

Below is a detailed, forward-thinking blueprint for building a **Report Templating Module (RTM)** in Django that allows non-technical users to design invoice templates based on predefined stored procedures. It draws lessons from leading reporting tools such as **SAP Crystal Reports 2025** and **Microsoft's SQL Server Reporting Services (RDL/RDLC)**, and then proposes a modern solution that surpasses them.

1. What major competitors offer

Competitor	Key strengths & recent changes (2025-2024)
SAP Crystal Reports 2025	<ul style="list-style-type: none">• New SAP BW (DSL) driver and start-page shortcuts to quickly create reports community.sap.com.• Query panel enhancements allow database-side sorting and ranking for faster report performance community.sap.com.• BI Launchpad tile for launching the classic designer and a desktop viewer with improved performance community.sap.com.• Security and platform updates community.sap.com.
Microsoft SQL Server Reporting Services (SSRS / RDL)	<ul style="list-style-type: none">• Consolidated under Power BI Report Server from SQL Server 2025; focus on paginated reports learn.microsoft.com.• SSRS 2022 emphasizes accessibility (Windows Narrator support), security enhancements, and an updated Angular web portal learn.microsoft.com.• Alt-text support for report elements learn.microsoft.com, custom headers learn.microsoft.com, and transparent data encryption for the catalog learn.microsoft.com.• Comments on reports and an OpenAPI-compliant REST API to integrate reports into applications learn.microsoft.com.
Modern PDF generation libraries	<ul style="list-style-type: none">• ReportLab: supports dynamic PDF generation, converting XML into PDF, vector graphics, and charts apitemplate.io.• Playwright: cross-browser PDF creation, headless execution and fine-grained control over page size, orientation and margins apitemplate.io.• PDFKit: simple wrapper around <code>wkhtmltopdf</code> supporting vector graphics, image embedding, text wrapping, and PDF security options apitemplate.io.• XHTML2PDF: converts HTML/CSS into PDFs with advanced styling, page breaks, headers and footers apitemplate.io.

These tools demonstrate the value of robust data connectors, interactive designers, advanced formatting (sorting, ranking, charts), accessibility, strong security and open APIs.

2. Core requirements for your Django RTM

1. **Template design must be WYSIWYG and intuitive** – non-developers should be able to drag-and-drop fields, tables, images, barcodes and charts onto a canvas, adjust styles,

and immediately preview the result.

2. **Connect to predefined stored procedures** – the module should list available procedures (for example, invoice procedures) and infer their output schema. Users need a clear interface to choose a procedure, define parameters, and bind returned fields to placeholders in their templates.
3. **Pixel-perfect, printable output** – support complex layouts (page headers/footers, nested tables, repeated sections) and export to PDF, Excel or HTML with accurate sizing and positioning.
4. **Dynamic formulas and expressions** – allow users to define conditional logic, grouping, totals and calculations (similar to Crystal Reports' formula editor and RDLC expressions).
5. **Multi-format charts** – embed bar/line charts and sparklines similar to competitor offerings; rely on open-source charting (Chart.js, Plotly) or Python libraries.
6. **Accessibility and internationalization** – alt-text for images and charts, language translation, date/number formatting; these features rival SSRS' accessibility improvements learn.microsoft.com.
7. **Security and versioning** – enforce row-level security based on user roles, log all template modifications, and store template versions for rollback.
8. **Open REST API** – external systems must be able to generate or schedule reports via API; this mirrors SSRS' REST API learn.microsoft.com.
9. **Scalability and multi-tenant support** – design for growth: asynchronous processing via Celery/Redis; optionally support multiple tenants if your ERP has multi-tenant architecture.

3. High-level architecture

3.1. Data & procedure layer

- **Procedure registry:** create a `StoredProcedure` model capturing procedure name, description, connection info, input parameters and output schema. A management script can introspect your DB and populate this table.
- **Data adapters:** wrap calls to procedures and other data sources (SQL views, REST APIs); unify them under a `DataSource` abstraction. Provide caching and concurrency

control.

3.2. Template design and storage

- **Template model:** store metadata (name, owner, creation date), version history (JSON field storing the design), associated procedure and parameter mapping.
- **WYSIWYG editor:** integrate a modern JS-based editor (e.g., GrapesJS or Tiptap with custom plugins) into Django. This editor should:
 - Display the output schema of the selected procedure and let users drag fields into the layout.
 - Provide blocks for static text, images, barcodes, page numbers, charts and tables.
 - Allow CSS styling (fonts, colours, alignment) and define page headers/footers.
 - Support repeating sections (e.g., invoice line items) and nested lists.
 - Include a formula builder for arithmetic, conditional logic and aggregation (similar to Excel).
- **Template engine:** represent templates as HTML/CSS with placeholders using Jinja2 or Django templates. When rendering, supply the procedure results and evaluate formulas.

