Thank you for sending this application, it was very interesting to solve.

I will be happy to discuss the code in greater details however I thought of sharing some design decision for the application and along with some other notes. For running the application readme page has the required instructions.

Backend Service

1. Postman collection is included as part of Git repository (Postman_Collection.json) .
2. The layering of application is inspired by Onion Architecture.
3. The dependencies in UI layer is kept at minimum , service dependencies are handled in Services Project . It helps in Keeping UI loosely coupled with infrastructure and other non-UI related dependencies.
4. To handle more Service complexities Service Interfaces should be refactored into its own project in future.
5. The business model (namespace Model) and database related model (namespace entity) is used across. To avoid confusion alias entity and model are used.
6. EF Core fluent API is used and encouraged to avoid tying the Persistence specific information like Primary key. Sample examples are shown for the team who will work on this and basic under Entity-Configurations.
7. Very generous db schema is used like nvarchar max , its expected whiles implementing the above this should be considered carefully as well.
8. `modelBuilder.ApplyConfigurationsFromAssembly(Assembly.GetExecutingAssembly())` is used so that developer does not have to register new entity changes.
9. Some basic example like ignoring DatePublished during DB Update Operation is included for reference. Although we will have restriction at API level its often required to be handle condition like this database level too.
10. ModelState.IsValid is not implemented intentionally.
    Its left for the team taking on the project with expectation they will implement complete Model Binding with check like over posting , HTTPS, CORS and other securities features.
11. Since this is application will initially be used for Pitch its developed in Happy Path with less edge case handling. However some important restriction is added like an admin user trying to modify resource of another admin user etc.
12. It is expected the team taking on this project will implement with global Error handling and logging or may be use Application Insight if hosting in Azure is a consideration.
13. Since password management was not in scope a basic but very extensible JWT authentication is implemented . It still usage some very bad practise like storing password in plain text etc which needs to be looked on during later stage/
14. Through Unit Test Case is not implemented but the example of xomplex and simple testing is included like setting up Mock, Automapper Profile and even In-Memory Database testing.
    It should be used as reference for complete implementation and good code coverage.
15. Implementation of Patch or even better OData is targeted for future release.
16. API will also support versioning and the full implementation will be done in code as well as in swagger documentation + UI

Frontend UI

1. Material-UI is used for more modern look and feel for Mr. Pressford.
2. The application is using ReactHooks like useState, useEffect and useReducer
   Hence all components are functional component .
3. There is a **known issue** while loading the page first time, there is no data shown initially simple page **refresh** should fix this.
   The reason is while making api call Bearer token is not getting passed and 401 is returned.
   Will fix this issue in later release.
4. Instead of creating separate component for User and Publisher and even for some UI like publish and edit same component is reused across .
   The inline logic is used to determine the UI like User should not see deleted, Publish button.
5. One growing future complexities and requirement we might move them to sperate components and sharing the common UI components.

Kind Regards,
Kumar Shantanu