The Async Nightmare

Toni Petrina
@to_pe

DEMO

Callback based

```
getStuff('http://fromhere.com', result => {

}, error => {

}
```

Promise based

```
getStuff('http://fromhere.com')
    .then(result => {

    })
    .catch(error => {

    })
```

Async-await

```
async process = () => {
    try {
        let result = await getStuff('http://fromhere.com')
    }
    catch(error){}
}
```

So where is the problem?

Problems

- Is the action done?
- Not done, any chance of knowing when it is being done?
- Any errors?
- What kind of results can be acquired?
- What about timeouts? Ambient cancellations? User cancellations?

DEMO

```
class Demo1 extends Component {
  state = {
    data: []
  }

  handleClick = () => {
    f.simpleDelayedList().then(r => this.setState({ data: r }))
  }

  render() {
    return (
      <div>
        <Button className="primary" role="button" onClick={this.handleClick}>
          Async me!
        </Button>

        <List divided relaxed>
          {this.state.data.map((item, index) => (
            <List.Item key={index}>
              Too lazy to random generate so I got {item}
            </List.Item>
          ))}
        </List>
      </div>
    )
  }
}
```

```
class Demo5 extends Component {
  state = {
    isFetching: false,
    isFailed: false,
    isFetched: false,
    data: []
  }

  handleClick = () => {
    this.setState({ isFetching: true, isFailed: false })
    f
      .randomDo(true)
      .then(r => this.setState({ data: r, isFetched: true, isFetching: false }))
      .catch(e => this.setState({ isFetching: false, isFailed: true }))
  }

  render() {
    return (
      <div>
        <Button
          className="primary"
          role="button"
          onClick={this.handleClick}
          disabled={this.state.isFetching}>
          Guess the future!
        </Button>

        <div className="clearfix" />

        {this.state.isFetching ? (
          "Loading..."
        ) : !this.state.isFetched ? (
          <p />
        ) : this.state.isFailed ? (
          "Error!"
        ) : this.state.data.length === 0 ? (
          "No results!"
        ) : (
          <List divided relaxed>
            {this.state.data.map((item, index) => (
              <List.Item key={index}>Random number: {item}</List.Item>
            ))}
          </List>
        )}
      </div>
    )
  }
}
```

Disable button

```
<Button
  className="primary"
  role="button"
  onClick={this.handleClick}
  disabled={this.state.isFetching}>
  Guess the future!
</Button>
```

