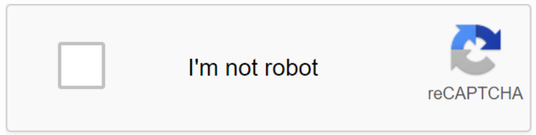

Continue

Cuba platform tutorial pdf

FacebookTwitterLinkedInGithubYoutubeAll articles ↓Share: Cuba Platform is an open source platform aimed at improving the process of developing business applications. It combines proven architecture, ready-to-use enterprise-grade components, and powerful instrumentation, so you can deliver modern web applications faster than ever before. In this fast start, we will scratch cuba's surface and develop a very simple but fully functional conference planning application. It will show you three main things to create any web application: how to design a data model, how thew manipulate data, how to create business logic, and finally how to create a user interface. For example, we'll create a simple but fully functional version of the app to help you schedule a conference session. In fact, this tutorial will only be enough to start your own KUBA app, so let's get started. In this tutorial, we'll use CUBA Studio, so install it before you begin and accept the trial subscription to have access to visual designers. Sample code repository: Https://github.com/cuba-platform/sample-session-planner.Creating blank projectSee an empty KUBA project using the appropriate Intellij IDEA menu. Name it SessionPlanner with project namespace - planner. We will use Java 11 as the default JDK. We will use an in-memory database - HSQLDB. Let's apply a trial subscription. It is free for 28 days and gives us some useful optional designers. Creating a data model The first task is to create entities. The business domain model has only two classes: Speaker and Session. The relationship is one to many. One speaker can conduct multiple sessions. Let's start with the Speaker entity. To do this, we can use the link on the project welcome page:Or just right-click the data model node in the KUBA project tree on the left side of the screen and select the new -> Entity.Enter entity name - Speaker and create attributes according to specifications:On CUBA we use standard JPA units so that you can create them using a code editor or visual designer. Just click the + icon and add attributes to the entity, CUBA Studio will generate class members for you. In cuba, you can specify the format for the proper display of the entity as a string in the UI by using the instance name. For the speaker we will choose both first name and last name. If we look at the Text tab at the bottom of the entity designer, we can only see the java class with JPA annotations. If the generated code can be updated as needed, the designer will be updated accordingly when you switch to the Designer tab. Additionally, you can see the DDL Viewer and you can create indexes as needed. For example, we expect multiple searches by To make this search more efficient, you can create a name field index. Let's move on and create a Session entity and combine it with the Speaker class. Speaker. Specification. The end time of the session will be the calculated value, which is the start date plus the duration. Take a look at adding a mandatory reference to the speaker. The relationship is many to one, so we'll define an association field called a speaker that references the speaker class. Finally, the field definition should look like this:Create a calculatednow attribute, we need to create an automatically calculated attribute for the end date and time of the session. First, add the getter method and name it getEndDate. Annotate the method @MetaProperty. For a calculated attribute, we need to specify the fields to be loaded. In our case, they are the initialdate and duration. Then just add calculations logic.@Transient @MetaProperty(related = {startDate, duration}) public LocalDateTime getEndDate() { return (startDate != null & duration != null) ? startDate.plusHours(duration) : null; } Note that Studio highlights the method because we need to specify a text label for the attribute. Press Alt+Enter and add text to the message package. That's it. Our domain model was created. Generating CRUD ScreensCUBA Studio includes a UI screen generation wizard that helps us create basic but useful UI screens:Browser - to display a list of entities in gridEditor - to edit one instance of an entity using a user interface similar to formFirst, we will create screens to work with speakers. Since this unit is quite simple, we will use the default parameters to create the screen. Start the wizard by clicking create screen on the Screens menu at the top of the entity designer. You can also start the Screen Generation Wizard by right-clicking the entity in the KUBA project tree and selecting New -> screen from the context menu. For the Speaker unit, we'll create a default browser and editor. Select Browser screens and entity editor on the Screen Templates tab of the wizard, and click Next. Since this unit is quite simple, we can accept the default parameters for creating the screen. As you can see, each screen consists of two parts: a controller, written in Java, which is responsible for internal screen logic and event handling, and an XML layout that defines the appearance of the screen. In our case, the browser will consist of speaker-browse.xml and SpeakerBrowse.java and editor - speaker-edit.xml and SpeakerEdit.java respectively. The source files are located under General User Interface -> menu screens in the KUBA project tree. Pay attention to the data section in the XML screen descriptors - determines how to retrieve data from the database. You can easily navigate between the screen controller, descriptor and connected entity using cuba studio using the buttons at the top of the window:Create a browser and editor for sessionRun the screen generation wizard, readonly=true> <collection id=speakersDc class=com.company.planner.entity.Speaker view=_local> <loader id=speakersDl> <query> <! [CDATA[select e from planner_Speaker e]]> </query> </loader> </collection> </data> </data> Browser screens and editor entities and stop in the unit browser view step. In CUBA, the entity view determines which fields will be retrieved from the database. You can define views in a separate file to use in different application modules, or to create built-in views when creating screens. Let's create a built-in view. Just select the necessary data: end date and speaker reference. In the next step, the necessary fields are already selected. You can see that the appropriate fields are added to the screens. Also, set the default duration value for the new session to one hour. To do this, go to the Component Inspector window and subscribe to the InitEntity event on the Handlers tab. Then set this value to code.