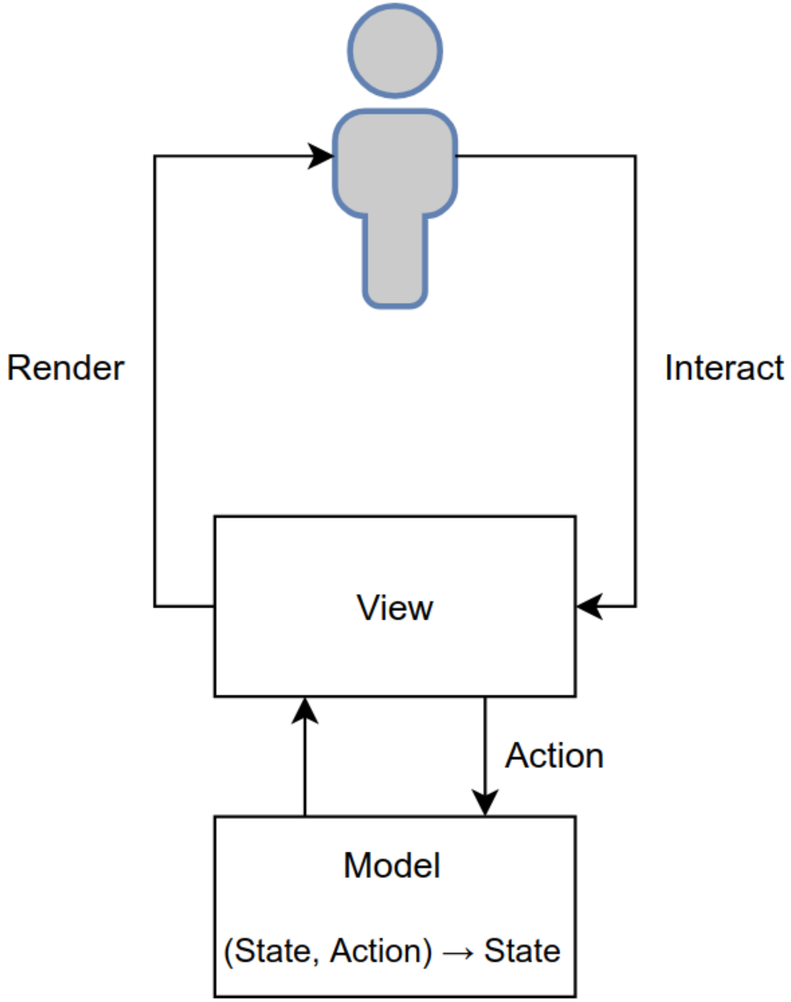
Actions



Actions

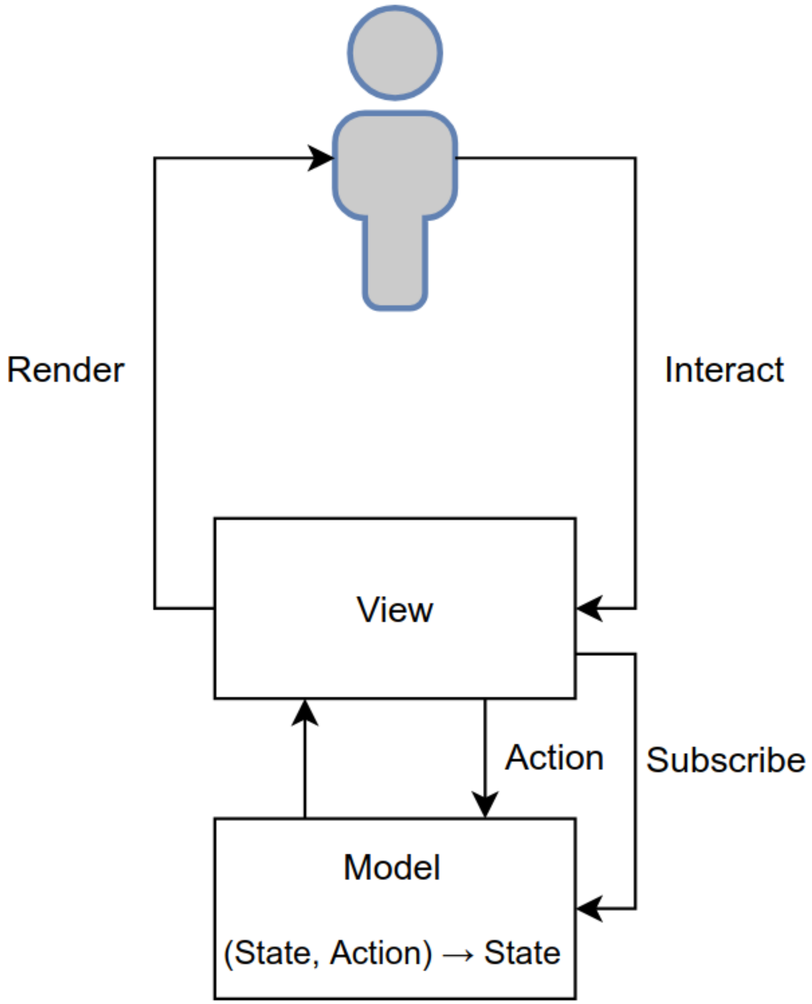


Store



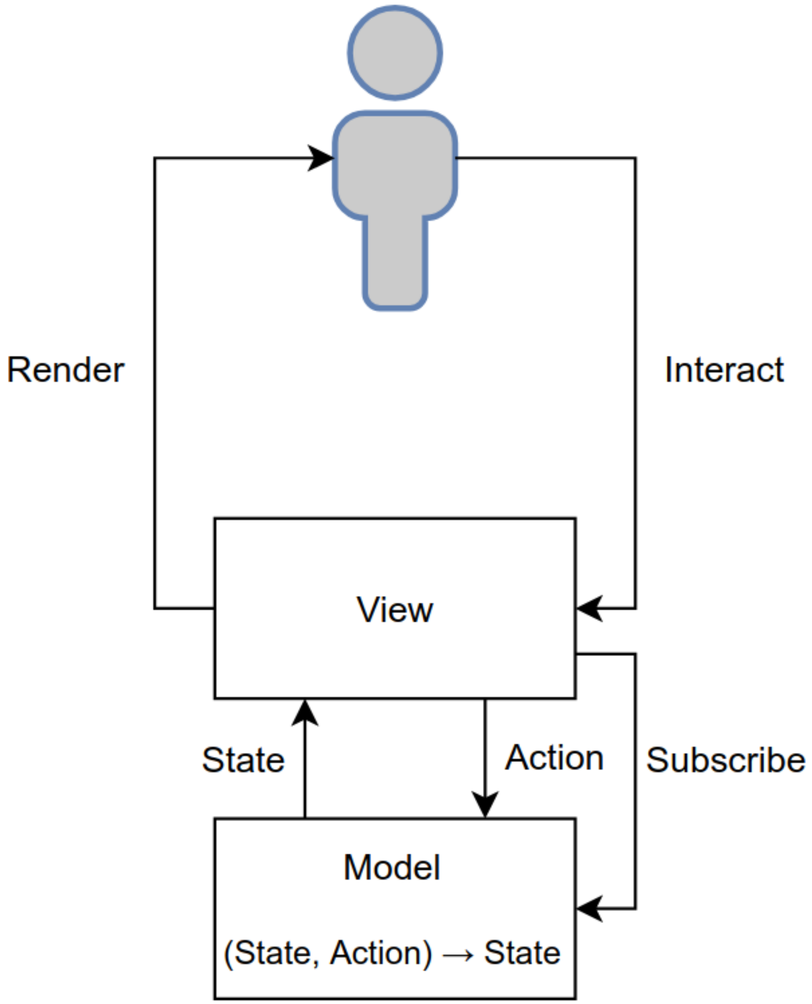

```
type ModelFold<S,A> = (S, A) => S

interface Store<S, A> {
  getState    : S
  subscribe   : (S => ()) => ()
  dispatch    : A => ()
}
```

