Note Subdomain

Table of Contents

This module (`incode-module-note`) provides the ability to attach `Note` objects to arbitrary domain entities. There are *no* requirements for those domain objects implement any interfaces. A subclass of the `NotableLink` abstract class is required (about 50 lines of boilerplate), acting as the "glue" between the note and the "notable" domain object.

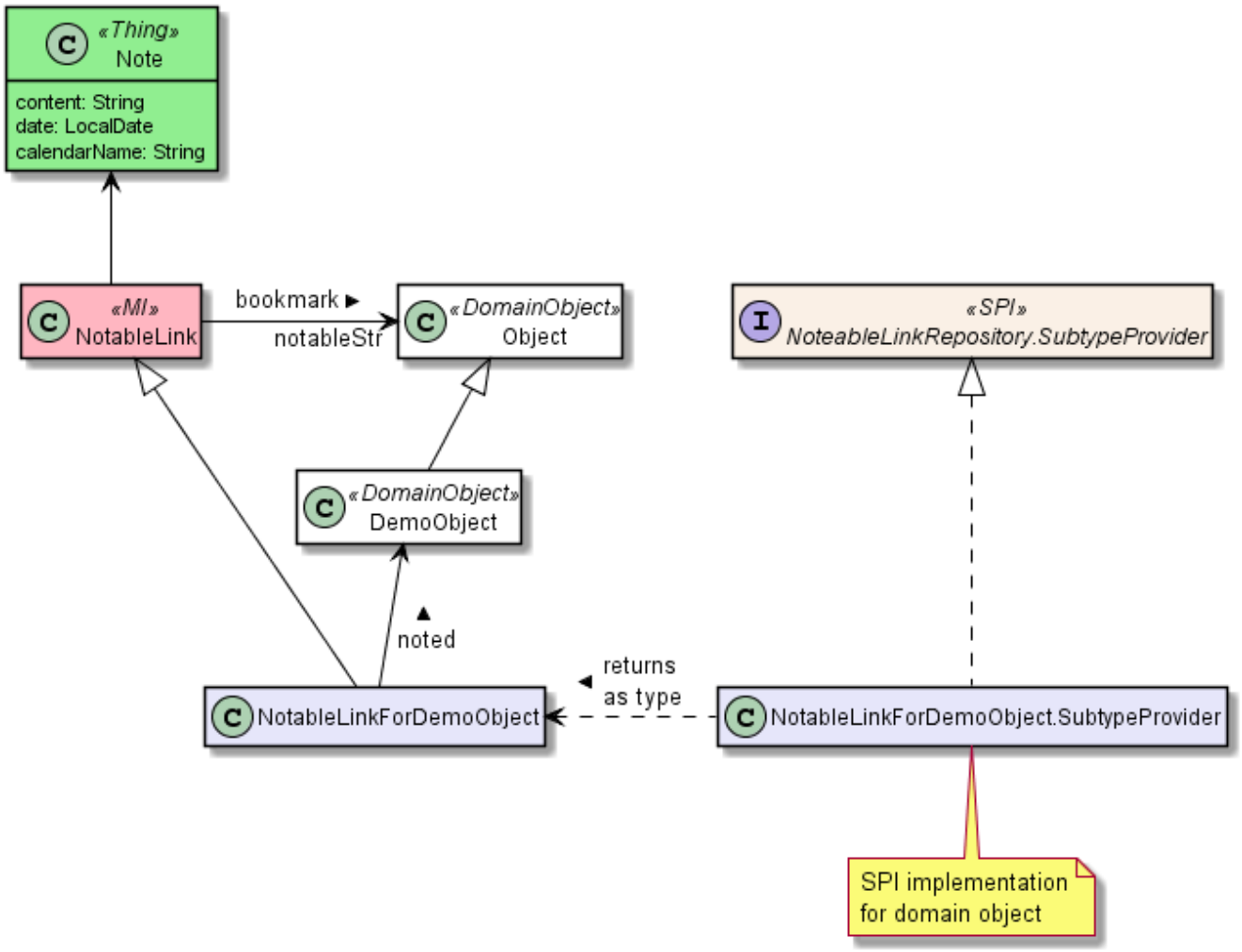
Each `Note` can optionally have a date associated with it, and corresponding (predefined) calendar. The `Note` implements the `CalendarEventable` interface (from the fullcalendar2 wicket component) meaning that they can then be rendered in a calendar view. This makes the notes useful for a simple task list.

The optional `CalendarNameRepository` SPI service is used to enumerate the calendars for any given "notable" domain object. The list of available calendars is determined by the "notable": if `Customer` and `Order` both are "notable", then the `CalendarNameRepository` could provide different calendar names for each. If no implementation of the `CalendarNameRepository` service can be found, then a single "default" calendar is used.

Any given "notable" domain object can have multiple `Note`s. As currently implemented, a "notable" domain object can only have one `Note` per named calendar.