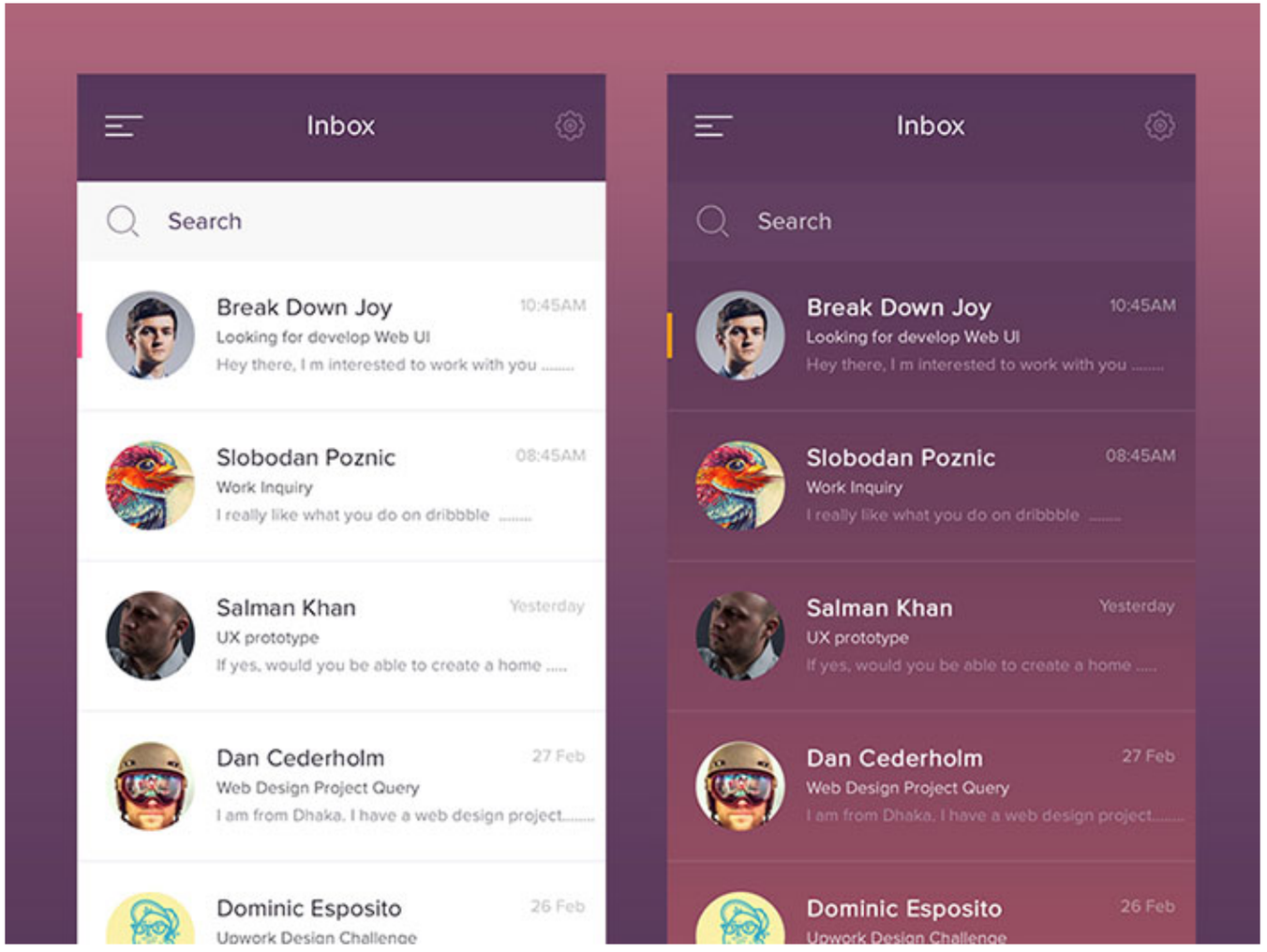
Show progress

```
this.state.isFetching
? “Loading..."
```