Store




```
type ModelFold<S,A> = (S, A) => S

interface Store<S, A> {
  getState    : S
  subscribe   : (S => ()) => ()
  dispatch    : A => ()
}
```

Store




```
type ModelFold<S,A> = (S, A) => S

interface Store<S, A> {
  getState    : S
  subscribe   : (S => ()) => ()
  dispatch    : A => ()
}
```

Action types in PixieDust

```
entity Num{
  value : Int = 0 (default)
}
entity Add{
  value : Int = lhs.value + rhs.value
}
relation Add.lhs 1 <-> Num
relation Add.rhs 1 <-> Num

component Add(add: Add) {
  button[onClick=reset()]{ "Reset" }
  @IntInput(add.lhs.value) "+"
  @IntInput(add.rhs.value) "="
  add.value

  action reset(){
    add {
      lhs { value = 0 }
      rhs { value = 0 }
    }
  }
}
```

Reset

add-example

Reset

0 +

0 =

0

Action types in PixieDust

```
entity Num{
  value : Int = 0 (default)
}
entity Add{
  value : Int = lhs.value + rhs.value
}
relation Add.lhs 1 <-> Num
relation Add.rhs 1 <-> Num

component Add(add: Add) {
  button[onClick=reset()]{ "Reset" }
  @IntInput(add.lhs.value) "+"
  @IntInput(add.rhs.value) "="
  add.value

  action reset(){
    add {
      lhs { value = 0 }
      rhs { value = 0 }
    }
  }
}
```

Reset

add-example

Reset

0 +

0 =

0

Update field (bidirectional mappings)

```
{"type": "setEntity_field", "id": someId, "value": someValue}
```

Component actions

```
{"type": "Component_action", "props": [...], "args": [...]}
```

Cache updates while rendering

```
{"type": "cacheUpdate[Component]", "updatedState": state}
```

Lazy rendering

```
entity Student {
    name : String
  }

entity Course {
  name : String
  avgGrade: Float? = avg(submissions.grade2)
}

entity Submission {
  grade: Float?
  grade2 : Float? =  if(pass) grade
  pass: Boolean = grade > 5.5 <+ false
}

relation Course.submissions *
   <-> 1 Submission.course
relation Submission.student 1
   <-> * Student.submissions
```

Reset

grades-example

Math

Math

Average grade: No grades

Student	Grade
Alice	
Bob	
Charlie	

Component

```
type Component<Args, S> =
    (Args, S) => (View, S)
```

Component

```
type Component<Args, S> =
    (Args, S) => (View, S)


component FooComponent(foo: Foo) {
  foo.attr
  foo.der
  foo.dv
}
```

Component

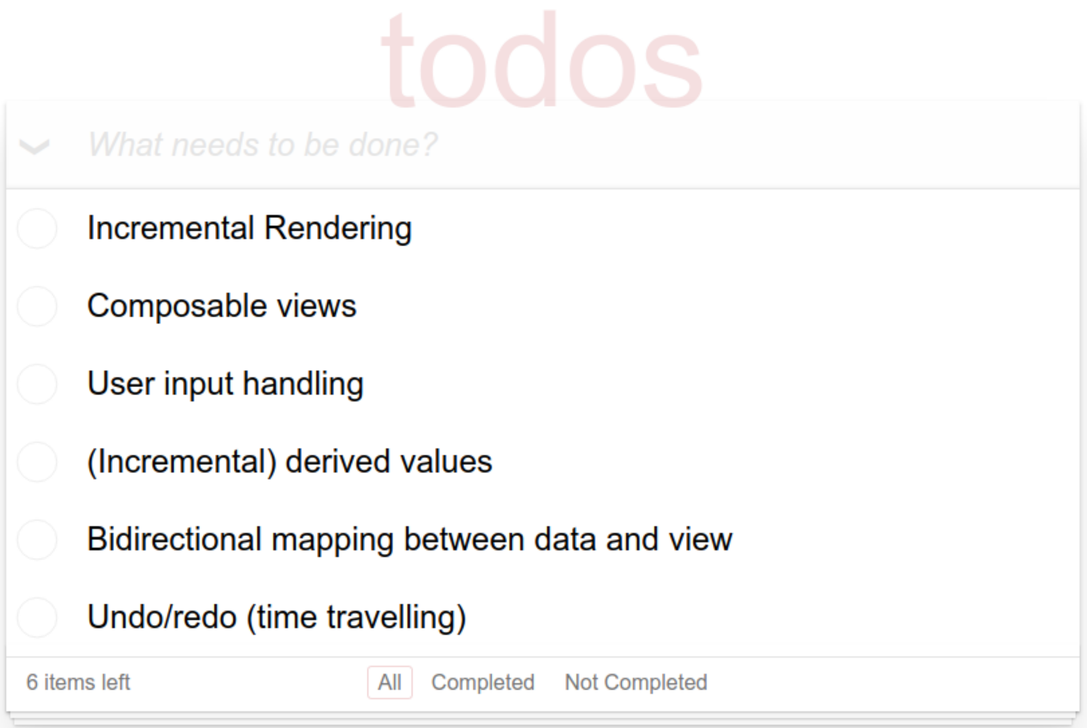
```
type Component<Args, S> =
    (Args, S) => (View, S)


component FooComponent(foo: Foo) {
  foo.attr
  foo.der
  foo.dv
}


interface ReactComponent<Props> {
    componentWillReceiveProps : Props => ()
    render : Props => ReactElement
}
```