Domain Model

The following class diagram highlights the main concepts:

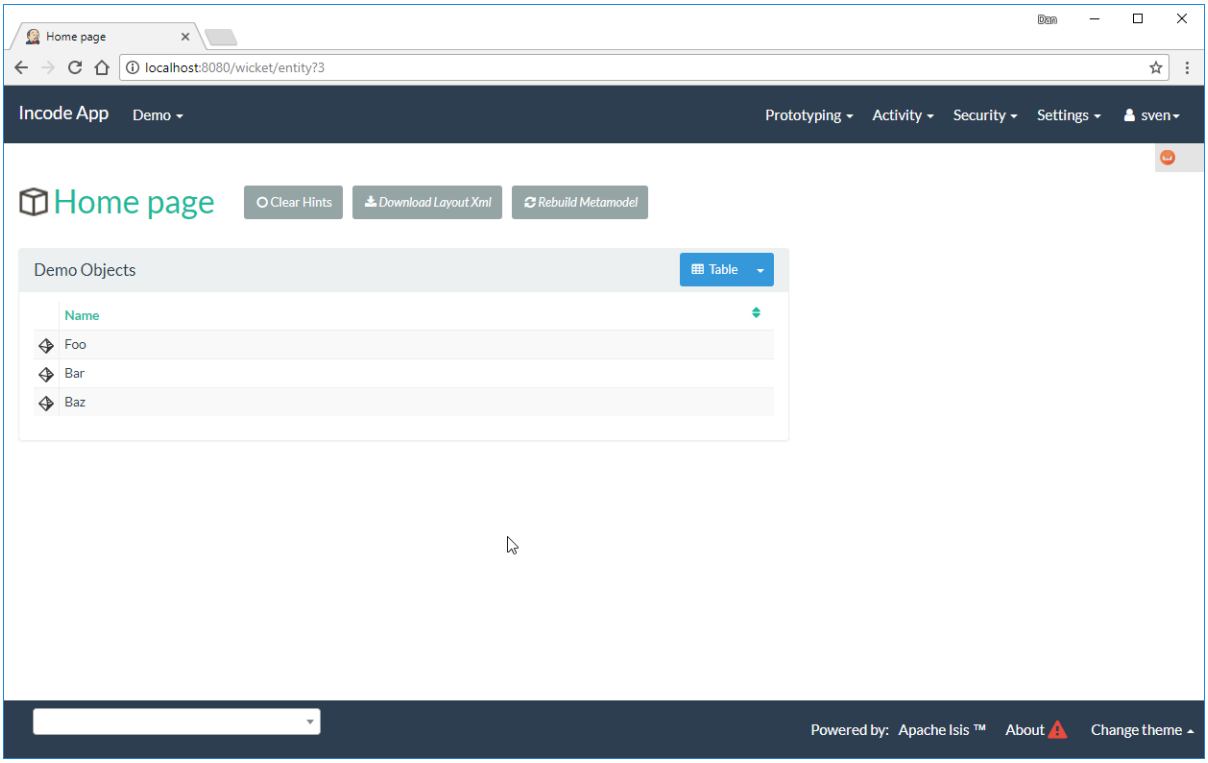


(The colours used in the diagram are - approximately - from Object Modeling in Color).

Screenshots

The module's functionality can be explored by running the quickstart with example usage using the `org.incode.domainapp.example.app.modules.ExampleDomDomNoteAppManifest`.

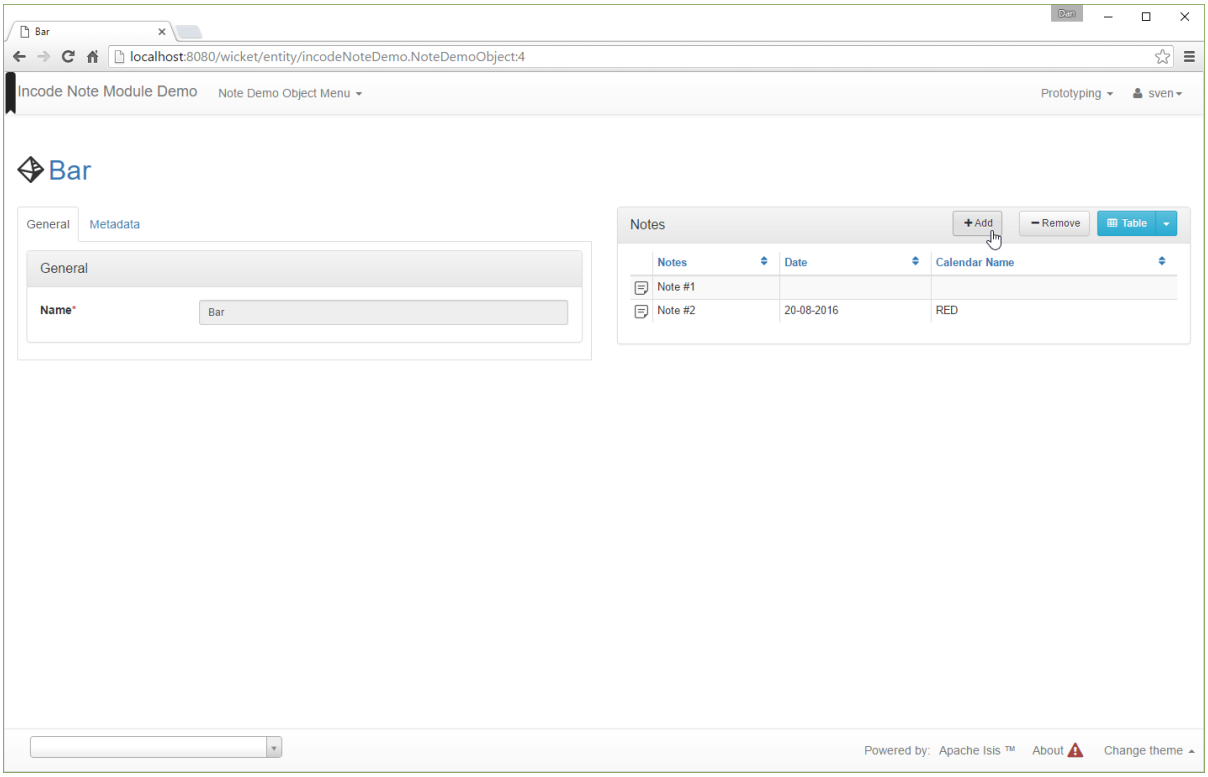
A home page is displayed when the app is run:



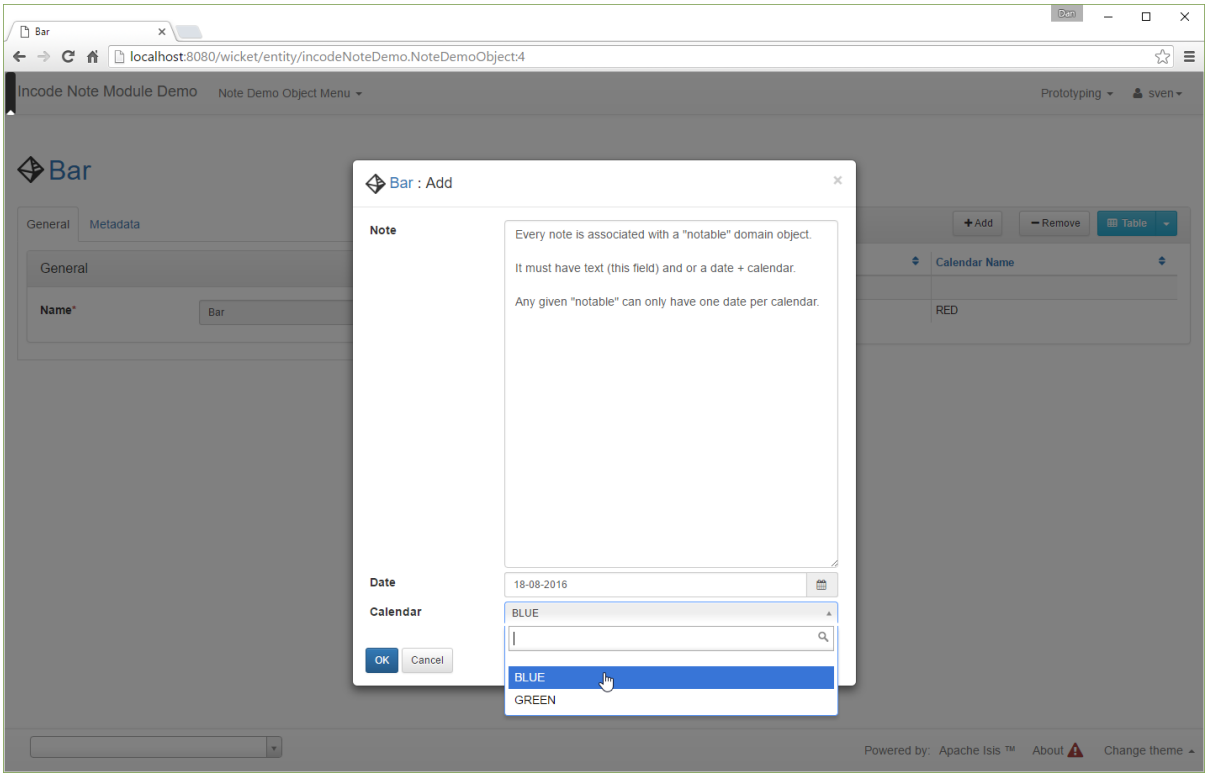
These are our "notable" domain objects.

The remaining screenshots below **do** demonstrate the functionality of this module, but are out of date in that they are taken from the original isisaddons/incodehq module (prior to being amalgamated into the incode-platform).

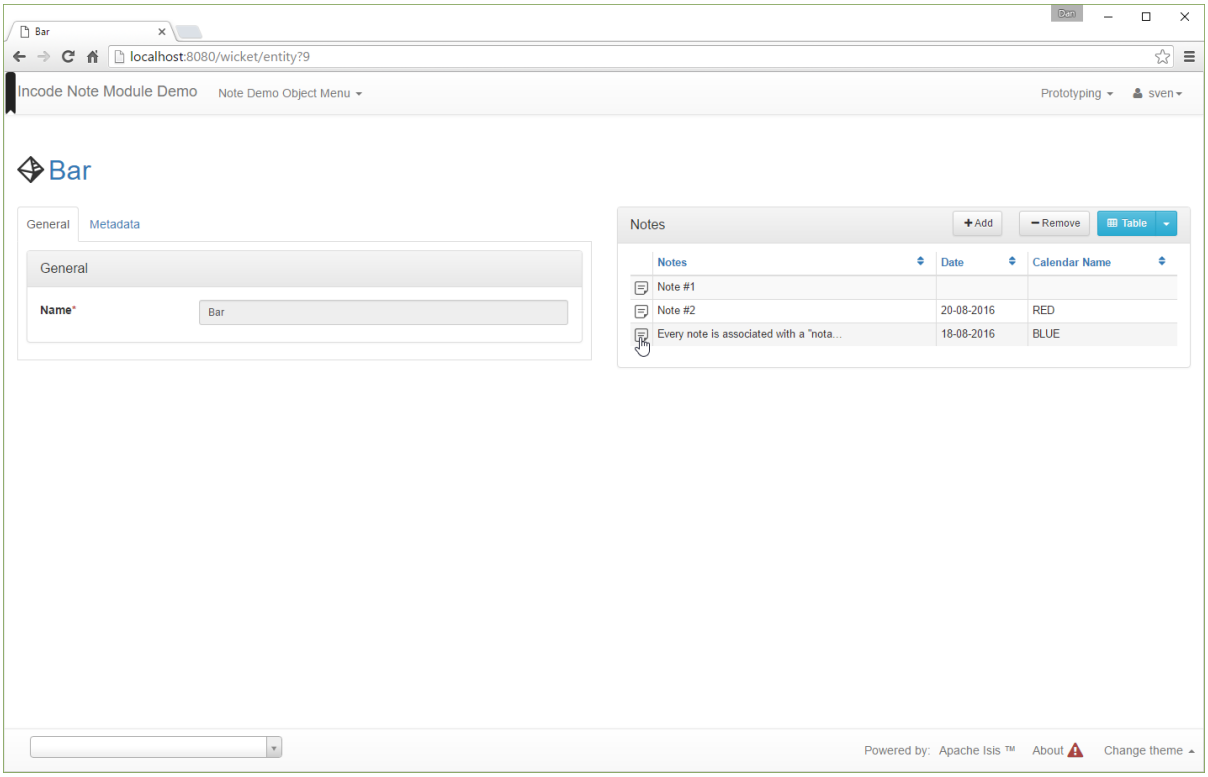
The fixture sets up some notes for each of these "notable" objects; these are displayed in a (contributed) `notes` collection. We can also add new notes using a (contributed) `addNote(…)` action:



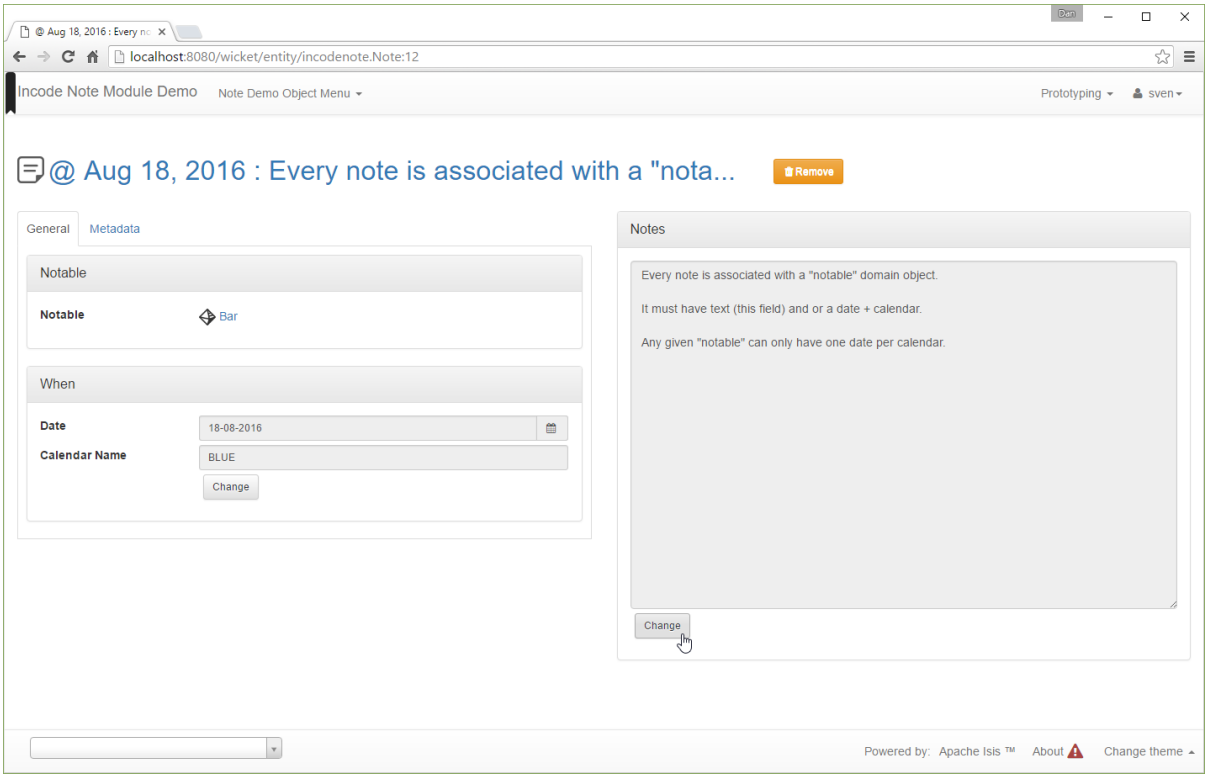
The action allows the note text and optionally a date/calendar to be specified. Every note must have either text and/or a date and calendar. Also, each "notable" can only associate one `Note` per calendar. The list of calendars is defined by the optional `CalendarNameRepository` SPI domain service, discussed below:



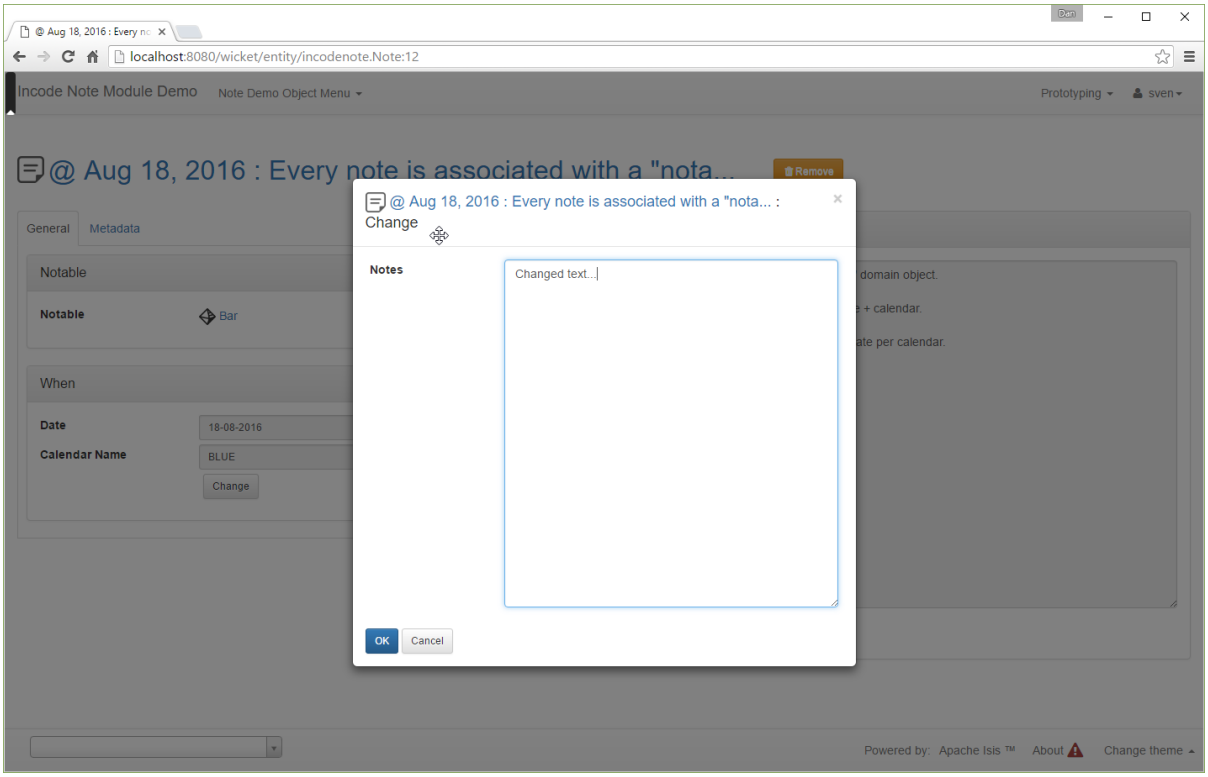
The notes for the "notable" domain object is added to. Each `Note` can also be viewed:



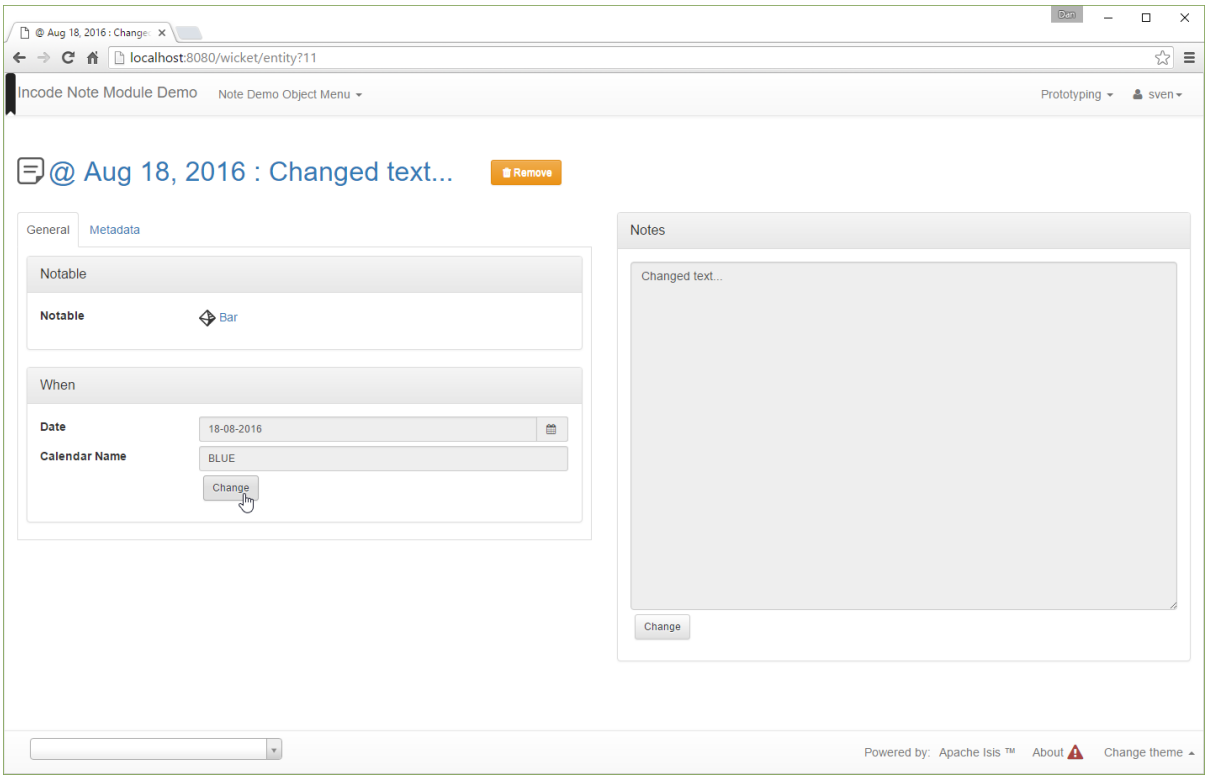
The `Note` shows the text and date/calendar, as well as the "notable" domain object that it is attached to.



