

Supporting Code Comprehension via Annotations: Right Information at the Right Time and Place

Marjan Adeli, Nicholas Nelson, Souti Chattopadhyay, Hayden Coffey, Austin Henley, and Anita Sarma

{adelima, nelsonni, chattops, anita.sarma}@oregonstate.edu

khcoffey1@vols.utk.edu, azh@utk.edu



 @nomaticdev



<https://nomatic.dev/>

Code Comprehension

- Code Comprehension:
 - Navigating through a codebase
 - Building a mental model of that code
 - Large portion of developers' activities
 - 58% of the effort [1]



[1] Xia, X., Bao, L., Lo, D., Xing, Z., Hassan, A. E., & Li, S. (2017). Measuring program comprehension: A large-scale field study with professionals. *IEEE Transactions on Software Engineering*, 44(10), 951-976.

Difficulties in code comprehension

Understanding code requires developers to:

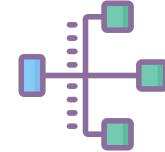
- Manage different types of artifacts



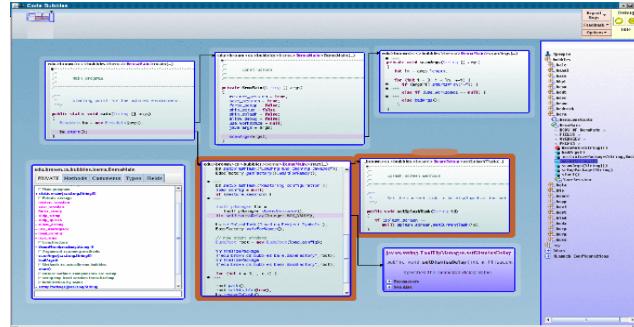
- Locate relevant information in different places



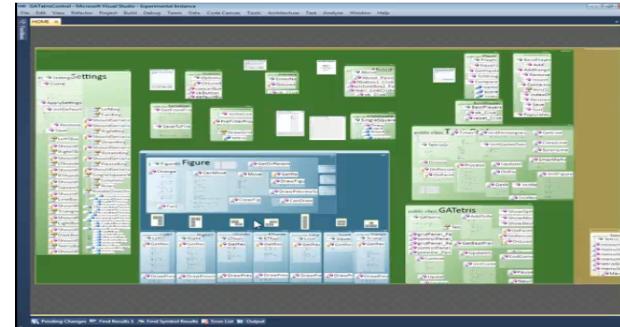
- Understand the relationships between artifacts