Component

```
type Component<Args, S> =
    (Args, S) => (View, S)


component FooComponent(foo: Foo) {
  foo.attr
  foo.der
  foo.dv
}


interface ReactComponent<Props> {
    componentWillReceiveProps : Props => ()
    render : Props => ReactElement
}
```

```
function lift(component){
  return class extends ReactComponent{
    var materialized: ReactElement;

    componentWillReceiveProps(props){
      this.materialize(props);
    }

    materialize(props) {
      var store = this.context.store;
      var state = store.getState();
      var result = component(props, state);
      this.materialized = result.view;
      if(result.state !== state)
        store.dispatch(
          {newState: result.state}
        );
    }

    render() {
      return this.materialized;
    }
  }
}
```

PixieDust


todos
What needs to be done?
Incremental Rendering
Composable views
User input handling
(Incremental) derived values
Bidirectional mapping between data and view
Undo/redo (time travelling)
6 items left
All
Completed
Not Completed

Slideshow

```
entity SlideShow{
  totalSlides: Int = current.slideNumber + current.slidesLeft <+ 0
}

entity Slide {
  title: String
  content: View?

  slideNumber : Int = previous.slideNumber + 1 <+ 1
  slidesLeft: Int = 1 + next.slidesLeft <+ 0
}

relation Slide.next ? <-> ? Slide.previous
relation Slide.slideshow 1 <-> * SlideShow.slides
relation SlideShow.current ? <-> ? Slide.inverseCurrent
```

Slideshow

```
entity SlideShow{
  totalSlides: Int = current.slideNumber + current.slidesLeft <+ 0
}

entity Slide {
  title: String
  content: View?

  slideNumber : Int = previous.slideNumber + 1 <+ 1
  slidesLeft: Int = 1 + next.slidesLeft <+ 0
}

relation Slide.next ? <-> ? Slide.previous
relation Slide.slideshow 1 <-> * SlideShow.slides
relation SlideShow.current ? <-> ? Slide.inverseCurrent

relation Slide.allNext0 = this ++ next.allNext0 <+ this <-> Slide.inverseAllNext0
relation Slide.allPrevious0 = previous.allPrevious0 ++ this <+ this <-> Slide.inverseAllPrevious0

relation Slide.allPrevious = allPrevious0 \ this <-> Slide.inverseAllPrevious
relation Slide.allNext =  allNext0 \ this <-> Slide.inverseAllNext
```

Slideshow

```
component Slide(slide: Slide) {
  h1 { slide.title }
  slide.content
  @SlideFooter(slide)
}
```