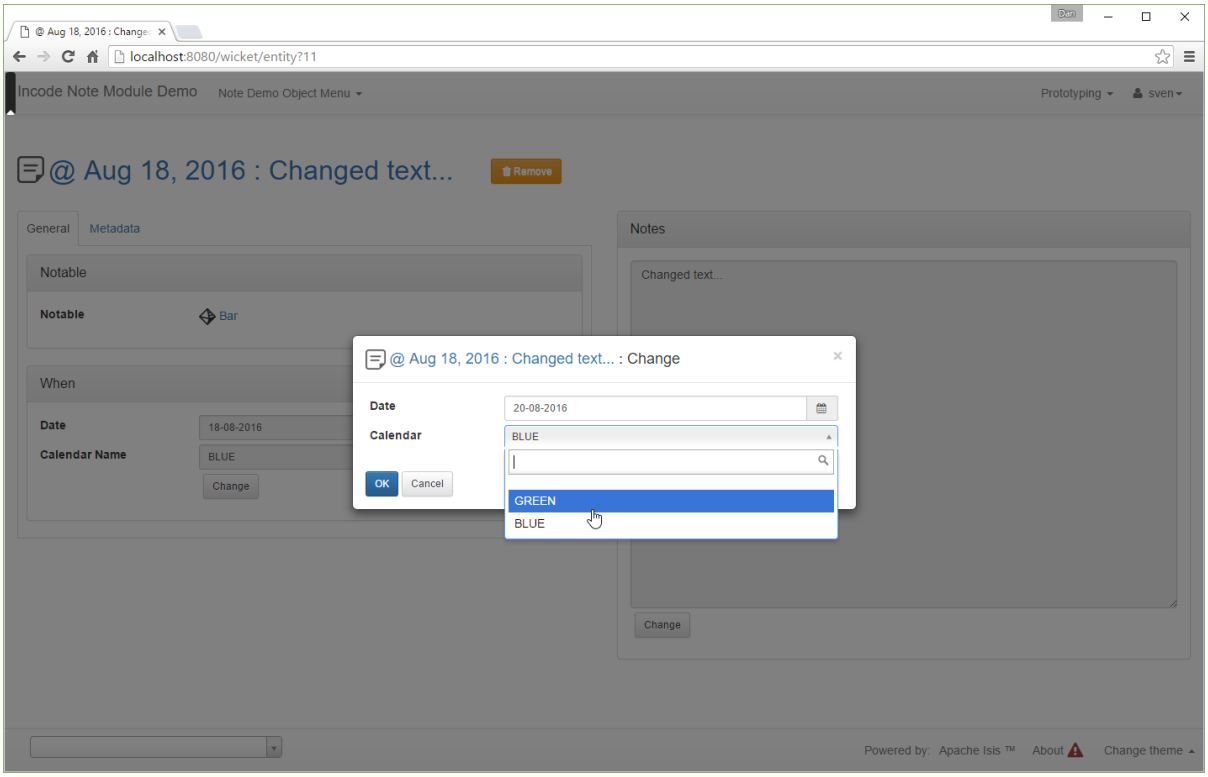
The `changeNote(…)` action allows the note text to be updated (or cleared/set to null if the note has a date/calendar):