3.3. Rendering and export

- **HTML preview:** render templates on the fly within the browser for instant feedback.
- **PDF generation:** choose a library that supports complex layouts and styling.
XHTML2PDF offers full HTML/CSS support and page breaksapitemplate.io, while
ReportLab can embed charts and vector graphicsapitemplate.io. A hybrid approach could use HTML-to-PDF via PDFKit or Playwright for quick exports and fall back to ReportLab for advanced graphing.
- **Chart rendering:** generate charts client-side using Chart.js for preview and then embed as images in the exported PDF. For server-side rendering, use Matplotlib or Plotly to produce SVG/PNG.

3.4. Parameter input and data binding

- **Dynamic forms:** for each stored procedure, auto-generate a form (with date pickers, dropdowns, etc.) to collect parameters. Provide default values and validation.
- **Data mapping:** after selecting a procedure, auto-populate a “fields panel” in the editor. Dragging a field onto the template creates a placeholder (e.g., `{
procedure.customer_name } }`) and sets up the data binding. For lists (invoice line items), allow repeating groups that iterate over the array returned by the procedure.

3.5. Versioning, scheduling and API

- **Version control:** each template update writes to a new version record. Provide diff/rollback functions and track who made changes.
- **Report generation jobs:** integrate Celery to run heavy rendering tasks asynchronously; store job status and allow users to download completed reports.
- **REST API:** expose endpoints to list templates, submit parameter values, and retrieve generated files. Document them using OpenAPI/Swagger so other services can integrate easily—something SSRS only recently added learn.microsoft.com.

3.6. Security and multi-tenancy

- **Authentication:** rely on Django’s auth system. Support SSO or OAuth2 if needed.
- **Authorization:** restrict which procedures and templates each role can access. For multi-tenant setups, scope queries by tenant ID (your ERP schema shows tenant/organization tables).
- **Data protection:** encrypt stored templates, enforce HTTPS, and optionally support PDF encryption features (PDFKit supports encryption apitemplate.io).
- **Audit trail:** log every report run with timestamp, user, parameters and status for compliance.

4. Implementation plan (phased)

1. **Analyse existing procedures** and define the `StoredProcedure` and `DataSource` models. Write a management command to introspect the DB and update the registry.

2. **Design database schema** for templates, versions, and user rights. Use Django migrations.
3. **Integrate WYSIWYG editor**: embed GrapesJS or another library into a Django template; write custom blocks for report elements (fields, loops, charts). Provide a side panel listing procedure fields and parameters.
4. **Build template engine**: implement HTML generation using Jinja2; parse the template JSON from the editor and produce a compiled HTML template.
5. **Implement data binding and formula evaluation**: supply the procedure result as context to Jinja2; implement a simple expression parser for formulas.
6. **Add preview & PDF export**: start with PDFKit or XHTML2PDF for converting HTML to PDF; ensure page breaks and headers/footers work [apitemplate.io](#); integrate ReportLab for charts and vector graphics [apitemplate.io](#).
7. **Develop parameter UI**: auto-generate forms; handle validation; feed parameters to the stored procedure call.
8. **Implement versioning**: store snapshots of template definitions; build UI to view history and restore older versions.
9. **Provide scheduling & asynchronous tasks**: configure Celery with Redis; allow users to schedule report runs and download results later.
10. **Expose REST API**: design endpoints for listing templates, submitting parameters, starting exports, and retrieving completed files; use DRF (Django REST Framework) with token-based auth.
11. **Security & multi-tenancy**: add permission checks to all endpoints; filter data by tenant; optionally encrypt generated PDFs.
12. **Enhance with AI & analytics (future work)**: incorporate natural-language query to help design templates; use generative AI to suggest layouts based on sample invoices. Provide dashboards to track report usage.

5. Why this approach surpasses competitors

- **Modern, browser-based designer** – unlike Crystal Reports' desktop-only designer, your RTM uses a web-based WYSIWYG editor accessible anywhere, eliminating client installations and aligning with remote work.

- **Flexibility in data sources** – the procedure registry can call SQL stored procedures, REST APIs or Python functions, whereas Crystal Reports focuses on SAP/BW drivers community.sap.com and SSRS is tied to SQL Server.
 - **HTML/CSS layout engine** – leveraging HTML5 and CSS gives you the ability to create highly customized, pixel-perfect layouts and responsive designs; advanced libraries like XHTML2PDF support page breaks and headers/footers apitemplate.io.
 - **Open APIs and automation** – your REST API for report generation parallels SSRS' new REST API learn.microsoft.com but is fully open-source and customizable.
 - **Extensible charting and analytics** – integrate Chart.js or Plotly for modern charts; Crystal Reports' chart types are relatively fixed.
 - **Accessibility and internationalization** – by following web standards and alt-text best practices you match SSRS' accessibility improvements learn.microsoft.com and go beyond by enabling multi-language templates via Django's i18n.
 - **Security and audit** – built-in encryption options (through PDFKit apitemplate.io) and row-level security ensure compliance; versioning and audit logs give traceability.
 - **Open-source ecosystem** – by building on Django and widely adopted libraries, you avoid vendor lock-in and benefit from community support.
-

By following this plan, you can create a powerful and user-friendly Report Templating Module within Django that not only meets the needs of invoice printing but also outperforms traditional tools like Crystal Reports and Microsoft RDL/RDLC.