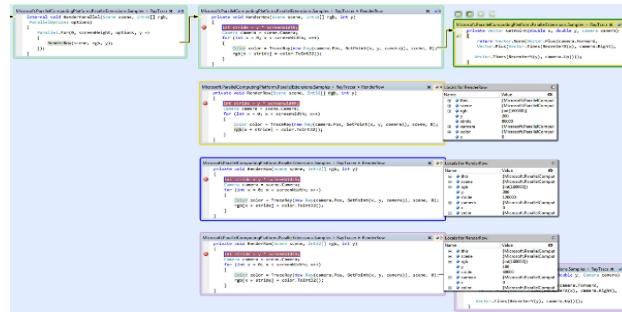
Facilitating code comprehension in IDEs



Code Bubbles



Code Canvas



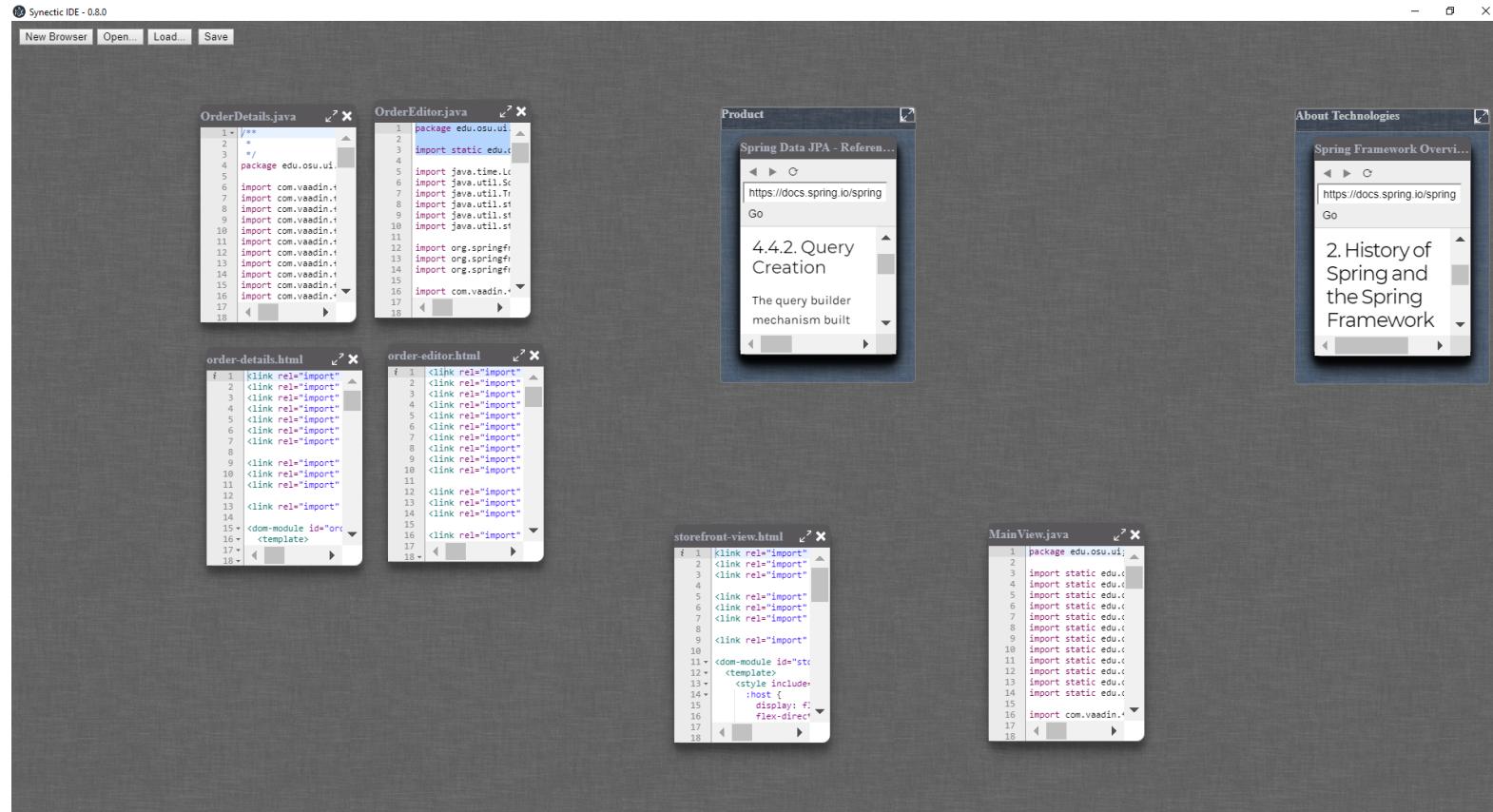
Debugger Canvas



Synectic

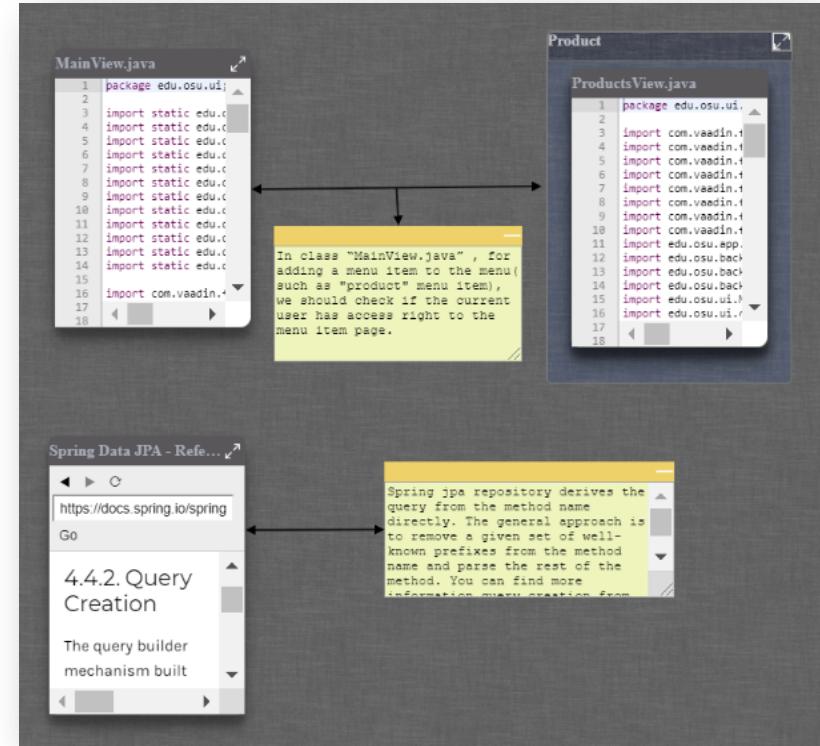


Synectic : A canvas-based IDE



Annotation Overlay

- **Annotation notes** for capturing design rationale, expected API usage patterns, corner-cases, etc.
- **Annotation links** for connecting notes to cards or groups
- Multiway connections between annotations and cards to describe relationships.



Study Design - RQs

- RQ1: How do annotations affect code comprehension among newcomers?
 - Do annotations increase the accuracy of responses?