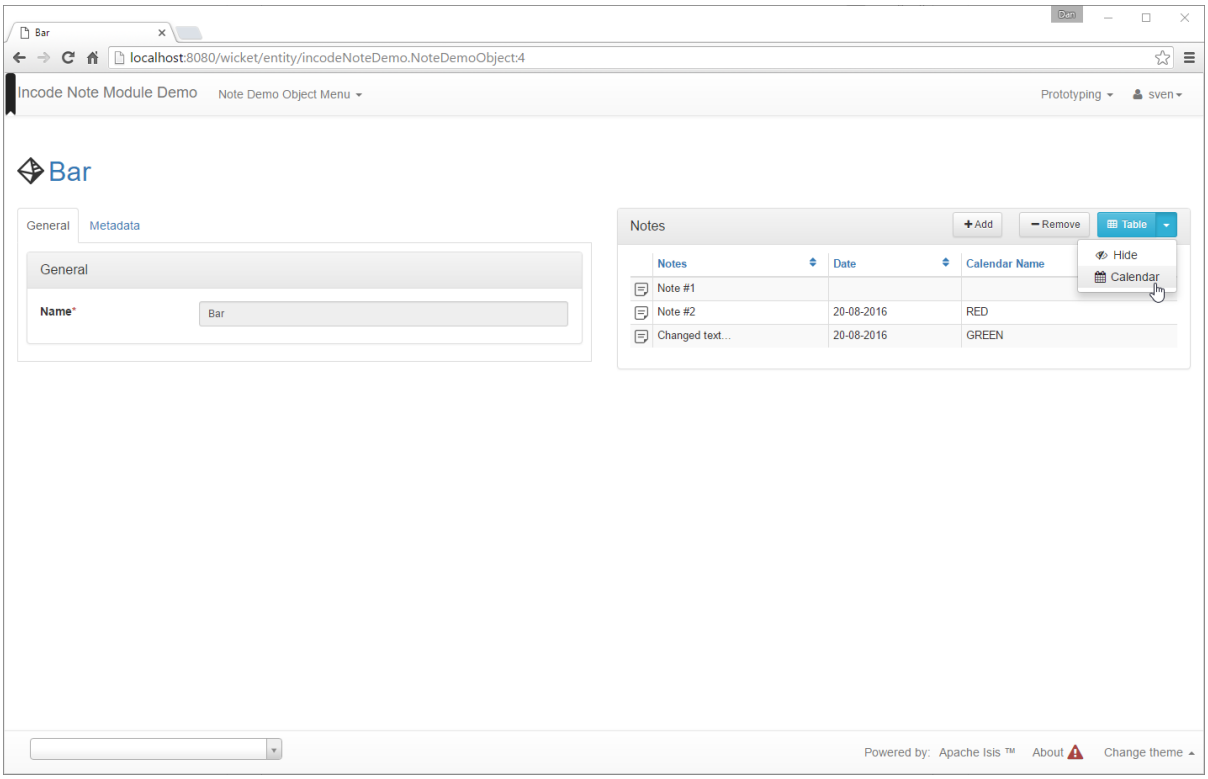
while the `changeDate(…)` action…



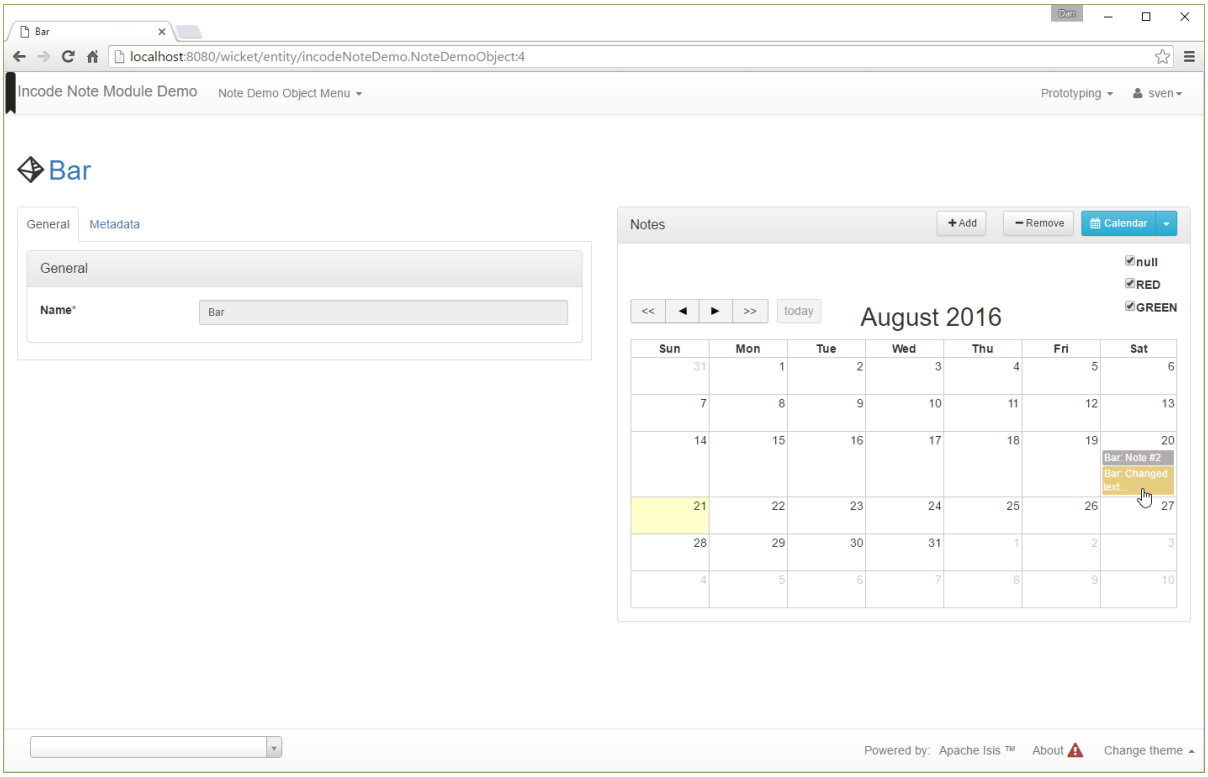
allows the note's date/calendar to be updated (or cleared/set to null if the note has text):



Each `Note` implements the fullcalendar2 component's `CalendarEventable` interface, meaning ...



... that it can be rendered on a calendar:


Bar
localhost:8080/wicket/entity/incodeNoteDemo.NoteDemoObject:4
Incode Note Module Demo
Note Demo Object Menu
Prototyping
sven
Bar
General
Metadata
General
Name*
Bar
Notes
+Add
−Remove
Calendar
null
RED
GREEN
<<
◄
►
>>
today
August 2016
Sun
Mon
Tue
Wed
Thu
Fri
Sat
Bar: Note #2
Bar: Changed text...
Powered by: Apache Isis ™
About
Change theme

How to configure/use

Classpath

Update your classpath by adding this dependency in your dom project's `pom.xml`:

```
<dependency>
    <groupId>org.incode.module.note</groupId>
    <artifactId>incode-module-note-dom</artifactId>
    <version>1.15.1.1</version>
</dependency>
```

Check for later releases by searching Maven Central Repo.

For instructions on how to use the latest `-SNAPSHOT`, see the contributors guide.

Bootstrapping

in the `AppManifest`, update its `getModules()` method, eg:

```
@Override
public List<Class<?>> getModules() {
    return Arrays.asList(
            ...
            org.incode.module.note.dom.NoteModule.class,
    );
}
```

For each domain object...

In order to be able to attach a note to a domain object, you need to:

- implement a subclass of `NotableLink` to hold a type-safe reference back to the domain object.

  This link acts as a type-safe tuple linking the domain object to the `Note`.

