Building Explainable User Interfaces

Tommaso Turchi
University of Pisa
Pisa, Italy
research@tommasoturchi.fyi

Alan Dix*
Swansea University
Swansea, Wales, United Kingdom
Cardiff Metropolitan University
Cardiff, Wales, United Kingdom
alanjohndix@gmail.com

Ben Wilson
Swansea University
Swansea, United Kingdom
b.j.m.wilson@swansea.ac.uk

Matt Roach
Swansea University
Swansea, United Kingdom
m.j.roach@swansea.ac.uk

Alessio Malizia†
University of Pisa
Pisa, Italy
alessio.malizia@oldsport.org

Abstract

The notion of explainable user interfaces (XUI) has been proposed as a way to address the growing complexity and unpredictability of many UIs. A key question is whether this aspirational goal can be achieved with reasonable amounts of development effort. This paper takes React as its focus as it is one the most widely used web development frameworks. It shows through practical demonstration that the existing event mechanisms within React can be repurposed with the addition of a small amount of annotation to implement one of the proposed XUI techniques.

CCS Concepts

• **Human-centered computing** → **Interaction design theory, concepts and paradigms**; **Systems and tools for interaction design**.

Keywords

user experience, explainable UI, user interface architecture, design, explainable AI, artificial intelligence

ACM Reference Format:
Tommaso Turchi, Alan Dix, Ben Wilson, Matt Roach, and Alessio Malizia. 2026. Building Explainable User Interfaces. In *Proceedings of the 2026 International Conference on Advanced Visual Interfaces (AVI '26), June 08–12, 2026, Venice, Italy*. ACM, New York, NY, USA, 3 pages. https://doi.org/10.1145/3811427.3811499

1 Introduction

This paper presents a first working implementation of the new concept of an explainable user interface (XUI), and discusses broader architectural issues that arise,

In a "futures track" paper at BCS HCI 2025, we introduced the notion of explainable user interfaces (XUI) [4]. This took the analogy of explainable AI (XAI) [1, 2], but argued that system behaviour in *any* user interface may be equally obscure or surprising irrespective of whether there is any AI involved; we are often left asking, "what just happened?", "what did I do?", "how do I do this again?" This is a paradigm shift from the myth of 'walk up and use', explicitly recognising that systems are rarely usable by everyone without aid. The paper discussed several types of XUI, including an envisionment video [3]) of one of these styles.

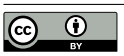
If we accept the *desirability* of XUI, this leaves open the question of how *easy* is it to achieve in practice. That is, we need to be able to augment existing frameworks and not add unreasonable cost during development. This question is addressed in three parts: **(1)** a generic software system is implemented over React, a common web development framework, that can collect suitable live information from any application to enable XUI; **(2)** a generic interface component is provided to present explanations at user controllable levels of detail for any application that uses (1); **(3)** an existing application is annotated using the facilities provided by (1) to demonstrate the feasibility of the techniques and assess the level of effort required.

2 Generic XUI platform

Choice of underlying platform – Many of the most popular web UI development frameworks, collectively known as reactive toolkits, are data-oriented employing a two-way binding between data, including remotely stored data, and HTML templates with special annotations. This is an extension to the observer pattern [5] found in the model–view relationship in MVC [6], but also operates in the reverse direction whereby changes to web fields (such as entering text) and rendered structures (such as reordering a list) are reflected back on the data. The most influential reactive frameworks arose directly out of large-scale web platforms: React by Facebook/Meta, AngularJs by Google and JsRender/JsViews by Miscrosoft, but are all now open source. While driven by practical concerns, reactive frameworks can be viewed through more theoretical UI architecture lenses in terms of their relationship to MVC and also as they share aspects with declarative [10, 12] and model-based development [7, 8, 16, 17].

React was chosen as a base framework for this proof-of-concept demonstrator in part because of its popularity[1], and in part due to the researchers' familiarity both with the framework itself and with the developer discussion forums relating to it.

*Also with Computational Foundry, Swansea University, UK.
†Also with Faculty of Logistics, Molde University College.



AVI '26, Venice, Italy

ACM ISBN 979-8-4007-2342-1/26/06
https://doi.org/10.1145/3811427.3811499

[1]React was ranked most popular in the 2025 Stack Overflow survey [15].