 - Do annotations reduce the time to task completion?
 - Do annotations reduce cognitive load?



User Study

Controlled lab study

- Between-subject design
- 22 participants (graduate students)
- 4 code comprehension tasks

Synectic treatment

11 participants

4 code comprehension tasks

Onboarding information added as annotations

Eclipse treatment

11 participants

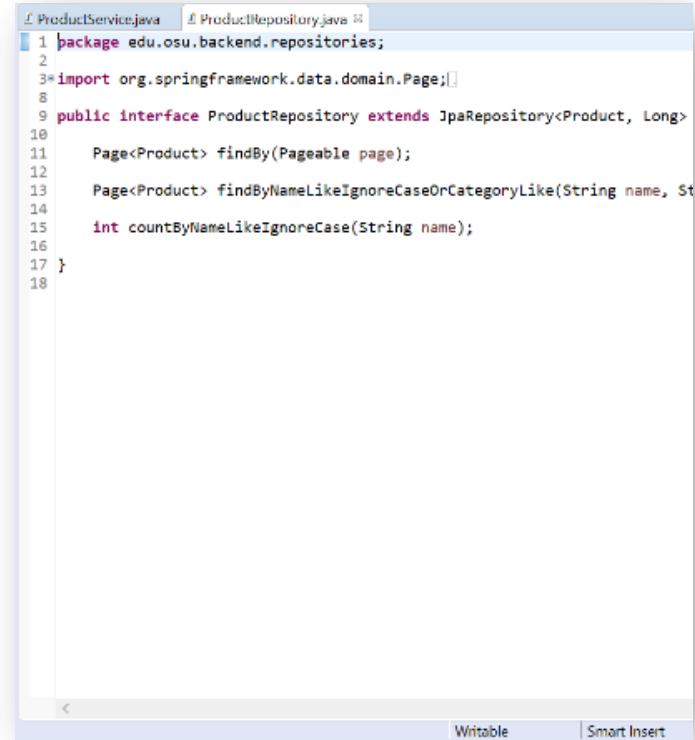
4 code comprehension tasks

Onboarding information added as text document



Study Design – Tasks

- Code Comprehension Task
 - Navigation portion
 - Comprehension portion
- Designed as onboarding tasks
 - Locating code related to a feature
 - Learning how to make changes to those features



