Building a complete PDF editor with features like form handling, canvas drawing, annotation tracking, OCR, document conversion, version control, and signature features and the technology stack that can be used to achieve them. Please note that building a complete PDF editor is a complex project, and the implementation details may vary based on specific requirements and support is a comprehensive task. Below is a detailed plan outlining the required technologies.

### Features:

**PDF Viewer and Editor: 30 hrs**

* + Display PDF documents.
  + Navigate through pages.
  + Zoom in/out.
  + Highlight text and add comments.
  + Insert, delete, and reorder pages.
  + Track changes and version control.

**Form Handling: 40 hrs**

* + Fillable form fields.
  + Support for various form field types (text fields, checkboxes, radio buttons, etc.).
  + Form data extraction and manipulation.

**Canvas for Drawing: 20 hrs**

* + Annotate PDFs with freehand drawings.
  + Add shapes, lines, and text on the canvas.

**Annotation Tracking: 30 hrs**

* + Track and manage annotations.
  + Collaborative annotation support for real-time collaboration.

**OCR (Optical Character Recognition): 30 hrs**

* + Extract text from PDF documents.
  + Convert images to text for OCR capabilities.

**Document Conversion: 50 hrs**

* + Convert Word and Excel documents to PDF.
  + Image to PDF conversion.

**JavaScript Library: 50 hrs**

* + Provide a comprehensive JavaScript library.
  + Token-based authentication for embedding in various projects.
  + Well-documented API for easy integration.

**Cross-Platform Compatibility: 20 hrs**

* + Web app support (compatible with major browsers).
  + Mobile app support for iOS and Android platforms.

**Signature and Certificates: 30 hrs**

* + Add digital signatures to PDF documents.
  + Certificate-based authentication.

### Technology Stack:

**Frontend:**

* + **JavaScript Framework:** Choose a modern JavaScript framework/library such as React, Angular, or Vue.js for the frontend development.
  + **PDF Rendering:** Use a PDF rendering library like PDF.js for displaying PDFs.
  + **Canvas Drawing:** Leverage HTML5 Canvas API for drawing functionalities.

**Backend:**

* + **Server-side Language:** Choose a backend language such as Node.js, Python, or Java.
  + **OCR Service:** Integrate with OCR services like Tesseract or use cloud-based OCR APIs.

**Document Conversion:**

* + Use libraries like **mammoth.js** for Word to PDF conversion.
  + For Excel to PDF, consider libraries like **exceljs**.

**Version Control:**

* + Implement a version control system on the server side to track changes.

**Authentication:**

* + Use token-based authentication (JWT) for secure embedding in various projects.

**Database:**

* + Choose a database (e.g., MongoDB, PostgreSQL) for storing document metadata, annotations, and user data.

**Collaboration:**

* + Implement real-time collaboration using technologies like WebSockets or a dedicated collaboration library.

**Cross-Platform Mobile App:**

* + Use a framework like React Native or Flutter for building cross-platform mobile applications.

**Signature and Certificates:**

* + Implement digital signature features using cryptographic libraries.
  + Integrate with certificate authorities for certificate-based authentication.

**Deployment:**

* + Deploy the application on cloud platforms like AWS, Azure, or Google Cloud for scalability and accessibility.

### Comparison with PSPDFKit:

Compare the features and capabilities of your PDF editor with PSPDFKit to ensure that your solution meets or exceeds the industry standards. Evaluate factors such as performance, user experience, and security.

Building such a comprehensive PDF editor requires careful planning, coding, and testing. It's advisable to break down the development into manageable milestones and conduct regular testing to ensure each feature works seamlessly. Additionally, consider ongoing maintenance and updates to keep up with evolving standards and technologies.