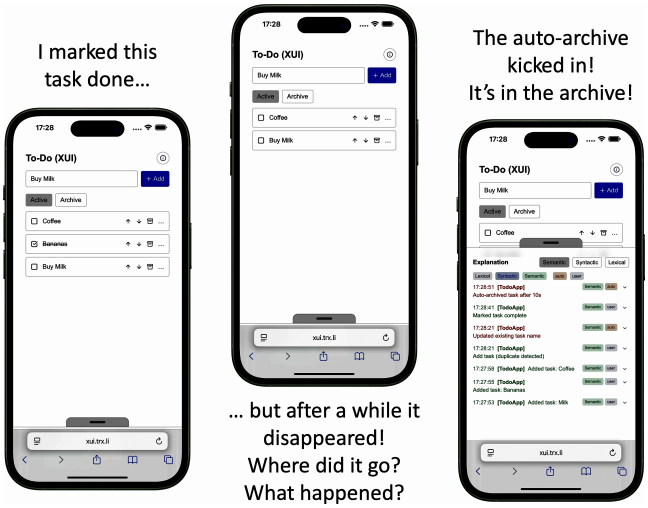


Figure 1: React to-do-list app with XUI

XUI data collection – The implementation leverages two key React mechanisms: Context and hooks. React Context provides shared data across a component tree without prop-drilling (i.e., explicitly passing data through every intermediate component level). Custom hooks encapsulate stateful logic for reuse. Our implementation wraps these primitives to add XUI functionality with minimal developer burden.

The architecture follows React's provider-consumer pattern. The `ExplanationContext` stores the event log and provides a `logEvent` function. The `ExplanationProvider` component wraps the application tree, maintaining the event log in its internal state and exposing this through React's Context API. Any descendant component can then access the logging functionality without requiring explicit prop threading through intermediate components. Global event listeners are attached within the Provider's initialization, ensuring lexical events are captured throughout the component tree. This centralized architecture means that even independently developed components automatically participate in the explanation system once wrapped by the Provider.

XUI visualisation – A basic user interface is also provided by the `ExplanationPanel` component which consumes the data gathered by the `ExplanationContext` to render the event log as an interactive UI, following closely the style in the original XUI envisionment video [3]. The application's existing interface is augmented by a tab that can be drawn up by the user showing the most recent events or a complete history (see Fig. 1). This can be viewed at multiple levels based on Seeheim's levels of abstraction [13]: semantic/functionality, syntactic/dialogue and lexical/presentation. The event stream includes both user-initiated and automatic events. The latter can be especially confusing for users.

While grounded in long-standing user interface understanding, this visualisation is not intended to be a definitive interface for XUI, more to demonstrate the *feasibility* of being able to present the kinds of information needed for UI explanation.

3 Demonstration application

Choice of application – In order to evaluate the potential of the generic XUI framework an existing application was adapted to use it. This involved adding the XUI data gathering and visualisation modules to the project (a single step) and then annotating the code. For this stage we chose the to-do-list application that is used throughout React documentation and libraries [9, 11, 14].

This application is relatively small, hence viable as a research demonstrator, but also extremely challenging for a number of reasons: **(1)** often novel frameworks and systems are demonstrated by 'toy' applications that are specially developed to demonstrate the new system, in this case it is an existing application; **(2)** the application was specifically created by the React community in order to demonstrate all the major features of React, including complex aspects such as asynchronous automatic actions; **(3)** the original code was developed by third parties and so not structured in order to be easily amenable to our XUI framework. It thus provides a realistic proof of concept.

Complexity of effort – Quantitatively, converting our to-do list application to support XUI required modifying 38 lines of code (24 deletions, 14 substantive changes) in a 334-line implementation. The changes fall into three categories: **(1)** replacing the state management hook call (one line per state variable), **(2)** adding description parameters to state updates, and **(3)** adding explicit `logEvent` calls for semantic events that don't correspond to state changes.

Where present `aria-label` attributes are used, and most updates are simple annotations of existing React calls, such as replacing `setTasks([...tasks, newTask])` with our `useExplainableState` wrapper: `setTasksWithExplain( [...tasks, newTask], "Added task: Buy milk")`. Importantly, the application logic itself — the actual state transformations — remains identical between versions.

4 Summary and further work

This paper has presented a first proof of concept of XUI demonstrating that a widely used development platform can be augmented to provide XUI capability, and that the cost of annotating an existing application to use this is not excessive. The use of an example application that includes automated actions is important as AI is becoming ubiquitous. The framework we have developed also shares features with those needed for accessibility, undo support and agentic AI suggesting the potential for coherent architectural support. The inspiration for XUI came from our own and widespread research into explainable AI (XAI) and in future work we intend to integrate XUI and XAI as well as empirically test the utility of XUI as a concept. We hope this work will also inspire others to explore different forms of XUI, so that future interfaces are really available for all.