Slideshow

```
Component Footer(slide: Slide) {
  action setCurrent(slide: Slide?) { slide.slideshow { current = slide }}
  action nextSlide(){ slide.slideshow { current = current.next }}
  action previousSlide(){ slide.slideshow { current = current.previous }}

  @KeyboardListener(
    slide.nextSlide
  , slide.previousSlide
  )

  footer {
    span { slide.slideNumber "/" slide.slideshow.totalSlides }
    div {
      button[onClick=previousSlide()] { "<" }
      for(s in slide.allPrevious)
        button[onClick=setCurrent(s) { s.slideNumber }
      span { slide.slideNumber }
      for(slide in slide.allNext)
        button[onClick=setCurrent(s){ s.slideNumber}
      button[onClick=nextSlide()] { ">" }
    }
  }
}
```

Slideshow

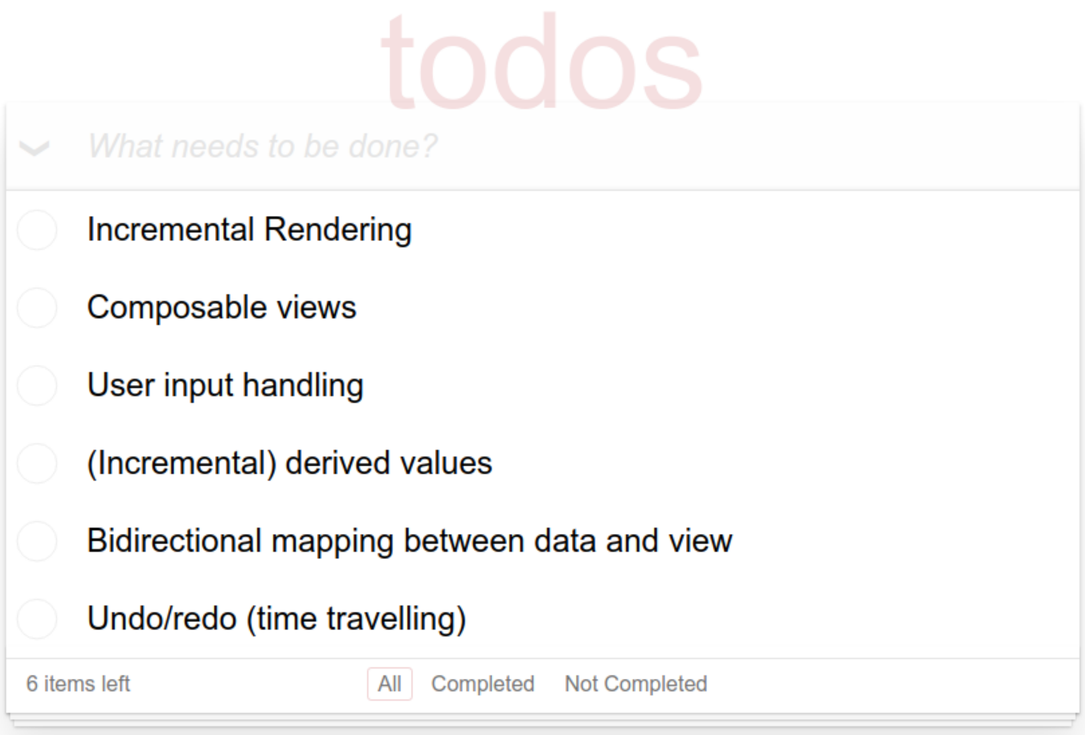
```
intro: Slide {
  slideshow = slideshow
  title = "PixieDust: Declarative incremental user interfaces for IceDust"
  content = @Center {
    @FixedWidth(800) {
      @TodoWithData()
    }
  }
}
slideshow { current = intro }

ui : Slide {
  slideshow = slideshow
  previous = intro
  title = "UI Pattern"
  content = @TwoColumn(
    @FixedWidthImage("/images/ui.svg", 600)
  , no value
  )
}

uiExtended : Slide {
  slideshow = slideshow
  previous = ui
  title = "UI Pattern"
```