Empty states

```
this.state.data.length === 0
? "No results!”
```

Design app states


Inbox
Search
Break Down Joy
10:45AM
Looking for develop Web UI
Hey there, I m interested to work with you .....
Slobodan Poznic
08:45AM
Work Inquiry
I really like what you do on dribbble .....
Salman Khan
Yesterday
UX prototype
If yes, would you be able to create a home .....
Dan Cederholm
27 Feb
Web Design Project Query
I am from Dhaka. I have a web design project.....
Dominic Esposito
26 Feb
Upwork Design Challenge
Inbox
Search
Break Down Joy
10:45AM
Looking for develop Web UI
Hey there, I m interested to work with you .....
Slobodan Poznic
08:45AM
Work Inquiry
I really like what you do on dribbble .....
Salman Khan
Yesterday
UX prototype
If yes, would you be able to create a home .....
Dan Cederholm
27 Feb
Web Design Project Query
I am from Dhaka. I have a web design project.....
Dominic Esposito
26 Feb
Upwork Design Challenge

```
public class LoginViewModel : ViewModelBase
{
    private string username, password;
    private bool isBusy;

    public string Username
    {
        get { return username; }
        set
        {
            if (Set(ref username, value))
                LoginCommand.RaiseCanExecuteChanged();
        }
    }

    public string Password
    {
        get { return password; }
        set
        {
            if (Set(ref password, value))
                LoginCommand.RaiseCanExecuteChanged();
        }
    }

    public bool IsBusy
    {
        get { return isBusy; }
        set
        {
            if (Set(ref isBusy, value))
            {
                LoginCommand.RaiseCanExecuteChanged();
            }
        }
    }

    public RelayCommand LoginCommand { get; }

    public LoginViewModel(IService service)
    {
        LoginCommand = new RelayCommand(Login, () => !IsBusy &&
                                                     !string.IsNullOrEmpty(Username) &&
                                                     !string.IsNullOrEmpty(Password));
    }

    private async void Login()
    {
        IsBusy = true;

        try
        {
            await Task.Delay(TimeSpan.FromSeconds(3));
        }
        finally
        {
            IsBusy = false;
        }
    }
}
```

```
await _service.LoginAsync(username, password);
```

```cs
public class RegisterViewModel : ViewModelBase
{
    public bool IsRegistering
    {
        get { return Get<bool>(); }
        private set
        {
            Set(value); RegisterCommand.RaiseCanExecuteChanged();
        }
    }
    public bool IsCheckingUsername
    {
        get { return Get<bool>(); }
        private set
        {
            Set(value);
            RegisterCommand.RaiseCanExecuteChanged();
        }
    }
    public bool IsUsernameAvailable
    {
        get { return Get<bool>(); }
        private set
        {
            Set(value);
            RegisterCommand.RaiseCanExecuteChanged();
        }
    }
    public bool IsUsernameTaken
    {
        get { return Get<bool>(); }
        private set
        {
            Set(value);
            RegisterCommand.RaiseCanExecuteChanged();
        }
    }
    public bool CanRegister
    {
        get
        {
            if (string.IsNullOrEmpty(Username) ||
                string.IsNullOrEmpty(Password))
                return false;

            if (IsRegistering || IsCheckingUsername || IsUsernameTaken)
                return false;

            return true;
        }
    }

    public string Username
    {
        get { return Get<string>(); }
        set
        {
            Set(value);
            RegisterCommand.RaiseCanExecuteChanged();
            CheckUsernameAvailable();
        }
    }

    public string Password
    {
        get { return Get<string>(); }
        set
        {
            Set(value);
            RegisterCommand.RaiseCanExecuteChanged();
        }
    }

    public string UsernameError { get { return Get<string>(); } set { Set(value); } }

    public RelayCommand RegisterCommand { get; }

    public RegisterViewModel()
    {
        RegisterCommand = new RelayCommand(Register, () => CanRegister);
    }

    private CancellationTokenSource checkingCts;

    private async void CheckUsernameAvailable()
    {
        checkingCts?.Cancel();
        checkingCts = new CancellationTokenSource();

        await CheckUsernameAvailable(Username, checkingCts.Token);
        checkingCts = null;
    }

    private async Task CheckUsernameAvailable(string username, CancellationToken token)
    {
        UsernameError = string.Empty;
        await Task.Delay(TimeSpan.FromMilliseconds(250));
        if (token.IsCancellationRequested)
            return;

        IsCheckingUsername = true;

        // simulate actual checking
        await Task.Delay(TimeSpan.FromSeconds(1));

        IsCheckingUsername = false;

        if (token.IsCancellationRequested)
            return;

        if (username.Contains("a"))
        {
            IsUsernameAvailable = false;
            IsUsernameTaken = true;
            UsernameError = $"username {username} taken";
        }
        else
        {
            IsUsernameAvailable = true;
            IsUsernameTaken = false;
        }
    }

    private async void Register()
    {
        IsRegistering = true;

        try
        {
            await Task.Delay(TimeSpan.FromSeconds(3));
            MessageBox.Show("registered");
        }
        finally
        {
            IsRegistering = false;
        }
    }
}
```