The full code of our application, as well as a demo, can be found on GitHub[2].

Acknowledgments

This work has been supported by the HORIZON Europe project TANGO - Grant Agreement n. 101120763. Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the European Health and

[2]https://github.com/tommasoturchi/xui-demo

Digital Executive Agency (HaDEA). Neither the European Union nor the granting authority can be held responsible for them.

References

[1] Amina Adadi and Mohammed Berrada. 2018. Peeking inside the black-box: a survey on explainable artificial intelligence (XAI). *IEEE access* 6 (2018), 52138–52160.

[2] Plamen P Angelov, Eduardo A Soares, Richard Jiang, Nicholas I Arnold, and Peter M Atkinson. 2021. Explainable artificial intelligence: an analytical review. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery* 11, 5 (2021), e1424.

[3] Alan Dix. 2025. XUI : eXplainabale User Interfaces – BHCI 2025. https://vimeo.com/1134246464z Vimeo video.

[4] Alan Dix, Tommaso Turchi, and Ben Wilson. 2025. Towards Explainable User Interfaces. In *38th International BCS Human-Computer Interaction Conference*. BCS Learning & Development, 262–266.

[5] Erich Gamma. 1995. *Design patterns: elements of reusable object-oriented software*. Pearson Education India.

[6] Glenn E. Krasner and Stephen T. Pope. 1988. A Cookbook for Using the Model-View-Controller User Interface Paradigm in Smalltalk-80. *JOOP* 1, 3 (August 1988).

[7] Gerrit Meixner and Gaëlle Calvary Joëlle Coutaz. 2014. *Introduction to Model-Based User Interfaces*. Working Group Note 07 January 2014. W3C. http://www.w3.org/TR/2014/NOTE-mbui-intro-20140107/.

[8] Gerrit Meixner, Fabio Paternò, and Jean Vanderdonckt. 2011. Past, present, and future of model-based user interface development. *i-com* 10, 3 (2011), 2–11.

[9] Meta Open Source. 2026. Learn React: You Might Not Need an Effect. https://react.dev/learn/you-might-not-need-an-effect

[10] Kate Moran and Sarah Gibbons. 2011. Generative UI and Outcome-Oriented Design. Nielsen Norman Group, March 22, 2024, https://www.nngroup.com/articles/generative-ui/.

[11] Mozilla. 2026. Beginning our React ToDo app. https://developer.mozilla.org/en-US/docs/Learn_web_development/Core/Frameworks_libraries/React_todo_list_beginning MDN Web Docs.

[12] Brad A. Myers, Dario Giuse, Andrew Mickish, Brad Vander Zanden, David Kosbie, Richard McDaniel, James Landay, Matthew Golderg, and Rajan Pathasarathy. 1994. The Garnet user interface development environment. In *Conference Companion on Human Factors in Computing Systems* (Boston, Massachusetts, USA) *(CHI '94)*. Association for Computing Machinery, New York, NY, USA, 457–458. doi:10.1145/259963.260472

[13] G. Pfaff and P.J.W. ten Hagen (Eds.). 1985. *Seeheim Workshop on User Interface Management Systems*. Springer-Verlag, Berlin.

[14] React Redux. 2024. Redux Fundamentals, Part 5: UI and React. https://redux.js.org/tutorials/fundamentals/part-5-ui-react

[15] Stack Overflow. 2025. 2025 Stack Overflow Developer Survey – Web frameworks and technologies. https://survey.stackoverflow.co/2025/technology#1-web-frameworks-and-technologies.

[16] Noi Sukaviriya, Srdjan Kovacevic, James D Foley, Brad A Myers, Dan R Olsen Jr, and Matthias Schneider-Hufschmidt. 1994. Model-based user interfaces: What are they and why should we care?. In *Proceedings of the 7th annual ACM symposium on User interface software and technology*. 133–135.

[17] Kashyap Todi, Gilles Bailly, Luis Leiva, and Antti Oulasvirta. 2021. Adapting User Interfaces with Model-based Reinforcement Learning. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems (CHI '21)*. Association for Computing Machinery, New York, NY, USA, Article 573. doi:10.1145/3411764.3445497