Slideshow

Reset

slides-program

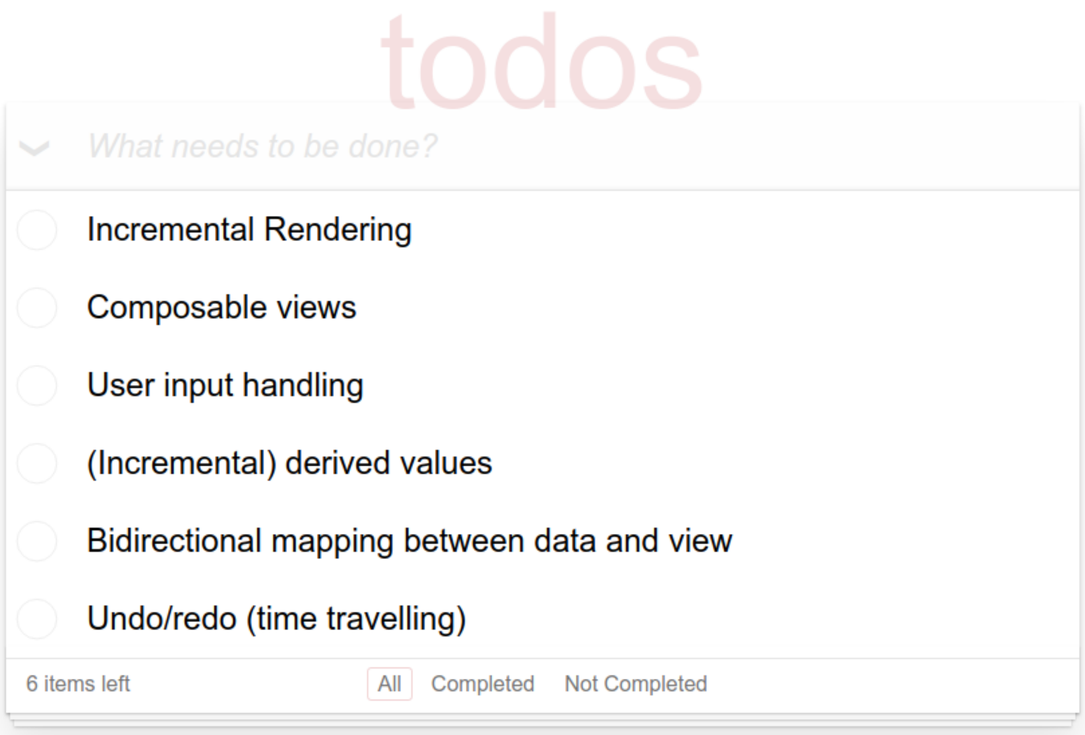
PixieDust: Declarative incremental user interfaces for IceDust



Reset

slides-program

PixieDust: Declarative incremental user interfaces for IceDust



Grades

```
module grades

config
  backend: PixieDust
  target: webpack

imports
  pixiedust/components/native/inputs {
    component OptFloatInput(ref value : Float?)
  }

  ../../components/Select {
    component Select(labels: String*, choice: Course*, ref selection: Course?)
  }

model
  entity App {

  }

  entity Student {
    name : String
  }

  entity Course {
    name : String
    avgGrade: Float? =  avg(submissions.grade2)
  }

  entity Submission {
    grade: Float?
    grade2 : Float? =  if(pass) grade

    pass: Boolean = grade > 5.5 <+ false
  }

  relation Course.submissions * <-> 1 Submission.course
  relation Submission.student 1 <-> * Student.submissions

  relation App.courses * <-> 1 Course.app
  relation App.selectedCourse ? <-> ? Course.inverseSelectedCourse

view
  component Course(c: Course) {
    h1{ c.name }
    h2 { "Average grade: " c.avgGrade as String <+ "No grades" }
```