@Subscribe void onInitEntity(InitEntityEvent<Session> event) { event.getEntity().setDuration(1); } Create a cuba select database -> Generate database scripts menu to create sql to create the database. You can review the generated scripts in a pop-up window before saving them as project files. Note that these scripts are part of the project, and you can find them in the node of the main data store in the project tree. You can modify scripts if you want to add certain specific things, such as additional indexes, or insert instructions for the initial data. Click Create Database to apply these scripts and create the database. In addition to application tables, CUBA creates system tables in which we store information about users, roles, tasks, etc. That's it. The database has been created. Running the application in development modeAboth application startup can be used or run the configuration at the top in the IDE window. From the main menu, select CUBA -> Start Application Server. After a while, you can access the application using a browser. The URL appears in the Run in Idea toolbox. In our case, it will be . Open the URL in your web browser and sign in to the app using admin as your username. The password is admin by default. Screens for manipulating units can be found in the Applications menu. Next, let's add some data to the database: several speakers and two sessions for them scheduled for the rest of the week. You can try entering an invalid email message for the speaker and see that the email validator is working as expected. The screens generated are good for basic operations, but in the real world the user interface is usually more complex. Customize the user interfaceDecoding a calendar view to view sessions in addition to grid view. On the browser screen, we'll add a tab control, put a calendar on it, and make it available to edit and reassemly the session using that calendar, as in the following image:Open session-browse.xml in the designer and wrap the session table in a worksheet in the Component Hierarchy window. Add one more tab page under TabSheet.Put calendar controls on it. Select tabsheet in the Component Hierarchy window and select expanded in </Session> </Session> Studio asks for an ID. In CUBA, we need identifiers to refer to a screen element in your code. Assign calendar cards to IDs and a tab for the tabs. Then set the session calendar and sessions table signatures for these tabs accordingly. Update calendar – Assign a data container and expand it. Finally, expand CUBA Table.In, UI components can be bound to entities and their properties. Let's associate the calendar with a collection of data downloaded on the screen. Use the search box to find the properties and bind:startDateProperty to start the DateendDateProperty session until the end of the DatecaptionProperty session to the subject session and descriptionProperty to the descriptionLet to make the calendar to display only working hours. In addition to the visual designer, you can use the XML tag editor. Let's add navigation buttons. To see changes to the user interface, you don't need to restart the app, just reopen the screen in the app. Cuba supports hot screen deployment. Now you can see the sessions are displayed in the calendar. By using the screen APIWhen interacting with the UI, events are generated, so CUBA gives you an API that you can use to subscribe to these events to handle them. Let's use a click on a calendar entry and call the session editor. In case of screen manipulation, CUBA provides the Screen Builder API that we will use. In the session-browse.xml file, select SessionsCalendar and go to the Handlers tab in the Component Inspector window. Select CalendarEventClickEvent and click the arrow to go to the controller. An empty method will be generated for you.@Subscribe(sessionsCalendar) public void onSessionsCalendarCalendarEventClick(Calendar.CalendarEventClickEvent<LocalDateTime> event) { } We need to call up the editor screen to change session properties. Let's use EditorScreenFacet. This is a component that provides the ability to pre-configure the editor screen. Browse to session-browse.xml and locate the editor screen in the component palette and move it below the window element in the Component Hierarchy window. Set the following settings:ID - sessionEditDialog;data container - sessionsDc;edit mode - EDIT;entity class - Session;open mode - DIALOG;screen class - SessionEdit.Go to the event handler. Click the Injection Injection button at the top of the window and select the SessionEdidialog service form Screen API section in the pop-up window. Another way to inject (and subscribe) - press alt+insert key combination in the editor and select the insert injection in the pop-up menu:To search for a service, just start typing its name, and then use the up and down keys to move between the services available for injection. Upload the session entity received in the event object for editing. Then we should show the editor. @Subscribe(sessionsCalendar) public void event) { sessionEditDialog.setEntityProvider(() -> (Session)</LocalDateTime> </LocalDateTime> </LocalDateTime> sessionEditDialog.show(); } That's it. You can reopen the session browser screen in the running application and call the editor by clicking on the session in the calendar. The editor does not look nice, so we need to adjust its width and height. In ide, open the XML descriptor screen, select dialogMode properties, and set the width and height of the property to auto. Go to the app and reopen the editor by closing it and clicking on the session in the calendar again. Now it looks nicer. Adding Business LogicNow we will use CUBA Studio to create a service that implements business logic and uses that service on