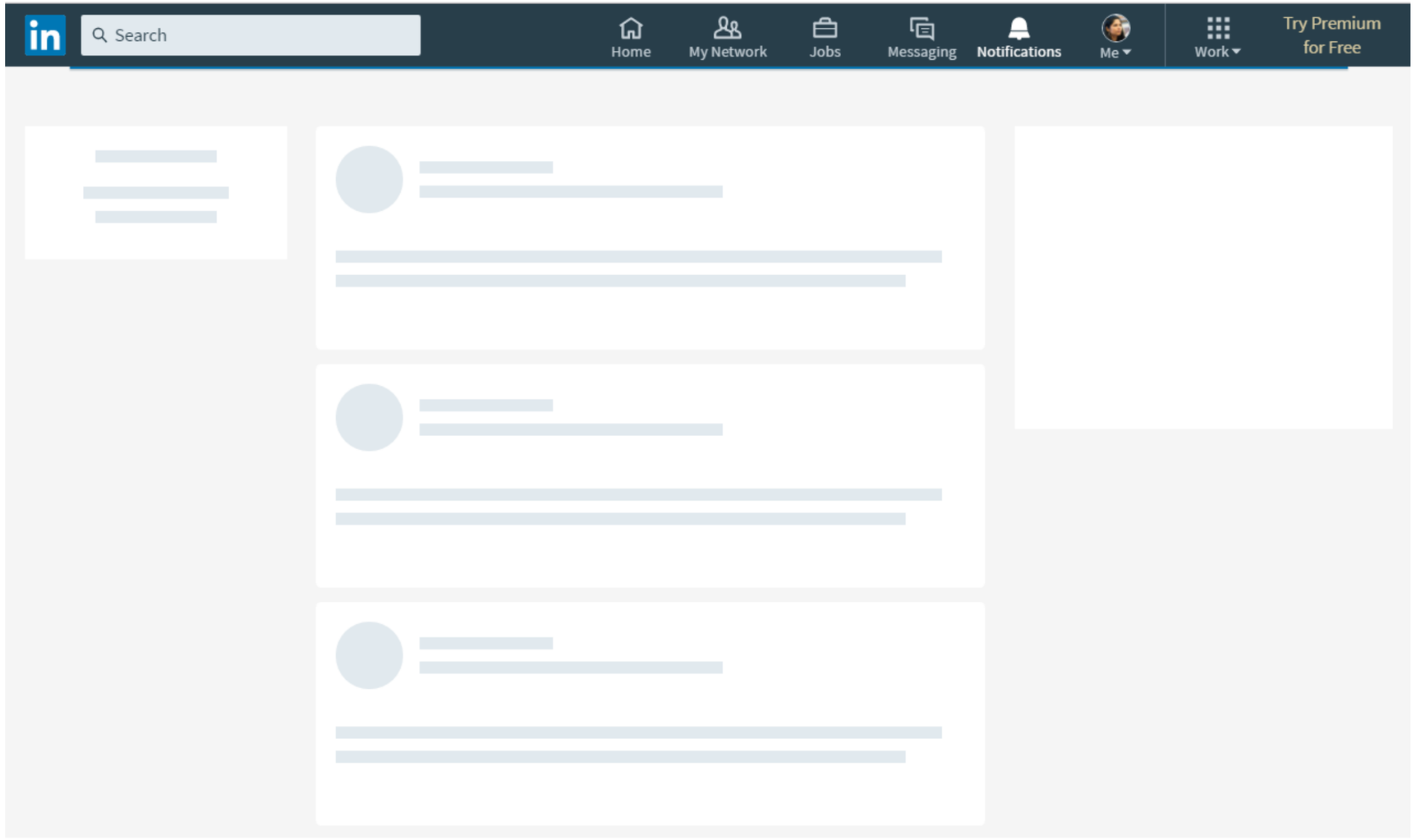
```cs
if (await _service.Register(email, password) == false)
    MessageBox.Show("registered");
```

Progress

- Sometimes you are just waiting…until you are done (e.g. web requests)
- Sometimes you know how much you still have to go (e.g. upload, download, processing)

DEMO

Skeleton screens



“You can’t fire me, I quit!”

–Someone

Canceling a promise

```
this.tokenSource = f.createTokenSource()

// our promise that can be cancelled!
let promise = f.delay(10000)

f
  .withCancel(promise, this.tokenSource.token)
  .then(() => this.setState({ isWorking: false }))
  .catch(() => this.setState({ isWorking: false }))


// cancel!
this.tokenSource.cancel()
```

Native support

- Some libraries offer native cancelations
- Bluebird, axios
- Fetch will get it too

Ambient cancellations

- Ignoring responses can bring tragedies.
- Can application be in inconsistent state if the promise completes too late?
  - Or if it fails?

DEMO

So how complicated can it get?

- The answer is: “it depends”
- No, the answer is: “a lot!”

Additional considerations

- Losing connectivity - how do we verify if started action actually finished
- Optimistic update - assume everything went ok, but what if it didn’t?

“Thanks?”

–Me