Reset

grades-example

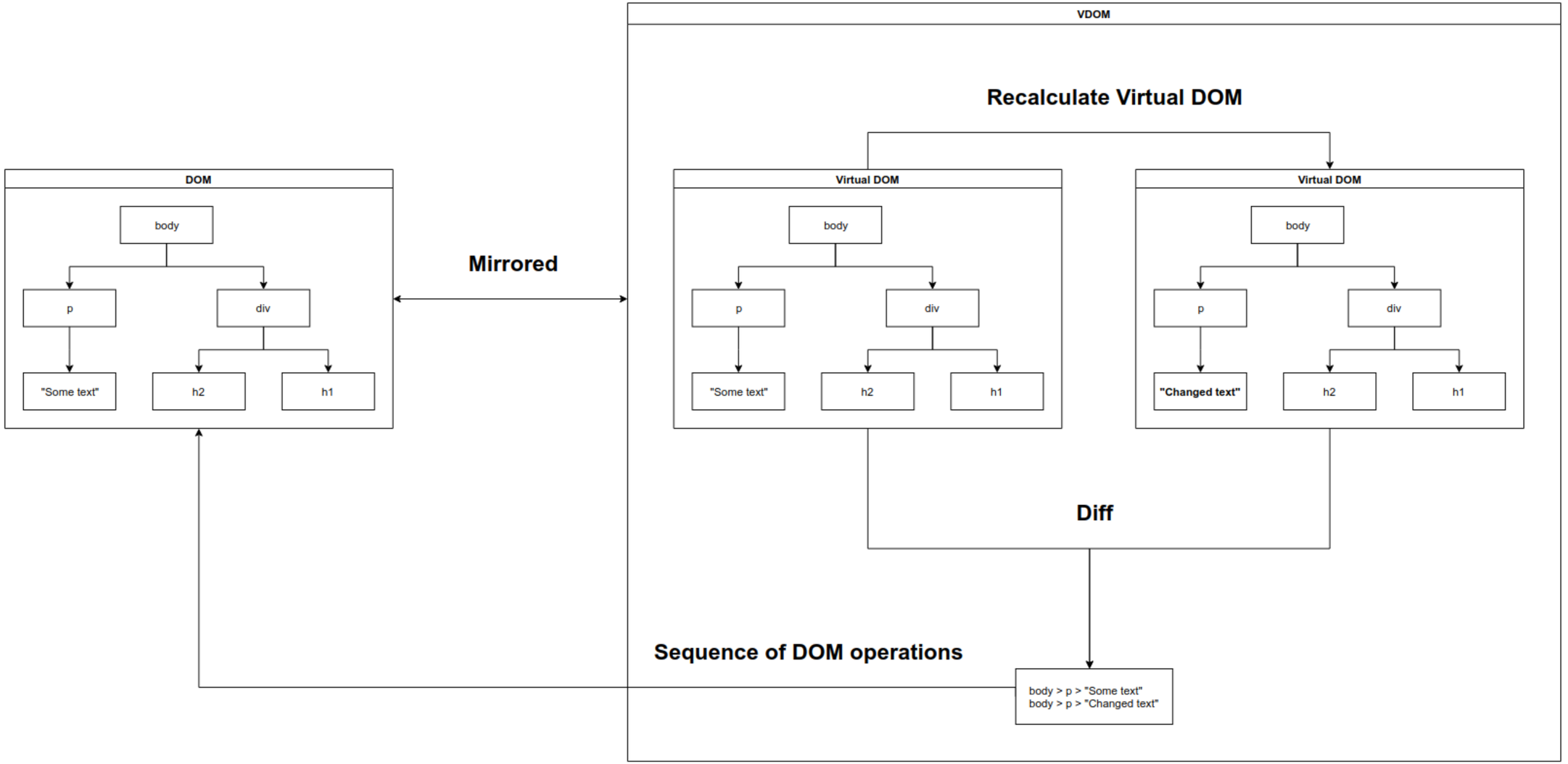
Math

Math

Average grade: No grades

Student	Grade
Alice	
Bob	
Charlie	

Virtual DOM


VDOM
Recalculate Virtual DOM
DOM
body
p
div
"Some text"
h2
h1
Mirrored
Virtual DOM
body
p
div
"Some text"
h2
h1
Virtual DOM
body
p
div
"Changed text"
h2
h1
Diff
Sequence of DOM operations
body > p > "Some text"
body > p > "Changed text"