The screenshot shows a code editor with two tabs: `ProductService.java` and `ProductRepository.java`. The `ProductRepository.java` tab is active, displaying the following Java code:

```
1 package edu.osu.backend.repositories;
2
3 import org.springframework.data.domain.Page;
4
5 public interface ProductRepository extends JpaRepository<Product, Long>
6 {
7     Page<Product> findBy(Pageable page);
8
9     Page<Product> findByNameLikeIgnoreCaseOrCategoryLike(String name, String category, Pageable page);
10
11     int countByNameLikeIgnoreCase(String name);
12
13 }
```

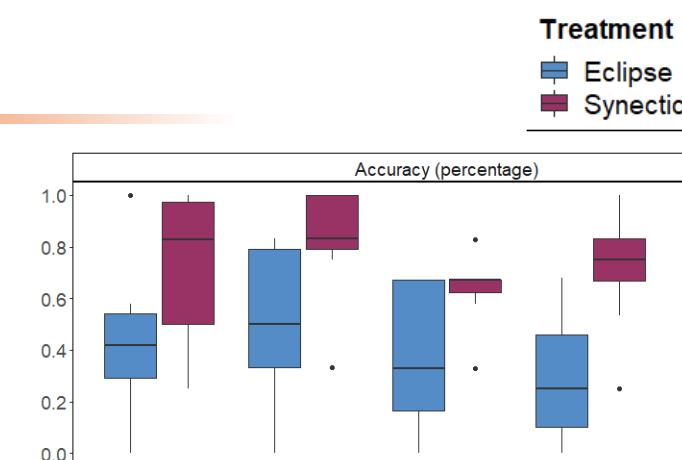
The code editor interface includes tabs for both files, a status bar at the bottom with "Writable" and "Smart Insert" buttons, and a navigation bar at the top.



Results

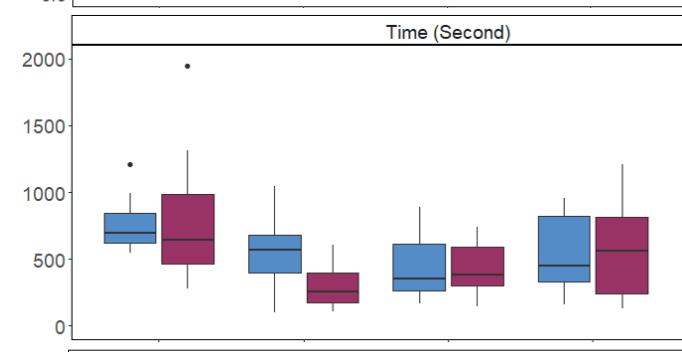
🎯 Accuracy

Rank Based Non-Parametric (RBNP) ANOVA test
(p -value < 0.001, statistic = 19.46488)



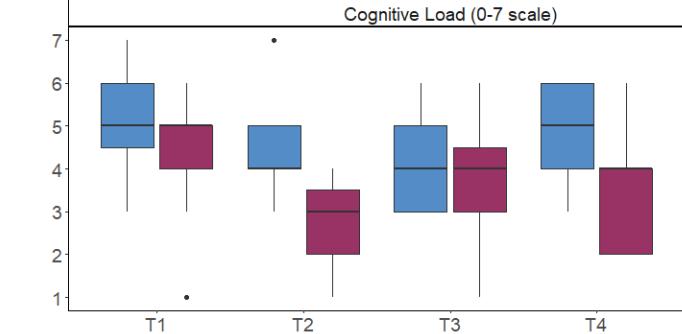
⌚ Time

RBNP ANOVA test
(p -value = 0.22, statistic = 1.607723)