the screen. This will be a session rescheduly service that ensures that one speaker does not have more than two sessions in one day. Right-click the service node in the KUBA project tree and choose New ->Service. This opens the create service dialog box. If you enter SessionService as the interface name, SessionServiceBean as the implementation name will be generated automatically. Create a rescheduleSession method in the interface, as in the code snippet below:public interface SessionService { String NAME = planner_SessionService; Reschedual scheduled session session(session session, LocalDateTime newStartDate); } The method accepts the session and the new date and time of the session. The service returns an updated session instance. To implement the method, open the code editor for the sessionservicebean class, you can find it in the middleware - node service in the KUBA project tree:You'll see that class is empty and invalid:Press Alt + Insert in class body and select implement methods in pop-up menu:Select reschlatleSsion method, code shortcut will be generated:@Service(SessionService.NAME) Public class SessionServiceBean implements SessionService { @Override public reschedule SessionSession(Session Session, LocalDateTime newStartDate) { return null; } First, calculate the start and end times of the day on which the session is planned.@Override public Reschedule SessionSession(Session Session, LocalDateTime newStartDate) { LocalDateTime dayStart = newStartDate.truncated(ChronoUnit.DAYS).withHour(8); LocalDateTime dayEnd = newStartDate.truncatedTo(ChronoUnit.DAYS).withHour(19); returns null; } For the service, we will use the CUBA API to access data - DataManager class. With this class, we create a JPQL query to see if there are any sessions scheduled for the speaker over a specified period of time and add parameter values to it. We then check the result of the query and, depending on the result, update the session with the new start date or simply return the original session instance. Inject DataManager into the service by pressing Alt+Insert in the class body and choose Inject from pop-up menu:Select datamanager in pop-up window:Method implementation is on the snippet below: @Override Session Reschemy Public Session Session,LocalDateTime newStartDate) { day = newStartDate.tnie(ChronoUnit.DAYS).withHour(8); newStartDate.tnie(ChronoUnit.DAYS).withHour(8); dayEnd = newStartDate.truncatedTo(ChronoUnit.DAYS).withHour(19); Long sessionsSameTime = dataManager.loadValue(select count(s) from planner_Session where + (s.startDate between :d ayStart and :d ayEnd) + and s.speaker = :speaker + and s.id <> :sessionId, Long.class) .parameter(dayStart, dayStart) .parameter(dayEnd, dayEnd) .parameter(speaker, session.getSpeaker()) .parameter(sessionId, session.getId()) .one(); if (sessionsSameTime >= 2) { return session; } session.setStartDate(newStartDate); returns dataManager.commit(session); } The service is ready, now let's add it to the session browser screen. It will be called for the drag-and-drop event in the calendar. In the session-browse.xml file, select SessionsCalendar and go to the Handlers tab in the Component Inspector window. Select CalendarEventMoveEvent and click the arrow to go to the controller. In a method that is subscribed to a drag-and-drop event, we add code to extract the session entity from the calendar event and pass it to the service to request that session. Then we will simply update this item in the public container.@Subscribe void onSessionsCalendarCalendarEventMove(Calendar.CalendarEventMove)<LocalDateTime> { Session session = sessionService.rescheduleSession((Session) event.getEntity(), event.getNewStart()); sessionsDc.replaceItem(session); In order for the new service to be redeplaceed, we need to restart the application, we can use the same run button in IDEA. After restarting we can open the session calendar - and voila! We have a drag-and-drop re-pull function for your calendar! Let's test it by adding an extra session and trying to get it off. By adding brandingW CUBA, you can update resource files and replace standard text. Let's update the text according to the business domain of the application - conference planning. Open the main message package file - messages.properties, which is under General UI - Main Message Package node in the KUBA project tree. This is a standard java .properties file, so you can edit it freely by adding new message keys and updating existing ones. Update messages to reflect the purpose of the application. The example is below:application.caption = Session Planner application.logoLabel = Session Planner application.logoImage = branding/app-icon-menu.png application.welcomeText = Welcome to session planning! loginWindow.caption = Login Login Session ScheduleWindow.welcomeLabel = Welcome to the Session Schedule! loginWindow.logoImage = branding/app-icon-login.png menu-config.application-planner = Sessions and speakers menu-config.planner_Speaker.browse=Speaker menu-config.planner_Session.browse=Sessions With cuba hot deployment function, all we need to do is log in to the app again to see the changes. MarketplaceSpryIad is equipped with a market that contains many components, thanks to you can easily add useful features such as charts or maps to support an existing application. You can install </LocalDateTime> </LocalDateTime> components directly from the studio via the CUBA -> Marketplace menu. Let's add an addition to helium. This is a new visual theme that you can use instead of standard themes. Just click install and apply the changes. Now we need to stop the application and update the database to apply the init scripts of the add-in. Launch the app and go to the settings screen. You can find the theme you added in the drop-down list. Select it and apply it. Sign in to the app again - the theme is already applied. You can also open the theme settings screen and change the settings using the preview. The ConclusionCUBA framework includes many useful APIs that help you build business applications quickly. This quick start shows only the basics of cuba and cuba studio framework. Thank you for your interest in CUBA Platform. Enjoy development with CUBA! Cuba!