- implement the `NotableLinkRepository.SubtypeProvider` SPI interface:

  ```
  public interface SubtypeProvider {
      Class<? extends NotableLink> subtypeFor(Class<?> domainObject);
  }
  ```

  This tells the module which subclass of `NotableLink` to use to attach to the domain object. The `SubtypeProviderAbstract` adapter can be used to remove some boilerplate.

- subclass `T_addNote`, `T_removeNote`, `T_notes` (abstract) mixin classes for the domain object.

  These contribute the "notes" collection and actions to add/remove notes for the domain object.

Typically the SPI implementations and the mixin classes are nested static classes of the `NotableLink` subtype.

For example, in the demo app the `NoteDemoObject` domain object can have notes attached to it by virtue of the `NotableLinkForDemoObject` subclass:

```
@javax.jdo.annotations.PersistenceCapable(identityType= IdentityType.DATASTORE, schema
="incodeNoteDemo")
@javax.jdo.annotations.Inheritance(strategy = InheritanceStrategy.NEW_TABLE)
@DomainObject
public class NotableLinkForDemoObject extends NotableLink {                    ①

    private NoteDemoObject demoObject;
    @Column( allowsNull = "false", name = "demoObjectId" )
    public NoteDemoObject getDemoObject() {                                     ②
        return demoObject;
    }
    public void setDemoObject(final NoteDemoObject demoObject) {
        this.demoObject = demoObject;
    }

    public Object getNotable() {                                                ③
        return getDemoObject();
    }
    protected void setNotable(final Object object) {
        setDemoObject((NoteDemoObject) object);
    }

    @DomainService(nature = NatureOfService.DOMAIN)
    public static class SubtypeProvider
                extends NotableLinkRepository.SubtypeProviderAbstract {         ④
        public SubtypeProvider() {
            super(NoteDemoObject.class, NotableLinkForDemoObject.class);
        }
    }

    @Mixin
    public static class _notes extends T_notes<NoteDemoObject> {                ⑤
        public _notes(final NoteDemoObject notable) {
            super(notable);
        }
    }
    @Mixin
    public static class _addNote extends T_addNote<NoteDemoObject> {
        public _addNote(final NoteDemoObject notable) {
            super(notable);
        }
    }
    @Mixin
    public static class _removeNote extends T_removeNote<NoteDemoObject> {
        public _removeNote(final NoteDemoObject notable) {
            super(notable);
        }
    }
}
```

① extend from `NotableLink`

② the type-safe reference property to the "notable" domain object (in this case `DemoObject`). In the RDBMS this will correspond to a regular foreign key with referential integrity constraints correctly applied.

③ implement the hook `setNotable(…)` method to allow the type-safe reference property to the "notable" (in this case `DemoObject`) to be set. Also implemented `getNotable()` similarly

④ implementation of the `SubtypeProvider` SPI domain service, telling the module which subclass of `NotableLink` to instantiate to attach to the owning domain object

⑤ mixins for the collections and actions contributed to the owning domain object

SPI

The `CalendarNameRepository` interface can optionally be implemented to specify the available calendars for each "notable" domain object.

For example, in the demo app this is implemented as:

```
@DomainService(nature = NatureOfService.DOMAIN)
public class CalendarNameRepositoryForDemo implements CalendarNameRepository {
    private final Map<Class<?>, List<String>> namesByClass = Maps.newHashMap();
    public CalendarNameRepositoryForDemo() {
        setCalendarNames(NoteDemoObject.class, "BLUE", "GREEN", "RED");
    }
    @Programmatic
    public void setCalendarNames(final Class<?> cls, final String... names) {
        namesByClass.put(cls, Lists.newArrayList(names));
    }
    @Override
    public Collection<String> calendarNamesFor(final Object notable) {
        return namesByClass.get(notable.getClass());
    }
}
```

If no implementation of this interface can be found, then the module provides a single "default" calendar for all "notable" domain objects.

UI Concerns

Suppressing/adding UI elements

Every property, collection and action has a corresponding domain event. Thus, a subscriber can be used to hide or disable UI representation of any domain object's members.

For example, the "content" property of a `Note` could be suppressed using the following service:

```
@DomainService(nature = NatureOfService.DOMAIN)
public class NotesDemoSuppressContentSubscriber extends AbstractSubscriber {
    @Subscribe
    public void on(Note.ContentDomainEvent ev) {
        switch (ev.getEventPhase()) {
        case HIDE:
            // uncomment as an example of how to influence the UI
            // (the content property should disappear)
            // ev.hide();
        }
    }
}
```

Conversely, new UI elements can be added using contributions and mixins.

Link class

The `NotableLink` object is not intended to be rendered directly in the UI. Rather, the `T_notes` mixin renders the referenced `Note`s instead.

Nevertheless (just in case there is a requirement to render the link object), the `NotableLink` allows its title, icon and CSS class to be specified using subscribers to UI event classes specific to the link class.

Other Services

The module provides the following domain services for querying notes:

- `NoteRepository`

  To search for notes by "notable" or in general within a date range

- `NotableLinkRepository`

  To search for `NotableLink`s, ie the tuple that links a `Note` with an arbitrary "notable" domain object. This repository is likely to be less useful than `NoteRepository`, but is crucial to the internal workings of the `incode-module-note` module.

Known issues

None known at this time.

Dependencies

Maven can report modules dependencies using:

```
mvn dependency:list -o -pl modules/dom/note/impl -D excludeTransitive=true
```

which, excluding the Incode Platform and Apache Isis modules, returns no direct compile/runtime dependencies.

From the Incode Platform it uses:

- poly library
- fullcalendar2 wicket component

The module also uses icons from icons8.