🧠 Cognitive Load

RBNP ANOVA test
(p -value = 0.003, statistic = 11.52591)



Discussion

- Quantitative results
 - Accuracy & Cognitive Load differences were significant
 - Time differences were not statistically significant
- Qualitative results
 - Sillito et al.'s four stages of comprehension model^[1] to explain comprehension
 - Information Foraging Theory (IFT) to explain navigation

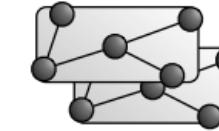
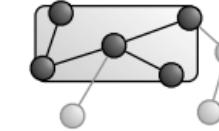
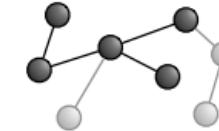
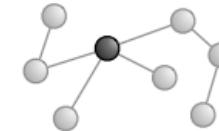
[1] Sillito, J., Murphy, G. C., & De Volder, K. (2008). Asking and answering questions during a programming change task. *IEEE Transactions on Software Engineering*, 34(4), 434-451.



Discussion – Stages of comprehension

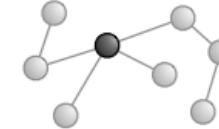
Sillito et al. identified 4 categories of comprehension:

1. Finding the initial focus point
2. Building on those focus points
3. Understanding the concepts between related entities
4. Understanding concepts across multiple groups of related entities



Discussion – Stages of comprehension

1. Finding the initial focus point



The screenshot shows the Synetic IDE interface with several windows open:

- New Browser**: Shows a note about MainView.java containing system menu items.
- About Technologies**: Shows a note about different technologies used in the project.
- StoreFront**: Contains two code editors:
 - OrderEditor.java**: A Java file with code for an order editor.
 - MainView.java**: A Java file with code for a main view. A callout box explains that it contains a method to add menu items to the main menu.
- Product**: Contains a code editor for "Spring Data JPA - Reference" and a note about query creation. A red arrow points from this window to the MainView.java window.
- 4.4.2. Query Creation**: A detailed note explaining the @Secured annotation and its role in access control.



Discussion – Stages of comprehension

2. Building on those focus points



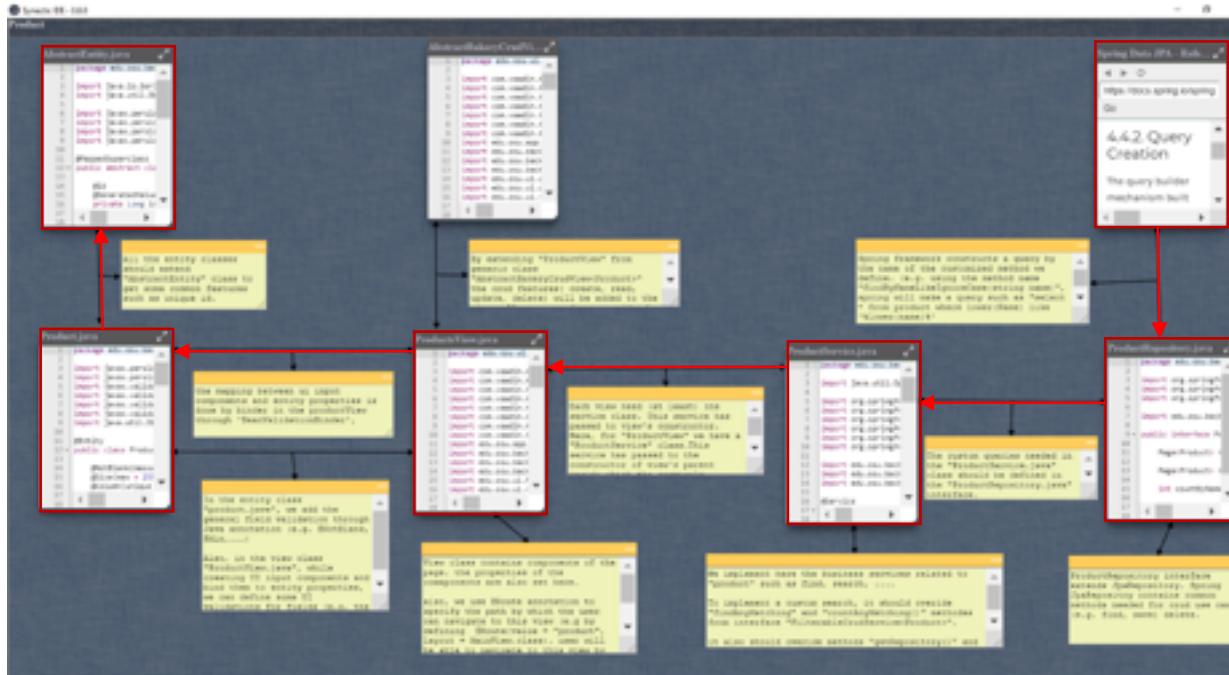
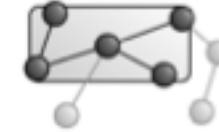
The screenshot shows a Java IDE (IntelliJ IDEA) with several projects open:

- Product**: Shows a class named `Product` with annotations like `@Entity`, `@Table`, and `@Id`. A callout notes: "Add this entity class to 'ProductRepository' class to get some common functions such as `save()`".
- ProductView**: Shows a class named `ProductView` with annotations like `@Entity`, `@Table`, and `@Id`. A callout notes: "By extending 'ProductView' from 'ProductViewBase' and implementing 'ProductViewInterface' the view interface will be added to the mapping between no input properties and view properties through inheritance (e.g. Immediate, Blaz...)".
- ProductService**: Shows a class named `ProductService` with annotations like `@Service`, `@Transactional`, and `@Autowired`. A callout notes: "Each class needs its own DAO interface class. This interface has to be passed to the constructor of view's parent".
- ProductController**: Shows a class named `ProductController` with annotations like `@Controller`, `@RequestMapping`, and `@GetMapping`. A callout notes: "The action classes needed as arguments for controller methods should be defined as 'ProductRepository.java'".
- ProductRepository**: Shows a class named `ProductRepository` with annotations like `@Repository`, `@Transactional`, and `@Query`. A callout notes: "We implement basic CRUD operations using 'JpaRepository'. To implement a custom method, we should create 'ProductRepositoryCustom' interface from interface 'ProductRepository' and then override certain methods (e.g. `findByName()`)".
- Spring Data JPA - Rule**: A browser window showing the URL <https://docs.spring.io/spring-data/jpa/docs/2.5.0/reference/html/>.



Discussion – Stages of comprehension

3. Understanding the concepts between related entities



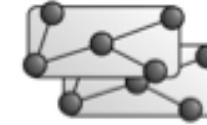
The screenshot shows the Spring IDE interface with several windows open, each containing code snippets and annotations:

- Product.java**:
 - Annotations: `@Entity`, `@Table(name = "product")`, `@Id`, `@GeneratedValue(strategy = GenerationType.IDENTITY)`.
 - Text: "All the entity classes must implement 'Serializable' class to get some common functions such as clone()."
- ProductView.java**:
 - Annotations: `@Entity`, `@Table(name = "product")`, `@Id`, `@GeneratedValue(strategy = GenerationType.IDENTITY)`.
 - Text: "The mapping between no input properties and entity properties is done by beans in view controller through 'TransientController'."
 - Text: "Is the entity class 'ProductView.java' and the properties are mapped through view controller i.e., 'TransientController'. Also, in the view class 'ProductView.java', while implementing the properties and need to set entity properties, we can define some UI representation for entity i.e., ->".
- ProductViewDetail.java**:
 - Annotations: `@Entity`, `@Table(name = "product")`, `@Id`, `@GeneratedValue(strategy = GenerationType.IDENTITY)`.
 - Text: "This class contains components of the page, the properties of the components are also set here. Also, we use Beans annotation to specify the path by which the component needs to be mapped to the entity using annotation i.e., 'path'.
 - Text: "layer = ModelLayer.class" come will be used to distinguish between components and entities."
- ProductController.java**:
 - Annotations: `@Controller`, `@RequestMapping(value = "/product")`.
 - Text: "The mapping between no input properties and entity properties is done by beans in view controller through 'TransientController'."
 - Text: "The action method needed as a class should be defined as 'TransientController.java'."
- ProductRepository.java**:
 - Annotations: `@Repository`, `@Transactional`.
 - Text: "We implement base class 'BaseRepository' relates to 'Product' such as find, search etc."
 - Text: "To implement a custom search, we should create 'TransientSearch' and 'TransientProduct' methods from interface 'TransientController'."
 - Text: "It also needs custom service 'TransientService' and its logic needs to be implemented."
- TransientController.java**:
 - Annotations: `@Controller`, `@RequestMapping(value = "/product")`.
 - Text: "TransientController contains common functions for both components i.e., find, search etc."



Discussion – Stages of comprehension

4. Understanding concepts across multiple groups of related entities



The screenshot shows the Synetic IDE interface with several windows open:

- MainView.java**: Contains code for adding menu items to the main menu.
- OrderEditor.java**: Shows the code for the OrderEditor component.
- MainView.java** (another instance): Shows annotations for securing menu items.
- Product**: A window showing the "Spring Data JPA - Reference" and "4.4.2. Query Creation".
- About Technologies**: A window showing the "Spring Framework Overview" and "2. History of Spring and the Spring Framework".
- StoreFront**: A window showing the "MainView.java" code again.

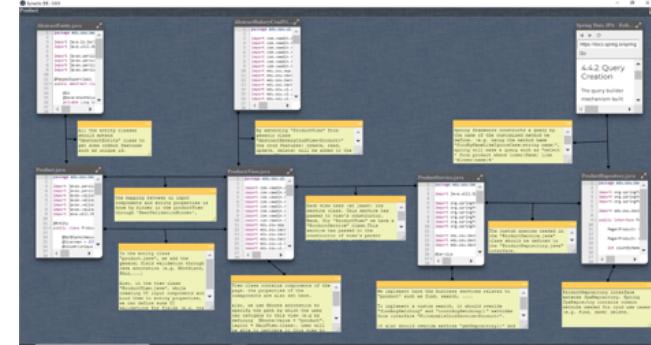
Annotations and callouts highlight specific parts of the code and windows, such as:

- "For the Store front page, we create template UI component using polymer library."
- "MainView.java contain the list of all system menu items which are accessible for the user has logged in. we add menu items to the main menu in the "configure()" method. for each menu item, we specified icon, title, and route(url)."
- "We have used different technologies in this project. here, you can find some general information about the general ones."
- "To get familiar with general pattern used for adding a new page, here we explain what we also project related artifacts to the project. We use a 3-tier architecture(view->service->repository)."
- "In class "MainView.java" for adding a menu item to the menu (such as "product" menu item), we should check if the current user has access right to the same item page by "SecurityUtil.isAccessGranted(ProductView.class)". The annotation "@Secured" on the above of the class declaration on "ProductView.java" tells the framework which roles can access this."



Summary

- Annotations in a canvas-based IDE resulted in:
 - Lower cognitive load among newcomers
 - More accurate comprehension responses
 - Required no additional time compared traditional IDEs
- Design challenges for annotations within IDEs:
 - Manage different types of artifacts
 - Locate relevant information in different places
 - Understand the relationships between artifacts



“Right information, at the right place, and the right time”



 @nomaticdev



<https://nomatic.dev/>